

BREATH OF THE RF FIELD

Hacking Amiibo with Software-Defined Radio

James Chambers

@jamchamb_





INTRODUCTION

AMIIBO®





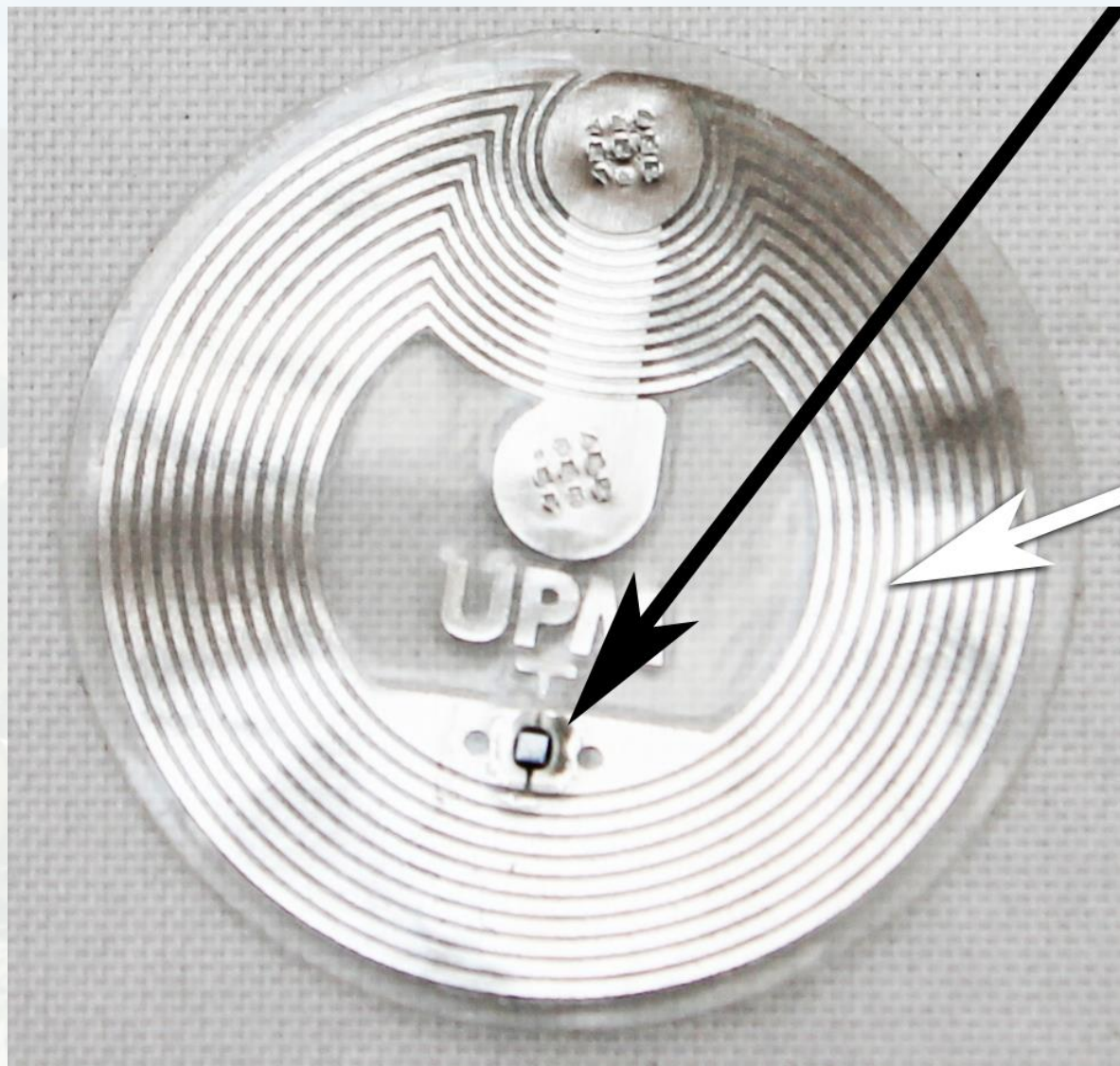
AMIIBO



AMIIBO

- Contain an NFC “tag”
- Near Field Communication (NFC)
 - Set of protocols for contactless power and data transfer between devices within a few centimeters of each other
- Amiibo use NTAG215
 - Part of the NTAG21x specification, a specific kind of NFC tag made by NXP Semiconductors





<https://gototags.com/blog/whats-the-difference-nfc-tags-v-nfc-chips/>

TWILIGHT HACK

- Buffer overflow triggered with horse name, which is set by user and kept in save file
 - Save file edited using crypto keys obtained from hardware exploit
 - Edit save to smash the stack
- Used in software-based attack chain for installing homebrew software manager



EponAA

See "Console Hacking 2008: Wii Fail" by Team Twiizers (fail0verflow)

TWILIGHT HACK



TWILIGHT BLACK



TWILIGHT HACK

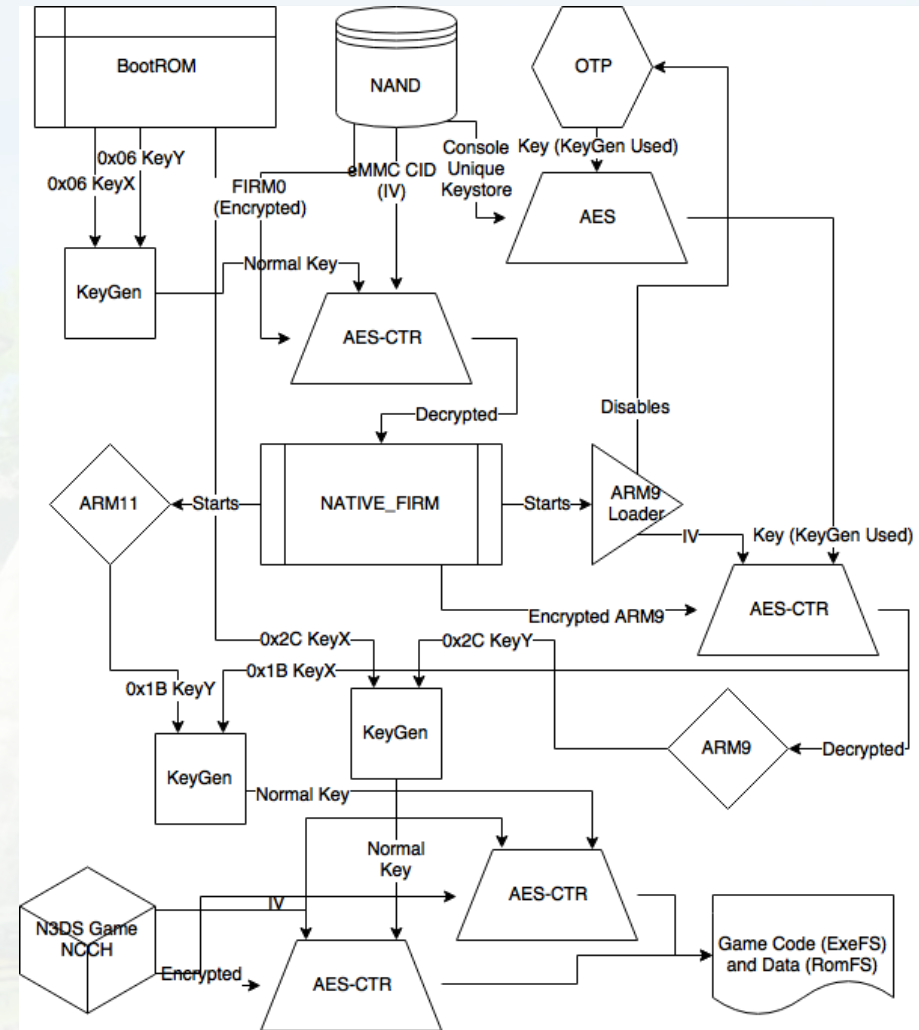
```
OK:
Getting SD card status: 00000005
sd_reset_card(): got reply = 63000000
Card status: 00000700 [STANDBY]
Selecting card:
Card status: 00000900 [TRANSFER]
sd_set_blocklength(512)
sd_set_bus_width()
SD card detected
USBGecko serial interface detected
Loading FAT:
DEM Name: MSDOS5.0, 38 reserved sectors, FAT32.
Fat size = 1df200
FAT starts at sector 0x125
/ starts at sector 0x 1f17
Reading boot.dol:
start cluster = 0000
start cluster = 0003
start cluster = 0515
Found boot.dol!
done, filesize is 1588672
** No elf image at address 90100000
DOL image detected?
relocating 00000100 to 802191a0 (2016 bytes)
relocating 000008e0 to 80219980 (1586400 bytes)
clearing BSS (at 00000000, 0 bytes...)
```


TWILIGHT HACK



TWILIGHT HACK

- Like smartphones, modern game consoles put users in “jail”
- Meant to prevent piracy and cheating
 - Consoles often sold at a loss, in hopes of profiting from games and services
- No custom software allowed



Yifan Lu – “The 3DS Cryptosystem”

<https://yifan.lu/2016/04/06/the-3ds-cryptosystem/>





Splatoon 2 Series amiibo



Use the Splatoon 2 amiibo to unlock the outfits below! The Splatoon 1 variants of these amiibo unlock the outfits from Splatoon 1!

amiibo can also be used for storing loadouts and settings to use it on any Switch console when you use the amiibo! The original amiibo for Splatoon 1 can be used for this function as well!

AMIIBO

- How easy would it be to clone or spoof them?
- Can the data they store be used for “save game” style exploits?
- Could they be used to trigger an exploit on the Nintendo Switch?

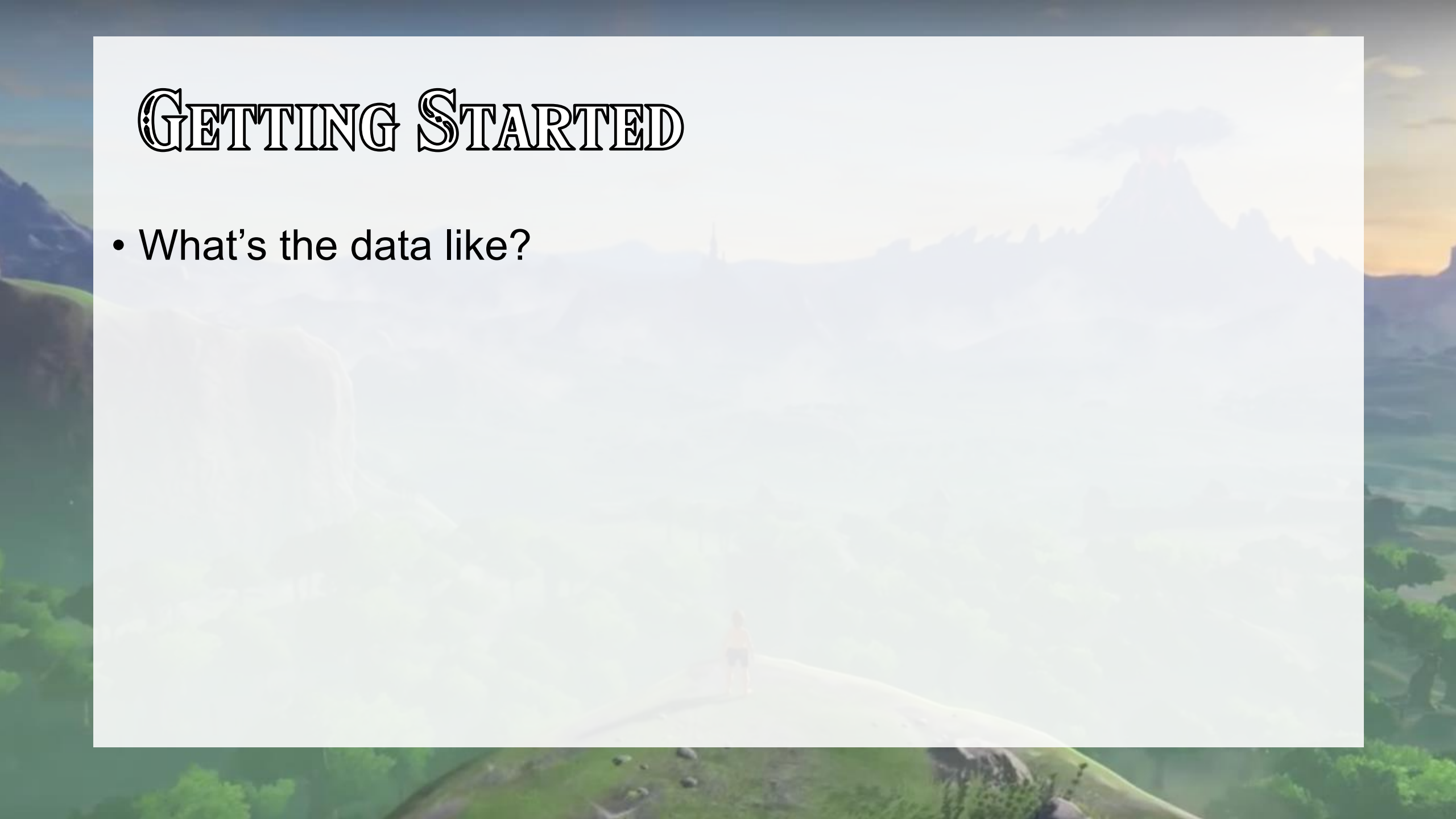


GETTING STARTED



GETTING STARTED

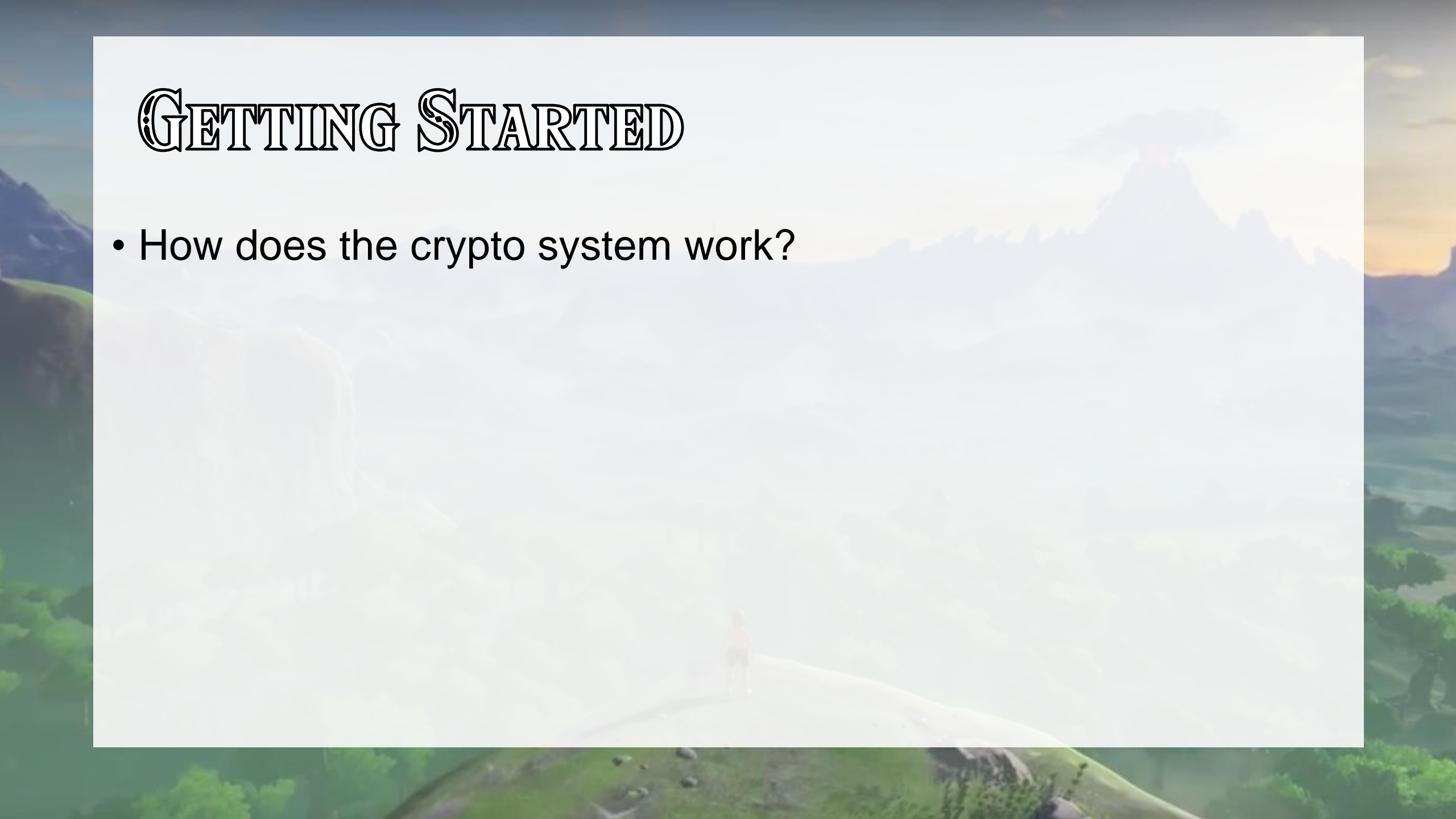
- What's the data like?



