PyAmiibo Documentation

Release 0.2.0

Toby Fleming

Jan 11, 2019

Contents:

PyAmiibo helps to read, modify and write dump files of Nintendo Amiibo figures. PyAmiibo is capable of parsing most NTAG properties, as well as some Amiibo data.

IMPORTANT: To decrypt and encrypt Amiibo data, you will need two master keys, commonly called `unfixed-info.bin` and `locked-secret.bin`. They are not provided.

You can also view the full PyAmiibo docs on ReadTheDocs.

It is based on Marcos Del Sol Vives' reverse engineering efforts of the Amiibo cryptography (amiitool, reddit).

So why does PyAmiibo exist? `amiitool` is a C binary, difficult to use in other tools (especially web-based tools). It also re-arranges the sections of the dump file when decrypting, and doesn't seem to support editing dumps (e.g. changing the UID). Even though PyAmiibo doesn't use any of `amiitool`'s code and contains a lot of my own research into the NTAG format and Amiibo data, it would not have been possible without Marcos' efforts.

CHAPTER 1

Usage

PyAmiibo is Python 3 only, if you get an error installing it this is the most likely reason.

```
pip install pyamiibo
```

PyAmiibo is mainly a library, but also contains some simple command-line tools:

```
$ # convert hexadecimal data to binary, note the quotes!
$ amiibo hex2bin "F1 A3 65 .." unfixed-info.bin
$ # get help for a subcommand
$ amiibo uid --help
$ # update the UID on an existing dump
$ amiibo uid old.bin "04 FF FF FF FF FF FF" new.bin
```

The master keys must be in the current directory for some commands to work!

It's also very easy to use in a script or interpreter session:

```
from amiibo import AmiiboDump, AmiiboMasterKey
with open('unfixed-info.bin', 'rb') as fp_d, \
        open('locked-secret.bin', 'rb') as fp_t:
    master_keys = AmiiboMasterKey.from_separate_bin(
        fp_d.read(), fp_t.read())

with open('dump.bin', 'rb') as fp:
    dump = AmiiboDump(master_keys, fp.read())

print('old', dump.uid_hex)
dump.unlock()
dump.uid_hex = '04 FF FF FF FF FF FF'
dump.lock()
dump.unset_lock_bytes()
print('new', dump.uid_hex)

with open('new.bin', 'wb') as fp:
    fp.write(dump.data)
```

CHAPTER 2

Development

Use pipenv to install the development dependencies, and make sure `flake8` and `pylint` pass before a PR is submitted.

```
pipenv install --three --dev
pipenv shell
isort -y
flake8 amiibo/
pylint amiibo/
sphinx-build -b html docs/ docs/_build
```

CHAPTER 3

Index

3.1 Master keys

IMPORTANT: To decrypt and encrypt Amiibo data, you will need two master keys, They are not provided, however there is validation to ensure you have the correct keys.

The keys are commonly called `unfixed-info.bin` (data key) and `locked-secret.bin` (tag key). Occasionally, these keys are joined for easier loading:

```
$ cat unfixed-info.bin locked-secret.bin > key.bin
```

These are binary files of 80 bytes each. Sometimes, they are distributed as hexadecimal bytes separated by spaces (e.g. `F1 A3 65 ..` etc). PyAmiibo can also parse this data into binary.

The tag master key is used to derive a Amiibo-specific tag key to sign fixed/locked information of the Amiibo, such as the UID, the Amiibo type.

The data master key is used to derive several Amiibo-specific data keys to sign and encrypt unfixed/unlocked information of the Amiibo, such as the name, the owner, and game data.

The signing operation is a HMAC-SHA256 using the derived keys, and the encryption operation is AES128 in counter mode using a derived key and initialisation vector.

3.1.1 Classes

3.2 Amiibo

Amiibos are Nintendo figures with small RFID tags inside them that enables integration with some Wii U, 3DS, and Switch games. With a compatible RFID/NFC reader, the data from the tags can be extracted. This is called a dump.

3.2.1 Dumps

The Amiibo RFID tags happen to be NXP Semiconductor's NTAG215 (see also the *NTAG* page), which can hold 540 bytes. So Amiibo dumps should be 540 bytes. Sometimes, the configuration pages are omitted, incomplete dumps are 520 bytes.

3.2.2 Password protection

NTAG215 offers password protection for memory access, which is used for Amiibos. The password is derived from the tag's 7 byte UID:

```
pw[0] = 0xAA ^ uid[1] ^ uid[3]
pw[1] = 0x55 ^ uid[2] ^ uid[4]
pw[2] = 0xAA ^ uid[3] ^ uid[5]
pw[3] = 0x55 ^ uid[4] ^ uid[6]
```

The first byte of the UID is always `0x04` for NXP tags, and so it doesn't make sense to use it in the password.

3.2.3 Cryptography

On top of the password protection NTAG215 offers, cryptography is used to encrypt and sign some sections of the user data in the tag. Marcos Del Sol Vives' reverse engineered the Amiibo cryptography in his excellent amiitool.

For more information, see the *Master keys* page.

3.2.4 Classes

3.3 NTAG

NTAG is NXP Semiconductor's name for a family of NFC RFID products. Amiibos use NTAG215 internally. NXP's NTAG213/215/216 datasheet is truly excellent and worth a read. (The code is based on Rev. 3.2 / 2 June 2015 / 265332.)

Many constants/properties for interpreting Amiibo/NTAG215 dumps are sourced from the datasheet; where possible the exact section is indicated.

3.3.1 ISO/IEC 14443-3

ISO/IEC 14443:2016 is the standard for "Identification cards – Contactless integrated circuit cards – Proximity cards", which NFC (and therefore NTAG) products implement. There are four parts:

- Part 1: Physical characteristics
- Part 2: Radio frequency power and signal interface
- Part 3: Initialization and anticollision
- Part 4: Transmission protocol

For interpreting NTAG dumps, only part 3 is interesting, because it details the way the UID is stored and validated.

Unfortunately, getting a PDF of the standard costs real money. They are also very tedious - stick to the NTAG213/215/216 datasheet instead!

3.3.2 Classes

- genindex
- modindex
- search