Dismantling iClass and iClass Elite

Flavio D. Garcia[1], Gerhard de Koning Gans[1], Roel Verdult[1], and Milosch Meriac[2]

[1] Institute for Computing and Information Sciences,
Radboud University Nijmegen, The Netherlands.
{flaviog,gkoningg,rverdult}@cs.ru.nl

[2] Bitmanufaktur GmbH, Germany.
milosch.meriac@bitmanufaktur.de

Abstract. With more than 300 million cards sold, HID iClass is one of the most popular contactless smart cards on the market. It is widely used for access control, secure login and payment systems. The card uses 64-bit keys to provide authenticity and integrity. The cipher and key diversification algorithms are proprietary and little information about them is publicly available. In this paper we have reverse engineered all security mechanisms in the card including cipher, authentication protocol and key diversification algorithms, which we publish in full detail. Furthermore, we have found six critical weaknesses that we exploit in two attacks, one against iClass Standard and one against iClass Elite (a.k.a., iClass High Security). In order to recover a secret card key, the first attack requires one authentication attempt with a legitimate reader and 2^{22} queries to a card. This attack has a computational complexity of 2^{40} MAC computations. The whole attack can be executed within a day on ordinary hardware. Remarkably, the second attack which is against iClass Elite is significantly faster. It directly recovers the master key from only 15 authentication attempts with a legitimate reader. The computational complexity of this attack is lower than 2^{25} MAC computations, which means that it can be fully executed within 5 seconds on an ordinary laptop.

1 Introduction

iClass is an ISO/IEC 15693 [20] compatible contactless smart card manufactured by HID Global. It was introduced in the market back in 2002 as a secure replacement of the HID Prox card which did not have any cryptographic capabilities. According to the manufacturer, more than 300 million iClass cards have been sold. These cards are widely used in access control of secured buildings such as The Bank of America Merrill Lynch, the International Airport of Mexico City and the United States Navy base of Pearl Harbor [9] among many others[3]. Other applications include secure user authentication such as in the naviGO system included in Dell's Latitude and Precision laptops; e-payment like in the FreedomPay and SmartCentric systems; and billing of electric vehicle charging

[3] http://hidglobal.com/mediacenter.php?cat2=2

such as in the Liberty PlugIns system. iClass has also been incorporated into the new BlackBerry phones which support Near Field Communication (NFC).

iClass uses a proprietary cipher to provide data integrity and mutual authentication between card and reader. The cipher uses a 64-bit diversified key which is derived from a 56-bit master key and the serial number of the card. This key diversification algorithm is built into all iClass readers. The technology used in the card is covered by US Patent 6058481 and EP 0890157. The precise description of both the cipher and the key diversification algorithms are kept secret by the manufacturer following the principles of security by obscurity. Remarkably, all iClass Standard cards worldwide share the same master key for the iClass application. This master key is stored in the EEPROM memory of every iClass reader. It is possible though to let HID generate and manage a custom key for your system if you are willing to pay a higher price. The iClass Elite Program (a.k.a., High Security) uses an additional key diversification algorithm and a custom master key per system which according to HID provides "the highest level of security" [19].

Over the last few years, much attention has been paid to the (in)security of the cryptographic mechanisms used in contactless smart cards [14,17,27,32]. Experience has shown that the secrecy of proprietary ciphers does not contribute to its cryptographic strength. Most notably the Mifare Classic, which has widespread application in public transport ticketing and access control systems, has been thoroughly broken in the last few years [6,11,14,16,26]. Other prominent examples include KeeLoq [4,22] and Hitag2 [7,30,32] used in car keys and CryptoRF [1,2,17] used in access control and payment systems. HID proposes iClass as a migration option for systems using Mifare Classic, boosting that iClass provides "improved security, performance and data integrity"[4]. For almost one decade after its introduction to the market, the details of the security mechanisms of iClass remained unknown.

Our contribution In this paper we have fully reverse engineered iClass's proprietary cipher and authentication protocol which we publish in full detail. This task is not trivial since it was first necessary to bypass the read protection mechanisms of the microcontroller used in the readers in order to retrieve its firmware. Furthermore we have found serious vulnerabilities in the cipher that enable an attacker to recover the secret key from the card by just wirelessly communicating with it. The potential impact of this attack is vast since other vulnerabilities in the key diversification algorithm allow an adversary to use this secret key to recover the master key, provided that he has mild computational power. Additionally, we have reverse engineered the iClass Elite key diversification algorithm which we describe in full detail. We show that this algorithm has even more serious vulnerabilities than the standard key diversification algorithm, allowing an attacker to directly recover the *master key* by simply communicating with a legitimate iClass reader. Concretely, we propose two attacks: one against iClass Standard and one against iClass Elite. Both attacks allow an adversary to recover the master key.

[4] http://www.hidglobal.com/pr.php?id=393

- The first attack exploits a total of *four* weaknesses in the cipher, key diversification algorithm and implementation. In order to execute this attack the adversary first needs to eavesdrop one legitimate authentication session between card and reader. Then it runs 2^{19} key updates and 2^{22} authentication attempts with the card. This takes less than six hours to accomplish when using a Proxmark III as a reader and recovers 24 bits of the card key. Finally, off-line, the attacker needs to search for the remaining 40 bits of the key. Having recovered the card key, the adversary gains full control over the card. Furthermore, computing the master key from the card key is as hard as breaking single DES [15].
- The second attack concerning iClass Elite exploits *two* weaknesses in the key diversification algorithm and recovers the master key directly. In order to run this attack the adversary only needs to run 15 authentication attempts with a legitimate reader. Afterwards, off-line, the adversary needs to compute only 2^{25} DES encryptions in order to recover the master key. This attack, from beginning to end runs within 5 seconds on ordinary hardware.

We have executed both attacks in practice and verified these claims and attack times. For eavesdropping and card emulation we used a Proxmark III (see http://www.proxmark.org) which costs approximately 200 USD.

