Desfire for Python Documentation

Release 0.1.0

Mikko Ohtamaa

March 31, 2016

Contents

Contents:

CHAPTER 1

MIFARE DESFire for Python

This package provides MIFARE DESFire native communication protocol for NFC cards.

Source code: https://github.com/miohtama/desfire

Documentation: https://desfire.readthedocs.org

In photo: MIFARE DESFire EV1 8kB blank card with Identive CLOUD 4500 F Dual Interface Reader

1.1 Features

- Compatibile with USB-based NFC readers via PCSC interface. PCSC API is available on Linux, OSX and Windows. Linux support includes support for Raspberry Pi.
- Compatibile with Android mobile phones and their built-in NFC readers. This is done using Kivy cross application Python framework and native Android APIs via pyjnius Python to Java bridging.
- Only some of the commands are implemented in the current alpha quality version, please feel free to add more.
- Compatible with Python 2 and Python 3

1.2 Background

The communication protocol specification is not public. The work is based on reverse engineering existing open source DESFire projects, namely Android host card emulation for DESFire and MIFARE SDK.

1.3 Author

Mikko Ohtamaa.

1.4 Credits

This package was created with Cookiecutter and the audreyr/cookiecutter-pypackage project template.

CHAPTER 2

Installation

Install with pip to your virtualenv.

2.1 Ubuntu Linux

Install libraries using a Python virtual environment.

You need pyscard and it's dependencies. For Ubuntu:

```
apt install swig swig3.0 libpcsclite-dev pcscd
```

pyscard must be installed by hand (see issue):

```
# Need github registerd SSH pubkey
git clone git@github.com:LudovicRousseau/pyscard.git
cd pyscard
python setup.py develop
```

Then install desfire:

```
pip install desfire
```

2.2 Android and Kivy

TODO

CHAPTER 3

Usage

The library provides abstraction over DESFire command set. The communication with a NFC card must be done with an underlying library or API. DESFire provides adapters for different connection methods.

- Create a native connection to NFC card using underlying libraries
- Wrap this connection to proper adapter as `desfire.device.Device` subclass
- Create a `desfire.protocol.DESFire` object for the device
- Use `desfire.protocol.DESFire` API methods

3.1 PCSC example

Below is an example how to interface with DESFire API using pcscd daemon and pycard library. It should work on OSX, Linux and Windows including Raspberry Pi:

```
#! /usr/bin/env python
from __future__ import print_function

import functools
import logging
import time
import sys

from smartcard.System import readers
from smartcard.CardMonitoring import CardMonitor, CardObserver
from smartcard.util import toHexString
from smartcard.CardConnectionObserver import ConsoleCardConnectionObserver

from desfire.protocol import DESFire
from desfire.pcsc import PCSCDevice

#: Setup logging subsystem later
logger = None


IGNORE_EXCEPTIONS = (KeyboardInterrupt, MemoryError,)


def catch_gracefully():
    """Function decorator to show any Python exceptions occured inside a function.
```

```
    Use when the underlying thread main loop does not provide satisfying exception output.
    """
    def _outer(func):

        @functools.wraps(func)
        def _inner(*args, **kwargs):
            try:
                return func(*args, **kwargs)
            except Exception as e:
                if isinstance(e, IGNORE_EXCEPTIONS):
                    raise
                else:
                    logger.error("Catched exception %s when running %s", e, func)
                    logger.exception(e)

        return _inner

    return _outer


class MyObserver(CardObserver):
    """Observe when a card is inserted. Then try to run DESFire application listing against it."""

    # We need to have our own exception handling for this as the
    # # main loop of pyscard doesn't seem to do any exception output by default
    @catch_gracefully()
    def update(self, observable, actions):

        (addedcards, removedcards) = actions

        for card in addedcards:
            logger.info("+ Inserted: %s", toHexString(card.atr))

            connection = card.createConnection()
            connection.connect()

            # This will log raw card traffic to console
            connection.addObserver(ConsoleCardConnectionObserver())

            # connection object itself is CardConnectionDecorator wrapper
            # and we need to address the underlying connection object
            # directly
            logger.info("Opened connection %s", connection.component)

            desfire = DESFire(PCSCDevice(connection.component))
            applications = desfire.get_applications()

            for app_id in applications:
                logger.info("Found application 0x%06x", app_id)

            if not applications:
                logger.info("No applications on the card")

        for card in removedcards:
            logger.info("- Removed: %s", toHexString(card.atr))


def main():
```

```
    global logger

    logging.basicConfig(level=logging.DEBUG)
    logger = logging.getLogger(__name__)

    logger.info("Insert MIFARE Desfire card to any reader to get its applications.")

    available_reader = readers()
    logger.info("Available readers: %s", available_reader)
    if not available_reader:
        sys.exit("No smartcard readers detected")

    cardmonitor = CardMonitor()
    cardobserver = MyObserver()
    cardmonitor.addObserver(cardobserver)

    while True:
        time.sleep(1)

    # don't forget to remove observer, or the
    # monitor will poll forever...
    cardmonitor.deleteObserver(cardobserver)


if __name__ == "__main__":
    main()
```

