Cryptanalysis of INCrypt32 in HID's iCLASSTM Systems

ChangKyun Kim, Eun-Gu Jung, Dong Hoon Lee, Chang-Ho Jung, and Daewan Han

ETRI, Korea {kimck, egjung, dlee, zangho, dwh}@ensec.re.kr

Abstract. The cryptographic algorithm called INCrypt32 is a MAC algorithm to authenticate participants, RFID cards and readers, in HID Global's iCLASS systems. HID's iCLASS cards are widely used contactless smart cards for physical access control. Although INCrypt32 is a heart of the security of HID's iCLASS systems, its security has not been evaluated yet since the specification has not been open to public. In this paper, we reveal the specification of INCrypt32 by reverse engineering an iCLASS card and investigate the security of INCrypt32. As a result, we show that the secret key of size 64 bits can be recovered using only 2^{18} MAC queries if the attacker can request MAC for chosen messages of arbitrary length. If the length of messages is limited to predetermined values by the authentication protocol, the required number of MAC queries grows to 2^{42} to recover the secret key.

Key words: INCrypt32, HID's iCLASS, RFID, reverse engineering, chosen message attack.

1 Introduction

Nowadays Radio Frequency Identification (RFID) is widely used in our life such as electronic payment system and access control. Since wireless communication is easy to intercept and tamper with, its security is a major concern to cryptographic researchers, especially when RFID systems use some proprietary cryptographic algorithms and unpublished protocols. The most representative examples are NXP's MIFARE Classic systems [10] and HID Global's iCLASS systems [8].

According to Kerckhoffs's principal, a proprietary algorithm does not enhance the security of the system [2]. In case of MIFARE Classic, Nohl and Plötz have reverse engineered the hardware of the MIFARE Classic card and introduced weaknesses in its proprietary primitive [6]. So far, several related papers [3, 5, 7, 1] have been published and MIFARE Classic system is considered to be fully broken. One can easily recover a secret key of a MIFARE Classic card and replicate it by using a card emulator such as Proxmark III and OpenPCD.

For the other case, Plötz and Meriac presented security weakness of the HID's iCLASS and extracted secret keys from the firmware images of an HID's iCLASS

reader [4]. However, they have not recovered the proprietary cryptographic algorithm and the details of security protocols. This means that although the attacker is only able to access (read and write) iCLASS cards, they cannot replicate cards because the diversified key of each card and the details of the proprietary cryptographic algorithm are required to replicate cards.

Our Contribution This paper describes the proprietary cryptographic algorithm called $INCrypt32^1$ of HID's iCLASS systems. With authors' best knowledge, this is the first published work to describe the details of the INCrypt32 algorithm. We fully recover the INCrypt32 algorithm and communication protocols.

Furthermore, we investigate the security of INCrypt32. Our study shows that INCrypt32 is vulnerable to chosen message attacks. If attacker is allowed to request MAC for arbitrary messages, then the secret key can be recovered within 2^{18} MAC queries. If the length of messages is limited to the specified value (determined by an authentication protocol), then the required number of MAC queries grows to 2^{42} .

Organization This paper is organized as follows. The next section briefly describes the structure of the HID's iCLASS card and the overview of the authentication protocol between iCLASS cards and readers [9]. In Section 3, we explain how to reveal INCrypt32 algorithm and introduce the full description of the algorithm. But we omit the details of reverse engineering procedure since it is beyond the scope of this paper. In Section 4, we present the weaknesses of INCrypt32. Finally, we conclude in Section 5.

2 Preliminaries

2.1 Structure of HID's iCLASS Cards

HID's iCLASS cards are widely used contactless smart cards for physical access control and are compliant with ISO 15693 and 14443B standards. The iCLASS card is fundamentally a memory card with simple security mechanisms for access control. The memory is divided into data blocks which are grouped into application areas.

The iCLASS 2K (256-byte) has 32 blocks configured with 2 application areas. This card does not allow for ISO 14443B standard. While the iCLASS 16K/2 has 256 blocks with 2 application areas, the iCLASS 16K/16 has 256 blocks with 16 application areas split into 8 pages evenly. These cards allow for both ISO 15693 and ISO 14443B standards. The memory map of each card is shown in Table 1.

² Authors Suppressed Due to Excessive Length

¹ We borrow the algorithm name 'INCrypt32' from http://www.insidesecure.com/ eng/content/download/502/3679/version/2/file/01_Flyer_MicroPass4006_BD. PDF, INSIDE Secure.