Related work Recently, Meriac proposed a procedure to read out the EEPROM of a PIC microcontroller, like the ones used in iClass readers [25]. The reverse engineering process described here builds upon this work. Garcia, de Koning Gans and Verdult in [15] have reverse engineered the key diversification algorithm of iClass and showed that it is possible to recover a master key when the adversary has full control (i.e., can execute arbitrary commands) over a legitimate iClass reader. They also showed that inverting the key diversification function in iClass is as hard as a chosen plaintext attack on single DES. During the course of our research Kim, Jung, Lee, Jung and Han have made a technical report [23] available online describing independent reverse engineering of the cipher used in iClass. Their research takes a very different, hardware oriented approach. They recovered most of the cipher by slicing the chip and analyzing the circuits with a microscope. Our approach, however, is radically different as our reverse engineering is based on the disassembly of the reader's firmware and the study of the communication behavior of tags and readers. Furthermore, the description of the cipher by Kim et al. is not correct. Concretely, their key byte selection function in the cipher is different from the one used in iClass which results in incompatible keys. Kim et al. have proposed two key recovery attacks. The first one is theoretical, in the sense that it assumes that an attacker has access to a MAC oracle over messages of arbitrary length. This assumption is unrealistic since neither the card nor the reader provide access to such a powerful oracle. Their second attack requires full control over a legitimate reader in order to issue arbitrary commands. Besides this assumption, it requires 2^{42} online authentication queries which, in practice, would take more than 710 years to gather. Our attacks, however, are practical in the sense that they can be executed within a day and require only wireless communication with a genuine iClass card/reader.

Overview This paper is organized as follows. Section 2 starts with a description of the iClass architecture, the functionality of the card, the cryptographic algorithms. Section 2.6 describes four weakness in the cipher, key diversification algorithm and implementation of iClass. All these weaknesses are exploited in Section 2.7 were we propose a key recovery attack against iClass. Section 3 studies iClass Elite. We first describe its key diversification algorithm and then we describe two weaknesses which are later exploited in Section 3.3 to mount an attack that recovers the master key. Finally, Section 4 gives concluding remarks.

2 iClass

An HID iClass card is in fact a pre-configured and re-branded PicoPass card produced by Inside Secure[5]. HID configures and finalizes the cards so that the configuration settings can no longer be modified. This section describes in detail the functionality and security mechanisms of iClass and it also describes the reverse engineering process. Let us first introduce notation.

Notation 2.1 *Throughout this paper* ϵ *denotes the empty bitstring.* $\oplus$ *denotes exclusive or.* $\boxplus$ *denotes addition modulo* 256*. Given two bitstrings* x *and* y*,* xy *denotes their concatenation. Sometimes we write this concatenation explicitly with* $x \cdot y$ *to improve readability.* $\overline{x}$ *denotes the bitwise complement of* x*.* 0^n *denotes a bitstring of* n *zero-bits. Furthermore, given a bitstring* $x \in (\mathbb{F}_2^k)^l$*, we denote with* $x_{[i]}$ *the* i*-th element* $y \in \mathbb{F}_2^k$ *of* x*. We write* y_i *to denote the* i*-th bit of* y*. For example, given the bitstring* $x =$ `0x010203` $\in (\mathbb{F}_2^8)^3$ *and* $y := x_{[2]}$ *then* $y =$ `0x03` *and* $y_6 = 1$*.*

Remark 1 (Byte representation). Throughout this paper, bytes are represented with their most significant bit on the left. However, the least significant bit is transmitted first over the air (compliant with ISO/IEC 15693). This is the same order in which the bits are input to the cryptographic functions. In other words, `0x0a0b0c` *is transmitted and processed as input* `0x50d030`*.*

2.1 Reverse engineering iClass

In order to reverse engineer the cipher and the key diversification algorithms, we have first recovered the firmware from an iClass reader. For this we used a technique introduced in [25] and later used in [15]. Next we will briefly describe this technique.

iClass readers, as many other embedded devices, rely on the popular PIC microcontroller to perform their computations. These microcontrollers are very versatile and can be flashed with a custom firmware. The (program) memory of the microcontroller is divided into a number of blocks, each of them having access control bits determining whether this block is readable/writable. Even when the PIC is configured to be non-writable, it is always possible to reset the access control bits by erasing the memory of the chip. At first glance this feature does

[5] http://www.insidesecure.com/eng/Products/Secure-Solutions/PicoPass

not seem very helpful to our reverse engineering goals since it erases the data on the memory. Conveniently enough, even when the most common programming environments do not allow it, the microcontroller supports erasure of a single block. After patching the PIC programmer software to support this feature, it is possible to perform the following attack to recover the firmware:

- Buy two iClass RW400 (6121AKN0000) readers.
- Erase block 0 on one of the readers. This resets the access control bits on block 0 to readable, writable.
- Write a small dumper program on block 0 that reads blocks $1, \ldots, n$ and outputs the data via one of the microcontroller's output pins.
- Use the serial port of a computer to record the data. This procedure recovers blocks $1, \ldots, n$.
- Proceed similarly with the other reader, but erasing blocks $1, \ldots, n$. This in fact fills each block with NOP operations.
- At the end of block n write a dumper program for block 0.
- At some point the program will jump to an empty block and then reach dumper program that outputs the missing block 0.

Once we had recovered the firmware, it was possible to use IDA Pro and MPLAB to reverse engineer the algorithms.

2.2 Functionality

iClass cards come in two versions called 2KS and 16KS with respectively 256 and 4096 bytes of memory. The memory of the card is divided into blocks of eight bytes as shown in Figure 2.1. Memory blocks 0, 1, 2 and 5 are publicly readable. They contain the card identifier id, configuration bits, the card challenge c_C and issuer information. Block 3 and 4 contain two diversified cryptographic keys $k1$ and $k2$ which are derived from two different master keys $\mathcal{K}1$ and $\mathcal{K}2$. These master keys are referred to in the documentation as debit key and credit key. The card only stores the diversified keys $k1$ and $k2$. The remaining memory blocks are divided into two areas, so-called applications. The size of these applications is defined by the configuration block.

