Closed Source Cryptography

Lukas Zobernig

25 March 2025

うせん 前 ふばやふばやふむやる

Intro

- Open Source Cryptography Workshop?
- Let's talk about closed source cryptography then.
- ► Where? Virtually all proprietary systems.
- ▶ Why? Corporate red tape, paranoia, security through obscurity, DRM, IP, etc.
- ▶ What? Lots and lots of "homegrown" cryptography. Amazing.

Reverse Engineering

How does one look at closed source cryptography?

- Primarily through reverse engineering a spectrum of difficulty.
- JavaScript easy.
- Compiled Java/.NET easy.
- ► Compiled C/C++/Rust/etc. trickier (decompilers can help).
- Synthesised IP (hardware) difficult (need expensive lab setup).

Many mistakes reoccur (for and between vendors). Often we can guess what could go wrong once we built a good library of mistakes. We will see examples of that.

Sony/IBM/Toshiba - Playstation 3

First released in 2006 (JP, NA) and 2007 (EU).

► GPU by Nvidia, CPU co-developed by Sony, Toshiba, and IBM.

▲□▶ ▲□▶ ▲□▶ ▲□▶ □ の00

- Cell Broadband Engine Architecture (CELL/B.E.) CPU.
 - PPC main cores.
 - SPU accelerator/compute cores.
 - Secure Boot (boots from **isolated** SPU).
- Lots and lots of custom Sony DRM.
- Security through obscurity.
- "Custom" cryptography.

- ▶ IBM released public versions of the CELL/B.E. manuals.
- Sony provided a mechanism to run Linux on the PS3, we can look at drivers.
- The CPU was also used in super-computing applications, SPU instruction set publicly documented.

▲□▶ ▲□▶ ▲ □▶ ▲ □▶ ▲ □ ● ● ● ●

Can develop tooling like disassemblers and emulators.



- Tricky since there was no decompiler for the SPU architecture.
- This was also long before Ghidra was published, so one would have had to develop their own.
- Thus, reverse engineering by hand.

Reverse Engineering Cryptography Algorithms

There are a few tricks.

- Most often standard algorithms implemented.
- ▶ AES, SHAx, HMAC, ECC, RSA, etc. all use characteristic constants.
- Ciphers and hashes have different algorithmic structure than signatures or key exchange.

▶ Often high-entropy data (keys, IVs,) referenced through inputs.