2K Memory		16K/2 Memory		16K 16 Memory		
Block	Data	Block	Data	Page	Block	Data
0	Card serial number	0	Card serial number	- 0	0	Card serial number
1	Configuration data	1	Configuration data		1	Configuration data
2	Stored value area	2	Stored value area		2	Stored value area
3	Key 1 (\mathcal{K}_d)	3	Key 1 (\mathcal{K}_d)		3	Key 1 (\mathcal{K}_d)
4	Key 2 (\mathcal{K}_c)	4	Key 2 (\mathcal{K}_c)		4	Key 2 (\mathcal{K}_c)
5	Application issuer data	5	Application issuer data		5	Application issuer data
6-18	Application area 1	6-18	Application area 1		6 - 18	Application area 1
19–31	Application area 2	19 - 255	Application area 2		19–31	Application area 2
				Page 1–7		

Table 1. iCLASS card memory map

The first 6 blocks contain special data. Block 0 contains the card serial number (CSN) used in the anti-collision procedure. Block 1 has card configuration information which contains a security option, application limit for secured page, and read/write access. The value stored in block 2 is for the electronic purse. Increment and decrement of this value should be authenticated using \mathcal{K}_c (Credit Key) and \mathcal{K}_d (Debit Key), respectively. Block 3 and 4 contain secret keys which are derived values from the master key and the CSN to create a unique key. These keys are used for an authentication with the reader to allow the execution of read and write commands. Data blocks from 6 to 18, application area 1, are protected by \mathcal{K}_d and the others, application area 2, are protected by \mathcal{K}_c .

2.2 Communication Protocols Using INCrypt32

MIFARE Classic makes use of an authentication protocol by means of the proprietary stream cipher CRYPTO-1. After an authentication protocol is performed successfully, all communications are encrypted by CRYPTO-1. However, iCLASS only makes use of the proprietary symmetric cryptographic algorithm INCrypt32 when an authentication and write commands are performed [9].

Authentication Protocol To get an access right over the iCLASS card, the card and the reader need to perform an authentication protocol. This protocol is based on INCrypt32 with the 8-byte key \mathcal{K}_d or \mathcal{K}_c . The authentication protocol is shown in Fig. 1.

Before the authentication protocol is performed, the reader derives a diversified key (\mathcal{K}_d or \mathcal{K}_c) for the card from the master key and the card serial number. The stored value of block 2 is sent by the card. The reader then sends a 4-byte random number (RND) with its 4-byte signature (MAC0_r) which is the half of

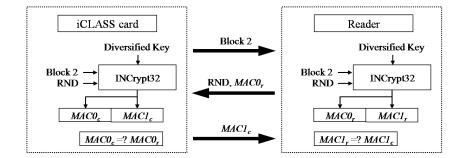


Fig. 1. Authentication protocol

an 8-byte MAC. At this point, the card can compute an 8-byte MAC in the same way. If $MAC0_r$ is correct, the card will answer the other 4-byte signature $(MAC1_c)$ that enables the reader to authenticate the card. Therefore, the authentication protocol needs to perform INCrypt32 with 12-byte input and 8-byte output data.

Write Protocol If the authentication protocol succeeds, the reader is able to read data blocks without an additional authentication procedure. However, to write an 8-byte data in a data block, the reader needs to perform INCrypt32 every time. The write protocol is described in Fig. 2.

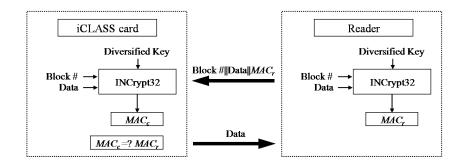


Fig. 2. Write protocol

Once the card is selected and authenticated, the reader is able to write the addressed memory block. The reader sends an 8-byte data with an 1-byte block address and a 4-byte signature (MAC_r). At this point, the card can compute a 4-byte signature MAC_c in the same way. If MAC_r is correct, the card writes the 8-byte data at the addressed memory and answers the data to be stored.

Therefore, the write protocol needs to perform INCrypt32 with 9-byte input and 4-byte output data.

3 Revealing the INCrypt32 Algorithm

We describe how to reveal the proprietary symmetric cryptographic algorithm INCrypt32 from its silicon implementation alone. The details of INCrypt32 will be described as well.