Block	Content	Denoted by
0	Card serial number	Identifier id
1	Configuration	
2	e-Purse	Card challenge c_C
3	Key for application 1	Diversified debit key $k1$
4	Key for application 2	Diversified credit key $k2$
5	Application issuer area	
$6 \ldots 18$	Application 1	HID application
$19 \ldots n$	Application 2	User defined memory

▨	publicly readable
░	write-only after authentication
□	read-write after authentication

Fig. 2.1. Memory layout of an iClass card

The first application of an iClass card is the *HID application* which stores the card identifier, PIN code, password and other information used in access control systems. Read and write access to the HID application requires a valid mutual authentication using the cipher to prove knowledge of $k1$. The master key of the HID application is a global key known to all iClass Standard compatible readers. The globally used key $\mathcal{K}1$ is kept secret by HID Global and is not shared with any customer or industrial partner. Recovery of this key undermines the security of all systems using iClass Standard. Two methods have been proposed [15,25] to recover this key. To circumvent the obvious limitations of having only a global master key, iClass Elite uses a different key diversification algorithm that allows having custom master keys. The details regarding iClass Elite can be found in Section 3. The second global master key $\mathcal{K}2$ is used in both iClass Standard and Elite systems and it is available to any developer who signs a non-disclosure agreement with HID global. It is possible to extract this key from publicly available software binaries [15]. In addition, the document [18] contains this master key and is available online. This key $\mathcal{K}2$ can be used by developers to protect the second application, although in practice, $\mathcal{K}2$ is hardly ever used or modified.

The card provides basic memory operations like read and write which have some non-standard behavior and therefore we describe them in detail.

- The `read` command takes as input an application number a and a memory block number n and returns the memory content of this block. This command has the side effect of selecting the corresponding key ($k1$ for application 1 or $k2$ for application 2) in the cipher and then it feeds the content of block n into the internal state of the cipher. Cryptographic keys are not readable. When the block number n corresponds to the address where a cryptographic key is stored, then `read` returns a bitstring of 64 ones.
- The `write` command takes as input a block number n, an eight-byte payload p and a MAC of the payload $\text{MAC}(k, n \cdot p)$. When successful, it writes p in memory and it returns a copy of p for verification purposes. This command has the side effect of resetting the internal state of the cipher. In addition, when the block number n corresponds to the address where a cryptographic key k is stored, the payload is XORed to the previous value instead of overwriting it, i.e., it assigns $k := k \oplus p$.

Therefore, in order to update a key k to k', the reader must issue a `write` command with $k \oplus k'$ as payload. In this way the card will store $k \oplus k \oplus k' = k'$ as the new key. On the one hand, this particular key update procedure has the special feature that in case an adversary eavesdrops a key update he is unable to learn the newly assigned key, provided that he does not know k. On the other hand this introduces a new weakness which we describe in Section 2.6.2.

Before being able to execute `read` or `write` commands on the protected memory of a card, the reader needs to get access to the corresponding application by running a successful authentication protocol described in Section 2.3. Cryptographic keys $k1$ and $k2$ can be seen as part of application 1 and 2, respectively. This means that in order to modify a key e.g., $k1$, the reader first needs to run a successful authentication with $k1$.

2.3 Authentication protocol

This section describes the authentication protocol between an iClass card and reader. This protocol is depicted in Figure 2.2 and an example trace is shown in Figure 2.3. First, during the anti-collision protocol, the reader learns the identity of the card *id*. Then, the reader chooses an application and issues a

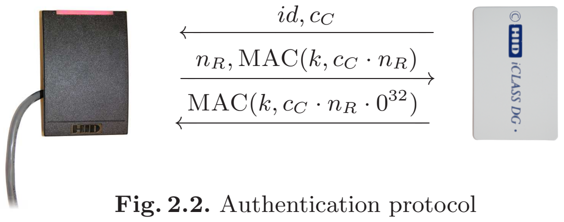
id, c_C
$n_R, \mathrm{MAC}(k, c_C \cdot n_R)$
$\mathrm{MAC}(k, c_C \cdot n_R \cdot 0^{32})$
HID iCLASS DG

Fig. 2.2. Authentication protocol

`read` command on the card challenge c_C. This c_C is called 'e-purse' in the iClass documentation [5] and it is a special memory block in the sense that it is intended to provide freshness. In the next step, the reader issues an `authenticate` command. This command sends to the card a reader nonce n_R and a MAC of the card challenge c_C concatenated with n_R. Finally, the card answers with a MAC of c_C, n_R followed by 32 zero bits. For more details on the MAC function see Section 2.4. After a successful authentication on c_C the reader is granted read and write access within the selected application.

Origin	**Message**	**Description**
Reader	`0C 00 73 33`	Read identifier
Tag	`47 47 6C 00 F7 FF 12 E0`	Card serial number *id*
Reader	`0C 01 FA 22`	Read configuration
Tag	`12 FF FF FF E9 1F FF 3C`	iClass 16KS configuration
Reader	`88 02`	Read c_C and select $k1$
Tag	`FE FF FF FF FF FF FF FF`	Card challenge c_C
Reader	`05 00 00 00 00 1D 49 C9 DA`	Authenticate with $n_R = 0, \mathrm{MAC}(k1, c_C \cdot n_R)$
Tag	`5A A2 AF 92`	Response $\mathrm{MAC}(k1, c_C \cdot n_R \cdot 0^{32})$
Reader	`87 02 FD FF FF FF FF FF FF FF CF 3B D4 6A`	Write on block `02`, $c_C - 1, \mathrm{MAC}(k1, \texttt{02} \cdot c_C - 1)$
Tag	`FF FF FF FF FD FF FF FF`	Update succesful

Fig. 2.3. Authenticate and decrement card challenge c_C using diversified key $k1$ = `0xE033CA419AEE43F9`

Remark 2. Since the card lacks a pseudo-random generator, the reader should decrement c_C after a successful authentication in order to provide freshness for the next authentication, see Figure 2.3. Note that this is not enforced by the card.

