© 2026 Yiguedan Philippe Andy Yaro



NFC Cartridges: Songs That Live on a Sticker

Yiguedan Philippe Andy Yaro

View GitHub Repository: github.com/andyyaro/flipper-nfc-cartridges

Let's connect! Email: yyaro@gmu.edu

LinkedIn: linkedin.com/in/andy-yaro

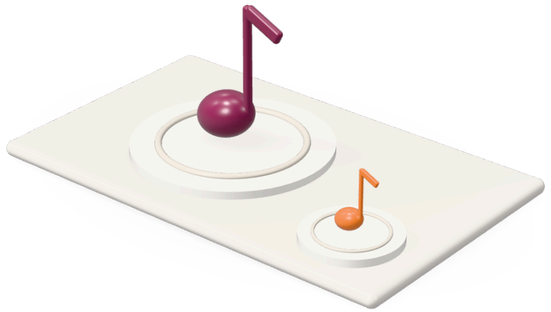


On 31 October 2026 I made the first commit of NFC Cartridges, an app for the Flipper Zero, a handheld radio multi-tool. It writes a whole song onto a cheap NFC sticker (NFC: Near Field Communication, a short-range radio) and plays it when the sticker is tapped to the Flipper.

The song is plain text in a standard tag format. The private, MIT-licensed repo also holds a test that runs on a computer, an experimental tool that turns the sung melody of a recording into song text, four sample songs, a logo and the source of a 76.5-second launch film.

For this deck, on 3 November 2026, I rebuilt the app, reran the test, simulated the burn order, reran the melody tool offline, and read the git history. Claude, Anthropic's AI, is credited on the film's commit; the repo does not record how the app's code was written. I am Yiguedan Philippe Andy Yaro.

A. Songs on a Sticker

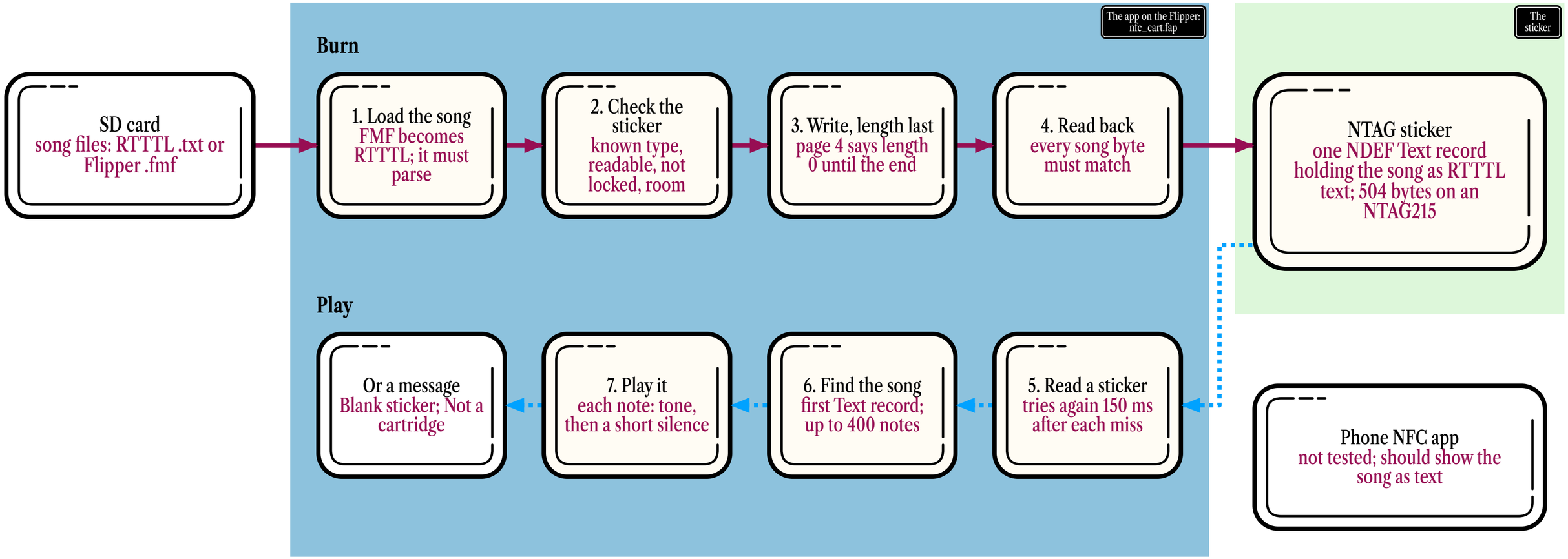


The Flipper Zero is a handheld multi-tool that reads and emulates radio protocols. NTAG stickers are low-cost NFC (short-range radio) tags built on chips from NXP.

The whole song lives on the sticker: Burn writes it from a file on the Flipper's SD (Secure Digital) memory card; Play reads a tapped sticker and plays it.

Phones use the same tag format, so they should show a cartridge's song as text; I have not tested a phone.