	seg003:00034ABF		.byte	0
	seg003:00034AC0	xmmword_34AC0:	.int128	0x637C777BF26B6FC53001672BFED7AB76
	seg003:00034AC0			; DATA XREF: sub_1D3C0+4+r
•	seg003:00034AC0			; sub_1D7C0+14+r
0	seg003:00034AD0	xmmword_34AD0:	.int128	0xCA82C97DFA5947F0ADD4A2AF9CA472C0
-	seg003:00034AD0			; DATA XREF: sub_1D3C0+Ctr
r2, r5, 0xC0	seg003:00034AD0			; sub_1D7C0+18+r
r39, xmmword_34AC0	seg003:00034AE0	xmmword_34AE0:	.int128)xB7FD9326363FF7CC34A5E5F171D83115
r20, r3	seg003:00034AE0			; DATA XREF: sub_1D3C0+10+r
r40, xmmword_34AD0	seg003:00034AE0			; sub_1D7C0+1Ctr
r36, XMINOTO_34AE0	seg003:00034AF0	xmmword_34AF0:	.int128)x4C723C31896059A071280E2EB27B275
r36 xmmord 34800	seg003:00034AF0			; DATA XREF: sub_1D3C0+14tr
r37, xmword 34810	seg003:00034AF0			; sub_107C0+20tr
r33, xmmword_34B20	seg003:00034B00	xmmword_34B00:	.int128	0x9832C1A1B6E5AA0523BD6B329E32F84
r34, xmmword_34B30	seg003:00034B00			; DATA XREF: sub_1D3C0+18tr
r30, xmmword_34B40	seg003:00034B00			; sub_1D7C0+24tr
r31, xmmword_34B50	seg003:00034B10	xmmword_34B10:	.int128	x53D100ED20FCB15B6ACBBE394A4C58CF
r32, xmmword_34860	seg003:00034B10			; DATA XREF: sub_1D3C0+1Ctr
r28, xmmword_34B70	seg003:00034B10			; sub_1D7C0+2Ctr
r29, xmmord_34880	seg003:00034B20	xmmword_34B20:	.int128	xD0EFAAFB434D338545F9027F503C9FA8
r26, xmmord 34840	seg003:00034B20	_		; DATA XREF: sub_1D3C0+20tr
	seg003:00034B20			; sub_1D7C0+30+r
r27, xmmword_34BB0	seg003:00034B30	xmmword_34B30:	.int128)x51A3408F929D38F5BCB6DA2110FFF3D2
sp, var_120(sp)	seg003:00034B30			; DATA XREF: sub_1D3C0+24tr
sp, sp, -0x120	seg003:00034B30			; sub_1D7C0+34tr
r2, loc_1D638	seg003:00034B40	xmmword_34B40:	.int128)xCD0C13EC5F974417C4A77E3D645D1973
	seg003:00034B40			; DATA XREF: sub_1D3C0+28tr
1	seg003:00034B40			; sub_1D7C0+38tr
F	seg003:00034B50	xmmword_34B50:	.int128)x60814FDC222A908846EEB814DE5E0BDB
	seg003:00034B50			; DATA XREF: sub_1D3C0+2Ctr
	seg003:00034B50			; sub_107C0+3Ctr
	seg003:00034B60	xmmword_34B60:	.int128)xE0323A0A4906245CC2D3AC629195E479
1	seg003:00034B60			; DATA XREF: sub_1D3C0+30tr
1	seg003:00034B60			; sub_107C0+40tr
1	seg003:00034B70	xmmword_34B70:	.int128)xE7C8376D8DD54EA96C56F4EA657AAE08
	seg003:00034B70			; DATA XREF: sub_1D3C0+34tr
	seg003:00034B70			; sub_1D7C0+48tr
	seg003:00034B80	xmmword_34B80:	.int128	xBA78252E1CA6B4C6E8DD741F4BBD8B8A
1	seg003:00034B80			; DATA XREF: sub_1D3C0+38tr
	seg003:00034B80			; sub_1D7C0+4Ctr
1	seg003:00034B90	xmmword_34B90:	.int128	x703EB5664803F60E613557B986C11D9E
	seg003:00034B90			; DATA XREF: sub_1D3C0+3Ctr
	seg003:00034B90			; sub_1D7C0+50+r
	000002.00024040	ummined 24040.	4.4170	L_E1E0A011CARA0EAAAB1E07EAFEEE30RE