2.4 The cipher

This section describes the cipher used in iClass. This cipher is interesting from an academic and didactic perspective as it combines two important techniques in the design of stream ciphers from the 80s and beginning of the 90s, i.e., Fibonacci generators and Linear Feedback Shift Registers (LFSRs).

The internal state of the cipher consists of four registers. Two of them, which we call left (l) and right (r) are part of the Fibonacci generator. The other two registers constitute linear feedback shift registers top (t) and bottom (b).

Definition 1 (Cipher state). *A* cipher state *of iClass s is an element of $\mathbb{F}_2^{40}$ consisting of the following four components: 1. the* left register $l = (l_0 \dots l_7) \in \mathbb{F}_2^8$*; 2. the* right register $r = (r_0 \dots r_7) \in \mathbb{F}_2^8$*; 3. the* top register $t = (t_0 \dots t_{15}) \in \mathbb{F}_2^{16}$*. 4. the* bottom register $b = (b_0 \dots b_7) \in \mathbb{F}_2^8$*.*

The cipher has an input bit which is used (among others) during authentication to shift in the card challenge c_C and the reader nonce n_R. With every clock tick a cipher state s evolves to a successor state s'. Both LFSRs shift to the right and the Fibonacci generator iterates using one byte of the key (chosen by the $select(\cdot)$ function) and the bottom LFSR as input. During this iteration each of these components is updated, receiving additional input from the other components of the cipher. With each iteration the cipher produces one output bit. The following sequence of definitions describe the cipher in detail; see also Figure 2.4.

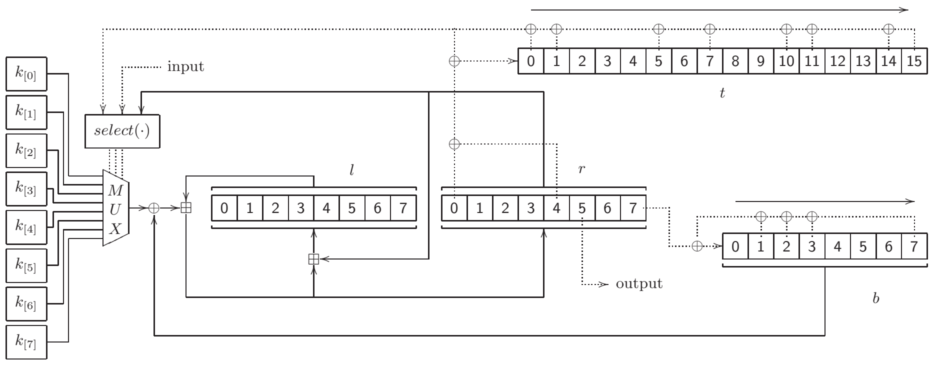

Fig. 2.4. The iClass cipher. Solid lines represent byte operations while dotted lines represent bit operations.

Definition 2. *The feedback function for the top register $T\colon \mathbb{F}_2^{16} \to \mathbb{F}_2$ is defined as $T(x_0x_1 \dots\dots x_{15}) = x_0 \oplus x_1 \oplus x_5 \oplus x_7 \oplus x_{10} \oplus x_{11} \oplus x_{14} \oplus x_{15}$. Similarly, the feedback function for the bottom register $B\colon \mathbb{F}_2^8 \to \mathbb{F}_2$ is defined as $B(x_0x_1 \dots x_7) = x_1 \oplus x_2 \oplus x_3 \oplus x_7$.*

Definition 3 (Selection function). *The selection function $select\colon \mathbb{F}_2 \times \mathbb{F}_2 \times \mathbb{F}_2^8 \to \mathbb{F}_2^3$ is defined as $select(x, y, r) = z_0z_1z_2$ where*

$$z_0 = (r_0 \wedge r_2) \oplus (r_1 \wedge \overline{r_3}) \oplus (r_2 \vee r_4)$$
$$z_1 = (r_0 \vee r_2) \oplus (r_5 \vee r_7) \oplus r_1 \oplus r_6 \oplus x \oplus y$$
$$z_2 = (r_3 \wedge \overline{r_5}) \oplus (r_4 \wedge r_6) \oplus r_7 \oplus x$$

Definition 4 (Successor state). *Let $s = \langle l, r, t, b\rangle$ be a cipher state, $k \in (\mathbb{F}_2^8)^8$ be a key and $y \in \mathbb{F}_2$ be the input bit. Then, the* successor cipher state *$s' = \langle l', r', t', b'\rangle$ is defined as*

$$t' := (T(t) \oplus r_0 \oplus r_4)t_0 \dots t_{14} \qquad l' := (k_{[select(T(t),y,r)]} \oplus b') \boxplus l \boxplus r$$
$$b' := (B(b) \oplus r_7)b_0 \dots b_6 \qquad r' := (k_{[select(T(t),y,r)]} \oplus b') \boxplus l$$

We define the successor function suc *which takes a key* $k \in (\mathbb{F}_2^8)^8$*, a state* s *and an input* $y \in \mathbb{F}_2$ *and outputs the successor state* s'*. We overload the function* suc *to multiple bit input* $x \in \mathbb{F}_2^n$ *which we define as*

$$\mathrm{suc}(k, s, \epsilon) = s$$

$$\mathrm{suc}(k, s, x_0 \dots x_n) = \mathrm{suc}(k, \mathrm{suc}(k, s, x_0 \dots x_{n-1}), x_n)$$

Definition 5 (Output). *Define the function output which takes an internal state* $s =< l, r, t, b >$ *and returns the bit* r_5*. We also define the function output on multiple bits input which takes a key* k*, a state* s *and an input* $x \in \mathbb{F}_2^n$ *as*

$$\mathit{output}(k, s, \epsilon) = \epsilon$$

$$\mathit{output}(k, s, x_0 \dots x_n) = \mathit{output}(s) \cdot \mathit{output}(k, s', x_1 \dots x_n)$$

$$\mathit{where}\ s' = \mathrm{suc}(k, s, x_0).$$

Definition 6 (Initial state). *Define the function init which takes as input a key* $k \in (\mathbb{F}_2^8)^8$ *and outputs the initial cipher state* $s =< l, r, t, b >$ *where*

$$t := \texttt{0xE012} \qquad l := (k_{[0]} \oplus \texttt{0x4C}) \boxplus \texttt{0xEC}$$

$$b := \texttt{0x4C} \qquad r := (k_{[0]} \oplus \texttt{0x4C}) \boxplus \texttt{0x21}$$