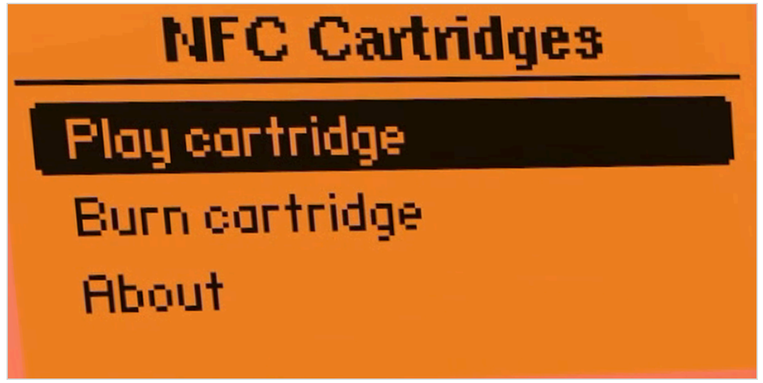
1. Burn and Play



From nfc_cart.c, cart_song.c and cart_ndef.c. NFC = Near Field Communication, short-range radio; .fap = a Flipper app file; ms = milliseconds; RTTTL = Ring Tone Text Transfer Language, a Nokia ringtone text format; FMF = Flipper Music Format; NDEF = NFC Data Exchange Format, the message format phones read from tags; NTAG = NXP's low-cost tag chips; SD = Secure Digital, the Flipper's memory card. Each note sounds for 90% of its length; the light-emitting diode (LED) blinks cyan as each note starts.



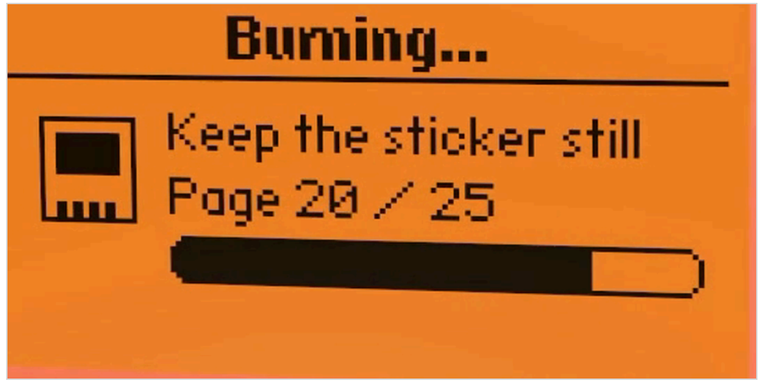
2. One Real Burn, One Real Play



Menu: Play cartridge, Burn cartridge, About.



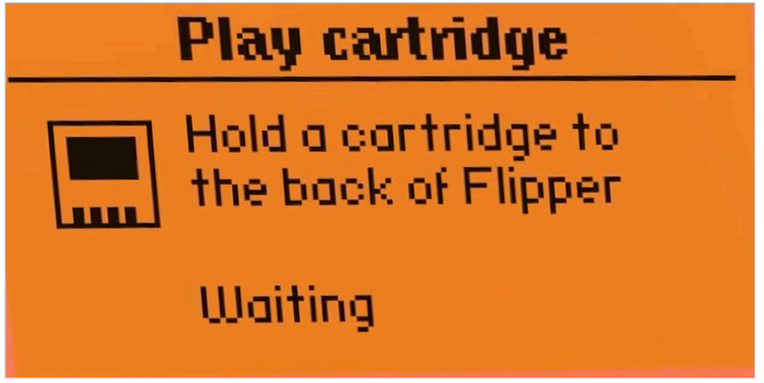
Burn cartridge: Ode to Joy needs 100 bytes.



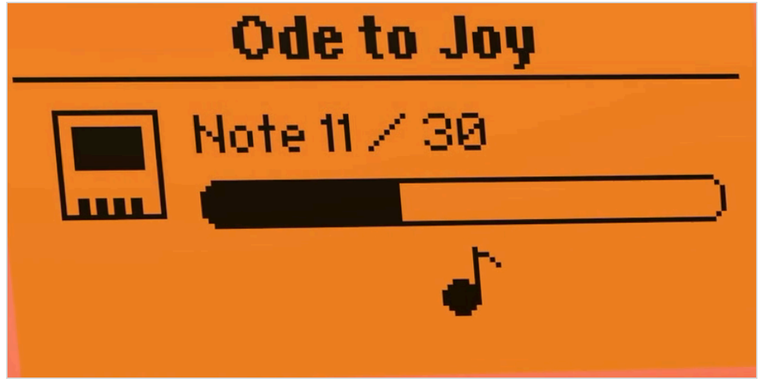
Burning: 20 of 25 song pages written.



Cartridge burned: read-back matched.



Play: waiting for a tapped sticker.



Playing the song read from the sticker: note 11 of 30.

Real screens from my Flipper Zero on 1 November 2026, streamed over USB (Universal Serial Bus) by a recording script, and cropped from the launch film, whose rendering softens the pixel font; one NTAG215 sticker, an NXP tag chip with 504 bytes for data, burned and played. The script sent the button presses; I put the sticker on the Flipper. The film's notes time this burn at 1.3 seconds from the first page to the success screen. Read-back = the app reading the sticker again and comparing; Back is ignored while burning.

B. The Format on the Sticker



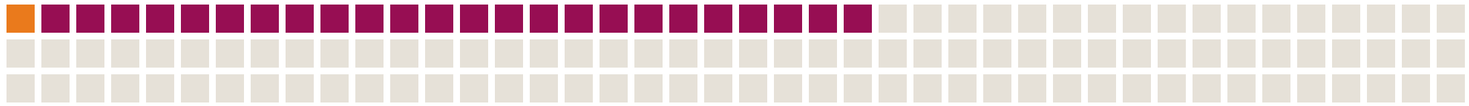
NTAG stickers, built on NXP tag chips, store data in 4-byte pages. User data starts at page 4, after the serial number, the lock bytes, which can freeze pages, and the capability container, which marks the format.