3.2 Continuous card connection

Here is another more advanced example. When the card is attached to the reader, keep connecting to the card continuously and decrease it's stored value file 1 credit per second until we have consumed all the credit.

```
#! /usr/bin/env python
from __future__ import print_function

import functools
import logging
import time
import sys
import threading

from rainbow_logging_handler import RainbowLoggingHandler

from smartcard.System import readers
from smartcard.CardMonitoring import CardMonitor, CardObserver
from smartcard.util import toHexString
from smartcard.CardConnectionObserver import ConsoleCardConnectionObserver
from smartcard.Exceptions import CardConnectionException

from desfire.protocol import DESFire
from desfire.pcsc import PCSCDevice

#: Setup logging subsystem later
logger = None
```

```
IGNORE_EXCEPTIONS = (KeyboardInterrupt, MemoryError,)


FOOBAR_APP_ID = 0x121314
FOOBAR_STORED_VALUE_FILE_ID = 0x01

#: FOOBAR consumer thread
consumer = None


def setup_logging():

    # Setup Python root logger to DEBUG level
    logger = logging.getLogger()
    logger.setLevel(logging.DEBUG)
    formatter = logging.Formatter("[%(asctime)s] %(name)s %(funcName)s():%(lineno)d\t%(message)s")

    # Add colored log handlign to sys.stderr
    handler = RainbowLoggingHandler(sys.stderr)
    handler.setFormatter(formatter)
    logger.addHandler(handler)

def catch_gracefully():
    """Function decorator to show any Python exceptions occured inside a function.

    Use when the underlying thread main loop does not provide satisfying exception output.
    """
    def _outer(func):

        @functools.wraps(func)
        def _inner(*args, **kwargs):
            try:
                return func(*args, **kwargs)
            except Exception as e:
                if isinstance(e, IGNORE_EXCEPTIONS):
                    raise
                else:
                    logger.error("Catched exception %s when running %s", e, func)
                    logger.exception(e)

        return _inner

    return _outer



class ConsumerThread(threading.Thread):
    """Keep debiting down stored value file on the card until its done."""

    def __init__(self):
        super(ConsumerThread, self).__init__()

        #: Array of cards with open connection in connection attribute
        self.cards = set()
        self.alive = True

    def attach_card(self, card):
        self.cards.add(card)
```

```
    def detach_card(self, card):
        if card in self.cards:
            self.cards.remove(card)

    @catch_gracefully()
    def run(self):

        while self.alive:

            # List of cards where we have lost connetion
            remove_cards = []

            for card in self.cards:
                card_id = toHexString(card.atr)
                desfire = DESFire(PCSCDevice(card.connection))
                try:
                    desfire.select_application(FOOBAR_APP_ID)
                    value = desfire.get_value(FOOBAR_STORED_VALUE_FILE_ID)
                    if value > 0:
                        logger.info("Card: %s value left: %d", card_id, value)
                        desfire.debit_value(FOOBAR_STORED_VALUE_FILE_ID, 1)
                        desfire.commit()
                    else:
                        logger.info("No value left on card: %s", card_id)

                except CardConnectionException:
                    # Lost the card in the middle of transit
                    logger.warn("Consumer lost the card %s", card_id)
                    remove_cards.append(card)
                finally:
                    pass

            for c in remove_cards:
                card_id = toHexString(card.atr)
                logger.debug("Consumer removing a bad card from itself: %s", card_id)
                self.detach_card(c)

