New interoperability for Disney Infinity NFC toys

Demonstration: writing on a Mac, reading on an Android phone

This demo shows new interoperability of Disney Infinity NFC toys. Previously, only Disney hardware and games had the ability to write data to Infinity NFC toys.

Any NFC reader can read the ATQA, SAK and UID of an NFC toy. These let you tell it's an Infinity, and its unique ID.

By knowing the algorithm used to set the read/write password (key A), we can interoperably write our own data to an Infinity NFC toy.

This video first shows a program which computes the key A and uses it to write new, custom text to an Infinity NFC toy. The program is running on a standard Apple laptop, using a commercial, off-the-shelf, USB NFC reader (the Identiv SCL3711).

Then, a standard Android phone with NFC support is shown, using the commercial, off-the-shelf, NXP TagInfo app from the Google Play store. Without the key A, the app is shown being unable to read the data on the toy. With the key A provided, the app is shown being able to read the custom text written on the toy.

The video has artificial delays inserted so the on-screen explanations can be read; computing the key A and writing to the NFC toy are nearly instantaneous otherwise, as with the Android portion of the demo.

Read Writing your own data to a Disney Infinity NFC toy to see a step-by-step workflow similar to the demo video, using standard software available on any Mac or Linux computer.

Computing the key A

In 2017, a researcher determined the algorithm used to compute the key A for Disney Infinity toys. It's a standard SHA-1 hash of the UID, with a prefix and postfix as a public or reused salt.

A clean room description of the algorithm is as follows:

> *Let a prefix be the 16-byte (32-character) hexadecimal representation of the integer computed by the multiplication of the four prime numbers 3 and 5 and 23 and 38,844,225,342,798,321,268,237,511,320,137,937*
>
> *Let a postfix be the 15-byte (30-character) hexadecimal representation of the integer computed by the multiplication of the three prime numbers 3 and 7 and 9,985,861,487,287,759,675,192,201,655,940,647*
>
> *Compute the SHA-1 digest of the 38 bytes encoded by the 76-character hexadecimal concatenation of the prefix and the UID and the postfix*
>
> *The key A for all sectors is 6 bytes, represented in hexadecimal as 12 characters: in order, the 4th and 3rd and 2nd and 1st and 8th and 7th bytes of the computed SHA-1 digest*

Sample inputs/outputs

UID	Key A
`0456263a873a80`	`29564af75805`
`049c0bb2a03784`	`c0b423c8e4c2`
`04a0f02a3d2d80`	`1e0615823120`
`04b40c12a13780`	`2737629f2ebe`
`04d9fb8a763b80`	`edb56de8a9fe`

An example of this implemented in Python 2 is as follows:

PYTHON
```python
#!/usr/bin/python

## infsha.py - Compute a key A
##
## Written in 2017 and 2018 by Vitorio Miliano
##
## To the extent possible under law, the author has dedicated all
## copyright and related and neighboring rights to this software to
## the public domain worldwide.  This software is distributed without
## any warranty.
##
## You should have received a copy of the CC0 Public Domain
## Dedication along with this software.  If not, see
## <http://creativecommons.org/publicdomain/zero/1.0/>.

import binascii, hashlib, re, sys

uidre = re.compile('^04[0-9a-f]{12}$', re.IGNORECASE)
magic_nums = [3, 5, 7, 23, 9985861487287759675192201655940647L, 38844225342798321268237511320137937L]

def calc_keya(uid, sector):
    if uidre.match(uid) is None:
        raise ValueError('invalid UID (seven hex bytes)')

    if sector < 0 or sector > 4:
        raise ValueError('invalid sector (0-4)')

    PRE = format(magic_nums[0] * magic_nums[1] * magic_nums[3] * magic_nums[5], '032x')
    POST = format(magic_nums[0] * magic_nums[2] * magic_nums[4], '030x')

    sha1 = hashlib.sha1()
    sha1.update(binascii.unhexlify(PRE + uid + POST))
    key = sha1.digest()

    return binascii.hexlify(key[3::-1] + key[7:5:-1])

if __name__ == '__main__':
    if len(sys.argv) > 2 and sys.argv[2] == '-eml':
        keys = calc_keya(sys.argv[1], 0)
        print ('0'*8+keys+'\n'+('0'*32+'\n')*3).join([keys]*5).join([(sys.argv[1]+'0'*18+'\n')+(('0'*32+'\n')*2),'0'*8+keys])
    elif len(sys.argv) > 1:
        print calc_keya(sys.argv[1], 0)
```

Use at your own risk

Consider writing your own data only to toys you no longer wish to use with the game.

Disney Infinity 3.0 on the PlayStation 3 console is very forgiving. If you write custom data to a Disney Infinity toy and then try and use it with DI3, the game will recognize the figure, treat it as brand new, and overwrite your data with its own.

I do not know what happens with Disney Infinity 2.0 or 1.0, or on other platforms.