00000000	04	B4	96	AE	EA	21	4B	80	00	48	0F	E0	F1	10	FF	EE	A5	00!K..H.....
00000012	00	00	86	FB	24	3F	9A	A8	29	10	E0	48	41	F9	F6	8A	E9	61\$?..) ..HA....a
00000024	D4	DD	01	3E	BD	B2	EA	DF	4D	88	27	56	C1	AF	3F	AF	1D	A0	...>....M.'V..?...
00000036	28	9A	3E	C6	B2	AC	77	77	55	62	7A	25	DD	63	94	11	7C	65	(.>...wwUzb%.c... e
00000048	86	DA	E8	9F	1B	84	DD	87	A9	78	E3	FA	19	27	00	00	00	26x...'...&
0000005a	00	02	0D	12	C5	09	F0	48	46	FC	0C	82	2D	E2	21	94	8D	7CHF...-...!...
0000006c	52	8F	66	30	08	24	E2	A0	1C	AC	4E	36	17	21	23	B3	C9	A0	R.f0.\$....N6.!#...
0000007e	BD	5C	69	DB	41	A6	94	B6	97	08	97	83	06	49	28	12	D8	49	..i.A.....I(..I
00000090	D8	F6	92	07	C3	B4	59	53	39	D6	AA	21	32	31	1B	17	2C	9BYS9..!21...,,
000000a2	80	38	8B	19	8D	23	C3	80	BF	E7	8F	68	51	95	18	94	1E	3C	.8...#.....hQ....<
000000b4	82	DB	14	53	98	20	14	41	56	A7	A1	A1	E4	60	A5	28	35	14	...S. .AV.....`. (5.
000000c6	4E	1E	82	59	4D	39	8B	E8	0C	3D	DC	DF	BC	9F	49	92	68	6A	N..YM9...=....I.hj
000000d8	2E	C0	32	BF	D6	19	95	7B	8D	EC	17	D7	1E	A2	A3	5C	5B	4F	..2....{.....\[O
000000ea	A1	91	D3	0F	05	49	EF	3F	7F	9F	A0	C6	76	70	DE	62	C3	09I.?....vp.b..
000000fc	38	76	FD	D0	0D	DF	AA	E5	D4	A5	34	8B	C6	42	C5	28	06	9B	8v.....4..B. (...
0000010e	CE	13	3F	44	6A	C4	CA	7F	F4	3C	D0	78	BE	F8	60	25	06	ED	..?Dj.....<.x..`%..
00000120	D1	01	5C	69	05	85	51	01	A9	B8	E0	6C	53	DF	19	6A	C3	C1	..i..Q....lS..j..
00000132	DF	11	08	7E	7D	B5	AF	13	52	F3	C7	DB	CC	8F	E6	13	48	03	...~}...R.....H.
00000144	AE	02	11	DA	18	3A	34	85	3C	54	78	AE	8B	04	C7	C2	68	CE:4.<Tx.....h.
00000156	14	E0	26	F1	35	3D	0E	14	F8	3C	26	F4	E1	0B	0D	88	23	D4	..&.5=...<&.....#.
00000168	37	43	C5	40	C5	A0	53	5C	9B	FD	2B	82	F5	F5	67	B9	EF	B9	7C.@...S\...+...g...
0000017a	1D	2F	09	9B	DD	98	CA	10	98	40	56	60	97	E9	58	5E	CA	EA	./.....@V`..X^...
0000018c	81	65	27	2F	24	DA	AA	DE	95	95	02	3F	A2	FF	15	F4	0C	5C	.e'/\$.....?.....\
0000019e	74	EB	0A	A0	16	59	B6	A7	A3	6B	A2	96	DF	60	F0	1E	DF	1F	t....Y...k....`....
000001b0	F2	23	7B	8B	2E	E7	CA	95	AC	47	33	E6	9B	A3	89	80	5F	B9	.#{.....G3....._.
000001c2	BC	32	99	E2	22	05	83	66	59	49	13	85	90	E0	EC	C0	97	1C	.2.."..fYI.....
000001d4	89	ED	28	AF	F3	88	97	6C	EF	75	FF	D1	35	08	8E	88	2C	A0	..(.....l.u..5....,,
000001e6</																			

GETTING STARTED

- How does the crypto system work?



GETTING STARTED

Crypto

Details we do know:

- Amiibo data is encrypted
- Unique keys for each Amiibo
- Can't copy data from one Amiibo to another
- AES and HMAC are involved
 - Encrypted data is also signed



GETTING STARTED

Crypto

- amiitool by socram8888 can perform encryption and decryption, but the crypto keys are kept private:
 - “Please note that **keys used for signing and encryption are copyright of Nintendo and therefore they can't be shared, and I won't share post them here, send them using a private message, or anywhere else.**”
- Often, the homebrew/game hacking community does not make crypto keys public to avoid enabling piracy.

GETTING STARTED

Crypto

- Private online APIs for decrypting/encrypting Amiibo
 - Crypto keys stored on server, inaccessible to user
- Need to get an API key from the dev
- Activity on API is monitored
- Operations would be throttled by response time, possible rate limiting, and server downtime or decommission
 - Every payload attempt would require waiting for an API request

GETTING STARTED

Crypto

- Amiibo cheat devices use a similar model
 - Online API performs all operations that require crypto
 - PowerSaves device has API authentication tied to hardware
 - “Reversing Powersaves for Amiibo”
<http://blog.ghettoha.xxx/reversing-powersaves-for-amiibo/>
- Cheat device manufacturers don't want to reveal the crypto secrets, they want you to buy their device



GETTING STARTED

- How can we read, copy, and tamper with the data?
- How can we simulate an Amiibo?

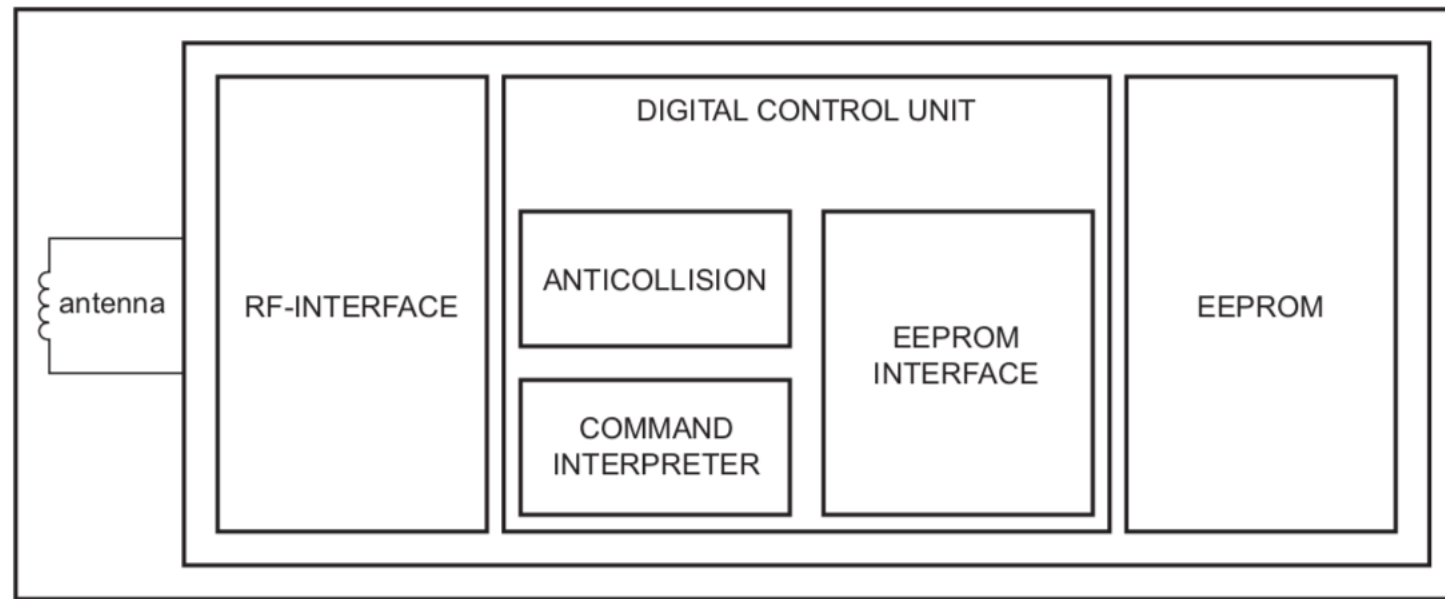
GETTING STARTED

Proxmark 3

- SDR device for RFID and NFC
- Primarily used to research proximity ID cards
- Open source, reprogrammable microcontroller and FPGA
- Can be used to simulate a reader/writer or tag



GETTING STARTED



aaa-006979

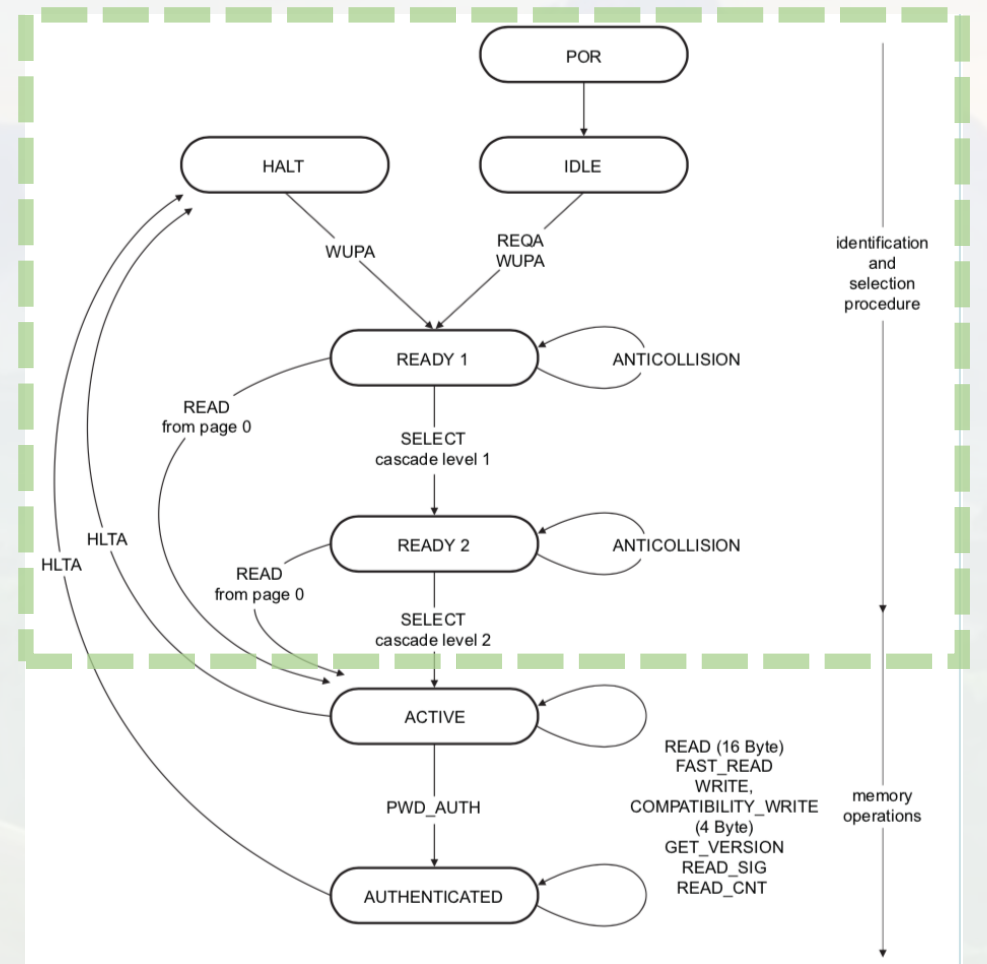
Fig 2. Block diagram of NTAG213/215/216

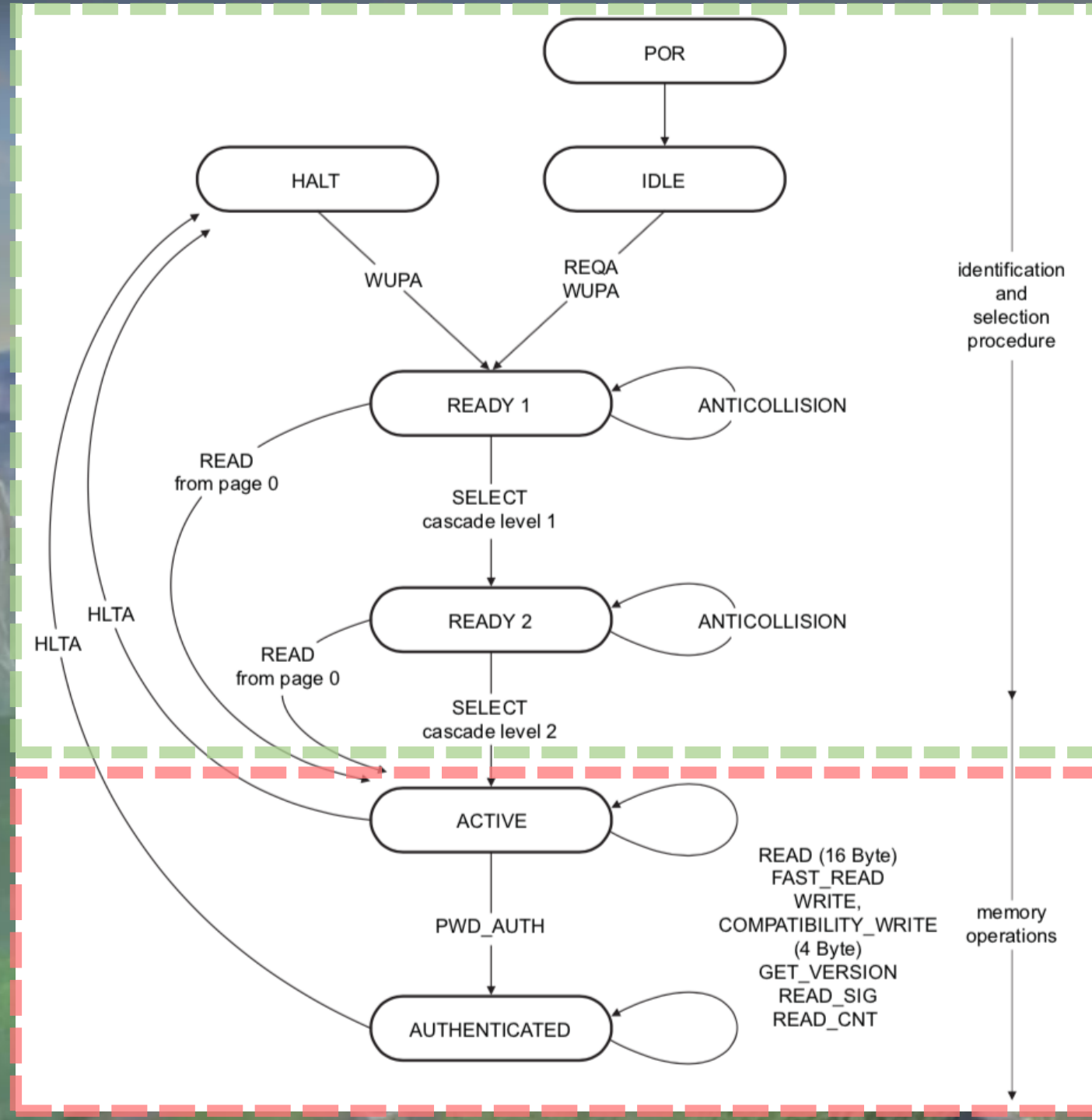
GETTING STARTED

Proxmark 3

- Has basic ISO 14443A simulation
- Supports initial wakeup, identification, and selection (anti-collision)
- Can tell an NFC reader its serial number. That's all.