3.1 Hardware Reverse-Engineering of iCLASS Cards

Several steps were taken for the hardware reverse-engineering of the iCLASS 2K card. After removing the card packaging, we decapsulated the card chip. For die delayering, we stripped off an upper layer to expose a lower layer. The picture of each layer was obtained with an auto-stage optical microscope at a magnification of 1,000x after die deprocessing, as shown in Fig. 3.

The chip on iCLASS cards is small with a total area of one square millimeter. This chip consists of three metal layers and a single poly layer and is implemented with 0.5μ m CMOS process. We extracted 108 different logic gates with different driving strengths by analyzing the poly layer picture and the first metal layer picture.

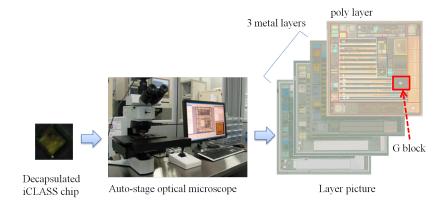


Fig. 3. iCLASS chip's layered picture and physical location of INCrypt32

To reconstruct the primary whole chip schematic, we got the interconnection information from the rest two layer pictures. Most of symmetric cryptographic algorithms usually use XOR/XNOR operations. This fact is a good approach to find the circuit including cryptography. Since the chip is visually divided into 18 blocks, we investigated the number of XOR/XNOR gates each block.

Since the block (we denote it by G block) near the right bottom corner isolated from other blocks has the largest number of XOR/XNOR gates, we guessed that G block would contain INCrypt32 (see Fig. 3). Fortunately, INCrypt32 was found at the guessed block. It contains single 16-bit LFSR, single 8-bit LFSR, two 8-bit full adders composed of 1-bit full adders, 16 1-bit D flip-flops, and an output block with complex Boolean functions.

Without reconstructing control circuits, the roles of input/ouput pins were derived from analyzing two types of captured signals of the communication protocol between the reader and the card. The first one is obtained from RF sniffing and the other is taken from decapsulated chip probing using a probestation with active probes, where probing pads are made by a Focused Ion Beam equipment.

3.2 Description of INCrypt32 Algorithm

We will describe bytes and words(2 bytes) as capital letters and describe a bit as a small letter. x_i means a *i*-th lsb of X. For example, a byte X is same to $x_7 \cdots x_0$.

INCrypt32 consists of internal 40-bit registers (P, Q, R, S), 64-bit key \mathcal{K} , and a state update function F. The details of internal registers and the key are as follows:

- P, Q: 1-byte registers, respectively.
- -R: 1-byte register. r_i is used as the state of 8-bit LFSR.
- -S: 2-byte register. s_i is used as the state of 16-bit LFSR.
- $-\mathcal{K} = (K_0, \cdots, K_7)$: 64-bit key, where each K_i is a byte.

In addition, a superscript of each register such as P^i represents the number of updating the state from the initial state (for example P^0). In other words, $(P^i, Q^i, R^i, S^i) = F(P^{i-1}, Q^{i-1}, R^{i-1}, S^{i-1})$. Moreover p_j^i means a *j*-th bit of P^i . We will use following notations for the description of F.

- -A + B: addition of bytes A and B modulo 2^8 .
- $A \oplus B$: byte-wise XOR of A and B.
- $-a \oplus b$: bit-wise XOR a and b.
- -ab: bit-wise AND of a and b.
- $-A \gg l$: right *l*-bits shift of *A*.
- $-a \ll l$: a byte(word) which is made from left *l*-bit shift of a bit *a*.
- $-\phi(u, v, w)$: an integer 4u + 2v + w.

For an input bit m_i and the key \mathcal{K} , F updates the internal state $(P^{i-1}, Q^{i-1}, R^{i-1}, S^{i-1})$ to the next state (P^i, Q^i, R^i, S^i) as follows.