The app wraps the song text in one Text record of NDEF, the standard tag data format. Ode to Joy takes 100 bytes: 25 pages.

The wrapping costs 10 bytes for a song of up to 247 bytes of text, and up to 15 bytes above that.

1. Ode to Joy, Byte by Byte

```
$ cat songs/ode_to_joy.txt
Ode to Joy:d=4,o=5,b=140:e,e,f,g,g,f,e,d,c,c,d,e,e.,8d,2d,e,e,f,g,g,f,e,d,c,c,d,e,d.,8c,2c   1
$ ../bin/pages songs/ode_to_joy.txt
song text: 90 bytes; with its NDEF wrapping: 100 bytes = 25 pages of 4 bytes
page   4  03 61 D1 01   .a..   2
page   5  5D 54 02 65   ]T.e
page   6  6E 4F 64 65   nOde   3
page   7  20 74 6F 20    to
page   8  4A 6F 79 3A   Joy:
page   9  64 3D 34 2C   d=4,
   ...    pages 10 to 26: 68 more bytes of song text
page  27  2E 2C 38 63   .,8c
page  28  2C 32 63 FE   ,2c.   4
NTAG215 user memory: pages 4 to 129 = 126 pages = 504 bytes; used 100 of 504   5
```



The NTAG215's 126 user pages: page 4 in orange, the song in berry.

RTTTL = Ring Tone Text Transfer Language, Nokia's ringtone text
NDEF = NFC (Near Field Communication) Data Exchange Format
NTAG215 = an NXP tag chip with 504 bytes of user memory
record, payload = one item in an NDEF message, and its content
UTF-8 = the common text encoding
page = 4 bytes of tag memory

1. The song is RTTTL text: a name; the default duration, octave and tempo; then 30 notes.

2. 03: an NDEF message follows; 61: it is 97 bytes long; D1: one short record of a standard type; 01: a 1-byte type name.

3. 5D: 93 bytes of payload; 54: "T", a Text record; 02: UTF-8, two-letter language; "en"; then the song.

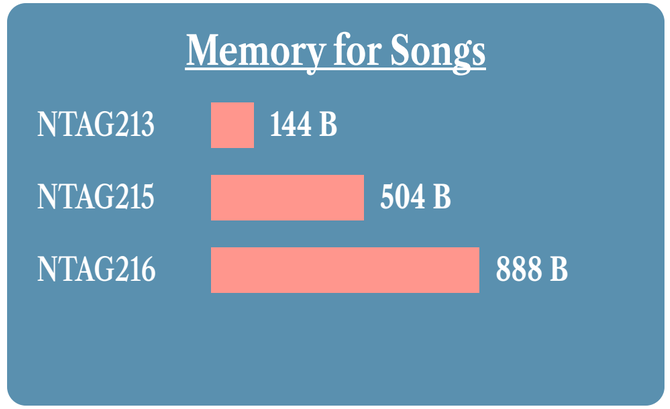
4. FE marks the end of the data.

5. 100 of 504 bytes, with the wrapping and FE: 25 of the NTAG215's 126 user pages.

pages is a C program written for this deck that calls the app's own encoder, cart_ndef.c; run offline on 3 November 2026. A short record has a payload of at most 255 bytes; from 248 bytes of text the message length takes 3 bytes, and from 253 the payload length takes 4.

2. What Fits on a Sticker

```
$ ../bin/songs songs/*.txt
title         notes   secs   text B    NDEF B  pages
Fur Elise        35   11.9      149       159     40
Ode to Joy       30   13.7       90       100     25
Toccata          16   13.9       71        81     21
Twinkle          42   24.0      111       121     31
largest song text that fits:  NTAG213 134 B of 144;  NTAG215 489 B of 504;  NTAG216 873 B of 888
a 429-note song: 873 B of text, 888 B of NDEF (fits NTAG216's 888): parse refused (400-note cap)
```

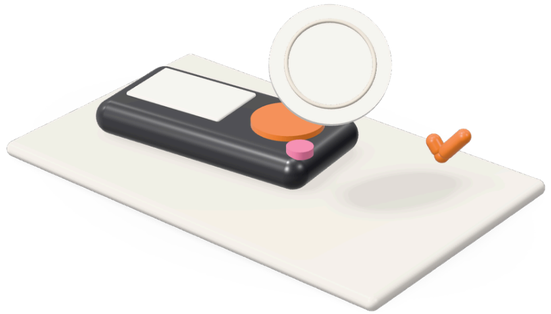


1. The four sample songs need 81 to 159 bytes. Three fit an NTAG213; Für Elise, at 159, does not.
2. The longest song text each sticker holds, after 10 to 15 bytes of wrapping.
3. A 429-note song fits an NTAG216, but the app caps songs at 400 notes and refuses it.

The README's guide: an NTAG213 holds a short jingle, an NTAG215 100 to 150 notes, an NTAG216 250 notes or more. In the samples a note takes 2.1 to 3.6 bytes, so an NTAG215 fits about 130 to 220 notes, at or above the README's figure.

songs is a C program written for this deck that runs the app's own cart_song.c and cart_ndef.c on a computer; run offline on 3 November 2026. notes counts sounded notes, not rests; secs = playing time; text B = bytes of RTTTL text; NDEF B = bytes of NDEF data before padding to whole pages; a page is 4 bytes; B = bytes. RTTTL = Ring Tone Text Transfer Language; NDEF = NFC Data Exchange Format; NFC = Near Field Communication; NTAG213/215/216 = NXP tag chips.