Need to roll our own NTAG21x simulator.





GETTING STARTED

To read, edit, and simulate Amiibo we need to:

- Make an NTAG215 simulator for Proxmark
- Obtain the Amiibo crypto keys
- Integrate Amiibo crypto operations with Proxmark simulator



+



CRYPTO CAVERNS

The image is a screenshot from a video game, likely 'Cryptic Caverns'. It depicts a dark, atmospheric cave interior. The architecture is composed of large, angular stone blocks, some of which are illuminated from within, creating a glowing effect. A central staircase leads up to a bright, glowing altar or shrine in the distance. The scene is lit by several glowing blue torches, which cast a cool, blue light. The overall mood is mysterious and ancient. The title 'CRYPTO CAVERNS' is overlaid in the center in a stylized, gothic font.

CRYPTO CAVERNS

- Jailbreaks already existed for 3DS
 - New 3DS has built-in NFC/Amiibo compatibility
 - (watch the “Breaking the 3DS” CCC presentation!)
- Reverse engineering the software will be much easier
 - Can get decrypted game and service binaries
 - Can tamper with processes



CRYPTO CAVERNS

- NTR CFW: Custom firmware for 3DS
- Adds a debugger to the 3DS
- Sadly, breakpoints don't work
- Limited to peek & poke
 - Read and tamper with memory
 - View registers at random moments

NTR CFW 3.3

```
Process Manager
Enable Debugger
Set Hotkey
* Take Screenshot
Real-Time Save (Experimental)
Power
Game Plugin
```

<http://44670.org/ntr>

NTR Debugger

Tools Credits Version

```
pid: 0x0000001c, pname: http, tid: 0004013000002902, kpobj: fff7a070
pid: 0x0000001d, pname: ssl, tid: 0004013000002f02, kpobj: fff7a2e0
pid: 0x0000001e, pname: cecd, tid: 0004013000002602, kpobj: fff7a550
pid: 0x0000001f, pname: friends, tid: 0004013000003202, kpobj: fff7a7c0
pid: 0x00000020, pname: ac, tid: 0004013000002402, kpobj: fff7aa30
pid: 0x00000021, pname: boss, tid: 0004013000003402, kpobj: fff7aca0
pid: 0x00000022, pname: act, tid: 0004013000003802, kpobj: fff7af10
pid: 0x00000023, pname: news, tid: 0004013000003502, kpobj: fff7b180
pid: 0x00000024, pname: ndm, tid: 0004013000002b02, kpobj: fff7b3f0
pid: 0x00000025, pname: nim, tid: 0004013000002c02, kpobj: fff7b660
pid: 0x00000026, pname: dlp, tid: 0004013000002802, kpobj: fff7b8d0
pid: 0x00000036, pname: NFC 0xDE, tid: 000400000dbef00, kpobj: fff7bb40
pid: 0x00000037, pname: nfc, tid: 0004013000004002, kpobj: fff7bdb0
pid: 0x00000038, pname: ro, tid: 0004013000003702, kpobj: fff7c020
```

end of process list.

> memlayout(pid=0x37)

null

valid memregions:

00100000 - 00168fff, size: 00069000

08000000 - 08000fff, size: 00001000

0fff8000 - 0fffffff, size: 00008000

end of memlayout.

Basic | EUR | USA | JPN | Gateshark | Debugging |

192.168.0.8

Connect

Hello! - Test connection

Disconnect

37

List processes

Memlayout

100000 -> 168FFF [69000]

filename

Dump memory

valid memregions:

00100000 - 00168fff, size: 00069000

08000000 - 08000fff, size: 00001000

0fff8000 - 0fffffff, size: 00008000

end of memlayout.

Addr

Read at Addr

4

Value

Write to Addr (HEX)

Value

Write to Addr (DEC)

CRYPTO CAVERNS

- Try to dump and reverse engineer the NFC service to:
 - Find the crypto keys
 - Fully understand crypto system
 - Understand how games, NFC service, and Amiibo interact
- The encrypted or decrypted Amiibo images may be accessible in dumped memory
- Static analysis of binary

CRYPTO CAVERNS

- Wrote a custom homebrew app that uses the NFC service:
 - Scan for Amiibo
 - Read its content
 - Request app data, tag info (can't request raw dump)



nfc_100000_168FFF ✕	nfc_FFF8000_FFFFFFFF ✕
0005db24	AE 10 F6 A9 0C C5 10 97 A2 D6 8A 7F A4 DB F5 35 5D 4A 62 8E
0005db38	A2 02 8D 97 1F 5B 4B E4 27 B9 79 16 C7 12 F1 7B DC 31 18 E5
0005db4c	BE 37 75 48 53 56 68 B7 04 3D 36 87 4A B3 49 81 19 19 00 00
0005db60	00 09 00 02 0D 12 B5 04 1A 90 97 F4 D0 17 87 08 42 FF E5 01
0005db74	54 2D B1 63 F1 BF E9 7A 96 F6 6A 6A 3C 82 69 58 C5 0D 45 73
0005db88	01 00 0F BD 00 00 00 04 5F 00 00 00 00 00 00 00 00 00 00
0005db9c	31 48 0F E0 F1 10 FF EE F2 40 7E 5F 75 1E D3 04 AE 86 3A A3
0005dbb0	AC F5 81 1B 86 88 A7 AD A4 49 27 FA 0C 3F C8 B4 75 E0 8D 38
0005dbc4	A5 00 02 00 30 31 00 01 00 3C 00 3C 83 F6 E7 95 00 50 00 65
0005dbd8	00 65 00 6B 00 61 00 6E 00 64 00 70 00 77 00 6E 03 00 00 30
0005dbec	C0 9E 3C 68 D2 B7 8C 4D 90 12 15 17 7C BB 8A 5D A9 22 00 00
0005dc00	48 00 53 00 77 00 61 00 67 00 6D 00 69 00 69 00 00 00 00 00
0005dc14	00 00 40 40 01 00 21 01 02 68 44 18 26 34 46 14 81 12 17 68
0005dc28	0D 00 00 29 00 52 48 50 4A 00 61 00 6D 00 65 00 73 00 00 00
0005dc3c	00 00 00 00 00 00 00 00 00 00 00 49 14 00 04 00 00 00 0E DF 00
0005dc50	00 02 10 11 0E 00 6B 87 E4 D7 AB 07 89 1D 9E 9E 50 40 01 33
0005dc64	98 EF 71 F7 2A 21 12 E9 71 C8 24 38 92 BC 82 47 8B C0 1F 07
0005dc78	00 00 00 00 DF DD 97 7B 00 00 00 00 00 00 FF FF FF 00 00 00 00
0005dc8c	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0005dca0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0005dcb4	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0005dcc8	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0005dcdc	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0005dcf0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0005dd04	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 31 59 FD 8B
0005dd18	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0005dd2c	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0005dd40	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 5D 4A 62 8E
0005dd54	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0005dd68	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0005dd7c	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0005dd90	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0005dda4	01 00 0F BD 00 00 00 04 5F 00 00 00 00 00 00 00 00 00 00 00 00
0005ddb8	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0005ddcc	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

```

.....5]Jb.
.....[K.'y....{.1..
.7uHSVh..=6.J.I.....
.....B...
T-.c....z..jj<.iX..Es
....._.....
lH.....@~_u.....:
.....I'..?..u..8
....0l...<.<....P.e
.e.k.a.n.d.p.w.n...0
..<h...M....|..].."
H.S.w.a.g.m.i.i.....
..@@..!..hD.&4F....h
...).RHPJ.a.m.e.s...
.....I.....
.....k.....P@.3
..q.*!..q.$8...G....
.....{.....
.....
.....lY..
.....]Jb.
.....
.....
.....
....._.....
.....
.....
.....

```

Decrypted Amiibo image

31	48	0F	E0	F1	10	FF	EE	F2	40	7E	5F	75	1E	D3	04	AE	86	3A	A3	1H.....@~_u.....:.	
AC	F5	81	1B	86	88	A7	AD	A4	49	27	FA	0C	3F	C8	B4	75	E0	8D	38I'..?..u..8	
A5	00	02	00	30	31	00	01	00	3C	00	3C	83	F6	E7	95	00	50	00	6501...<.<.....P.e	
00	65	00	6B	00	61	00	6E	00	64	00	70	00	77	00	6E	03	00	00	30	.e.k.a.n.d.p.w.n...0	
C0	9E	3C	68	D2	B7	8C	4D	90	12	15	17	7C	BB	8A	5D	A9	22	00	00	..<h...M... ...]."...	
48	00	53	00	77	00	61	00	67	00	6D	00	69	00	69	00	00	00	00	00	H.S.w.a.g.m.i.i.....	
00	00	40	40	01	00	21	01	02	68	44	18	26	34	46	14	81	12	17	68	..@@...hD.&4F....h	
0D	00	00	29	00	52	48	50	4A	00	61	00	6D	00	65	00	73	00	00	00	...) RHPJ.a.m.e.s...	
00	00	00	00	00	00	00	00	00	00	49	14	00	04	00	00	00	00	0E	DF	00I.....

Plaintext 16-bit character strings:
nickname, owner, Mii names

Offset	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	0123456789ABCDEF
000000h	00	48	0F	E0	F1	10	FF	EE	2A	B1	40	B6	F1	8F	2E	7F	H=α±> ε*⊗ ±Å.Δ
000010h	F3	F7	70	10	71	AE	98	5D	F9	EA	99	30	CB	E5	BE	4B	≤≈p>q<ÿ]•Ω00TσJ K
000020h	94	2D	48	46	1A	A0	1F	6D	A5	00	01	00	10	31	00	01	ö-HF→á▼mÑ ☺ ▶1 ☺
000030h	22	7B	22	7B	83	F6	E7	95	00	6A	00	69	00	67	00	67	"{"{â÷τò j i g g
000040h	00	6C	00	79	00	31	00	00	00	00	00	00	03	00	00	30	l y 1 ♥ 0
000050h	C0	9E	3C	68	D2	B7	8C	4D	90	12	15	17	7C	BB	8A	5D	Ls<hTtîME±§z ηè]
000060h	A9	22	00	00	48	10	53	00	77	00	61	00	67	00	6D	00	~" H>S w a g m
000070h	69	00	69	00	00	00	00	00	00	00	67	39	01	00	67	03	i i g9⊗ g♥
000080h	31	67	63	16	74	14	45	12	86	10	16	64	4D	00	1C	25	lgc=τ¶E±â→=dM L%
000090h	D2	49	48	50	4A	00	61	00	6D	00	65	00	73	00	00	00	T IHPJ a m e s
0000A0h	00	00	00	00	00	00	00	00	00	00	46	63	2F	7C	4A	10	Fc/ J>
0000B0h	4E	B8	1C	12	00	01	45	D6	7E	EE	D0	73	60	CA	84	A6	NηL± ⊗Eπ~εLs`=äa
0000C0h	0F	94	55	94	C3	7A	B6	D2	B9	A0	D3	C3	99	8D	DB	04	*öUö z T á Öi ◆
0000D0h	B2	78	49	FC	95	81	05	94	FC	5F	A2	04	EE	05	8F	73	⊗xI"òü♣ö" ó♦ε♣As
0000E0h	8E	0A	3B	CA	32	9A	B8	CF	59	7C	FE	7B	3E	82	3D	F0	ÄQ;=2Üj±Y ■{ >é==
0000F0h	13	CC	C0	E8	80	88	10	8D	8E	C1	98	64	D8	B2	E3	B4	!! LøÇê>îÄ-ÿd+⊗π
000100h	11	60	34	8E	51	11	D1	17	BF	79	A2	19	83	89	55	7A	◀4ÄQ←τ±ηyó±âëUz
000110h	21	B1	F1	22	44	A7	F2	06	F9	AE	3B	70	01	54	F7	A7	!⊗±"Dº≥♣•«;p⊗T≈º
000120h	99	4A	6A	DE	4D	AE	9F	01	FA	A9	28	07	E6	F7	0A	04	ÖJj M«f⊗.~(•μ≈♣♦
000130h	6C	1E	42	26	C1	C0	6F	69	6D	2E	95	A2	D5	61	15	6B	l▲B&LLoim.òòFa§k
000140h	82	FA	7F	38	1F	76	2C	73	12	EF	27	9B	7E	EB	20	01	é•Δ8▼v, s±n'ç~δ ☺
000150h	A4	36	03	E3	11	D1	E6	D8	DB	00	70	1E	18	FC	2C	99	ñ6♥π←τμ+■ p▲†" ,Ö
000160h	93	BD	85	EB	78	D1	E5	F5	F1	CF	13	D1	99	D4	EB	82	ô àδxτσ ±±!!τÖ kδé
000170h	0E	93	FE	9E	5D	31	5A	B0	14	A7	5D	1E	05	0E	5A	16	♪ô♣♣]1Z:¶¶]▲♣♪Z-
000180h	C2	8E	07	C1	65	35	89	D2	51	0D	F2	42	BD	14	3F	07	τÄ•±e5ëτQJ≥B ¶?•
000190h	15	EE	10	85	FD	8E	3B	61	63	D4	D6	D1	74	C7	F0	E3	§ε→à²Ä;acLττt ≡π
0001A0h	74	FD	0F	0A	CC	29	1C	D9	09	2C	4D	50	3F	17	F3	00	t²*⊗)LJ O,MP?±≤
0001B0h	67	56	08	17	1D	A0	28	9A	3E	C6	B2	AC	77	77	55	62	gVQ±→á(Ü> ⊗¼wwUb
0001C0h	7A	25	DD	63	94	11	7C	65	86	DA	E8	9F	1B	84	DD	87	z% cö-> eäτøf-ä ç
0001D0h	A9	78	E3	FA	04	B4	96	AE	EA	21	4B	80	19	27	00	00	~xπ•♦ û«Ω!KÇ↓'
0001E0h	00	26	00	02	0D	12	C5	09	F0	48	46	FC	0C	82	2D	E2	& ⅈ± O≡HF"qé-Γ
0001F0h	21	94	8D	7C	52	8F	66	30	08	24	E2	A0	1C	AC	4E	36	!öì RÄf0\$ΓáL¼N6
000200h	17	21	23	B3	C9	A0	BD	5C	01	00	0F	BD	00	00	00	04	±!# Fá ⊗ * ♦
000210h	5F	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	

Signature, settings,
timestamps, write counter

Amiibo nickname;
Registered owner Mii

Application ID, hash

Game/Application data

Signature, serial, character
ID

CRYPTO CAVERNS

- Loaded up the NFC service binary in Hopper
- Searched for intelligible strings
- Immediately found... something suspicious
 - “locked secret” and “unfixed infos”



CRYPTO CAVERNS

"unfixed infos", 0



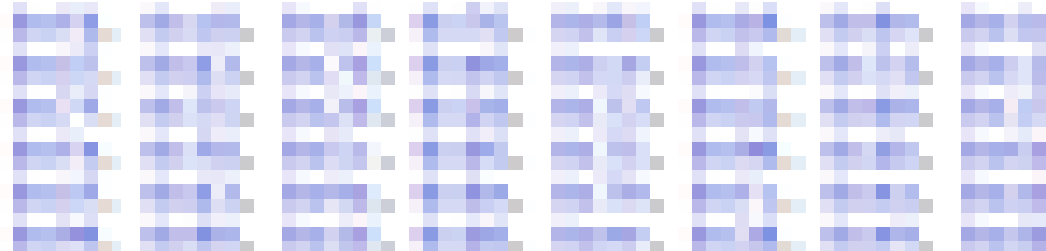
"locked secret", 0



"unfixed infos", 0



"locked secret", 0



CRYPTO CAVERNS

- Need to figure out how these bytes are used
- Difficulties with static analysis
 - Poor automatic code identification due to the way the service binary is set up
 - No main entry point that calls down to all the other code
 - Series of service call stubs that call isolated functions

CRYPTO CAVERNS

- Broken debugger, tons of unidentified code, disassembler crashes on Ctrl+F... what can I do?