sub 103C0: var 120= -0x120 var 100= -0x100

ceqi lga

٦ŕ lga

lga

lga

lga

lqa

lqa

lqa

lqa

lqa lga

lqa

lga

lqa

lga

304

						_		
								
	🏐 🕰 🗷			🏐 🕰 🔀			🍘 🕰 🔀	
			11					
	loc_1D588:			loc_1D488:		- 1	loc_1D760:	
	and	r60, r2, r22		rotmi	r3, r19, -0x1F	- 1	ai	r32, sp, 0x120+var_100
	shufb	r63, r2, r19, r41		shufb	r2, r17, r4, r43	- 1	ai	r31, r9, 0x20
	ai	r21, r21, -1		and	r77, r5, r41	- 1	а	r26, r9, r32
	shufb	r61, r2, r19, r42		shufb	r6, r17, r4, r42	- 1	lnop	
	а	r17, r18, r20		shufb	r14, r5, r17, r2	4	а	r22, r8, r20
	shufb	r62, r2, r19, r43		shufb	r4, r5, r17, r44		lqx	r30, r31, sp
	cgti	r16, r21, -1		а	r11, r19, r3		ai	r10, r10, -1
	shufb	r59, r2, r19, r23		shufb	r76, r5, r17, r4	5	lqd	r28, 0x10(r26)
	andbi	r54, r63, 0x1F		selb	r13, r6, r2, r22		ai	r9, r9, 0x40
	lqx	r44, r18, r24		lr	r5, r21		lqd	r29, 0x20(r26)
	andbi	r65, r63, 0x20	1	rotmai	r12, r11, -1	-1	cgti	r21, r10, -1
	lnop			andbi	r75, r13, 0x1F		stqx	r30, r8, r20
	andbi	r64, r63, 0x40		andbi	r7, r13, 0x20		ai	r8, r8, 0x30
	shufb	r15, r26, r27, r54	1	lnop			shufb	r27, r28, r29, r13
	ceqbi	r49, r65, 0x20		andbi	r16, r13, 0x40		stqd	r27, 0x10(r22)
	shufb	r57, r39, r40, r54		shufb	r67, r26, r27, r	75	lqd	r25, 0x30(r26)
	ceqbi	r45, r64, 0x40		shli	r72, r12, 4		lqd	r24, 0x20(r26)
	shufb	r58, r38, r35, r54		shufb	r78, r39, r40, r	75	shufb	r23, r24, r25, r12
	clgtbi	r10, r63, 0x7F		ceqbi	r66, r7, 0x20		stqd	r23, 0x20(r22)
	shufb	r52, r36, r37, r54	11	shufb	r79, r38, r35, r	75		
	xor	r55, r61, r62		ceqbi	r61, r16, 0x40		loc_1D7AC:	
	shufb	r53, r33, r34, r54		shufb	r73, r36, r37, r	75	brnz	r21, loc_1D760
	xor	r56, r59, r60		clgtbi	r57, r13, 0x7F		-	
	shufb	r50, r30, r31, r54	11	shufb	r74, r33, r34, r	75		
	selb	r46, r57, r58, r49		xor	r70, r14, r4			
	shufb	r51, r32, r28, r54	411	shufb	r68, r30, r31, r	75		
	xor	r13, r55, r56		selb	r63, r78, r79, r	66		
1	shufb	r48, r29, r25, r54	4	shufb	r69, r32, r28, r	75		
	selb	r47, r52, r53, r49		xor	r71, r76, r77			
	ai	r18, r18, 0x10		shufb	r65, r29, r25, r	75		
	selb	r14, r50, r51, r49		selb	r64, r73, r74, r	66		
l	selb	r11, r48, r15, r49		lqx	r59, r72, r23			
1	selb	r9, r46, r47, r45		xor	r47, r70, r71	-1		
	selb	r4, r14, r11, r45		selb	r60, r68, r69, r	66		
	selb	r8, r9, r4, r10		selb	r62, r65, r67, r	66		
	xor	r12, r44, r8		selb	r56, r63, r64, r	61		
1	xor	r5, r13, r12		selb	r58, r60, r62, r	61		
	lnop			andc	r48, r59, r22			
	tr	r2, r5		selb	r49, r56, r58, r	57		
1	stqd	r5, 0x10(r17)		ai	r19, r19, 1	-1		
	Inop	r127		lixor	r46. r48. r49			

◆□▶◆圖▶★≧▶★≧▶ ≧ の�?

arg A= A	
arg_o= o	
il	r12, 0
lqd	r11, 0x50(r3)
ilhu	r15, 0x6745
cdd	r13, arg_0(sp)
ilhu	r14, 0xEFCD
hbr	loc_1FEAC, lr
iohl	r15, 0x2301
lr	r5, r3
iohl	r14, 0xAB89
cwd	r6, arg_0(sp)
il	r4, 0
stqd	r15, 0(r3)
ilhu	r10, 0x1032
shufb	r8, r12, r11, r13
ilhu	r9, 0xC3D2
stqd	r14, 0x10(r3)
ilhu	r3, 0x98BA
iohl	r10 , 0x5476
iohl	r3, 0xDCFE
stqd	r8, 0x50(r5)
iohl	r9, 0xE1F0
lqd	r7, 0xA0(r5)
nop	r127
stqd	r3, 0x20(r5)
il	r3, 0
stqd	r10, 0x30(r5)
stqd	r9, 0x40(r5)
shufb	r2, r4, r7, r6
stqd	r2, 0xA0(r5)
loc 1FEAC:	
bi	lr