C. Burning Safely



A sticker can be pulled away in the middle of a burn. The app writes page 4 with a length of 0 (after page 3, if blank), then the rest of the song, then page 4 again with the real length.

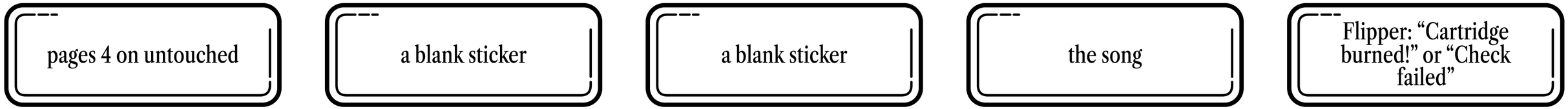
Until that last write, the app's parser sees an empty message, so a burn cut short (torn) reads as blank, not as half a song.

Then the app reads the sticker back and compares every song byte it wrote.

1. Length Last, Then Read Back



The app's parser then sees:



Why It Works

The app's parser treats an NDEF message of length 0 as empty and shows "Blank sticker"; a phone should too, though I have not tested one.

Back is ignored while burning, so a user cannot cancel a burn halfway.

What It Does Not Cover

A tear in the middle of one page write, such as a page-4 write or a first-time page-3 write, is not covered.

A torn re-burn loses the old song as well. No real sticker was pulled away on purpose.

From nfc_cart.c, lines 315 to 366; Ode to Joy on an NTAG215 (an NXP tag chip, 504 bytes) shown. Page 3 holds the capability container; E1 marks NDEF, NFC Data Exchange Format; NFC = Near Field Communication; parser = the code that reads the sticker. If all 3 tries of the last write get no reply, the app shows "Burn failed: Sticker moved away" though the song may be on the sticker.

2. In Simulation, a Torn Burn Reads as Blank

```
$ ../bin/tear songs/ode_to_joy.txt
sticker starts blank: 03 00 FE
stop  1  page 4 = 03 00 D1 01 (length 0)      reads as blank
stop  2  + page 5                             reads as blank
  ...    stops 3 to 24: one page each
stop 25  + page 28                            reads as blank
stop 26  page 4 = 03 61 D1 01 (real length)   reads the new song
interrupted stops that read as blank: 25 of 25
$ ../bin/tear songs/ode_to_joy.txt songs/fur_elise.txt | sed -n '1p;$p'
sticker starts with: "Fur Elise"
interrupted stops that read as blank: 25 of 25
$ ../bin/tear ../melody/out/OdeSynth.txt | sed -n '2p;$p'
stop  1  page 4 = 03 FF 00 00 (length 0)      reads as blank
interrupted stops that read as blank: 69 of 69
```

1. From blank: after page 4 with length 0, and after each later page, the app's parser reads a blank sticker.
2. Only the 26th write, page 4 with the real length, makes the song appear: 25 of 25 stops read blank.
3. Over an old song, Für Elise: every stop also reads blank, so the old song is lost, not mixed in.
4. A 259-byte song (274 bytes of NDEF) uses the long length form, 03 FF 00 00: all 69 interrupted stops read blank.

tear is a C program written for this deck. It re-creates the write order of nfc_cart.c, lines 315 to 338, stopping between whole page writes, on a simulated 504-byte NTAG215 user memory, and asks the app's own parser, cart_ndef.c, what it reads after each write. It is a simulation, not a real sticker; run offline on 3 November 2026. The long song is the melody tool's output from slide D4.

i

stop = the moment after one page write
page 4 = the first user page; it holds the length
long length form = 03 FF and a 2-byte length, once the message reaches 255 bytes
03 ... FE = the start and end marks of NDEF data
NDEF = NFC (Near Field Communication) Data Exchange Format
NTAG215 = an NXP tag chip with 504 bytes of user memory

3. Checks, and What They Miss

Checked Before Writing

The tag is a type the app knows.

Every user page could be read; if not, “password-protected or unreadable”.

The lock bits are clear; on NTAG213, 215 and 216, the extra lock bytes too.

Page 3 is blank, or marks NDEF with read and write access.

The song fits in the sticker’s memory.

Missed

The code also lets NTAG203 and most MIFARE Ultralight tags through, though the README names only NTAG213/215/216; NTAG203 and Ultralight EV1 extra lock bytes go unchecked.

A tag protected only against writing is not caught first: the burn fails with “Sticker moved away”, leaving it unchanged, or blank if protection starts after page 4.

It never compares the song with the data size page 3 declares.

It overwrites any unlocked NDEF sticker, even one holding a web link, without asking.

From nfc_cart.c, lines 106 to 122 and 263 to 342; found by reading the code, not tested on such tags. Lock bits make pages read-only permanently; page 3 is the capability container, which marks the tag’s format and size; NTAG, NTAG203 and MIFARE Ultralight are NXP tag chips of one family; NDEF = NFC (Near Field Communication) Data Exchange Format. A sticker with a broken song shows “Not a cartridge”.

D. Build and Tests



The app builds with ufbt (micro Flipper Build Tool), the official build tool, against firmware 1.2.0, with every warning treated as an error.

One host test, run on a computer, covers the NDEF code; NDEF is the tag data format. It builds a short and a long record, reads them back and rejects broken data.

Nothing in the repo tests the song parser, the burn order or phone reading; slide C2's simulation is my own check.

