MIFARE Sector Decoder

Plug-in User Guide

Contents

Read-a-Card

Overview

Read-a-Card plug-ins give you the ability to extend the application's functionality to meet your own custom needs. By default, Read-a-Card will read the CSN/UID of a contactless MIFARE card. But if instead you wish to read, for example, a specific sector of a MIFARE card, then Read-a-Card can return that data, formatted according to your requirements, using a software plug-in. You can also optionally store access keys securely in hardware.

The MIFARE Sector Decoder plug-in supports reading and decoding IDs stored in sectors on MIFARE 1K, 4K, Ultralight, Ultralight C and NTAG203 card type families.

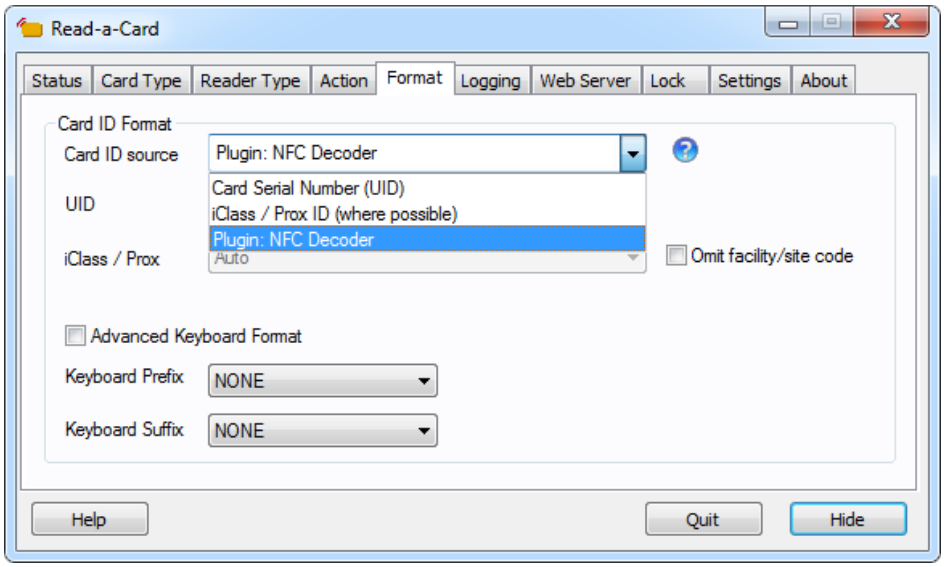
This document will guide you through the configuration and use of the Read-a-Card MIFARE Sector Decoder plug-in.

Read-a-Card and plug-in DLLs

Read-a-Card supports plug-in extensions in the form of DLL files. These DLL files must reside in the "Plugins" folder within the Read-a-Card application folder. (e.g. C:\Program Files\Read-a-Card\Plugins).

The MIFARE Sector Decoder plug-in is included with recent versions of the Read-a-Card (v3.0.7 onwards). The plug-in DLL (RACFormatDecodeA.dll) is copied to the Plugins folder when Read-a-Card is installed.

Read-a-Card will display all available plug-ins in the Card ID source drop-down box on the Format tab. All plug-ins will have the "Plug-in:" prefix. Only one plug-in can be in use at any time.



Some plug-ins may require that Read-a-Card is licensed for the plug-ins to function; others may require specific configuration data. If a plug-in is unable to work for either of these reasons, it will appear in the drop-down list “greyed out”.

Clicking on the help icon (?) next to the format drop-down box will pop-up a information dialog which indicates the current selected plug-in format name, version, format name and other features.

Plug-in configuration

Read-a-Card format plug-ins may be configured by using a Windows configuration file.

In the case of the MIFARE Sector Decoder plug-in, this file is named “RACFormatDecodeA.ini”. This ini file should be located in the same folder as the Read-a-Card ini file:

On Windows XP/2000:
C:\Documents and Settings\All Users\Application Data\Read-a-Card\

On Windows Vista/Windows 7/Windows 8 :
C:\ProgramData\Read-a-Card\

In order for this plug-in to work, this file must at least contain the following:

```
[Config]
ReadFromIniFile=1
PluginVersion=1.2
PluginName=MIFARE Sector Decode
```

For a complete example configuration file, see the end of this document.