            time.sleep(1)


class MyObserver(CardObserver):
    """Observe when a card is inserted. Then try to run DESFire application listing against it."""

    @catch_gracefully()
    def update(self, observable, actions):

        (addedcards, removedcards) = actions

        for card in addedcards:
            logger.info("+ Inserted: %s", toHexString(card.atr))

            connection = card.createConnection()
            connection.connect()
            card.connection = connection.component

            # This will log raw card traffic to console
            connection.addObserver(ConsoleCardConnectionObserver())
```

```
            # connection object itself is CardConnectionDecorator wrapper
            # and we need to address the underlying connection object
            # directly
            logger.debug("Opened connection %s", connection.component)

            desfire = DESFire(PCSCDevice(connection.component))
            applications = desfire.get_applications()

            if FOOBAR_APP_ID in applications:
                consumer.attach_card(card)
            else:
                logger.warn("DESFire card doesn't have the required application. Maybe not properly i

        for card in removedcards:
            logger.info("- Removed: %s", toHexString(card.atr))
            consumer.detach_card(card)


def main():
    global logger
    global consumer

    setup_logging()
    logger = logging.getLogger(__name__)

    logger.info("Insert MIFARE Desfire card to any reader to get its applications.")

    available_reader = readers()
    logger.info("Available readers: %s", available_reader)
    if not available_reader:
        sys.exit("No smartcard readers detected")

    consumer = ConsumerThread()
    consumer.start()

    cardmonitor = CardMonitor()
    cardobserver = MyObserver()
    cardmonitor.addObserver(cardobserver)

    try:
        while True:
            time.sleep(1)
    finally:
        consumer.alive = False

    # don't forget to remove observer, or the
    # monitor will poll forever...
    cardmonitor.deleteObserver(cardobserver)


if __name__ == "__main__":
    main()
```

CHAPTER 4

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

4.1 Types of Contributions

4.1.1 Report Bugs

Report bugs at https://github.com/miohtama/desfire/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

4.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" is open to whoever wants to implement it.

4.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "feature" is open to whoever wants to implement it.

4.1.4 Write Documentation

Desfire for Python could always use more documentation, whether as part of the official Desfire for Python docs, in docstrings, or even on the web in blog posts, articles, and such.

4.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/miohtama/desfire/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

4.2 Get Started!

Ready to contribute? Here's how to set up *desfire* for local development.

1. Fork the *desfire* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/desfire.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv desfire
$ cd desfire/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 desfire tests
$ python setup.py test
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

4.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.6, 2.7, 3.3, 3.4 and 3.5, and for PyPy. Check https://travis-ci.org/miohtama/desfire/pull_requests and make sure that the tests pass for all supported Python versions.

CHAPTER 5

Credits

5.1 Development Lead

- Mikko Ohtamaa <mikko@opensourcehacker.com>

5.2 Contributors

None yet. Why not be the first?

CHAPTER 6

History

6.1 0.2 (2016-03-30)

- Added data file read and write

6.2 0.1 (2016-03-07)

- First release on PyPI.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search