1. A Clean Build Against Firmware 1.2.0

```
$ grep -o '"version": "[^"]*"' $UFBT_HOME/current/components.json
"version": "1.2.0"
$ ufbt -c > /dev/null 2>&1; ufbt 2>&1 | grep -vE 'scons|^$'
        CC      /home/claude/run_fl_ev/repo/nfc_cart/cart_ndef.c
        CC      /home/claude/run_fl_ev/repo/nfc_cart/cart_song.c
        CC      /home/claude/run_fl_ev/repo/nfc_cart/nfc_cart.c
        CDB     /home/claude/run_fl_ev/repo/nfc_cart/.vscode/compile_commands.json
        LINK    /tmp/fl_work/ufbt_home/build/nfc_cart_d.elf
        INSTALL /home/claude/run_fl_ev/repo/nfc_cart/dist/debug/nfc_cart_d.elf
        APPMETA /tmp/fl_work/ufbt_home/build/nfc_cart.fap
        FAP     /tmp/fl_work/ufbt_home/build/nfc_cart.fap
        FASTFAP /tmp/fl_work/ufbt_home/build/nfc_cart.fap
        INSTALL /home/claude/run_fl_ev/repo/nfc_cart/dist/nfc_cart.fap
        APPCHK  /tmp/fl_work/ufbt_home/build/nfc_cart.fap
                Target: 7, API: 79.2
$ stat -c '%s bytes  %n' dist/nfc_cart.fap
19572 bytes  dist/nfc_cart.fap
$ python3 -I ../../checks/flags.py .vscode/compile_commands.json
cart_ndef.c  -Wall -Wextra -Werror -Os
cart_song.c  -Wall -Wextra -Werror -Os
nfc_cart.c   -Wall -Wextra -Werror -Os
```

1. The SDK, the kit for building Flipper apps, is official firmware 1.2.0.
2. ufbt compiles the app's three C files for the Flipper's processor, an ARM Cortex-M4, and packs them into a FAP.
3. The check passes: target 7, the Flipper Zero's hardware, and API 79.2, the firmware interface the app needs.
4. The installable app: 19,572 bytes.
5. All three files built with -Werror, which turns any warning into a failure.

i

ufbt = micro Flipper Build Tool
SDK = software development kit
API = application programming interface
FAP = Flipper Application Package, the installable file
-Wall -Wextra = turn on most warnings
-Os = optimize for size

Run offline on 3 November 2026 with ufbt 0.2.6; its SDK and compiler were downloaded earlier in this review. Not installed on a device for this deck; the film's capture log shows the app started on my Flipper on 1 November 2026.

2. One Host Test, Passing

```
$ cc -Wall -Wextra -I nfc_cart tests/test_ndef.c nfc_cart/cart_ndef.c -o test_ndef && ./test_ndef
ok: 36 byte song -> 46 byte NDEF (short record)   1
ok: 479 byte song -> 494 byte NDEF (long record)
all NDEF tests passed
$ gcc -g -fsanitize=address,undefined -I nfc_cart tests/test_ndef.c nfc_cart/cart_ndef.c -o \
    /tmp/t_san && /tmp/t_san
ok: 36 byte song -> 46 byte NDEF (short record)   2
ok: 479 byte song -> 494 byte NDEF (long record)
all NDEF tests passed
$ cc -DNDEBUG -I nfc_cart tests/test_ndef.c nfc_cart/cart_ndef.c -o /tmp/t_nd && /tmp/t_nd
ok: 36 byte song -> 46 byte NDEF (short record)   3
ok: 479 byte song -> 494 byte NDEF (long record)
all NDEF tests passed
$ grep -c 'assert(' tests/test_ndef.c
8      4
```

1. The README's command: short and long records round-trip; all checks pass.
2. Rebuilt with sanitizers, which catch memory errors as it runs: still passes.
3. Rebuilt with -DNDEBUG, the checks are compiled out, and it still prints "passed".
4. The test has 8 assert lines; 4 run twice, in a loop.

Covered

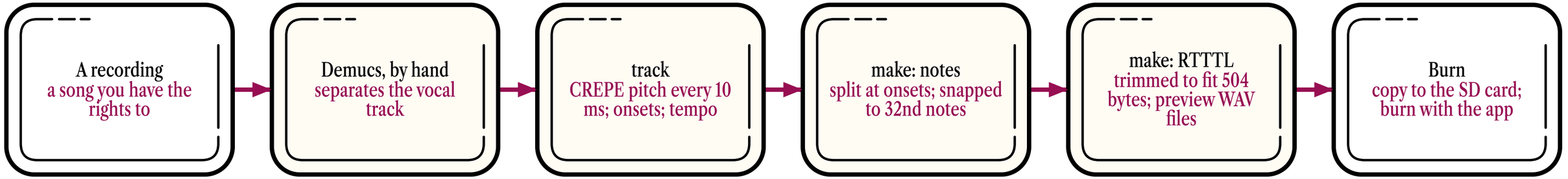
Building short and long Text records and reading them back; a buffer too small; a blank sticker's bytes; broken data.

Not Covered

The RTTTL and FMF song code, the sticker checks, the write order, the read-back and phones; no CI.

Run offline on 3 November 2026 with GCC 13.3, the GNU C compiler; a host test runs on a computer, not the Flipper. assert is C's standard check macro; -DNDEBUG switches all of them off; CI = continuous integration, tests run on every change; RTTTL and FMF are the two song formats; NDEF = NFC (Near Field Communication) Data Exchange Format.

3. The Melody Tool



The README Says

Experimental; results vary a lot.