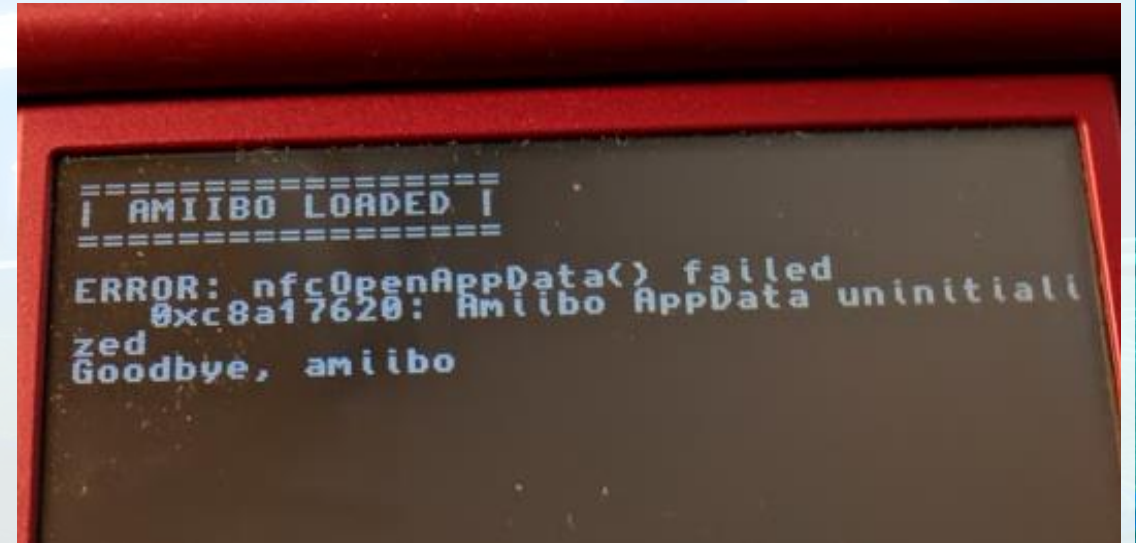
CRYPTO CAVERNS

- Use existing reverse-engineered NFC service IPC calls from the homebrew library to figure out the locations of related code in the binary dump



CRYPTO CAVERNS

- 0xC8A17620 “Invalid state” error code appears often when calling functions at the wrong time



CRYPTO CAVERNS

- Add new features to custom app to help identify code in the binary
- Version 1
 - Add a bunch of button bindings to trigger all known IPC calls
 - Connect debugger
 - Replace “invalid state” error code with `0xDEADBEEF` and call random functions
 - Find where `0xDEADBEEF` error shows up



NFC Status: Scanning in progress

NFC tag ID: 04 b4 96 ea 21 4b 80

Country code: 00

Initialized 01/03/2018

Owner Mii: Mr. Mii

Nickname: jiggly

App data:

00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00

L::	Init app data	R: Write app data
A::	Repopulate app data	D-Left: Smash data
B::	nfcExit()	D-Right: reset scan state
X::	nfcLoadAmiiboData()	D-Up: Cycle IPC
Y::	nfcStartScanning()x2	D-Down: Comm Status
	Press Start to exit.	

=====

CRYPTO CAVERNS

- Version 2
 - Replace the “invalid state” error codes with address of the function that sets them
 - Button mash and find lots more addresses

CRYPTO CAVERNS

- Version 3
 - Programmatically rewrite select function prologues to immediately return address of function
 - Works even if not in invalid state
 - Observe function addresses in error codes
 - Navigate down found branch and select more functions to rewrite

```
def make_ops(writer, target, pid):  
    start_addr = target[0]  
    regs = target[1]  
  
    payload = ()  
    payload += (regs, 0xb5)      # push <regs>  
    payload += (0x78, 0x46)     # mov r0, pc  
    payload += (0x06, 0x38)     # sub r0, r0, #6 ; fix PC  
    payload += (regs, 0xbd)     # pop <regs>  
  
    writer(start_addr, payload, pid)  
    return 'nc.write(0x%x, %s, pid=0x%x)' % (target[0],  
                                              str(payload),  
                                              pid)
```


CRYPTO CAVERNS

```
## Populate Amiibo data ##
ERROR: nfcGetTagInfo() failed
ERROR: nfcGetAmiiboSettings()
0x00113800: Unknown NFC err
ERROR: nfcGetAmiiboSettings()
0x001137dc: Unknown NFC err
ERROR: nfcOpenAppData() failed
0x001138f8: Unknown NFC err
## Init App Data ##
ERROR: nfcInitializeWriteAppDa
0x00113878: Unknown NFC err
## Test IPC calls ##
Calling 0x00:
invalid command
Result: 0xd900182f
## Test IPC calls ##
Calling 0x01:
cmdbuf[0] = 0x00010040
cmdbuf[1] = 0x00000000
cmdbuf[2] = 0x00025800
Result: 0x00113b24
```

```
stub_nfcGetAmiiboSettings:
00113800    push    {r4, lr}
00113802    mov     r4, r1
00113804    movs    r1, #0x11
00113806    lsls    r1, r1, #0x5
00113808    adds    r0, r0, r1
0011380a    ldrb     r0, [r0]
0011380c    cmp     r0, #0x2
0011380e    beq     stub_nfcGetAmiiboSettings+20

00113810    ldr     r0, = 0xc8a17600
00113812    pop     {r4, pc}
; endp

00113814    bl      usedBeforeAmiiboAPICall
00113818    mov     r1, r4
0011381a    bl      nfcGetAmiiboSettings
0011381e    pop     {r4, pc}
; endp
00113820    dd      0xc8a17600
```


keyinfo_stuff_happens_here:

```
push    {r0, r1, r2, r3, r4, r5, r6, r7, lr}
cmp     r2, #0x40
sub     sp, #0x54
mov     r4, r1
mov     r5, r3
beq     0x114674
```

0x114674:

```
movs    r1, #0x40
add     r0, sp, #0x10
blx     0x11db28
movs    r2, #0x40
mov     r1, r4
add     r0, sp, #0x10
blx     memcpy
ldr     r0, = 0x1ff80000
ldrb    r0, [r0, #0x14]
lsls    r0, r0, #0x1f
asrs    r0, r0, #0x1f
adds    r0, r0, #0x1
beq     0x114696
```

0x114692:

```
ldr     r4, = 0x135584
b       0x11469a
```

0x114696:

```
ldr     r4, = 0x135584
adds    r4, #0x6a
```

ENVINFO [\[edit\]](#)

Bit	Description
0	Clear for developer unit, set for retail.(See 0x1FF80015)
1	IsJtagConnected
2-7	?

Check last bit of
ENVINFO
in config memory

Pick developer or
retail key info

HMAC Algorithm

Keys longer than *blockSize* are shortened by hashing them
`if (length(key) > blockSize) then`
 `key ← hash(key) //Key becomes outputSize bytes long`

Keys shorter than *blockSize* are padded to *blockSize* by padding with zeros on the right
`if (length(key) < blockSize) then`
 `key ← Pad(key, blockSize) //pad key with zeros to make it blockSize bytes long`

`o_key_pad = key xor [0x5c * blockSize] //Outer padded key`
`i_key_pad = key xor [0x36 * blockSize] //Inner padded key`

`return hash(o_key_pad || hash(i_key_pad || message)) //Where || is concatenation`

```
hmac_shal:
push    {r0, r1, r2, r3, r4, r5, r6, r7, lr}
sub     sp, #0x1fc
sub     sp, #0xc8
mov     r6, r3
mov     r7, r2
ldr     r5, [sp, #0x2e8 + arg_4]
ldr     r1, [sp, #0x2e8 + ret_addr]
r4, sp, #0x220
r5, #0x40
bls     0x128c16
```

```
0x128c0c:
movs    r2, #0x40
mov     r0, r4
blx     0x128c28
```

```
0x128c16:
mov     r2, r5
mov     r0, r1
blx     memcpy
r0, #0x40
r1, r0, r5
r0, r4, r5
sub     11a7b0
```

```
0x128c28:
movs    r0, #0x0
movs    r3, #0x36
add     r5, sp, #0x260
```

```
0x128c2e:
ldrb    r2, [r4, r0]
eors    r2, r3
strb    r2, [r5, r0]
adds    r0, r0, #0x1
cmp     r0, #0x40
blo     0x128c2e
```

```
0x128c3a:
movs    r2, #0x40
mov     r1, r5
mov     r0, sp
blx     memcpy_big
r1, r7
r0, sp, #0x40
r2, r6
r7, r0
blx     memcpy
r2, r6
adds    r2, #0x40
mov     r1, sp
add     r0, sp, #0x2a0
bl      sub_12cde8
movs    r0, #0x0
movs    r1, #0x5c
```

```
0x128c60:
ldrb    r2, [r4, r0]
eors    r2, r1
strb    r2, [r5, r0]
adds    r0, r0, #0x1
cmp     r0, #0x40
blo     0x128c60
```



```
int lots_of_amiibo_crypto(int arg0, int arg1) {
    sp = sp - 0x440;
    r5 = arg0;
    memcpy(sp + 0x2c, arg1 + 0x10, 0x24);
    memcpy(sp + 0x1b8, arg1 + 0x34, 0x20);
    memcpy(sp + 0x1e0, arg1 + 0x54, 0x2c);
    memcpy(sp + 0xc, arg1 + 0x80, 0x20);
    memcpy(sp + 0x50, arg1 + 0xa0, 0x168);
    var_108 = sp + 0x20c;
    memcpy(sp + 0x20c, arg1 + (0x41 << 0x3), 0x14);
    sub_11db28(sp + 0x220, 0x40);
    memcpy(sp + 0x240, arg1 + 0x60, 0x20);
    *(sp + 0x2c) = 0xa5;
    *(sp + 0x2d) = 0x0;
```

Hopper pseudo-code of NFC
service

```
memcpy(0xc + 0x1b4, arg1 + 0x1b4, 0x034);
}

void nfc3d_amiibo_tag_to_internal(const uint8_t* tag) {
    memcpy(intl + 0x000, tag + 0x008, 0x008); //
    // e & Lock bytes & capability container
    memcpy(intl + 0x008, tag + 0x080, 0x020); //
    memcpy(intl + 0x028, tag + 0x010, 0x024); //
    memcpy(intl + 0x04c, tag + 0x0A0, 0x168); //
    memcpy(intl + 0x1B4, tag + 0x034, 0x020);
    memcpy(intl + 0x1D4, tag + 0x000, 0x008);
    memcpy(intl + 0x1DC, tag + 0x054, 0x02C);
    // ...
}
```

amiitool

CRYPTO CAVERNS

- Tried to figure out how to use it with amiitool, but there isn't enough data to build a full key file
- Using the content I did have, I tried searching for the keys online...

```
        ; DEVELOPER KEY INFO
"unfixed infos", 0
[REDACTED]
"locked secret", 0
[REDACTED]
        ; RETAIL KEY INFO
"unfixed infos", 0
[REDACTED]
"locked secret", 0
[REDACTED]
```

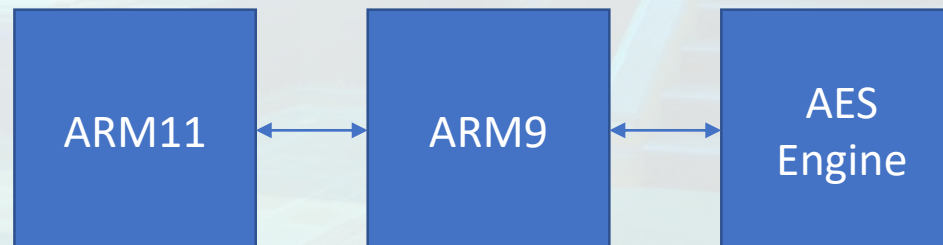

CRYPTO CAVERNS

- Found some interesting notes...
- Found the whole key file...
- It contains a value not in the NFC service dump; what is it?
- What hasn't shown up yet?
The 3DS's dedicated AES hardware

```
    ; DEVELOPER KEY INFO
    "unfixed infos", 0
    ; type string
    ; "magic bytes"
    ; HMAC/DRBG key
    "locked secret", 0
    ; type string
    ; "magic bytes"
    ; HMAC/DRBG key
    ; AES nonce and counter
    ; RETAIL KEY INFO
    "unfixed infos", 0
    ; type string
    ; "Magic bytes"
    ; HMAC / DRBG key
    "locked secret", 0
    ; type info
    ; "Magic bytes"
    ; HMAC / DRBG key
    ; AES nonce and counter
```

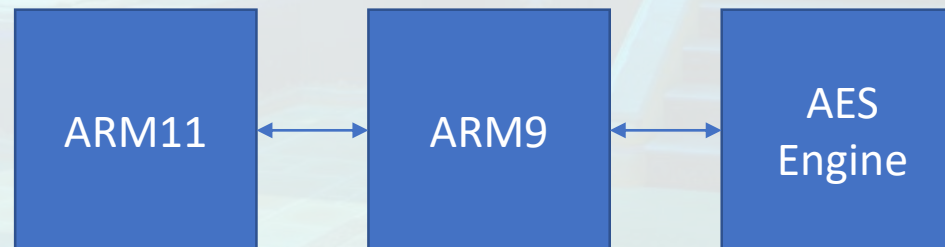
CRYPTO CAVERNS

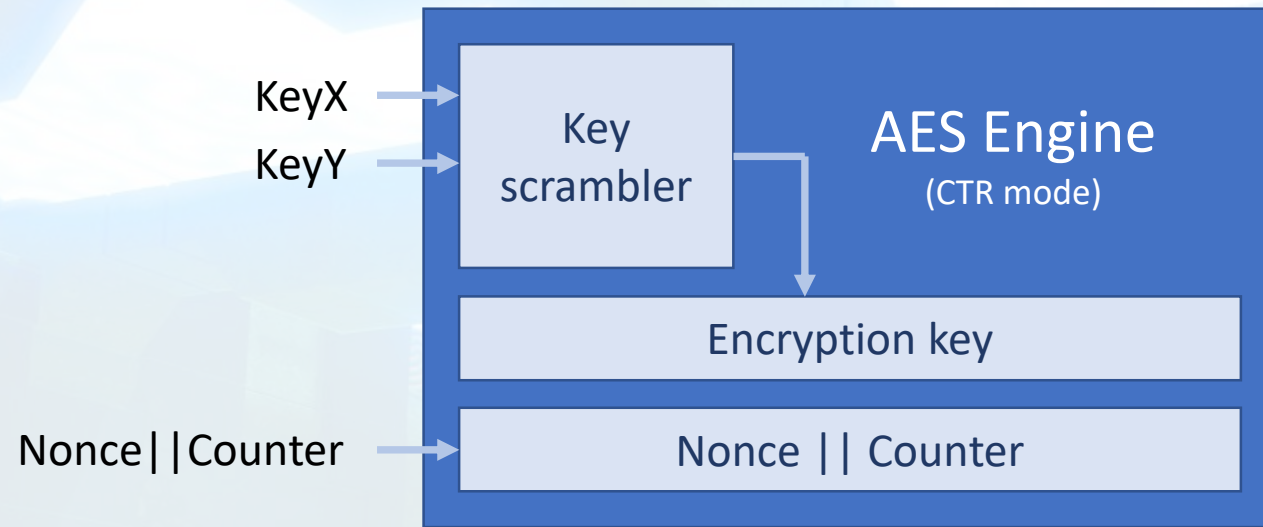
- Two key components are sent to AES engine
 - “keyX” set by bootloader
 - “keyY” set by firmware after it’s decrypted
- Setting keyY triggers generation of the “normal” key
- Normal key can be used to perform crypto operations in the engine, but can’t be read back out