Definition 7. *Define the function* MAC: $(\mathbb{F}_2^8)^8 \times \mathbb{F}_2^n \to \mathbb{F}_2^{32}$ *as*

$$\mathrm{MAC}(k, m) = \mathit{output}(k, \mathrm{suc}(k, \mathit{init}(k), m), 0^{32})$$

2.5 Key diversification

This section describes in detail the built-in key diversification algorithm of iClass. Besides the obvious purpose of deriving a card key from a master key, this algorithm intends to circumvent weaknesses in the cipher by preventing the usage of certain 'weak' keys. In order to compute a diversified key, the iClass reader first encrypts the card identity *id* with the master key $\mathcal{K}$, using single DES. The resulting ciphertext is then input to a function called *hash0* which outputs the diversified key k.

$$k = \mathit{hash0}(\mathrm{DES}_{\mathrm{enc}}(\mathit{id}, \mathcal{K}))$$

Here the DES encryption of *id* with master key $\mathcal{K}$ outputs a cryptogram c of 64 bits. These 64 bits are divided as $c = \langle x, y, z_{[0]}, \dots, z_{[7]} \rangle \in \mathbb{F}_2^8 \times \mathbb{F}_2^8 \times (\mathbb{F}_2^6)^8$ which is used as input to the *hash0* function. This function introduces some obfuscation by performing a number of permutations, complement and modulo operations, see Figure 2.5. Besides that, it checks for and removes patterns like similar key bytes, which could produce a strong bias in the cipher. Finally, the output of *hash0* is the diversified card key $k = k_{[0]}, \dots, k_{[7]} \in (\mathbb{F}_2^8)^8$.

Remark 3. The DES implementation used in iClass is non-compliant with the NIST standard [12] in the way of representing keys. According to the standard, a DES key is of the form $\langle k_0 \dots k_6 p_0, k_7 \dots k_{13} p_1, \dots, k_{47} \dots k_{55} p_7 \rangle$ where $k_0 \dots k_{55}$ are the actual key bits and $p_0 \dots p_7$ are parity bits. Instead, in iClass a DES key is of the form $\langle k_0 \dots k_{55} p_0 \dots p_7 \rangle$.

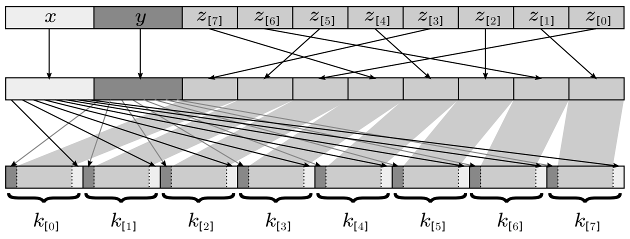

Fig. 2.5. Schematic representation of the function *hash0*

The following sequence of definitions describe the function *hash0* in detail. This function is included here for the sake of completeness. The details over this construction are not necessary to understand the attacks presented in Section 2.7 and Section 3.3.

Definition 8. *Let the function check*: $(\mathbb{F}_2^6)^8 \to (\mathbb{F}_2^6)^8$ *be defined as*

$$check(z_{[0]} \dots z_{[7]}) = ck(3, 2, z_{[0]} \dots z_{[3]}) \cdot ck(3, 2, z_{[4]} \dots z_{[7]})$$

where ck: $\mathbb{N} \times \mathbb{N} \times (\mathbb{F}_2^6)^4 \to (\mathbb{F}_2^6)^4$ *is defined as*

$$\begin{aligned} ck(1, -1, z_{[0]} \dots z_{[3]}) &= z_{[0]} \dots z_{[3]} \\ ck(i, -1, z_{[0]} \dots z_{[3]}) &= ck(i-1, i-2, z_{[0]} \dots z_{[3]}) \\ ck(i, j, z_{[0]} \dots z_{[3]}) &= \begin{cases} ck(i, j-1, z_{[0]} \dots z_{[i]} \leftarrow j \dots z_{[3]}), & z_{[i]} = z_{[j]}; \\ ck(i, j-1, z_{[0]} \dots z_{[3]}), & otherwise. \end{cases} \end{aligned}$$

Definition 9. *Define the function permute*: $\mathbb{F}_2^n \times (\mathbb{F}_2^6)^8 \times \mathbb{N} \times \mathbb{N} \to (\mathbb{F}_2^6)^8$ *as*

$$\begin{aligned} permute(\epsilon, z, l, r) &= \epsilon \\ permute(p_0 \dots p_n, z, l, r) &= \begin{cases} (z_{[l]} + 1) \cdot permute(p_0 \dots p_{n-1}, z, l+1, r), & p_n = 1; \\ z_{[r]} \cdot permute(p_0 \dots p_{n-1}, z, l, r+1), & otherwise. \end{cases} \end{aligned}$$

Definition 10. *Define the bitstring* $\pi \in (\mathbb{F}_2^8)^{35}$ *in hexadecimal notation as*

$$\begin{aligned} \pi =& \mathtt{0x0F171B1D1E272B2D2E333539363A3C474B} \\ & \mathtt{4D4E535556595A5C636566696A6C71727478} \end{aligned}$$

Each byte in this sequence is a permutation of the bitstring `00001111`. *Note that this list contains only the half of all possible permutations. The other half can be computed by taking the bit complement of each element in the list.*

Finally, the definition of *hash0* is as follows.