Fast or heavily processed vocals often do not survive.

Only burn songs you have the rights to.

This Review Found

The size target is fixed at 504 bytes, an NTAG215.

Its scores compare the notes with its own analysis of the vocal, not with the true melody.

No audio and no tests in the repo; its graphics-chip path is for Apple chips only.

If singing fills most of the file at even loudness, its loudness gate can drop every note.

From tools/song_to_rtttl.py, 266 lines of Python. Demucs = a model that splits a song into vocals and accompaniment; CREPE = a neural-network pitch tracker; onset = where a note starts; a 32nd note is the shortest step it uses; RTTTL = Ring Tone Text Transfer Language; WAV = an audio file; SD = the Flipper's memory card; NTAG215 = an NFC (short-range radio) sticker chip with 504 bytes for data.

4. The Melody Tool on a Synthetic Song

```
$ python3 -I ../checks/make_synth.py in 4.0
vocals.wav and mix.wav: 21.71 s, 30 notes, first note at 4.00 s   (1)
$ python3 -I ../repo/tools/song_to_rtttl.py track in/vocals.wav in/mix.wav cache/
tracked 2172 frames on cpu; 52 vocal onsets; tempo 139.7   (2)
$ python3 -I ../repo/tools/song_to_rtttl.py make cache/ 4.0 18.5 OdeSynth out/
OdeSynth: 38 notes, grid 139.7 BPM in 32nds, 274/504 bytes, covers 13.6s of 14s   (3)
  raw notes vs vocal : onset F1 0.71 | pitch ok 100% | voiced coverage 93%
  on-grid  vs vocal : onset F1 0.80 | pitch ok 100% | voiced coverage 83%   (4)
  OdeSynth:d=8,o=5,b=140:e.,32p,e.,32p,32e,f.,16p,g.,16p,g.,16p,f.,32p,32f,e.,16p,d.,16p,c.,16p,c.,16p,d.,16p,e....
$ ../bin/songs out/OdeSynth.txt | head -2
title         notes   secs   text B    NDEF B  pages
OdeSynth         38   13.5      259       274     69   (5)
```

(1) Input: a voice-like tone playing the repo's Ode to Joy, made for this deck: 30 notes, at 140 beats per minute.

(2) The track step ran on the processor: it found 52 note starts for 30 notes, and a tempo of 139.7 beats per minute.

(3) Output: 38 notes and 28 rests in 274 bytes, which fits an NTAG215. The 8 extra notes are short pieces at note edges.

(4) "pitch ok 100%" uses the tool's own pitch track, not the true melody, and ignores octave.

(5) The app's parser accepts the result: 38 notes, 13.5 seconds.

Run offline on 3 November 2026 with PyTorch, a machine-learning library, on the processor; Demucs was skipped because the input is already a vocal track. With 0.5 seconds of silence around the same melody instead of 4, the tool reported "no vocal notes in that section". onset F1 scores how well note starts match, from 0 to 1; BPM = beats per minute; NDEF = NFC Data Exchange Format; NFC = Near Field Communication; NTAG215 = a 504-byte NFC sticker chip.

E. The Film and the Credits

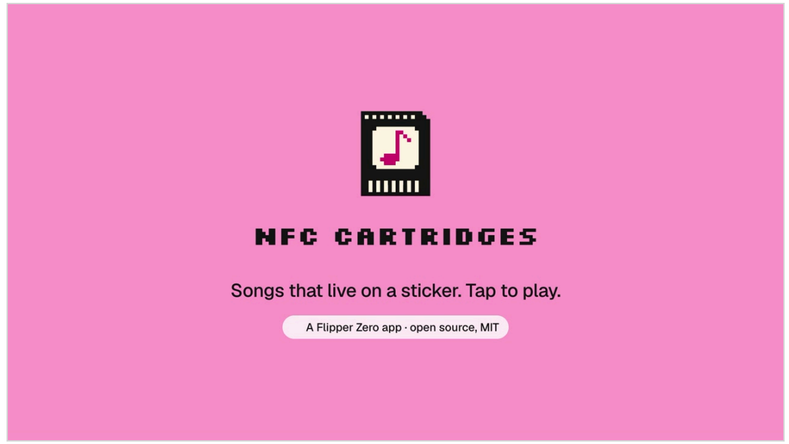


A 76.5-second launch film, added on 2 November 2026, shows real screens from my Flipper inside a 3D model, with an AI narrator.

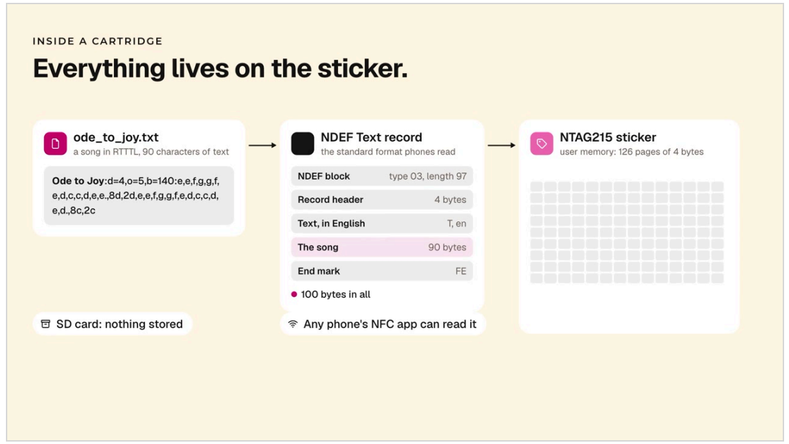
The film's commit credits Claude, Anthropic's AI, as co-author. The voice is Kokoro-82M, an AI text-to-speech model.