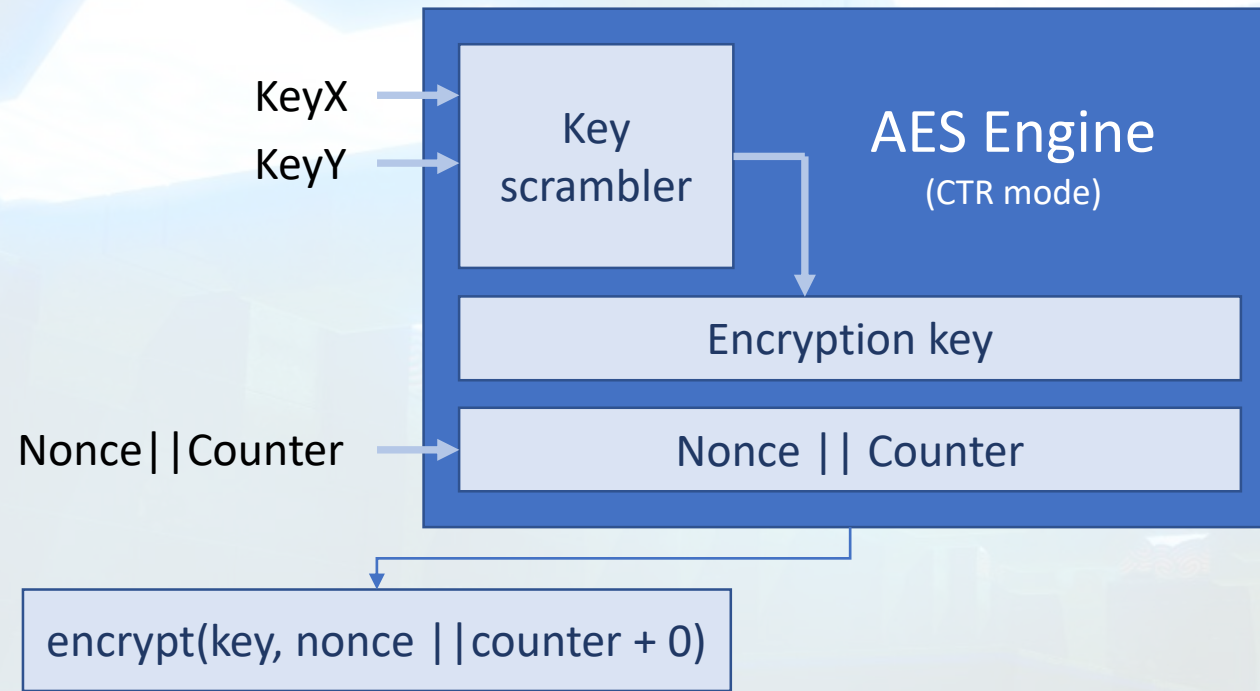


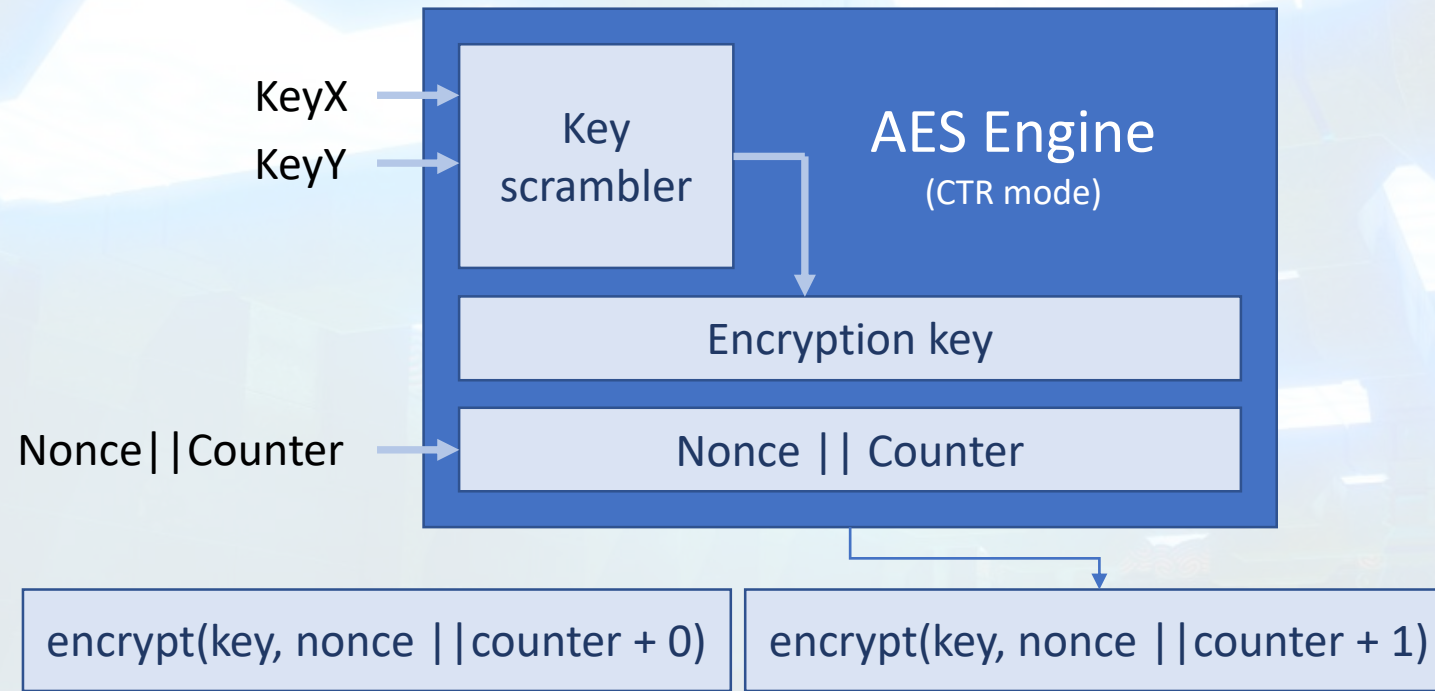
CRYPTO CAVERNS

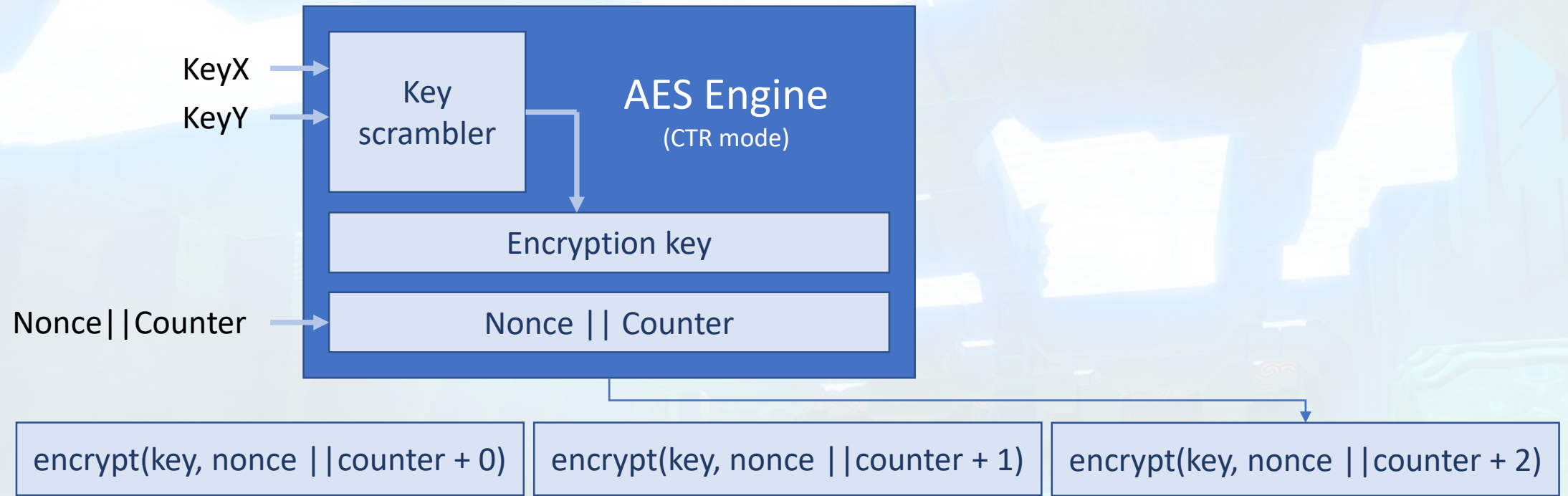
- “Key scrambling” algorithm performed by the AES engine was reversed by derrek, plutoo, smealum; presented in “Breaking the 3DS” at CCC in 2015
- All normal keys can be derived by applying this algorithm to the keyX and keyY components
- AES crypto could then be performed without the 3DS hardware, but before that...

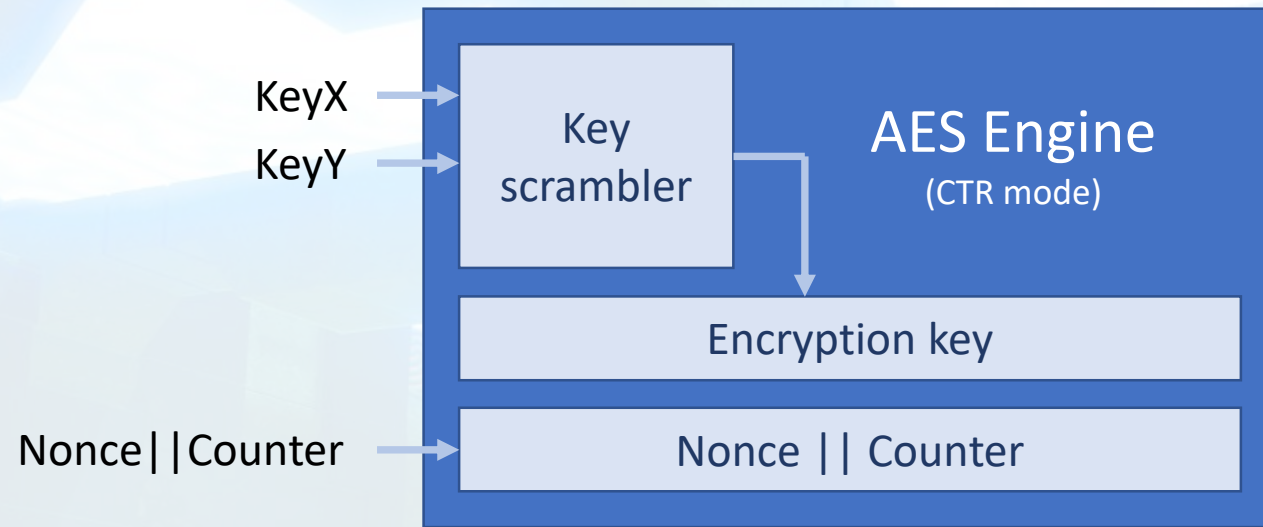










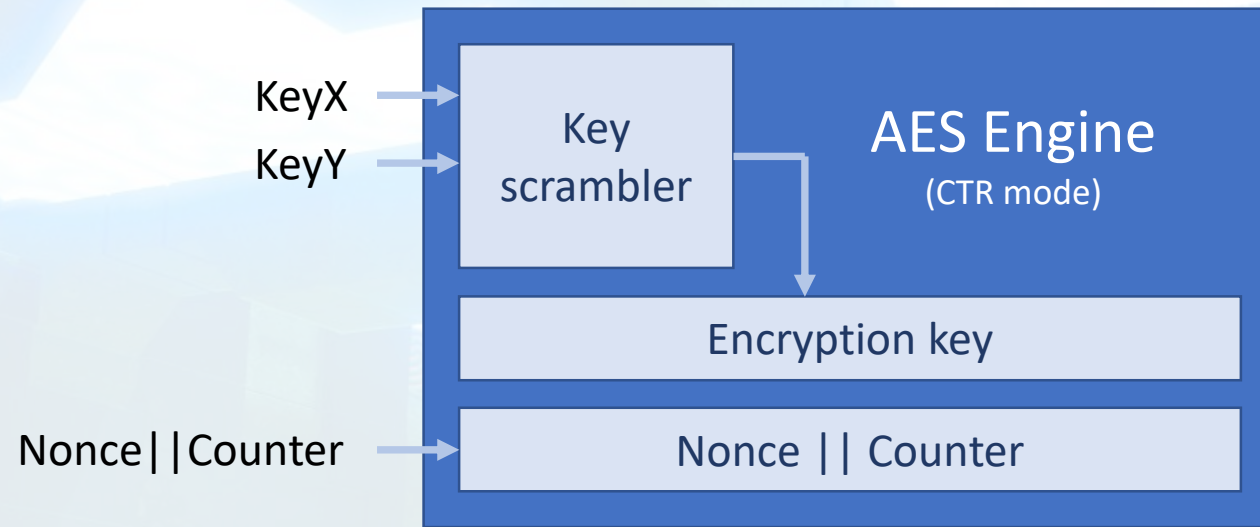


encrypt(key, nonce || counter + 0)

encrypt(key, nonce || counter + 1)

encrypt(key, nonce || counter + 2)

plaintext buffer



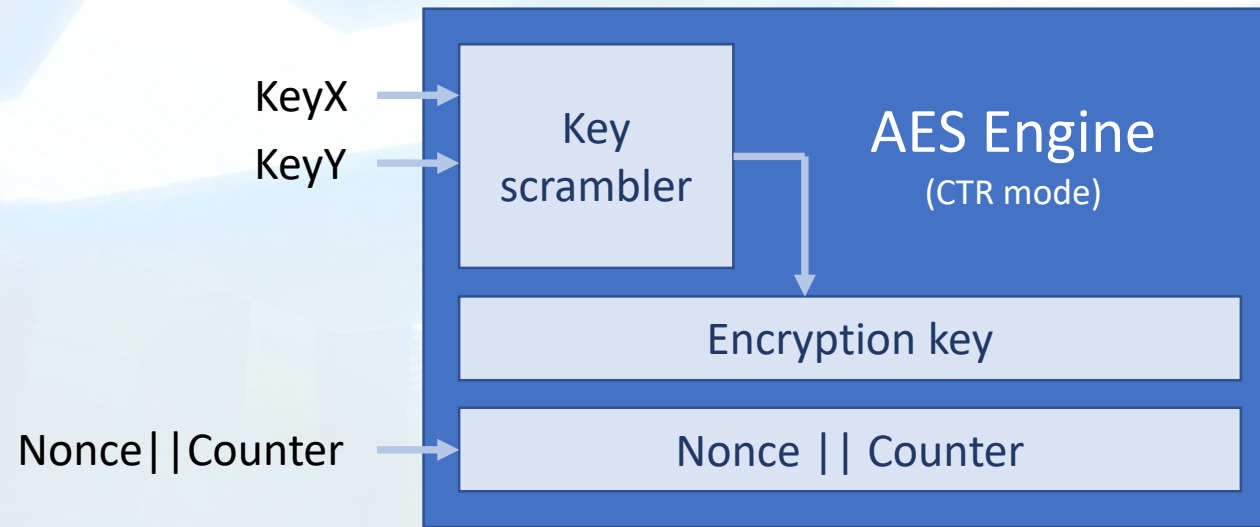
$\text{encrypt}(\text{key}, \text{nonce} || \text{counter} + 0)$