var_A0= -6	9AX0
var_80= -0)x80
var_40= -0)x40
var_20= -0	0x20
var_10= -0	0x10
arg_10= ()x10
hbra	loc 1D4A4,
stqd	r80, var 10(sp)
lr i	r80, r4
stqd	r81, var_20(sp)
lr .	r81, r3
stqd	lr, arg_10(sp)
stqd	sp, var_A0(sp)
ai	sp, sp, -0xA0
ai	r3, sp, 0xA0+var_40
loc 104A4:	
brsl	lr.
il.	r2, -1
brnz	r3, loc 10500
	· · · · · · · · · · · · · · · · · · ·
😻 🕰 🗷	
11	r5, 0
nbra	LOC_ID4E4, LOC_ID40
43	ro, 3
il Lnop	
il lnop	
il lnop	
il lnop	
il lnop	d d d c 0:
il lnop	d 4C0: r8, r5, r80
il lnop Doc_1D a ai	4C0: r8, r5, r80 r6, r6, -1
il lnop loc_lD a ai ai	4C0: r8, r5, r80 r6, r6, -1 r2, r5, 0x20
il lnop loc_lD a ai ai lqd	4C0: r8, r5, r80 r6, r6, -1 r2, r5, 0x20 r7, 0x80(r8)
il lnop loc_lD a ai ai lqd cgti	d 440: r6, r5, r50 r6, r6, -1 r2, r5, 0x20 r7, 0x80(r8) r4, r6, -1
il lnop loc_lD a ai ai lqd cgti ai	<pre>40: r8, r5, r80 r6, r6, -1 r2, r5, 0x80(r8) r4, r6, -1 r5, r5, 0x10</pre>
il lnop loc_lD a ai ai lqd cgti ai xorbi	4C0: r8, r5, r80 r6, r6, -1 r2, r5, 0x20 r4, r6, -1 r4, r6, -1 r3, r5, 0x10 r3, r5, 0x10
il lnop loc_lD a ai ai lqd cgti ai xorbi stqx	4C0: r6, r5, r50 r6, r6, -1 r2, r5, 0x20 r7, 0x80(r3) r4, r6, -1 r5, r5, 0x10 r3, r7, 0x55 r3, r2, sp
il lnop loc_lD a ai lqd cgti ai xorbi stqx nop	d +460: r6, r5, r50 r7, 0x80(r6) r4, r6, -1 r5, r5, 0x20 r3, r5, 0x30 r3, r2, sp r127
il lnop loc_lD a ai ai lqd cgti ai stqx nop loc_lD	4(0): r6, r5, r5, r80 r6, r6, -1 r2, r5, 0:20 r7, 0:80(r8) r4, r6, -1 r5, r5, 0:20 r3, r7, 0:80(r8) r3, r7, 0:82 r3, r2, sp r127 454;
il loop loc_1D a ai lqd cgti ai xorbi stqx nop loc_1D brnz	4(0): r8, r5, r10 r6, r6, -1 r2, r5, 020 r7, 0.00(r6) r4, r6, -1 r5, r5, 0210 r7, 0.00(r6) r3, r7, 0.55 r3, r2, sp r127 r454: r4, loc 10400

▲□ ▶ ▲ 臣 ▶ ▲ 臣 ▶ ○ 臣 ○ の Q @

<pre>sha1_init:</pre>	
arg_0= 0	
il	r12, 0
lqd	r11, 0x50(r3)
ilhu	r15, 0x6745
cdd	r13, arg_0(sp)
ilhu	r14, 0xEFCD
hbr	loc_1FEAC, lr
iohl	r15, 0x2301
lr	r5, r3
iohl	r14, 0xAB89
cwd	r6, arg_0(sp)
il	r4, 0
stqd	r15, 0(r3)
ilhu	r10, 0x1032
shufb	r8, r12, r11, r
ilhu	r9, 0xC3D2
stqd	r14, 0x10(r3)
ilhu	r3, 0x98BA
iohl	r10, 0x5476
iohl	r3, 0xDCFE
stqd	r8, 0x50(r5)
iohl	r9, 0xE1F0
lqd	r7, 0xA0(r5)
nop	r127
stqd	r3, 0x20(r5)
il	r3, 0
stqd	r10, 0x30(r5)
stqd	r9, 0x40(r5)
shufb	r2, r4, r7, r6
stqd	r2, 0xA0(r5)
loc 1FEAC:	
bi	lr