The first three commits carry no AI credit line; that alone does not show how the app was written.

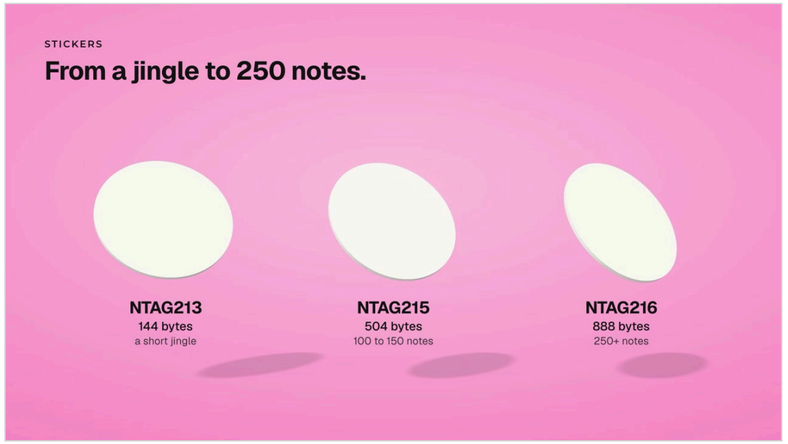
1. The Launch Film



The title card says “open source, MIT”.



Inside a cartridge, with “Any phone’s NFC app can read it”.



The three sticker sizes.

Real

The Flipper’s screens, streamed from my device over USB on 1 November 2026.

The burn and play timing, from that recording.

The burn used one real NTAG215 sticker.

Rebuilt

The Flipper and the sticker are 3D models.

The tune is synthesized from the app’s rules, not recorded; 15 of 30 notes are heard.

The waiting screen is one frame, held.

Overstated

“Open source, MIT”: the repo is private.

“Any phone’s NFC app can read it”: untested.

“Locked or password-protected: refused”: only in part.

Frames from the 76.5-second film (1920 × 1080, 27.2 MB), a release file of the private repo, chosen without the 3D Flipper model, whose license is unconfirmed. Made with HyperFrames, which renders web pages to video, GSAP, a JavaScript animation library, and three.js, a 3D library. USB = Universal Serial Bus; MIT = a permissive open-source license; NFC = Near Field Communication; NTAG215 = a 504-byte NFC sticker chip.

2. Who Did What

Me

Git author of all four commits, from the app on 31 October to the film on 2 November.

Put my blank NTAG215 sticker on the Flipper for the 1 November screen recording.

Gave directions on the film's sound and labels, which its source files quote.

AI Tools

Claude, Anthropic's AI: credited co-author of the film's commit: 39 files of film source, notes and a README preview.

Kokoro-82M, an AI voice, narrates; Whisper, a speech-recognition model, checked each take against the script.

Others' Work

The Flipper 3D model: most likely blazitt's, under CC BY 4.0; unconfirmed, and not credited on the end card.

Music "Tom" by Arulo and seven sound effects, from Mixkit's free library.

Fonts under the SIL Open Font License; three.js, GSAP and HyperFrames.

Not recorded: how the app's C code, the test, the melody tool and the logo were made. The first commit adds 16 files, 1,522 lines, at once, and the logo sheet names no maker.

From the commit messages and media/launch-film/. CC BY 4.0 = Creative Commons Attribution, which requires credit; blazitt = the Sketchfab user the model's listing names; Mixkit = a free stock-media site; MIT = a permissive open-source license; NTAG215 = an NXP NFC sticker chip.

3. Four Commits, One Co-Author Line

```
$ TZ=America/New_York git log --reverse --date=format-local:'%d %b %Y %I:%M %p' --format='%h  %ad  %s'
1fefcbc  31 Oct 2026 06:17 PM  NFC Cartridges: songs that live on an NFC sticker
4af23d6  31 Oct 2026 08:01 PM  Add MIT license
70693bf  01 Nov 2026 10:57 AM  Add project logo
a00364c  02 Nov 2026 03:02 PM  Add the launch film to the README, with its source
$ git log --reverse --shortstat --format=%h | grep . | paste - -
1fefcbc  16 files changed, 1522 insertions(+)
4af23d6  2 files changed, 25 insertions(+)
70693bf  25 files changed, 150 insertions(+)
a00364c  39 files changed, 2840 insertions(+)
$ git log --format=%an | sort | uniq -c
      4 Andy Yaro
$ git log --format=%B | grep -c 'Co-Authored-By'
1
$ wc -l nfc_cart/*.c nfc_cart/*.h tests/test_ndef.c tools/song_to_rtttl.py
  140 nfc_cart/cart_ndef.c
  220 nfc_cart/cart_song.c
  667 nfc_cart/nfc_cart.c
   25 nfc_cart/cart_ndef.h
   24 nfc_cart/cart_song.h
   48 tests/test_ndef.c
  266 tools/song_to_rtttl.py
 1390 total
```

1. Four commits: the app and license on 31 October, the logo on 1 November, the film on 2 November.
2. The first adds 1,522 lines at once, so the history does not show how the app grew.
3. One author name on all four.
4. One co-author line: the film's, naming Claude.
5. The app is 1,076 lines of C and headers; the test 48; the tool 266 lines of Python.