Definition 11. *Let the function hash0*: $\mathbb{F}_2^8 \times \mathbb{F}_2^8 \times (\mathbb{F}_2^6)^8 \to (\mathbb{F}_2^8)^8$ *be defined as* $hash0(x, y, z_{[0]} \dots z_{[7]}) = k_{[0]} \dots k_{[7]}$ *where*

$$
\begin{aligned}
z'_{[i]} &= (z_{[i]} \bmod (63-i)) + i && i = 0\dots 3\\
z'_{[i+4]} &= (z_{[i+4]} \bmod (64-i)) + i && i = 0\dots 3\\
\hat{z} &= \mathit{check}(z')\\
p &= \begin{cases} \overline{\pi_{[x \bmod 35]}}, \ x_0 = 1;\\ \pi_{[x \bmod 35]}, \ \mathit{otherwise}.\end{cases}\\
\tilde{z} &= \mathit{permute}(p, \hat{z}, 0, 4)\\
k_{[i]} &= \begin{cases} y_i \cdot \overline{\tilde{z}_{[i]}} \cdot p_i + 1, \ y_i = 1;\\ y_i \cdot \tilde{z}_{[i]} \cdot \overline{p_i}, \qquad \mathit{otherwise}.\end{cases} && i = 0\dots 7
\end{aligned}
$$

2.6 Weaknesses

This section describes weaknesses in the design and implementation of iClass that are later exploited in Section 2.7 to mount a key recovery attack.

2.6.1 Weak keys The cipher has a clear weakness when the three rightmost bits of each key byte are the same. Let us elaborate on that.

Proposition 1. *Let β be a bitstring of length three. Then, for all keys $k \in \mathbb{F}_2^{64}$ of the form $k = \alpha_{[0]}\beta \dots \alpha_{[7]}\beta$ with $\alpha_{[i]} \in \mathbb{F}_2^5$ the cipher outputs a constant C_β.*

This is due to the fact that only the three rightmost bits of register r define the output of the cipher and only the rightmost bit of r influences register b. But these, in turn, are only influenced by the three rightmost bits of the key bytes. This means that the 5 leftmost bits of r and the 5 leftmost bits of each key byte affect only the key byte selection, but for the key under consideration this does not affect the output. The same holds for c_C and n_R as they are just input to the $\mathit{select}(\cdot)$ function. Figure 2.6 shows the corresponding MAC value for each possible β. The manufacturer seems to be aware of this feature of the cipher since the function *hash0*, used in key diversification, prevents such a key from being used. Although, this weakness combined with the weakness described in Section 2.6.2 and 2.6.3 result in a vulnerability exploited in Section 2.7.

β	$C_\beta = \text{MAC}(k, c_C \cdot n_R)$
000	`BF 5D 67 7F`
001	`10 ED 6F 11`
010	`53 35 42 0F`
011	`AB 47 4D A0`
100	`F6 CF 43 36`
101	`59 7F 4B 58`
110	`1A A7 66 46`
111	`E2 D5 69 E9`

Fig. 2.6. Corresponding MAC for each value of β

2.6.2 XOR key update weakness In order to update a card key, the iClass reader does not simply send the new key to the card in the clear but instead it sends the XOR of the old and the new key (See Section 2.2). This simple mechanism prevents an attacker from eavesdropping the new key during key update. Although, this key update mechanism introduces a new weakness, namely, it makes it possible to make partial modifications to the existing key. A key update should be an atomic operation. Otherwise, it allows an adversary to split the search space in a time-memory trade-off. Moreover, in case the cipher has some weak keys like the ones described in Section 2.6.1, it allows an adversary to force the usage of one of these keys.

2.6.3 Privilege escalation weakness Several privilege escalation attacks have been described in the literature [10, 24]. The privilege escalation weakness in iClass also concerns the management of access rights over an application within the card. After a successful authentication for application 1 has been executed, the reader is granted read and write access to this application. Then, it is possible to execute a `read` command for a block within application 2 without loosing the previously acquired access rights. More precisely, when a `read` command is issued for a block n within application 2, with $n \neq c_C$, this returns a sequence of 64 ones which indicates that permission is denied to read this block. Surprisingly, this read attempt on application 2 does not affect the previously acquired access rights on application 1. This `read` command though, has the side effect of loading the key $k2$ into the internal state of the cipher. In particular, from this moment on the card accepts `write` commands on application 1 that have a valid MAC computed using key $k2$.

2.6.4 Weak key diversification on iClass The key diversification algorithm of iClass was reverse engineered by Garcia et al. in [15]. This algorithm uses a combination of single DES and a proprietary function called *hash0*, described in Section 2.5. Furthermore, the authors show that the function *hash0* is not one-way nor collision resistant. In fact, it is possible to compute the inverse function $hash0^{-1}$ having a modest amount (on average 4) of candidate pre-images. They also show that once a card key is known, recovering an iClass master key is not harder than a chosen plaintext attack on single DES. After careful inspection of the function *hash0* it becomes clear that this function attempts to fix the weak key weakness presented in Section 2.6.1. The function *hash0* makes sure that, when looking at the last bit of each key byte, exactly four of them are zeros (and the other four of them are ones). Due to this restriction there are only $\frac{8!}{(4!)^2} = 70$ possibilities for the last bits of each key byte, instead of $2^8 = 256$, reducing the entropy of the key by 1.87 bits.

2.7 Key recovery attack on iClass

This section shows how the weaknesses described in Section 2.6 can be exploited. Concretely, we propose an attack that allows an adversary to recover a card key by wirelessly communicating with a card and a reader. Once the card key has been recovered, the weak key diversification weakness described in Section 2.6.4 can be exploited in order to recover the master key. Next, we describe the attack on the card key in detail.