Plug-in name

The Read-a-Card format name displayed in the Card ID source drop-down box can optionally be configured in the ini file as follows:

```
[Config]
FormatUIName=My name for this plug-in
```

(note that there must only be one [Config] section in the ini file).

Read sector data configuration

For each of the main card types, a unique ini file section name is used to configure how data is to be read from the card. This allows different card types to have different data locations, keys and format conversion.

The following sections are recognised:
[MIFARE_1K]
[MIFARE_4K]
[MIFARE_UL]

The MIFARE_UL type covers MIFARE Ultralight, Ultralight C, NTAG203 and other similar related card types.

Note that not all readers support reading data from Ultralight family cards beyond the first 64 bytes.

For example, with a MIFARE 1K card the following options might be used:

```
[MIFARE_1K]
ReadData=1              ; Set this to 1 to read sector data from this card type
StartSector=0           ; Integer. The sector to start reading from.
StartBlock=1            ; Integer. The block to start reading from.
StartOffset=0           ; Integer. The byte offset within the block to start reading at
BytesToRead=8           ; Integer. Number of bytes to read
OutputFormat=0          ; Integer. How to format the resulting data (see below)
KeyType=1               ; Integer:  0 = ASCII String, 1 = Binary data as ASCII hex pairs
KeyA=1A2B3C4D5E6F       ; Optional KeyA
KeyB=FFFFFFFFFFFF       ; Optional KeyB
```

To enable the reading of a configured file or block/s the ReadData option must be set to 1 for every card type. If this option is not set for the appropriate card type, the card's serial number (UID) will be returned.

The memory in a MIFARE 1K or 4K card is organized in numbered sectors. Each sector contains 4 or 16 blocks (depending on card type and sector number) and every block contains 16 bytes of data.

Examples:

```
StartSector=3
StartBlock=1
```

The above configuration will start reading from sector 3, block 1 (effectively this is block number 13).

Any combination of sector and block numbering can be used. For example:

```
StartSector=
StartBlock=13
```

The above configuration will read the same block as the previous example.

A block of memory in a MIFARE 1K or 4K card contains 16 bytes of data (bytes numbered 0 to 15). The plug-in can be configured to start a read from a certain byte number within that block by configuring the StartOffset.

By contrast, the memory of MIFARE Ultralight family cards is arranged in blocks of 4 bytes.

Reading data that runs over multiple sectors and/or blocks will be automatically handled (e.g. BytesToRead may be greater than 16).

The resulting bytes will be formatted according to the OutputFormat value. This provides the same formatting options for formatting the sector data (or UID) that Read-a-Card offers for formatting the UID. It can have the following values:

0 Decimal reversed (default)
This option takes the last 4 bytes of data (or all bytes if fewer than 4 have been read) and interprets them as a little-endian binary 32 bit number (i.e. the last byte in the sequence has the most significant value). The resulting 32 bit value is formatted as a 10 digit decimal number.

1 Decimal
This option takes the first 4 bytes of data (or all bytes if fewer than 4 have been read) and interprets them as a big-endian binary 32 bit number (i.e. the first byte in the sequence has the most significant value). The resulting 32 bit value is formatted as a 10 digit decimal number.

2 Hex reversed
This option outputs all bytes of the data in hexadecimal (two ASCII numeric or uppercase hex characters per byte) in reverse order.

3 Hex standard
This option outputs all bytes of the data in hexadecimal (two ASCII numeric or uppercase hex characters per byte) in the same order that they were read.

4 Decimal 64 bit
This option takes the first 8 bytes of data (or all bytes if fewer than 8 have been read) and interprets them as a little-endian binary 64 bit number (i.e. the last byte in the sequence has the most significant value). The resulting 64 bit value is formatted as a decimal number with no leading zeros.