$\text{encrypt}(\text{key}, \text{nonce} || \text{counter} + 1)$

$\text{encrypt}(\text{key}, \text{nonce} || \text{counter} + 2)$



plaintext buffer



$\text{encrypt}(\text{key}, \text{nonce} || \text{counter} + 0)$

$\text{encrypt}(\text{key}, \text{nonce} || \text{counter} + 1)$

$\text{encrypt}(\text{key}, \text{nonce} || \text{counter} + 2)$

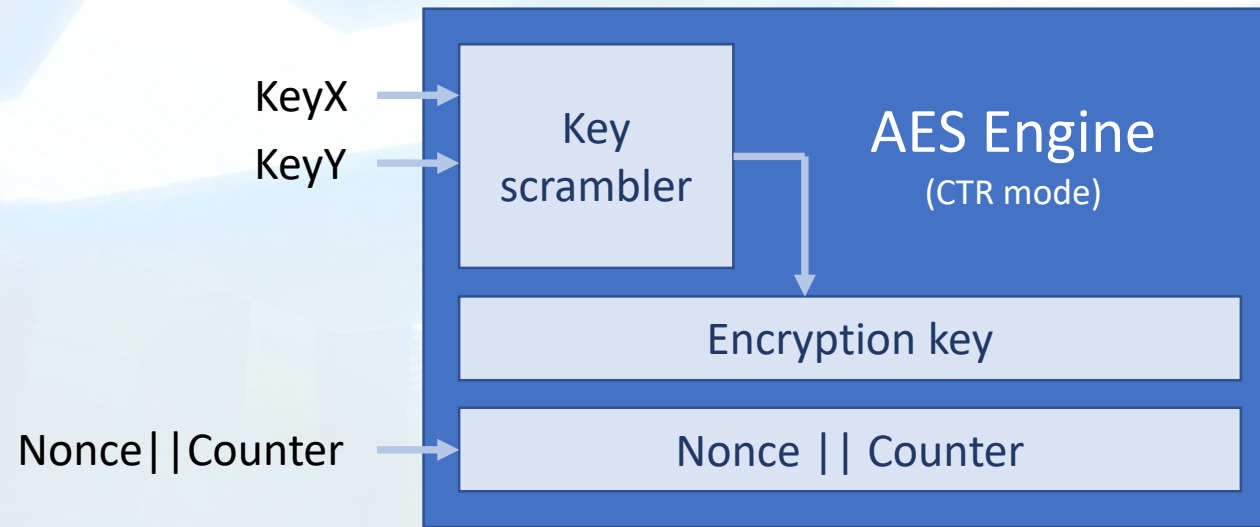


plaintext buffer

=

ciphertext


$$A \oplus 0 = A$$



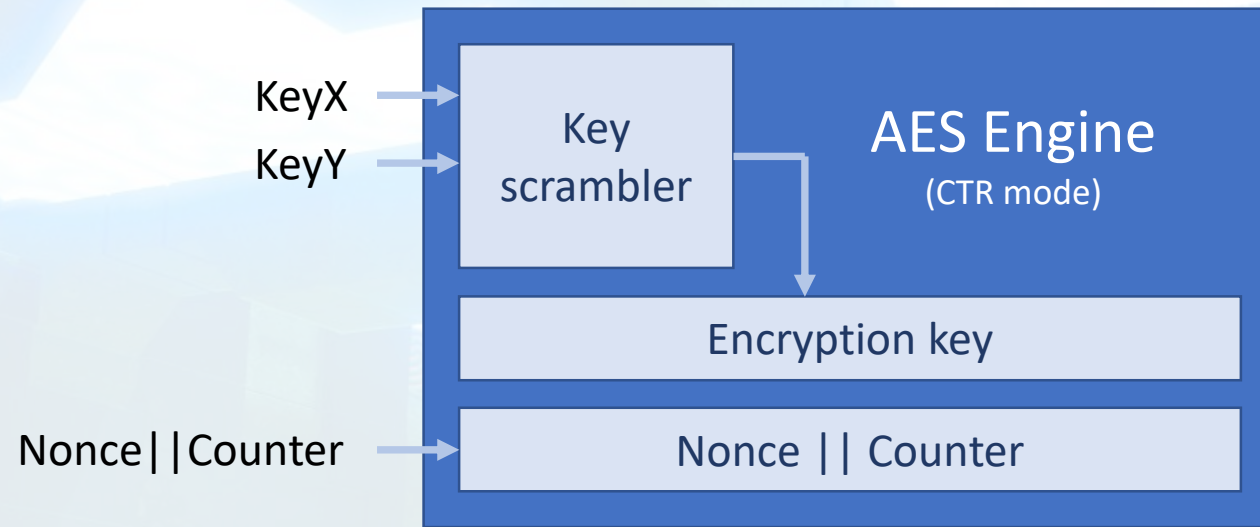
$\text{encrypt}(\text{key}, \text{nonce} || \text{counter} + 0)$

$\text{encrypt}(\text{key}, \text{nonce} || \text{counter} + 1)$

$\text{encrypt}(\text{key}, \text{nonce} || \text{counter} + 2)$



zeros



encrypt(key, nonce || counter + 0)

encrypt(key, nonce || counter + 1)

encrypt(key, nonce || counter + 2)



zeros

=

encrypt(key, nonce || counter + 0)

encrypt(key, nonce || counter + 1)

encrypt(key, nonce || counter + 2)

- A XOR pad can be used to perform CTR mode encryption and decryption without access to the AES engine hardware
- Only way to do crypto externally until hardware AES engine's key scrambling algorithm was cracked

“XOR pad”

encrypt(key, nonce || counter + 0)

encrypt(key, nonce || counter + 1)

encrypt(key, nonce || counter + 2)

CRYPTO CAVERNS

- Wrote a Python script for trying out a list of different keys from the hardware keys dump, with the ability to adjust AES-CTR internals
- Try encrypting zeros with different keys, nonce, and counters based on NFC service dump values until the known good XOR pad result is found
- With the following setup, it works:
 - The encryption key is the composite NFC service key from AES chip
 - The nonce and initial counter values are from the NFC binary
 - The counter is in big endian representation during AES-CTR
- Mysterious value is 32 byte XOR pad using NFC-specific key

CRYPTO CAVERNS

The Amiibo Crypto system

Amiibo data has two partitions:

1. Tag (“locked secret”)
 - NTAG215 serial number
 - Character ID
 - Unique 32 byte sequence
2. Data (“unfixed infos”)
 - Settings, bit flags
 - Registered owner Mii data
 - Game/application data

CRYPTO CAVERNS

The Amiibo Crypto system

1. Generate two sets of AES-CTR parameters and HMAC keys with a Deterministic Random Bit Generator
 - One set per partition
2. Sign each partition with its generated HMAC key
3. Encrypt data partition with AES-CTR using its generated AES key, nonce, and counter
 - Tag is stored plaintext; ignore that set of AES values
4. Rearrange the buffer for storage in NFC tag EEPROM

Reverse the process to decrypt (rearrange buffer, generate keys, decrypt data partition, check HMACs)

Amiibo

- Write counter
- Serial number
- Unique 32 bytes

3DS AES engine

Encrypt unique 32 bytes

HMAC Deterministic Random Bit Generator

type string || (write counter) magic bytes ||
serial || Encrypted unique 32 bytes

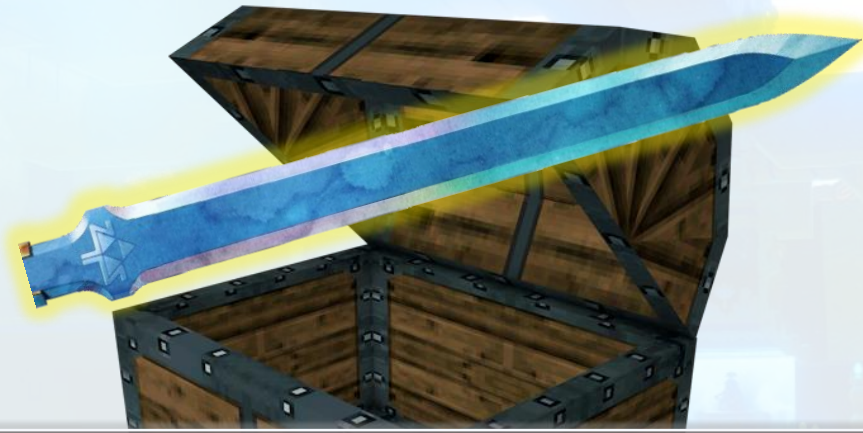
NFC Service

- AES parameters
- HMAC parameters
- Partition name
- Partition magic bytes

Amiibo unique signing and encryption keys

AES key || AES nonce || AES counter || HMAC key

CRYPTO CAVERNS



You got the **Amiibo crypto system!**
It's incredibly convoluted!

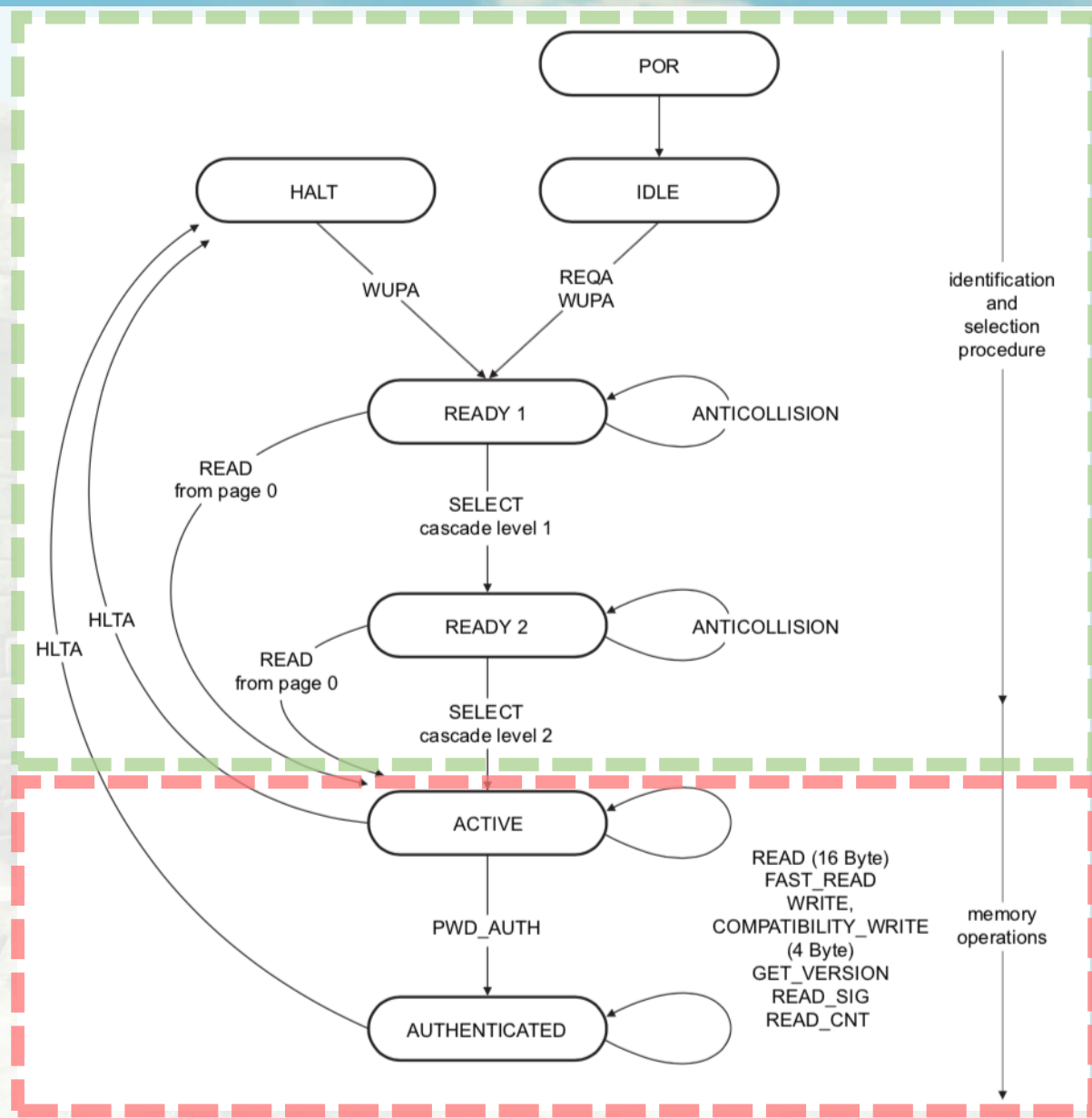
PROXMARK PLATEAU



PROXMARK PLATEAU

- Use the Proxmark sniffer and data sheets to figure out protocol
- NXP's NTAG21x data sheet is public
- Proxmark traffic display already recognizes many common ISO 14443 and NFC tag commands
 - Not always compatible with specific tag in use; may need changes

[illegible]

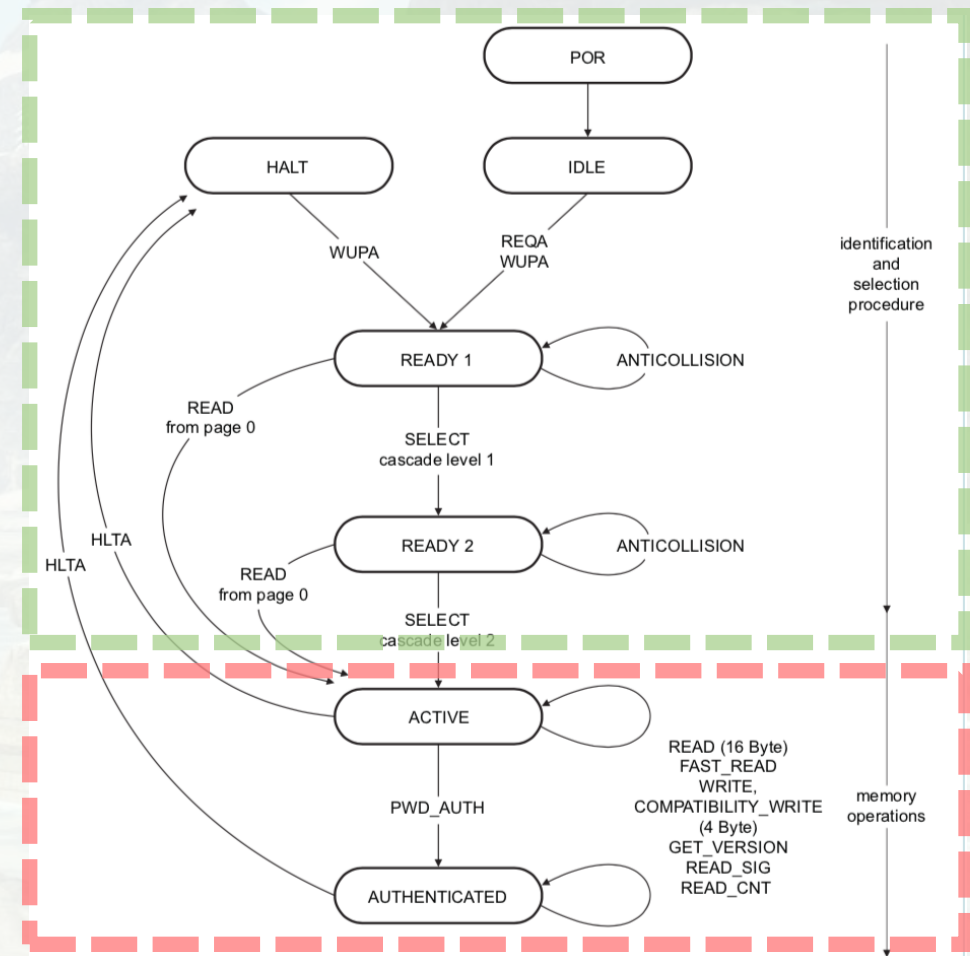


Implemented in ISO14443a simulator

Specific to NTAG21x

PROXMARK PLATEAU

- Commands to implement:
 - READ
 - FAST_READ
 - WRITE
 - GET_VERSION
 - PWD_AUTH
 - READ_SIG
- Buffer for tag EEPROM contents