	it:
var A0= -0xA0	
var_80= -0x80	
var_40= -0x40	
var_20= -0x20	
var_10= -0x10	
arg_10= 0x10	
hbra	loc_1D4A4, shal_fina
stqd	r80, var_10(sp)
lr	r80, r4
stqd	r81, var_20(sp)
lr	r81, r3
stqd	lr, arg_10(sp)
stqd	sp, var_A0(sp)
a1	sp, sp, -0xA0
ai	r3, sp, 0xA0+var_40
loc_1D4A4:	
brsl	<pre>lr, sha1_final</pre>
ii ii	r2, -1
brnz	r3, loc_10500
🕘 🕰 🗷	
11	r5, 0
hbra	Loc_1D4E4, Loc_1D4C0
	FB 3
11	10, 5
11 lnop	
11 lnop	
11 Inop	
11 lnop @ 🕰 🗷 loc_1D4C0: a	r8, r5, r80
11 Inop	r8, r5, r80 r6, r6, -1
11 Inop () 22 22 Ioc_1D4C0: a ai ai ai	r8, r5, r80 r6, r6, -1 r2, r5, 9x20
1L Lnop C1D4C0: a ai lqd Lqd	r8, r5, r80 r6, r6, -1 r2, r5, 0x20 r7, 0x80(r8)
1L lnop loc_lD4C0: a ai lqd cgti	r8, r5, r80 r6, r6, -1 r2, r5, 0x20 r7, 0x80(r8) r4, r6, -1
il Inop Inoc_1D4C0: a ai ai lqd cqti ai vechi	r8, r5, r80 r6, r6, -1 r2, r5, 0x20 r7, 0x80(r8) r4, r6, -1 r5, r5, 0x10 r5, r5, 0x10
il Inop Ioc_lD4C0: a ai ai lqd cqti ai xorbi xorbi	r8, r5, r80 r6, r6, -1 r2, r5, 0x20 r7, 0x80(r8) r4, r6, -1 r5, r5, 0x10 r3, r7, 0x55 r3, r7, 0x55
it Inop	r8, r5, r80 r6, r6, -1 r2, r5, 020 r7, 0380(r8) r4, r6, -1 r5, r5, 0310 r3, r7, 0550 r3, r2, 90
il Inop Dec_ID4C0: a ai iloc_LD4C0: a ai cgti ai xorbi stqx nop	r8, r5, r80 r6, r5, r80 r7, 0:80(r6) r4, r6, -1 r5, r5, 0:20 r3, r2, 0:55 r3, r2, 0:55 r3, r2, 55 r3, r2, 10
11 lnop loc_104C0: a ai lod cgti ai xorbi stqx nop loc_104E4:	r8, r5, r80 r6, r6, -1 r2, r5, 0x20 r7, 0x30(r8) r4, r6, -1 r5, r5, 0x10 r3, r7, 0x50 r3, r2, sp r127
ll lnop loc_lD4C0: ai ai iql cgti ai xorbi stqx nop loc_lD4E4: brnz	r8, r5, r80 r6, r6, -1 r2, r5, 0.20 r7, 0.080(r8) r4, r6, -1 r5, r5, 0.10 r3, r7, 0.55 r3, r2, sp r127 r4, 1oc_1D4C0

▲母 ▶ ▲ 臣 ▶ ▲ 臣 ▶ ○ 臣 ○ の Q @

And of course sometimes you get lucky and developers forget to strip symbols.