Run on 3 November 2026 on a copy of the repo. Times are U.S. Eastern. A commit is a saved version in git, a version-control tool; Co-Authored-By is a commit line that credits a co-author. Emails and the film commit's session link are left out. insertions = lines added; .c and .h = C source and header files.

F. Proof and Limits



Checked on 3 November 2026: the build, the host test, the bytes, the burn order in a simulation, and the melody tool on a synthetic song.

Not checked: a phone, a sticker pulled away, or the smallest and largest stickers, the NTAG213 and NTAG216. The only hardware record is one burn and play, recorded for the film.

The last slides set my notes against the repo, list risks and fixes, and say what the work shows.

1. My Claims Against the Repo

My notes said	The repo shows	So I now say
When: 31 October 2026	App, songs, test and tool on 31 Oct; logo 1 Nov; film 2 Nov	First commit 31 October; logo and film by 2 November
Burns songs onto cheap NTAG stickers, plays on tap	Yes; one NTAG215 burn and play on my Flipper	Yes, shown once on an NTAG215
The whole song is an NDEF Text record holding RTTTL	Yes; FMF songs become RTTTL first	Yes, as stated
Phones can read and write cartridges too	No phone code, test or record	Phones should read them; untested
Burns are tear-safe	Length written last; simulated; last write unprotected	A torn burn should read as blank
Verified by reading back	Every song byte compared, up to 3 reads	Yes, as stated
A host test for the NDEF code	Passes; its checks vanish with -DNDEBUG	A host test of the NDEF encoder and parser
An experimental tool turns a recording's vocal melody into a cartridge	It writes RTTTL from a vocal track; you burn it	It turns a vocal track into RTTTL; results vary
"Open source, MIT" (film)	MIT, but the repo is private	MIT-licensed; repo private for now
"Locked or password-protected: refused" (film)	Lock bits: yes, except NTAG203's extra ones; write-only passwords: no	Refuses locked NTAG21x and unreadable stickers

From my career notes, plus two film claims, checked on 3 November 2026. NFC = Near Field Communication; NDEF = NFC Data Exchange Format; NTAG = NXP's NFC sticker chips (NTAG21x = 213, 215 and 216); RTTTL = Ring Tone Text Transfer Language; FMF = Flipper Music Format; MIT = a permissive open-source license; -DNDEBUG switches C's assert checks off.

2. Risks Found in This Review

1. **“Open source” while private**
The film says it and shows the repo’s address, but the repo is private; visitors get a “not found” page.

2. **Untested claims stated as fact**
Phones, tear safety on a real sticker, password refusal, NTAG213 and NTAG216.

3. **How the app was made is unrecorded**
Only the film’s commit names Claude; the first three say nothing either way.

4. **A third-party model in the film**
The Flipper 3D model came with no license; likely CC BY 4.0, credited in the README but not on the end card.

5. **A copyrighted song’s file on screen**
The film’s file browser shows a file named after Wintergatan’s “Marble Machine”; it is not played or in the repo.

6. **Personal data in commits**
All four carry my personal email; the film’s also has a link to a private Claude session.

7. **Silent overwrite**
Burning overwrites any unlocked NDEF sticker, such as one holding a web link, without asking.

8. **Burns tags its README does not list**
NTAG203 and most Ultralight tags pass the type check; some extra lock bytes go unchecked.

Found in this review. CC BY 4.0 = Creative Commons Attribution, which requires credit; NFC = Near Field Communication; NDEF = NFC Data Exchange Format; NTAG213 and NTAG216 = 144- and 888-byte sticker chips; MIFARE Ultralight and NTAG203 are other NXP tag chips.

3. What to Fix Next

1. **Fix the “open source” line**
   Say “MIT-licensed”, or make the repo public first.

2. **Test phones**
   Read and write a cartridge with two phone NFC apps, and record which work.

3. **Pull a sticker on purpose**
   Stop real burns at different pages and log what Play then reads.

4. **Burn NTAG21x only**
   Refuse other types; check page 3’s declared size and the password settings.

5. **Ask before overwriting**
   Show what the sticker holds, and confirm first.

6. **Test the song code and write order**
   Host tests for both; stop the test building with -DNDEBUG; add CI.

7. **Credit or replace the 3D model**
   Confirm its source and license; credit it on the end card, with the license and changes.

8. **Record how it was built**
   Say in the README how the app and film were made, and which AI tools helped, if any.

Rough priority: public claims and real-sticker tests first, then code, then credits. NFC = Near Field Communication; NTAG21x = NTAG213, 215 and 216; CI = continuous integration, tests run on every change.

4. What It Shows, and What It Does Not

It Shows

A Flipper app in C that builds cleanly against the official kit and ran on my device.

A standard format on the sticker: one NDEF Text record of plain RTTTL; 100 bytes for Ode to Joy.

A write order that, simulated with stops between page writes, never leaves half a song; a byte-by-byte read-back.

A launch film built from one real screen recording, with its source in the repo.

It Does Not Show

A phone reading or writing a cartridge.

A sticker pulled away mid-burn, or an NTAG213 or NTAG216 on real hardware.

Tests beyond the NDEF code.

Who or what wrote the app's code.

A small Flipper app with a clear idea, a standard format and a careful write order, shown working once on real hardware. Next: test phones and pulled stickers, add tests for the song code, and fix the "open source" line.

NFC = Near Field Communication; NDEF = NFC Data Exchange Format; RTTTL = Ring Tone Text Transfer Language; NTAG213 and NTAG216 = the smallest and largest NTAG21x stickers, 144 and 888 bytes.