!PROXMARK !PLATEAU

- State of the codebase: spaghetti
- A lot of the work is just figuring out how everything is set up
 - There's no developer documentation
- ~~Modular, self-contained NTAG21x logic~~
- Weld NTAG215 simulator on to ISO 14443a simulator

PROXMARK PLATEAU

GET_VERSION

Rdr	60	f8	32									ok	EV1 VERSION
Tag	00	04	04	02	01	00	11	03	01	9e		ok	

- “The GET_VERSION command has no arguments and replies the version information for the specific NTAG21x type.”
- NTAG215 info: 00 04 04 02 01 00 11 03 (+CRC-16)
- Add pre-cached response with 16 bit CRC value
 - Cyclic Redundancy Check: used for detecting transmission errors
 - Pre-caching necessary for low latency commands
 - Modulation must be calculated for data bytes before the timeout (usually 5ms)

PROXMARK PLATEAU

READ

Rdr	30	03	99	9a												ok	READBLOCK(3)		
Tag	f1	10	ff	ee	a5	00	03	00	4a	0b	34	18	05	4e	3a	06	ok		
	ed	1f																ok	

- Read four pages, beginning at specified page
 - A page is a four byte block of data. 16 bytes are returned.
 - An NTAG215 has 135 pages (540 bytes total)
- Only used to read the capability container in page 3, so we don't need to implement everything (locked pages, rolling over after end of memory, etc.)
- Biggest hurdle is implementing a memory buffer to store the EEPROM data
- Get buffer dynamically, calculate CRC-16, and send it back

PROXMARK PLATEAU

Tag EEPROM Buffer

- The Proxmark has a “big buffer” with custom malloc
- Undocumented card memory section that can be preserved while freeing the rest of the memory
- Can be populated from client using USB commands
- Need to set up interface in Lua to handle reading/writing to card memory buffer
- Update simulator to use card memory buffer when simulating NTAG215

PROXMARK PLATEAU

FAST_READ

Rdr	3a	00	3b	90	df												ok	READ RANGE (0-59)
Tag	04	3d	36	87	4a	b3	49	81	31	48	0f	e0	f1	10	ff	ee		
	a5	00	03	00	4a	0b	34	18	05	4e	3a	06	aa	1b	70	b6		

- Read from given start page to end page
- Reading all pages at once is allowed, but the 3DS only reads up to 60 pages at a time
 - Uses three separate FAST_READ commands to get all memory
- Should send a NAK if request is out of range, but the consoles never do this
 - Still implement some bounds checks to prevent crashes
 - Poor signal results in noisy, corrupted commands
- Get buffer, calculate CRC-16, send it back

PROXMARK PLATEAU

WRITE

- Write to a single page
- Update 4 bytes in the card memory buffer and send ACK
- Could potentially maintain state to see whether password unlock was used, but it's not necessary at the moment

PROXMARK PLATEAU

PWD_AUTH

Rdr	1b	dd	d0	a9	67	5c	60	ok	PWD-AUTH KEY: 0xdd0a967
Tag	80	80	64	16					

- 32 bit password for unlocking write capability
 - One of NTAG21x's built-in security features
- After 7 failed attempts, pages will be permanently locked
- Respond with 80 80 (+CRC-16) to accept password
- We can simply accept any password for the simulator
 - But it would still be interesting to know how passwords are calculated for more accurate simulation, or for writing to a real Amiibo

!PROXMARK !PLATEAU

PWD_AUTH

- With SDR you can supply an arbitrary serial number and get back the password when the console attempts to authenticate
- Serials 0000000000000000 and 0400000000000000 give back AA55AA55...

Serial / UID	Password
04 00 00 00 00 00 00	AA 55 AA 55
04 AA 00 00 00 00 00	00 55 AA 55
04 AA 55 00 00 00 00	00 00 AA 55
04 AA 55 AA 00 00 00	AA 00 00 55

- It's based on XORing serial number bytes
 - $A \oplus 0 = A$; $A \oplus A = 0$

!PROXMARK !PLATEAU

PWD_AUTH

Serial/UID	Password
04 FF 00 00 00 00 00	55 55 AA 55
04 00 FF 00 00 00 00	AA AA AA 55
04 00 00 FF 00 00 00	<u>55</u> 55 <u>55</u> 55
04 00 00 00 FF 00 00	AA <u>AA</u> AA <u>AA</u>
04 00 00 00 00 FF 00	AA 55 <u>55</u> 55
04 00 00 00 00 00 FF	AA 55 AA <u>AA</u>

!PROXMARK !PLATEAU

PWD_AUTH

Serial/UID	Password (XOR AA55AA55)
04 FF 00 00 00 00 00	FF 00 00 00
04 00 FF 00 00 00 00	00 FF 00 00
04 00 00 FF 00 00 00	<u>FF</u> 00 FF 00
04 00 00 00 FF 00 00	00 <u>FF</u> 00 FF
04 00 00 00 00 FF 00	00 00 <u>FF</u> 00
04 00 00 00 00 00 FF	00 00 00 <u>FF</u>

!PROXMARK !PLATEAU

PWD_AUTH

Serial/UID	Password (XOR AA55AA55)
04 00 00 FF 00 00 00	FF 00 FF 00
04 FF 00 FF 00 00 00	00 00 FF 00
04 FF 00 FF 00 FF 00	00 00 00 00

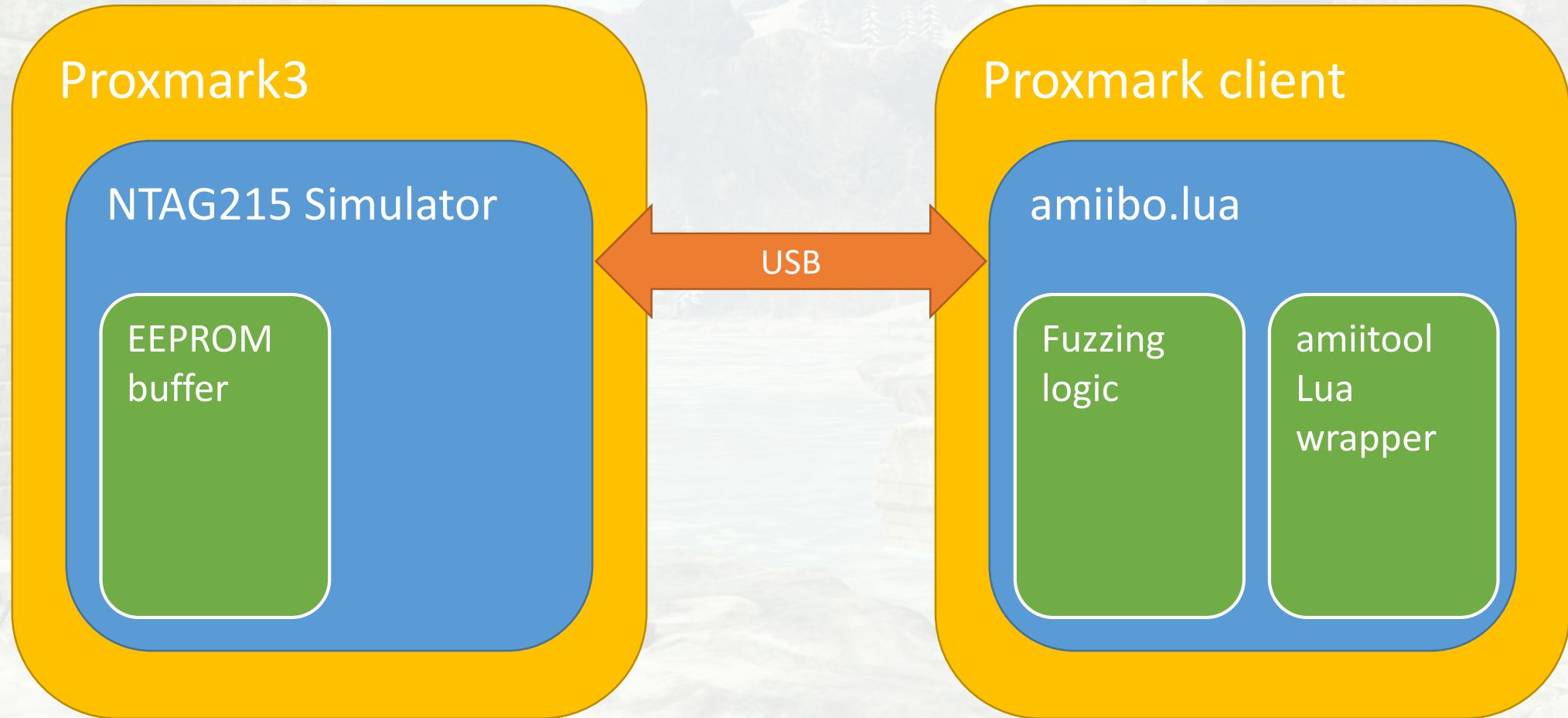
```
xormask = '\xaa\x55\xaa\x55'  
for i = 0 to 3:  
    pwd[i] = xor(uid[i+1], uid[i+3], xormask[i])
```


PROXMARK PLATEAU

Amiibo integration

- Proxmark client has a Lua script interface
 - Can write and update Lua programs without recompiling client
 - Easier to implement, has OOP-ish features
- Wrote Lua wrapper for amiitool and compiled as a library
- Amiibo object in Lua
 - Uses amiitool Lua module for packing and unpacking data
- Card memory interface for populating and reading card memory
- Features
 - Dump and unpack data from Amiibo
 - Load and simulate image
 - Save/restore state of card memory

PROXMARK PLATEAU



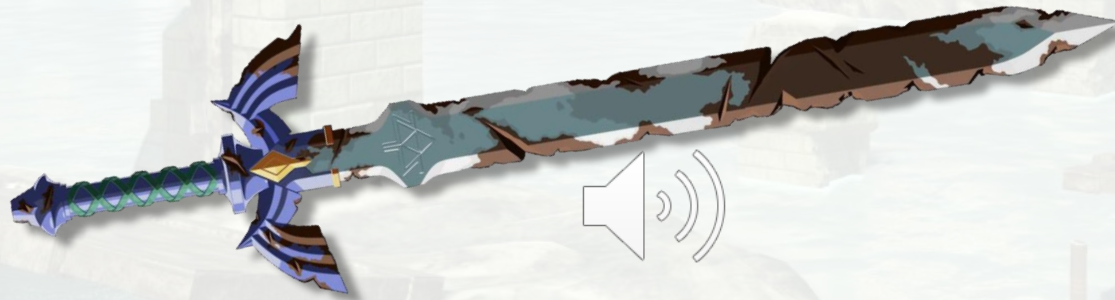
!PROXMARK !PLATEAU



You got the **NTAG215 simulator**!
Why does it smell like spaghetti?

PROXMARK PLATEAU

- Works fine on the 3DS, but when the Switch finally comes out in March:
 - ECDSA signature of the tag's serial number is actually checked with the READ_SIG command
 - The Wii U also uses this check, but I hadn't been testing on it
 - The ID card shaped Proxmark HF antenna works very poorly with the Switch



- It's not very effective.