CellCryptoSpu ecc. mpn. uc2mpn. 161 cellCryptoSpu ecc mpn mpn2ui 192 7 cellCryptoSpu ecc set pabr 7 cellCryptoSpu ecc set basepoint 7 cellCryptoSpuSha1Hash 7 cellCryptoSpuSha1Hmac CellCryptoSpuSha1HmacFinal 7 cellCryptoSpuSha1HmacInit CellCryptoSpuSha1HmacTransform 7 cellCryptoSpuSha1Init cellCryptoSpuSha1Transform 7 cellCryptoSpuAesDecryptParallel8 7 cellCryptoSpuAesDecrypt 7 cellCryptoSpuAesDecKeySet 7 cellCryptoSpuAesEncKeySet CellCryptoSpuAesEncryptParallel8 cellCryptoSpuAesEncrypt cellCryptoSpu ecc check is in order T cellCryptoSpu ecc check is in GEp CellCryptoSpu ecc check is in E 7 cellCryptoSpu ecc check curve type 7 cellCryptoSpu ecc ec bittest 7 cellCryptoSpu ecc ec bitlength f cellCryptoSpu ecc ec is zero4ic CellCryptoSpu ecc. ec. set. zero4ic cellCryptoSpu ecc ec moy cellCryptoSpu ecc ec NA2MI CellCryptoSpu ecc. ec. MI2NA 7 cellCryptoSpu ecc ec dbl cellCryptoSpu ecc ec add 7 cellCryptoSpu ecc_ec_mul_sim CollCovintoSnuSha1Einal cellCryptoSpuSha1OneBlk

7 cellCryptoSpu_ecc_mpn_sub_64_64 cellCryptoSpu ecc mpn sub CellCryptoSpu ecc. mpp. sub 256-256 CellCryptoSpu ecc. mpn sub 192 192 7 cellCryptoSpu ecc mpn mods 192 CellCryptoSpu ecc mpn addm 192 192 CellCryptoSpu ecc mpn subm 256 256 cellCryptoSpu ecc mpn subm 192 192 7 cellCryptoSpu ecc mpn mul 128 32 7 cellCryptoSpu ecc mpn mul ui CellCryptoSpu ecc.mpp. mul. 160-32 CellCryptoSpu ecc.mpn.mul 160 32 2 7 cellCryptoSpu ecc mpn spu shi CellCryptoSpu ecc mpn spu shr 7 cellCryptoSpu ecc mpn shl 7 ifl spu ecc mpn mul 128 r cellCryptoSpu ecc mpn mul 256 256 7 cellCryptoSpu ecc mpn shr CollCovotoSpu ecc. mpp. mods 256 cellCrvptoSpu ecc mpn mod 322 161 7 cellCryptoSpu ecc mpn divm 256 CellCryptoSpu ecc mpn inv 256 CellCryptoSpu ecc mpn inv mod2to32 7 cellCryptoSpu ecc mpn cal pd 7 cellCryptoSpu ecc mpn MM 160 7 cellCryptoSpu_ecc_mpn_MR_160 7 cellCryptoSpu ecc mpn mulR mod 160 7 cellCryptoSpu ecc mpp mul mod 161 161 cellCryptoSpu_ecc_mpn_uc2ui CellCryptoSpu ecc.mpn uc2mpn 160 7 cellCryptoSpu ecc mpn uc2mpn 161 7 cellCryptoSpu ecc mpn mpn2ui 192 T ----

CellCryptoSpuAesCbcCfb128Decrypt cellCryptoSpuAesCtrMode CellCryptoSpu ecc ecdsa veri 7 cellCryptoSpuEccEcDsaVeri cellCryptoSpuEccEcDsaVeriCType cellCryptoSpu ecc mpn zero clear cellCryptoSpu ecc mpn set ui 7 cellCryptoSpu ecc mpn is zero cellCryptoSpu ecc mpp is one cellCryptoSpu_ecc_mpn_mov 7 cellCryptoSpu_ecc_mpn_cmp cellCryptoSpu ecc mpn is even CellCryptoSpu ecc mpn add 128 128 cellCryptoSpu ecc mpn add 128 128 no cin cellCryptoSpu ecc mpn add 256 256 cellCryptoSpu ecc mpn add 192 192 cellCryptoSpu ecc mpn add 128 128 2 cellCryptoSpu ecc mpn add 128 128 2 no cin cellCryptoSpu ecc mpn add 128 128 2 no cout. CellCryptoSpu ecc mpn add r cellCryptoSpu ecc mpn sub 128 128