In order to recover a target card key $k1$ from application 1, an attacker A proceeds as follows. First, A eavesdrops a legitimate authentication trace on the e-purse with key $k1$, while making sure that the e-purse is not updated. If the reader attempts to update the e-purse, this can be prevented by playing as man-in-the-middle or by simply jamming the e-purse update message. Next, the adversary replays this authentication trace to the card. At this point the adversary gains read and write access to application 1. Although, in order to actually be able to write, the adversary still needs to send a valid MAC with $k1$ of the payload. To circumvent this problem, the adversary proceeds as described in Section 2.6.3, exploiting the privilege escalation weakness. At this point the adversary still has read and write access to application 1 but he is now able to issue `write` commands using MACs generated with the known key $k2$ to write on application 1. In particular, A is now able to modify $k1$ at will. Exploiting the XOR key update weakness described in Section 2.6.2, the adversary modifies the card key $k1$ into a weak key by setting the three rightmost bits of each key byte the same. Concretely, the adversary runs $2^{3\times7} = 2^{21}$ key updates on the card with $\Delta = 0^5\delta_{[0]} \ldots 0^5\delta_{[6]}0^8 \in \mathbb{F}_2^{64}$ and $\delta_{[i]} = abc \in \mathbb{F}_2^3$ for all possible bits a, b and c. One of these key updates will produce a weak key, i.e., a key of the form $k = \alpha_{[0]}\beta \ldots \alpha_{[7]}\beta$ with $\alpha_{[i]} \in \mathbb{F}_2^5$. Exploiting the weak key weakness described in Section 2.6.1, after each key update A runs 8 authentication attempts, one for each possible value of β, using the MAC values shown in Figure 2.6. Note that a failed authentication will not affect the previously acquired access rights. As soon as an authentication attempt succeeds the card responds with a MAC value that univocally determines β as stated in Proposition 1. Knowing β the adversary is able to recover the three rightmost bits of $k1_{[i]}$ by computing $\beta \oplus \delta_{[i]}$ for $i = 0 \ldots 6$. Furthermore, the three rightmost bits of $k_{[7]}$ are equal to $\beta \oplus 000 = \beta$. In this way, the attacker recovers $3 \times 8 = 24$ bits of $k1$ and only has to search the remaining 40 bits of the key, using the legitimate trace eavesdropped in the beginning.

This attack can be further optimized. The restriction on the last bit of each byte imposed by *hash0*, described at the end of Section 2.6.4, reduces the number of required key updates from 2^{21} to almost 2^{19}. Therefore, it reduces the total number of authentication attempts to $2^{19} \times 8 = 2^{22}$. Once the attacker has recovered the card key $k1$, as we already mention in Section 2.6.4, recovering the master key is just as hard as breaking single DES.

3 iClass Elite

HID introduces iClass Elite (a.k.a. High Security) as the solution for “those who want a boost in security” [8]. iClass Elite aims to solve the obvious limitations of having just one single world-wide master key for all iClass systems. Instead, iClass Elite allows customers to have a personalized master key for their own system. To this purpose, HID has modified the key diversification algorithm, described in Section 2.5 by adding an extra step to it. This modification only affects the way in which readers compute the corresponding card key but does not change anything on the cards themselves. Section 3.1 describes this key diversification algorithm in detail. Then Section 3.2 describes two weaknesses that are later exploited in Section 3.3.

3.1 Key diversification on iClass Elite

This section describes the key diversification algorithm of iClass Elite. We first need to introduce a number of auxiliary functions and then we explain this algorithm in detail.

Definition 12 (Auxiliary functions). *Let us define the bit-rotate left function* $rl\colon \mathbb{F}_2^8 \to \mathbb{F}_2^8$ *as* $rl(x_0 \dots x_7) = x_1 \dots x_7 x_0$. *Similarly, define the bit-rotate right function* $rr\colon \mathbb{F}_2^8 \to \mathbb{F}_2^8$ *as* $rr(x_0 \dots x_7) = x_7 x_0 \dots x_6$. *Furthermore, define the nibble-swap function* $swap\colon \mathbb{F}_2^8 \to \mathbb{F}_2^8$ *as* $swap(x_0 \dots x_7) = x_4 \dots x_7 x_0 \dots x_3$.

Definition 13. *Let the function* $hash1\colon (\mathbb{F}_2^8)^8 \to (\mathbb{F}_2^8)^8$ *be defined as* $hash1(id_{[0]} \dots id_{[7]}) = k_{[0]} \dots k_{[7]}$ *where*

$$\begin{aligned}
k_{[i]} &= k'_{[i]} \bmod 128, & i &= 0 \dots 7\\
k'_{[0]} &= id_{[0]} \oplus \dots \oplus id_{[7]} & k'_{[4]} &= \overline{rr(id_{[4]} \boxplus k'_{[2]})} + 1\\
k'_{[1]} &= id_{[0]} \boxplus \dots \boxplus id_{[7]} & k'_{[5]} &= \overline{rl(id_{[5]} \boxplus k'_{[3]})} + 1\\
k'_{[2]} &= rr(swap(id_{[2]} \boxplus k'_{[1]})) & k'_{[6]} &= rr(id_{[6]} \boxplus (k'_{[4]} \oplus \mathtt{0x3C}))\\
k'_{[3]} &= rl(swap(id_{[3]} \boxplus k'_{[0]})) & k'_{[7]} &= rl(id_{[7]} \boxplus (k'_{[5]} \oplus \mathtt{0xC3}))
\end{aligned}$$

Definition 14. *Define the rotate key function* $rk\colon (\mathbb{F}_2^8)^8 \times \mathbb{N} \to (\mathbb{F}_2^8)^8$ *as*

$$\begin{aligned}
rk(x_{[0]} \dots x_{[7]}, 0) &= x_{[0]} \dots x_{[7]}\\
rk(x_{[0]} \dots x_{[7]}, n+1) &= rk(rl(x_{[0]}) \dots rl(x_{[7]}), n)
\end{aligned}$$

Definition 15. *Let the function* $hash2\colon (\mathbb{F}_2^8)^8 \to (\mathbb{F}_2^{64})^{16}$ *be defined as* $hash2(k_{[0]} \dots k_{[7]}) = y_{[0]} z_{[0]} \dots y_{[7]} z_{[7]}$ *where*

$$\begin{aligned}
z_{[0]} &= \mathrm{DES}_{enc}(\mathcal{K}^{cus}, \overline{\mathcal{K}^{cus}}); & z_{[i]} &= \mathrm{DES}_{dec}(rk(\mathcal{K}^{cus}, i), z_{[i-1]}) & i &= 1 \dots 7\\
y_{[0]} &= \mathrm{DES}_{dec}(z_{[0]}, \overline{\mathcal{K}^{cus}}); & y_{[i]} &= \mathrm{DES}_{enc}(rk(\mathcal{K}^{cus}, i), y_{[i-1]}) & i &= 1 \dots 7
\end{aligned}$$

Next we introduce the selected key. This key is used as input to the standard iClass key diversification algorithm. It is computed by taking a selection of bytes from $hash2(\mathcal{K}^{cus})$. This selection is determined by each byte of $hash1(id)$ seen as a byte offset within the bitstring $hash2(\mathcal{K}^{cus})$.