$$\begin{split} P^{i} &= (X^{i-1} \oplus R^{i-1}) + Q^{i-1} \\ Q^{i} &= P^{i-1} + P^{i} \\ R^{i} &= (R^{i-1} \gg 1) \oplus ((r_{4}^{i-1} \oplus r_{5}^{i-1} \oplus r_{6}^{i-1} \oplus x_{0}^{i-1} \oplus q_{0}^{i-1}) \ll 7) \\ &e &= s_{0}^{i-1} \oplus s_{1}^{i-1} \oplus s_{4}^{i-1} \oplus s_{5}^{i-1} \oplus s_{8}^{i-1} \oplus s_{10}^{i-1} \oplus s_{14}^{i-1} \oplus s_{15}^{i-1} \\ S^{i} &= (S^{i-1} \gg 1) \oplus ((e \oplus p_{3}^{i} \oplus p_{7}^{i}) \ll 15) \\ &u &= p_{0}^{i} \oplus p_{4}^{i} \oplus p_{1}^{i} p_{3}^{i} \oplus p_{2}^{i} p_{4}^{i} \oplus e \\ &v &= p_{3}^{i} \oplus p_{5}^{i} \oplus p_{6}^{i} \oplus p_{3}^{i} p_{5}^{i} \oplus p_{4}^{i} p_{6}^{i} \oplus p_{5}^{i} p_{7}^{i} \\ &w &= m_{i} \oplus p_{0}^{i} \oplus p_{1}^{i} \oplus p_{2}^{i} \oplus p_{5}^{i} \oplus p_{6}^{i} \oplus p_{7}^{i} \oplus p_{0}^{i} p_{2}^{i} \oplus p_{5}^{i} p_{7}^{i} \oplus e \\ &X^{i} &= K_{\phi(u,v,w)} \end{split}$$

In the above we used dummy variables X^i, e, u, v, w , which do not exist in the real iCLASS card. F is depicted also in Fig. 4.

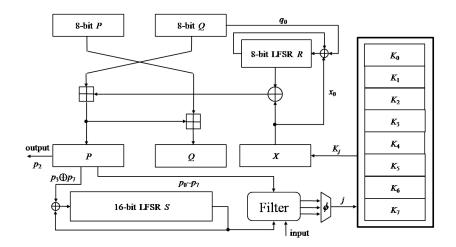


Fig. 4. State update function F of INCrypt32

INCrypt32 consists of an initialization step and a MAC computation step. The both steps use the same state update function F. The difference is that the former step initializes the internal state with each input message bit without output and the latter step outputs MAC bits assuming that input bit is 0. Overall process of INCrypt32 is described in Algorithm 1.

We were able to verify that the reconstructed INCrypt32 is exactly same one implemented in iCLASS cards by comparing output data of our reconstructed INCrypt32 with RF sniffing data. The diversified key used for verification of INCrypt32 was extracted from probing signals of input pads.

Algorithm 1 INCrypt32 algorithm

```
Require:
    1. \mathcal{K} = (K_0, K_1, \cdots, K_7): the secret key
    2. (m_1, m_2, \cdots, m_\alpha): arbitrary input message of length \alpha
    3. (b_1, b_2, \cdots, b_\beta): MAC output of length \beta
Ensure:
    1. initial state: P^0 = 0xcb; Q^0 = 0x21; R^0 = 0x4c; S^0 = 0xe012; X^0 = K_0
 1: Init_State((m_1, m_2, \cdots, m_\alpha))
 2: Compute_MAC()
 3: procedure INIT_STATE((m_1, m_2, \cdots, m_\alpha))
       for i=1 to \alpha do
 4:
           State_update(m_i)
 5:
 6:
       end for
 7: end procedure
 8: procedure COMPUTE_MAC()
9:
       for i=1 to \beta do
10:
            State_update(0)
           b_i = p_2^{\alpha + i}
11:
12:
        end for
13: end procedure
```

4 Cryptanalysis of INCrypt32

In this section we describe two kinds of key recovery attacks on INCrypt32. The purpose of our attacks is to recover the secret key. In our attacks, the attacker is able to request MAC for arbitrary messages of his/her choice. The first attack allows messages of *any* length, especially very short messages. It requires only about 2^{18} MAC queries to recover the secret key. In the second attack, the ability of the attacker is limited that he/she can only request MAC for messages of the fixed length (eg. 72 or 96 bits) as implemented in HID's iCLASS systems. Thus the second attack scenario is more natural. It requires about 2^{42} MAC queries to recover the key.

We would like to mention that we did not break the algorithm in *real* HID's iCLASS systems. The system does not allow MAC queries for messages of *arbitrary* length. Even in the second attack scenario, the required time for MAC query is too long to mount the attack since the speed of communications between cards and readers is very slow.