5 Decimal 64 bit reversed
This option takes the last 8 bytes of data (or all bytes if fewer than 8 have been read) and interprets them as a big-endian binary 64 bit number (i.e. the first byte in the sequence has the most significant value). The resulting 64 bit value is formatted as a decimal number with no leading zeros.

6 ASCII

This option outputs all bytes of data as ASCII printable characters. Any byte values that do not represent ASCII printable characters will be output as a '.' character.

MIFARE access keys

With MIFARE 1K and 4K cards, access to a sector's data may require one or two authentication keys. These keys (KeyA and KeyB) can be configured for each card type. The keys can be expressed as either an ASCII character string or as a binary value hex pairs indicated by the KeyType setting:

0 = ASCII String e.g. my key
1 = Binary data as ASCII hex pairs e.g. 6d79206b6579

For example:

```
KeyType=0
KeyA=my key
```

Or

```
KeyType=1
KeyA=6d79206b6579
```

If they are required, the KeyType KeyA and KeyB options should be specified under the appropriate section for the card type that they are to be used with (thereby allowing different keys to be used with different card types).

Securing your access keys

It may not be appropriate to expose the access keys for your cards in a plaintext configuration file. For this reason, we can supply an encrypted, secured copy of your configuration file on a smart card security module (or "SAM"). When configured and licensed in this manner, the Read-a-Card MIFARE Sector Decoder plug-in will read all of its configuration data, including the access keys, from the SAM in a secure manner.

Please contact Dot Origin or your supplier for more information on how to obtain a SAM containing your specific configuration options.

Decoding facility codes and card numbers

In some cases, it may be required to extract a facility code and card number from the bytes read from the card's memory. This plug-in provides a number of ways of doing this. The facility code and card number can then be output as required using Read-a-Card's advanced keyboard format or URL/HTTP/command actions. If these features are used, the resulting ID (as shown on Read-a-Card's status tab or inserted using

the %n option in the various Read-a-Card actions) will comprise the facility code followed by the card number.

As with most of the options, different options can be applied to each card types.

DigitMask

A DigitMask value can be used to indicate how the digits of the result of the OutputFormat process should be distributed between the facility code and card ID. For example:

```
DigitMask=xxfff-cccccx
```

The DigitMask is applied to the result of formatting the data according to the OutputFormat value. Only f (facility code) and c (card number) characters in the digit mask have any meaning. Any other characters indicate that the corresponding digit is to be ignored.

So, for example, with the above DigitMask setting specified, if the result of reading and formatting the data is the value “123456789012”, the resulting facility code will be 345 and the resulting card number will be 78901.

The following options allow the facility code and/or card number to be formatted with leading zeros:

```
FACDigits=4
CardNumDigits=7
```

Using the above example, these options would yield a facility code of 0345 and a card number of 0078901. The resulting ID (combined facility code and card number) would be 03450078901.

BitMask

More complex bitwise facility code and card number decoding can be performed with the BitMask option. Note that the BitMask and DigitMask options cannot be used together for a given card type (if both are specified then only the DigitMask option will be used).

```
BitMask=-----pfffffffffffffcccccccccccccccccccp--------
ByteOrder=0
BitOrder=0
```

The BitMask value is applied bitwise to the binary data read from the card, only characters c and f are used to indicate bits to be assigned to the card number or facility code respectively. All other characters indicate a bit to be ignored.

The OutputFormat option is not used in this case. Instead the bytes read from the card are first converted into a 64 bit integer value. The ByteOrder option determines

whether the first or last byte should be considered the most significant (0=first byte is most significant, 1=first byte is least significant).

The BitOrder value may then used to reverse the order of the bits in the resulting facility code and card number.

BitOrder=0 implies that the leftmost character of the BitMask is matched to the most significant bit of the data.