Reverse Engineering - SPU



<□▶ <□▶ < □▶ < □▶ < □▶ < □ > ○ < ○

PS3 DRM - ECDSA Signatures

ECDSA signing in a nutshell:

- ▶ Given elliptic curve *E*, point $G \in E$ of order *n*, signer private key $k \in [1, n-1]$.
- For signing a message *m*:
 - Sample random $t \in [1, n-1]$,
 - compute (x, y) = tG,
 - $\blacktriangleright \quad \text{compute } r = x \mod n$
 - compute $s = t^{-1}(H(m) + rk) \mod n$,
 - output (r, s) as the signature.

What did Sony do? Keep t fixed.

- For different m, m' we then have signatures (r, s) and (r, s'), respectively.
- ► Solve for $t = (s s')^{-1}(H(m) H(m')) \mod n$.
- Solve for **private key** $k = r^{-1}(st H(m)) \mod n$.

Nintendo - DS/DSi/3DS

- DS first released in 2004, DSi in 2008, and 3DS in 2011.
- Mostly custom designed hardware by Nintendo Research & Engineering.

▲□▶ ▲□▶ ▲□▶ ▲□▶ □ の00

- Lots and lots of custom Nintendo DRM.
- Security through obscurity.
- "Custom" cryptography.

Nintendo - DS/DSi/3DS

- These use a hardware AES engine for encryption/decryption of sensitive data.
- Keys are written to keyslots by secure boot stages, keyslots are write-only.
- Some keyslots are flushed by the hardware AES engine whenever any part of the key is written to.
 - Encrypt some plaintext M, and record corresponding ciphertext C^* .
 - Overwrite one byte of the key with $b \in [0, 255]$, encrypt same M to obtain C_b .

- Compare C^* with C_b , if they match we discovered one byte of the key.
- Otherwise, chose different $b \in [0, 255]$, repeat until all bytes known.

- Other keyslots are derived from a hardware key scrambler K_i = F(X_i, Y_i) for inputs slots X_i, Y_i, and some "secret" function F.
- Encrypting a chosen plaintext for carefully chosen X, Y pairs recovers $F(X, Y) = (((X \ll 2) \oplus Y) + C) \ll 87$, for some constant C.
- ► A SAT solver can find a solution to such a system quickly.
- This was also used to recover secret X_i values set by the bootrom, some keys K_i did leak from another device (used for interoperability).

Nintendo/Nvidia - Switch

▶ First released in 2017.

- ► SoC (CPU & GPU) by Nvidia.
 - ARM main CPU cores.
 - Secure Boot from BPMP (Boot and Power Management Processor).
 - ARM TrustZone runs tiny "hypervisor" mostly responsible for cryptography.

- Lots and lots of custom Nintendo DRM.
- Security through obscurity.
- "Custom" cryptography.

- ▶ SoC contains hardware AES engine for encryption/decryption.
- ► Keys are written to keyslots by secure boot stages, keyslots are write-only.
- Keyslots are flushed by the hardware AES engine whenever any word (32bit) of the key is written.

We have already seen this problem earlier. Partial key overwrites is a flaw many hardware crypto engines have.

Nvidia - TSEC

Nvidia makes graphics cards.

- ▶ These cards need to talk HDCP (i.e. HDMI DRM).
- > TSEC is a security co-processor handling cryptography.
- ▶ It supports secure boot for programs that can access sensitive hardware keys.
- To load such a program, one needs to write a signature to a hardware register before jumping to the program.
- ▶ The signature is essentially a Davies-Meyer hash of the input program.
- The signature is computed into another hardware register, but not cleared on verification failure.

Outro

- All of the points and bugs mentioned can be found documented in various places on the internet.
- ▶ We have glossed over many details, please go read up on them.
- Try to keep cryptography open and documented, hackers will figure it out in any case.

Questions?