4.1 How to Choose Messages

INCrypt32 initializes the internal state with each bit of the input message without output. Each m_i affects the key selecting process as follows.

$$u = p_{0}^{i} \oplus p_{4}^{i} \oplus p_{1}^{i} p_{3}^{i} \oplus p_{2}^{i} p_{4}^{i} \oplus e$$

$$v = p_{3}^{i} \oplus p_{5}^{i} \oplus p_{6}^{i} \oplus p_{3}^{i} p_{5}^{i} \oplus p_{4}^{i} p_{6}^{i} \oplus p_{5}^{i} p_{7}^{i}$$

$$w = m_{i} \oplus p_{0}^{i} \oplus p_{1}^{i} \oplus p_{2}^{i} \oplus p_{5}^{i} \oplus p_{6}^{i} \oplus p_{7}^{i} \oplus p_{0}^{i} p_{2}^{i} \oplus p_{5}^{i} p_{7}^{i} \oplus e$$

$$X^{i} = K_{\phi(u,v,w)}$$
(1)

According to the above Equation (1), the attacker can control the w value by selecting m_i as he/she wants. In particular, w can be selected to be zero by selecting m_i as follows.

$$m_i = p_0^i \oplus p_1^i \oplus p_2^i \oplus p_5^i \oplus p_6^i \oplus p_7^i \oplus p_0^i p_2^i \oplus p_5^i p_7^i \oplus e.$$

In other words, the indices of the selected key byte determined by $\phi(u, v, w)$ can be always even number. Thus if the attacker knows the half of the secret key (K_0, K_2, K_4, K_6) , he/she can initialize the internal state for the selected message. It is easy to show that m_i is uniquely determined if (K_0, K_2, K_4, K_6) is given. We call the uniquely determined message bit for the half key by the even message bit. Similarly, even messages of arbitrary length consist of only even message bits. Note that the even message of a given length is uniquely determined for a half key.

On the other hands, the attacker knows the internal state after initialization for even messages assuming that he/she knows the half key. Then he/she also computes the first bit of MAC for the message since the output bit is derived from the internal state after initialization. Therefore the attacker can guess the half of the key and filter out the wrong key by comparing the first bit of two MACs (computed and queried) for even messages. Our attack strategy is as follows.

- 1. Guess the half of the secret key $\mathcal{HK} = (K_0, K_2, K_4, K_6)$.
 - (a) Find the even messages and tweak messages for the half key \mathcal{HK} and compute the first bit of MACs of each message assuming the guessed key is correct. Tweak messages will be explained later.
 - (b) Request the correct MACs for the *even messages* and *tweak* messages.
 - (c) Compare the first bit of two MACs (computed and queried) and filter out the wrong half key until finding the correct one.
- 2. Determine the rest of the secret key.

4.2Attack with Short Messages

In the first attack, we request MAC for short messages of length at most 33 bits. At first, we define some notations for a given half key $\mathcal{HK} = (K_0, K_2, K_4, K_6)$.

- m_i : even message bits for $i = 1, 2, \cdots$.
- $-M_i = (m_1, m_2, \cdots, m_i)$: even message of length *i*. $-c_i = p_2^{i+1}$ for the even message M_i . It is the first bit of the computed MAC. It should be identical with the first bit of the queried MAC for M_i if the half kev \mathcal{HK} is correct.

Making a Precomputation Table As described in Section 4.1, we have to determine even message bits m_i $(i = 1, 2, \dots, 16)$ for a given half key \mathcal{HK} . This process can be done independent of the target card. Thus we can make a precomputation table with even message bits m_i and additional information c_i associated with 32-bit half keys in advance.

To make a table, each \mathcal{HK} is regarded as a index (memory address) and $(m_1, m_2, \cdots, m_{16} \mid c_1, c_2, \cdots, c_{16})$ is stored as a 32-bit data for the index \mathcal{HK} . Hence we need a memory space of size $4 \cdot 2^{32}$ bytes (16GB). Note that the distribution of the data is uniform in the sense that the number of elements in the table with the first *i* bits are identical is about 2^{32-i} for $i = 1, \cdots, 16$ and that c_i can be 0 or 1 with the probability 1/2.