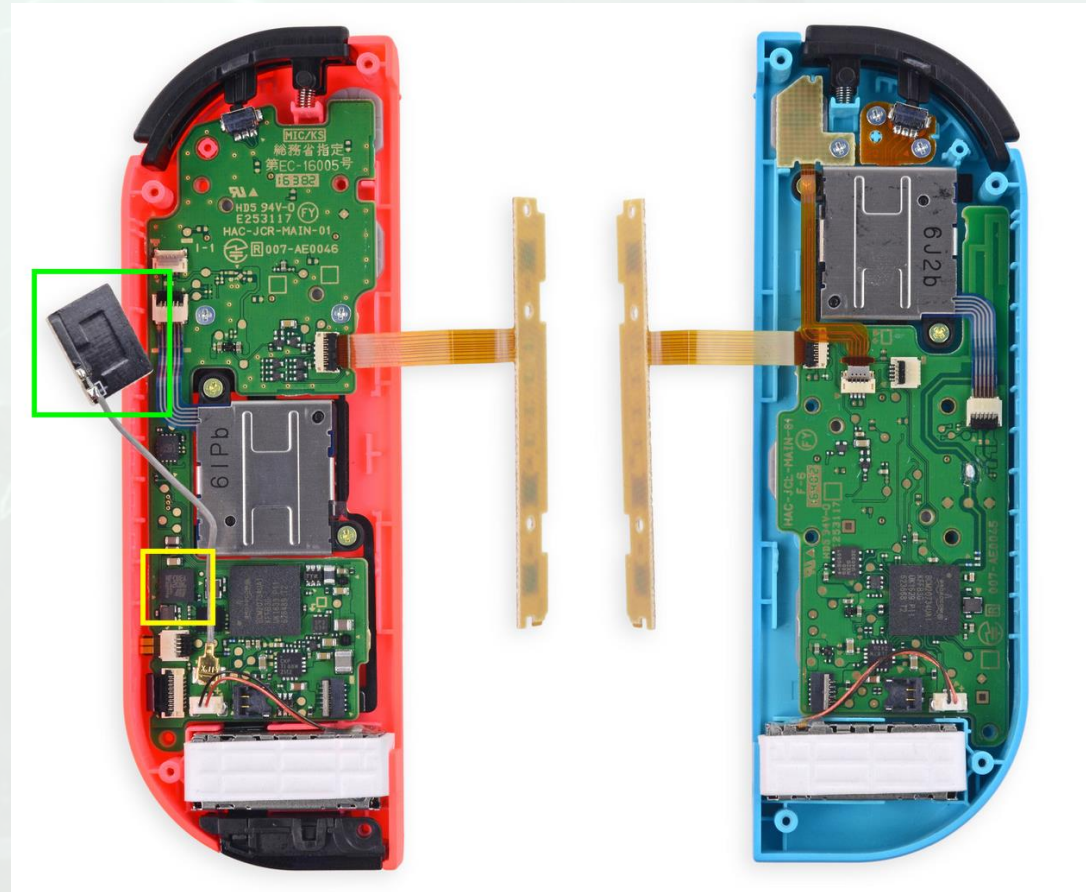


ANTENNA PROBLEMS

ANTENNA PROBLEMS



ANTENNA PROBLEMS

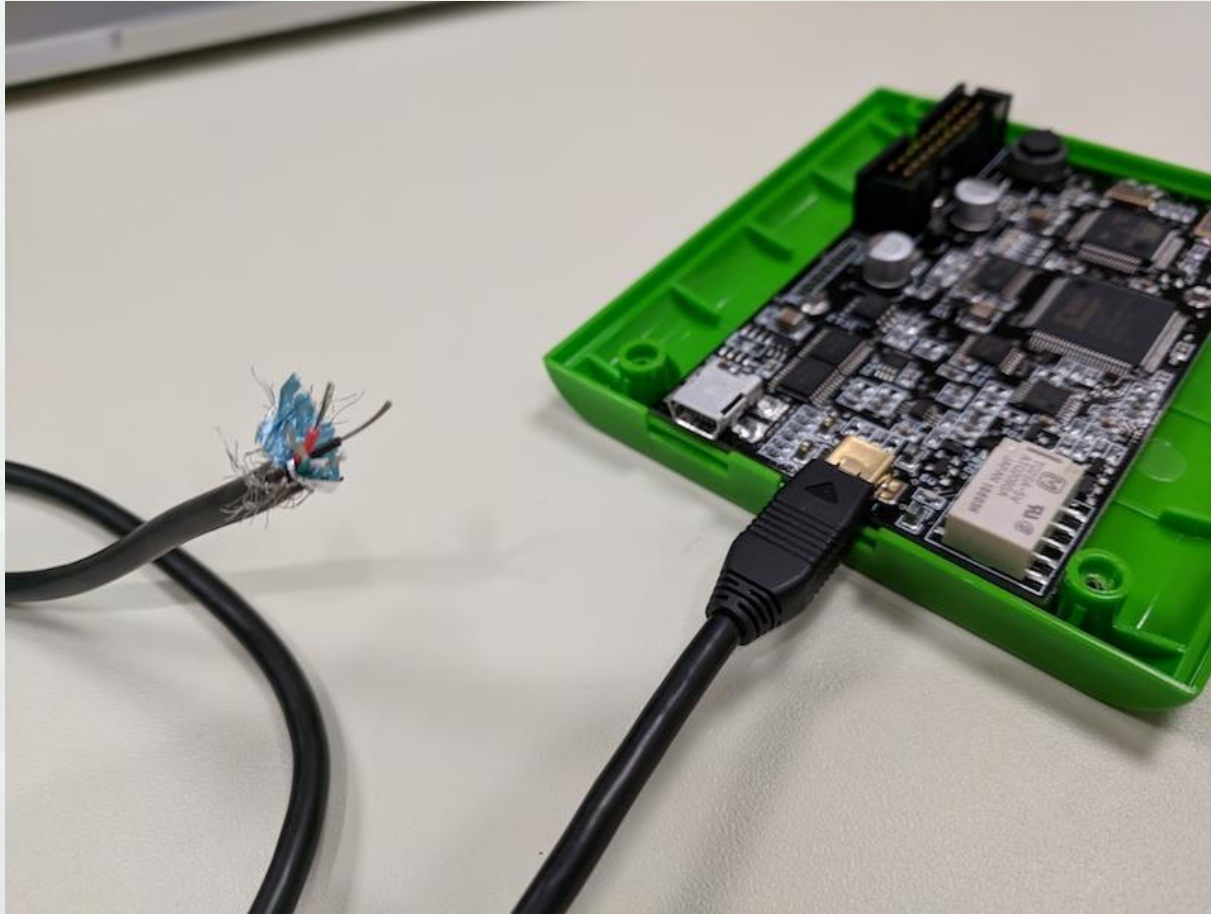


ANTENNA PROBLEMS

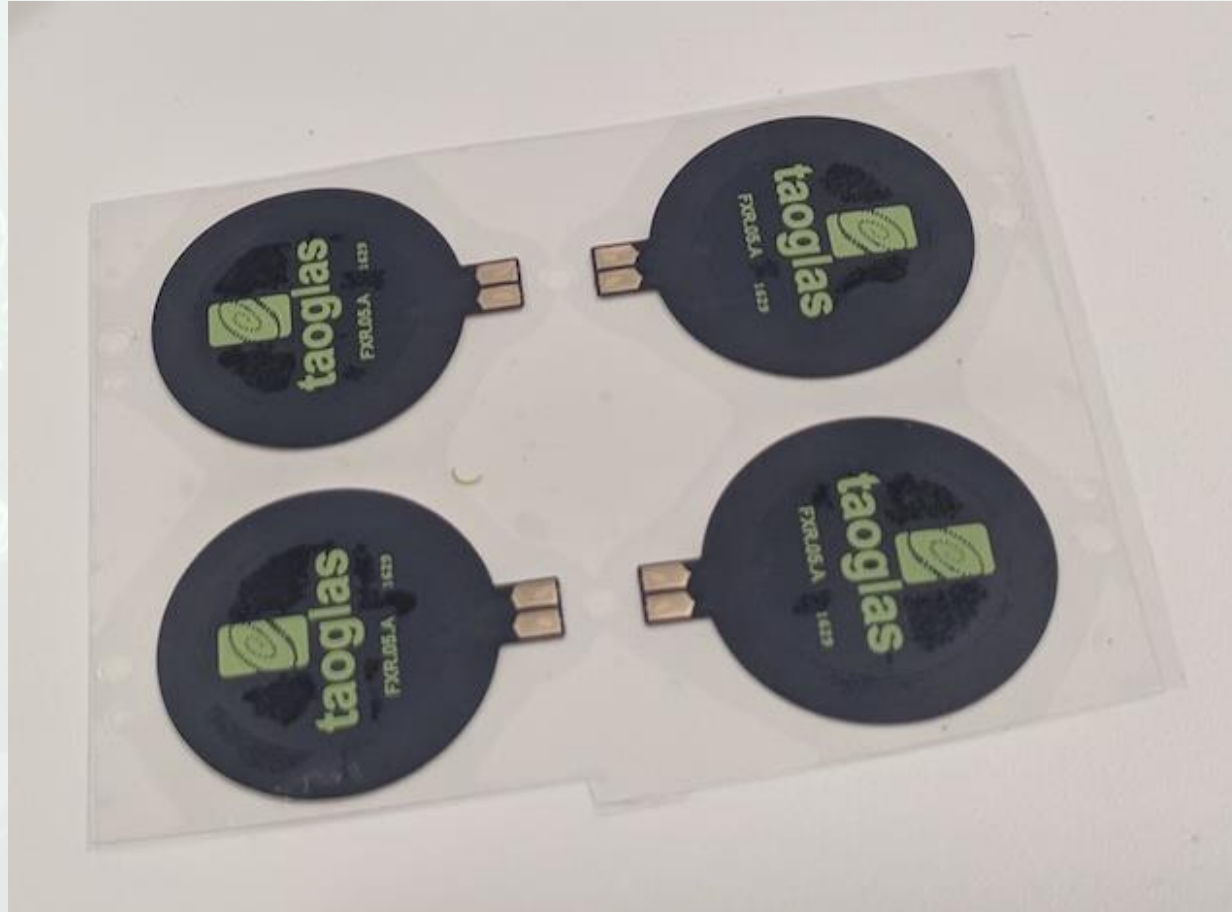
Official Proxmark3 instructions: rip open cable and wrap it around until it works



MAKING AN ANTENNA



MAKING AN ANTENNA



MAKING AN ANTENNA

- Fail: Signal is not strong enough to read tags or be picked up by console.
 - Got 0.1 V, need around 12V.
- Can't just buy any "13.56 MHz antenna" and attach it: tuning must factor in the rest of the device
 - Antenna forms inductor-capacitor circuit with capacitor on the Proxmark3 board
 - Capacitor has a capacitance of 47 picofarads (pF). Need to adjust this to match the inductance of a different antenna.



MAKING AN ANTENNA



2. Specifications

Flexible PCB Near-Field Communications Antenna		
Frequency	13.56	MHz
Inductance @ 13.56 MHz	15.9	μ H

Mechanical	
Antenna Dimensions	Diameter: 26.4 mm; Thickness: 0.24 mm
RoHS Compliant	Yes
Adhesive	3M 467
Weight	201.7mg

MAKING AN ANTENNA

- Current resonant frequency: $\frac{1}{2\pi \sqrt{15.9 \mu H \times 47 pF}} = 5.822 MHz$
- Desired resonant frequency: 13.56 MHz
- Solving for capacitance with desired frequency: need 8.67 pF.
- Need to reduce capacitance without changing Proxmark board.

MAKING AN ANTENNA

- Reduce capacitance by adding capacitors in series:

$$\frac{1}{\frac{1}{47pF} + \frac{1}{C_2} + \frac{1}{C_3} + \dots + \frac{1}{C_n}} = 8.67pF$$

- Ideal: add a 10.6 pF capacitor. Hard to order one that specific though.
- Alternative: get closest approximation with a series of whole-number capacitance levels available in small quantities online

MAKING AN ANTENNA

```
[ jchambers@LT8220 ~/Documents/amiibo ]  
[$ ./caps.py 8.67 --included 47 --allowed 9 10 11 12 15 16 18 20 22 24 27 30 33 36 39  
Closest set of 1 caps gives 8.913793pF  
[47.0, 11.0]  
Closest set of 2 caps gives 8.641471pF  
[47.0, 36.0, 15.0]  
Closest set of 3 caps gives 8.679351pF  
[47.0, 33.0, 33.0, 30.0]  
Closest set of 4 caps gives 8.074890pF  
[47.0, 39.0, 39.0, 39.0, 39.0]  
Closest set of 5 caps gives 6.689781pF  
[47.0, 39.0, 39.0, 39.0, 39.0, 39.0]  
Closest set of up to 5 caps gives 8.679351pF  
[47.0, 33.0, 33.0, 30.0]  
[ jchambers@LT8220 ~/Documents/amiibo ]  
$
```


MAKING AN ANTENNA



- 13.9 MHz resonant frequency with spare 10pF capacitor
- Boosts strength from 0.1 to 7V @ 13.56 MHz

MAKING AN ANTENNA

```
proxmark3> script run amiibo
--- Executing: amiibo.lua, args ''
Loaded retail keys from all_in_one_keys.bin
Tag type:      NXP MIFARE Plus 2k
Tag UID:       043D36874AB349813148
Tag len:       540
Figure ID:     19190000000900020d
Settings init: true
Nickname:      Peekandpwn
Appdata writes: 60
UID:           043d364ab34981
Write key:     ddd0a967

-----Finished
proxmark3> hw tune

Measuring antenna characteristics, please wait....#db# DownloadFPGA(len: 42096)
.....
# LF antenna:  0.00 V @ 125.00 kHz
# LF antenna:  0.00 V @ 134.00 kHz
# LF optimal:  0.00 V @ 12000.00 kHz
# HF antenna:  6.91 V @ 13.56 MHz
# Your LF antenna is unusable.
# Your HF antenna is marginal.
proxmark3> █
```


ORIGINALITY CHECK



ORIGINALITY CHECK

1.3 Security

- Manufacturer programmed 7-byte UID for each device
- Pre-programmed Capability container with one time programmable bits
- Field programmable read-only locking function
- ECC based originality signature
- 32-bit password protection to prevent unauthorized memory operations

ORIGINALITY CHECK

- ECDSA signature of tag's serial number
 - Elliptic Curve Digital Signature Algorithm
- Signature proves tag originates from NXP Semiconductors
- Prevents simple counterfeiting
 - Can't make signature for new serial without NXP private key
- Ideal benefit: A valid Amiibo NFC tag must be produced by NXP and initialized by Nintendo. Amiibo data is bound and unique to the tag it's created for.
 - No one else can produce or duplicate Amiibo.

ORIGINALITY CHECK

- Can't use an arbitrary serial number without the ECDSA signature created with NXP's private key
- Implications for simulating, fuzzing:
 - Some games limit usage by serial number and time (Breath of the Wild)
 - Need to amass a collection of serial/signature pairs...
 - or rewrite with the same serial number and redo encryption each time
 - Can't simulate with arbitrary serial number ☹
- ... or can you?



ORIGINALITY CHECK

How do cheat devices do it?

- PowerSaves offers “serial randomization” cheat for bypassing rate limit in Breath of the Wild
 - Can only scan an individual Amiibo once per in-game day
- Cheat devices *could* harvest serial/signature pairs from users' Amiibo...
 - *Haven't verified this, it's just a possibility*
 - Amiibo image uploaded to API for each operation

ORIGINALITY CHECK

Crypto implementation errors?

- Reusing the same “random number” in ECDSA will compromise the private key
 - See fail0verflow’s PlayStation 3 hacking talk
- Used example SDK code to parse out the nonce parameter from signature
- They don’t appear to reuse the nonce
 - in the small sample I checked

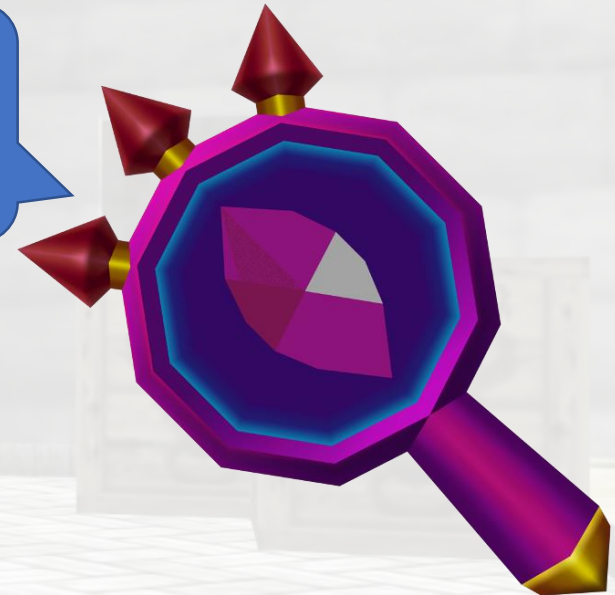
ORIGINALITY CHECK

Protocol or application logic flaws?

Verification process:

1. Select tag by ID (anti-collision)
2. Read NXP tag signature
3. Validate selected ID with signature
4. Read entire image from EEPROM
5. Generate keys for image
6. Perform HMAC validation of image

Can you
spot the
bug?



ORIGINALITY CHECK

Protocol or application logic flaws?

Verification process:

1. Select tag by ID (anti-collision)
2. Read NXP tag signature
3. Validate selected ID with signature
4. Read entire image from EEPROM
5. Generate keys for image
6. Perform HMAC validation of image



ORIGINALITY CHECK

- Supply any known ID and signature pair during selection
- Any Amiibo data can be returned through the read commands afterwards, regardless of the contained tag ID
 - Patched on Switch in 5.0.0; no fix for Wii U
- *No crypto keys necessary*
 - Don't need to do any crypto rewriting to load an image
 - Arbitrary serial number in tag data



System Settings

Data Management

Users

Mii

amiibo

Themes

Notifications

Sleep Mode



You can edit and delete data saved to amiibo.

Register Owner and Nickname

Delete Game Data

Reset amiibo



B Back

A OK



jchambers — jamchamb@ubuntu: ~/Documents — ssh jamchamb@192.168.251.138 — 116x9

— jamchamb@ubuntu: ~/Documents — ssh jamchamb@192.168.251.138

```
#db# 02
#db# Received unknown command (len=1):
#db# 00
#db# Received unknown command (len=2):
#db# 7f 00
#db# Trace full; clearing.
#db# Button press
#db# 0 0 b13
proxmark3> 
```


FUTURE WORK

- Finish antenna for Switch
- Examine new layer of encryption added to application data in games like Splatoon 2
- Create test harness for fuzzing on the Switch, 3DS



TOOL RELEASE

Get amiimikyu at

<https://github.com/nccgroup/proxmark3-amiimicyou>

amiitool Lua wrapper at

<https://github.com/jamchamb/amiitool>

THANKS TO...

- Jeff Dileo
- Nolan Ray