BitOrder=1 implies that the rightmost character of the BitMask is applied to the least significant bit of the data and that the bit order is reversed (i.e. the least significant bit is treated as the most significant in the resulting facility code and card number)

So for example, reading 6 bytes of data with the following setting:

BytesToRead=6

The following six bytes of data are read from the card's memory:
C6 8B 06 92 48 14

ByteOrder=1 will reverse the byte order resulting in a 64 bit hex value of

144892068BC6

In binary this is:
0001 0100 0100 1000 1001 0010 0000 0110 1000 1011 1100 0110

Matching this to a BitMask and BitOrder as follows:
BitMask=-----pfffffffffffffccccccccccccccccccp
BitOrder=0

We get:
000101000100100010010010000001101000101111000110
-----pfffffffffffffccccccccccccccccccp

Facility code =000100100010001b = 489h = 1161 (decimal)
Card number = 0010000001101000101b = 10345h = 66373 (decimal)

The following options allow the facility code and/or card number to be formatted with leading zeros:
FACDigits=4
CardNumDigits=6

So using the above values the resulting facility code would be 0489, the card number would be 066373 and the overall combined ID would be 0489066373.

As another example, consider the following BitMask with BitOrder=1 used on the same data:
BitMask=pccccccccccccccccccfffffffffffffp
BitOrder=1
ByteOrder=1

In this case we would get the following match:
0001010001001000100100100000011010001011110001 10
pccccccccccccccccccfffffffffffffp

Note that the BitMask is right-aligned with the data due to BitOrder=1. This option also reverses the bit order giving:

Facility code =11000111101000b = 31E8h = 12776 (decimal)
Card number = 10110000001001001 00b = 58124h = 360740 (decimal)

In this case the FACDigits=4 and CardNumDigits=6 options will not cause any additional leading zeros to be added because the values are already contain at least this number of digits.

The resulting combined ID will be 12776360740.

Fixed facility codes

A fixed facility code for all cards of a particular type can be configured by specifying the following setting:

FAC=123 ; (String. The fixed facility code)

Read-a-Card actions

You can dictate which Read-a-Card actions are taken (depending on the type of card) using the ActionMask setting. This allows the plug-in settings for a card type to suppress actions settings that would otherwise be enable by Read-a-Card's configuration settings.

ActionMask=00 (Hex pair e.g. "7E" for Action Mask 0111 1110)

The following is the bitwise representation of the Action Mask:

Action Mask:
X X X X X X X X
7 6 5 4 3 2 1 0
Bit 7 = Not Used
Bit 6 = Log Action 0 = Enable, 1 = Disable
Bit 5 = URL/HTTP Action 0 = Enable, 1 = Disable
Bit 4 = Command Action 0 = Enable, 1 = Disable
Bit 3 = Sound Action 0 = Enable, 1 = Disable

Bit 2 = Clipboard Action 0 = Enable, 1 = Disable
Bit 1 = Keyboard Action 0 = Enable, 1 = Disable
Bit 0 = Not used

Example configuration file

; Here is example content for the RACFormatDecodeA.ini configuration file:
[Config]
ReadFromIniFile=1
PluginVersion=1.2
PluginName=MIFARE Sector Decode
FormatUIName=ACME Ltd Card Format

[MIFARE_1K]
ReadData=1
StartSector=0
StartBlock=1
StartOffset=4
OutputFormat=3
BytesToRead=8
FormatName=MIFARE 1K (sector decode)
ByteOrder=1
BitOrder=0
DigitMask=fffccccc
FACDigits=3
CardNumDigits=5

[MIFARE_4K]
ReadData=1
StartSector=0
StartBlock=1
BytesToRead=16
FormatName=MIFARE 4K (sector decode)
ByteOrder=1
BitOrder=0
BitMask=------ffffffffffffffcccccccccccccccccccp--------
FACDigits=4
CardNumDigits=6
KeyType=1
KeyA=123456789abc
KeyB=FFFFFFFFFFFF

[MIFARE_UL]
ReadData=1
StartBlock=4
BytesToRead=4
FormatName=MIFARE Ultralight (sector decode)