Reducing the Number of Candidates of the Half Key This is the first step of online phase. Now we are ready to request MACs for short messages. We request MACs for 1-bit messages (0) and (1) respectively. Let the first bit of the MACs be b_0 and b_1 respectively. Then we can filter out the half keys such that the data has forms of $(0, *, \cdots | b_0 \oplus 1, *, \cdots)$ and $(1, *, \cdots | b_1 \oplus 1, *, \cdots)$ where * can be any value. As a result, about 2^{31} candidates can survive.

Next, we request MACs for all 2-bit messages (0,0), (0,1), (1,0), and (1,1) respectively. Let the first bit of the MACs be b_{00} , b_{01} , b_{10} , and b_{11} respectively. Then we also filter out the survived keys such that the data has forms of $(0,0,*,\cdots | *,b_{00} \oplus 1,*,\cdots)$, $(0,1,*,\cdots | *,b_{01} \oplus 1,*,\cdots)$, $(1,0,*,\cdots | *,b_{10} \oplus 1,*,\cdots)$, and $(1,1,*,\cdots | *,b_{11} \oplus 1,*,\cdots)$. As a result, about 2^{30} candidates can survive.

In the same manner, we request MACs for all the messages of length at most 16 bits and can filter out wrong key candidates by comparing the first bit of MACs and the precomputed table. Finally, we have about 2^{16} candidates of the half key. The required number of MAC queries is $2 + 2^2 + \cdots + 2^{16} = 2^{17} - 2$.

Determining the Correct Half Key After above filtering, we have 2^{16} candidates of the half key and associated 16-bit even messages. In the second phase, we verify the correctness of each candidate in turn.

Let \mathcal{HK} be one of the survived keys. We determine 17-th even message bit m_{17} and compute c_{17} as described in the previous section. Then we request MAC for $M_{17} = (m_1, \dots, m_{17})$ and compare the first bit of the MAC with c_{17} . If they are different, then \mathcal{HK} can be filtered out. The probability that a wrong key passes the test is 1/2. If the candidate passes the test, we proceed the same test for longer even messages M_{18}, \dots, M_{33} . If \mathcal{HK} is correct, all the tests should be passed. Since the probability that a wrong key passes all the tests is $1/2^{17}$, we can expect that the only correct key survives. The required number of MAC queries is $2^{16} + 2^{15} + \dots + 1 = 2^{17} - 1$.

Determining the Rest of the Key In this phase, we will find new *tweak* messages slightly different from even messages. Let m_1 be the first even message bit and 2l be the associated key index, i.e. $X^1 = K_{2l}$.

We guess K_{2l+1} . Let T_1 be $(m_1 \oplus 1)$ which is the first *tweak* message. Since we guessed K_{2l+1} , we can update the internal state and compute c_1 for T_1 . Then we request MAC for T_1 and compare the first bit of the MAC with c_1 . If the both values do not coincide, the guessed K_{2l+1} is not correct. Otherwise, we proceed to find the next *tweak* message T_i for $i \geq 2$. The *i*-th message bit is selected such that the key index should be even number. We denote such a message bit by \tilde{m}_i for $i \geq 2$. Thus T_i has the following form.

$$T_i = (m_1 \oplus 1, \tilde{m}_2, \cdots, \tilde{m}_i).$$

Then we can check the correctness of the guessed value K_{2l+1} . We expect the only one value would be survived after 9 consecutive tests and it would be the correct K_{2l+1} . The required number of MAC queries is about $2^8 + 2^7 + \cdots + 1 = 2^9 - 1$.

The other secret keys with odd index can be found in the similar way. The required number of MAC queries is also same respectively. Therefore the total number to determine the rest of the key is about $2^{11} - 4$.

4.3 Attack with Messages of the Fixed Length

The messages of the authentication protocol in real environments are formatted and have fixed lengths (72 or 96 bits). Hence the previous attack cannot be applied to HID's iCLASS systems. In the second attack, we assume that the attacker can request MAC for messages of the fixed length determined by the authentication protocol.

In this attack, we guess the half key $\mathcal{HK} = (K_0, K_2, K_4, K_6)$, and we find the unique even message of given length (eg. 72 bits) for \mathcal{HK} and compute additional information $c = p_2^{73}$ to compare with the first bit of the MAC. As the previous attack, we request MAC for the even message and filter out wrong candidates by comparing the first bit of MACs. Thus we have 2^{31} candidates of the half key. Let (m_1, \dots, m_{72}) be the even message for a survived \mathcal{HK} and v_i be the index of selected key byte in *i*-th updating process, i.e. $X^i = K_{v_i}$. Then $v_i \in \{0, 2, 4, 6\}$.

We will simultaneously find the rest of the secret key with odd index in the process of checking the validity of \mathcal{HK} . For example, we start with K_1 . Let $I_0 = \{i_1, \dots, i_k\}$ be the set of indices *i* such that $v_i = 0$.

For a possible value of K_1 , we have to find the tweak message T_j for each $j \in I_0$ such that the first $v_j - 1$ bits are identical to the even message, the next bit is flipped, and the other bits are selected as *tweak* message bits as the previous subsection.

$$T_j = (m_1, \cdots, m_{v_j-1}, m_{v_j} \oplus 1, \tilde{m}_{v_j+1}, \cdots, \tilde{m}_{72}).$$

Since we know $\mathcal{HK} = (K_0, K_2, K_4, K_6)$ and K_1 , we can initialize the internal state with the tweak message T_j and compute $c_j = p_2^{73}$. We request MAC for T_j and compare the first bit of the MAC with c_j . If both values are identical, we repeat the test for another $j \in I_0$. We have to carry out the test until we find a value of K_1 such that the test passes for all $j \in I_0$.

If we can not find a value of K_1 passing the above test, then the candidate \mathcal{HK} must be incorrect. The probability that wrong (\mathcal{HK}, K_1) passes the test is $1/2^{n_0}$ where n_0 is the number of elements in I_0 . n_0 is expected about 18 = 72/4.

If we find a candidate K_1 , we proceed the above test for K_3 , K_5 , and K_7 in turn in the same way. The wrong 64-bit key may pass the above test with probability $1/2^{72}$, thus it hardly happens. Therefore the survived value is the full secret key. The required number of MAC queries is at most $2^{31} \cdot 4(2^8 + 2^7 + \cdots + 1) \simeq 2^{42}$.

5 Conclusion

Although INCrypt32 plays a key role in the security of HID's iCLASS systems, it has not been evaluated the security since the structure of INCrypt32 is not known to public. In this paper, we revealed the unknown algorithm by reverse engineering iCLASS cards. Unfortunately, we showed that there are some security flaws in cryptographic sense. The secret key of iCLASS cards can be recovered using MAC queries for chosen messages. The number of MAC queries is 2^{18} or 2^{42} which depends on the attack scenario.

Note that our analysis for INCrypt32 is not realistic yet since our attack requires many (in real environments) MAC queries for unauthorized messages. But our work shows that a proprietary cryptographic algorithm does not enhance the security of a system. Moreover, if INCrypt32 were used in other applications which allow very short messages, then the security of a whole system could be totally compromised.

References

- Garcia, F.D., de Koning Gans, G., Muijrers, R., van Rossum, P., Verdult, R., Schreur, R.W., and Jacobs, B.: Dismantling MIFARE Classic. In: Jajodia, S., Lopez, J. (eds.) ESORICS 2008. LNCS, vol. 5283, pp. 97-114. Springer, Heidelberg (2008)
- Kerckhoffs, A.: La cryptographie militaire. Journal des Sciences Militaires IX. 5–38 (1883)
- de Koning Gans, G., Hoepman, J.-H., Garcia, F.D.: A practical attack on the MIFARE Classic. In: Grimaud, G., Standaert, F.-X. (eds.) CARDIS 2008. LNCS, vol. 5189, pp. 267–282. Springer, Heidelberg (2008)
- 4. Meriac, M.: Heart of darkness exploring the uncharted backwaters of HID iCLASSTM security. In: 27th Chaos Communication Congress (2010)
- Nohl, K., Evans, D., Starbug, Plötz, H.: Reverse-engineering a cryptographic RFID tag. In: 17th USENIX Security Symposium 2008. pp. 185–193. (2008).
- Nohl, K., Plötz, H.: Mifare, little security despite obscurity. In: 24th Chaos Communication Congress (2007)
- 7. Teepe, W., Nohl, K.: Making the best of MIFARE Classic (manuscript, 2008)
- 8. HID Global: http://www.hidglobal.com/technology.php?tech_cat=2\ &subcat_id=9\&techno_id=2
- INSIDE Secure.: PicoPass 2KS, http://66.7.214.212/~orangeta/datasheet/ Inside/DSPicopass2KSV1-0.pdf

10. NXP Semiconductors.: MIFARE standard 4KByte card IC functional specification. February (2007)