Definition 16. *Let* $h \in (\mathbb{F}_2^8)^{128}$. *Let* $k^{sel} \in (\mathbb{F}_2^8)^8$ *be the selected key defined as*

$$h := hash2(\mathcal{K}^{cus}); \qquad k^{sel}_{[i]} := h_{[hash1(id)_{[i]}]} \qquad i = 0 \dots 7$$

The last step to compute the diversified card key is just like in iClass (see Section 2.5) $k := hash0(\mathrm{DES}_{enc}(k^{sel}, id))$.

3.2 Weaknesses in iClass Elite

This section describes two weaknesses in the key diversification algorithm of iClass Elite. These weaknesses are exploited in Section 3.3 to mount an attack against iClass Elite that recovers the custom master key.

3.2.1 Redundant key diversification on iClass Elite Assume that an adversary somehow learns the first 16 bytes of $hash2(\mathcal{K}^{cus})$, i.e., $y_{[0]}$ and $z_{[0]}$. Then he can simply recover the master custom key $\mathcal{K}^{cus}$ by computing

$$\mathcal{K}^{cus} = \overline{\mathrm{DES}_{enc}(z_{[0]}, y_{[0]})} .$$

Furthermore, the adversary is able to verify that he has the correct $\mathcal{K}^{cus}$ by checking whether $z_{[0]} = \mathrm{DES}_{enc}(\mathcal{K}^{cus}, \overline{\mathcal{K}^{cus}})$.

3.2.2 Weak key-byte selection on iClass Elite Yet another weakness within the key diversification algorithm of iClass Elite has to do with the way in which bytes from *hash2*$(\mathcal{K}^{cus})$ are selected in order to construct the key k^{sel}.

As described in Section 3.1, the selection of key bytes from *hash2*$(\mathcal{K}^{cus})$ is determined by *hash1*(id). This means that only the card's identity determines which bytes of *hash2*$(\mathcal{K}^{cus})$ are used for k^{sel}. This constitutes a serious weakness since no secret is used in the selection of key bytes at all. Especially considering that, for some card identities, the same bytes of *hash2*$(\mathcal{K}^{cus})$ are chosen multiple times by *hash1*(id). In particular, this implies that some card keys have significantly lower entropy than others. What is even more worrying, an adversary can compute by himself which card identities have this feature.

3.3 Key recovery attack on iClass Elite

In order to recover a master key $\mathcal{K}^{cus}$, an attacker proceeds as follows. First, exploiting the weakness described in Section 3.2.2, the adversary builds a list of chosen card identities like shown in Figure 3.1. This table shows a list of 15 card identities and their corresponding key-byte selection indices *hash1*(id). These card identities are malicious. They are chosen such that the resulting key k^{sel} has very low entropy (in fact, it is possible to find several tables with similar characteristics). For the first card identity in the table, the resulting key k^{sel} is build out of only three different bytes from *hash2*$(\mathcal{K}^{cus})$, namely `0x00`, `0x01` and `0x45`. Therefore, this key has as little as 24 bits of entropy (instead of 56). Next, the adversary will initiate an authentication protocol run with a legitimate reader, pretending to be a card with identity $id =$ `0x000B0FFFF7FF12E0` as in the table. Following the authentication protocol, the reader will return a message containing a nonce n_R and a MAC with k. The adversary will repeat this procedure for each card identity in the table, storing a tuple $< id, n_C, n_R, \text{MAC} >$ for each entry. Afterwards, off-line, the adversary tries all 2^{24} possibilities for bytes `0x00`, `0x01` and `0x45` for the first key identity. For each try, he computes the resulting k and recomputes the authentication run until he finds a MAC equal to the one he got from the reader. Then he has recovered bytes `0x00`, `0x01` and `0x45` from *hash2*$(\mathcal{K}^{cus})$. The adversary proceeds similarly for the remaining card identities from the table. Although, this time he already knows bytes `0x00`, `0x01` and `0x45` and therefore only two bytes per identity need to be explored. This lowers the complexity to 2^{16} for each of the remaining entries in the table. The bytes that need to be explored at each step are highlighted with boldface in the table. At this point the adversary

card identity *id*	*hash1*(id)
`00 0B 0F FF F7 FF 12 E0`	**`01 01 00 00 45 01 45 45`**
`00 04 0E 08 F7 FF 12 E0`	**`78 02`** `00 00 45 01 45 45`
`00 09 0D 05 F7 FF 12 E0`	**`7B 03`** `00 00 45 01 45 45`
`00 0A 0C 06 F7 FF 12 E0`	**`7A 04`** `00 00 45 01 45 45`
`00 0F 0B 03 F7 FF 12 E0`	**`7D 05`** `00 00 45 01 45 45`
`00 08 0A 0C F7 FF 12 E0`	**`74 06`** `00 00 45 01 45 45`
`00 0D 09 09 F7 FF 12 E0`	**`77 07`** `00 00 45 01 45 45`
`00 0E 08 0A F7 FF 12 E0`	**`76 08`** `00 00 45 01 45 45`
`00 03 07 17 F7 FF 12 E0`	**`69 09`** `00 00 45 01 45 45`
`00 3C 06 E0 F7 FF 12 E0`	**`20 0A`** `00 00 45 01 45 45`
`00 01 05 1D F7 FF 12 E0`	**`63 0B`** `00 00 45 01 45 45`
`00 02 04 1E F7 FF 12 E0`	**`62 0C`** `00 00 45 01 45 45`
`00 07 03 1B F7 FF 12 E0`	**`65 0D`** `00 00 45 01 45 45`
`00 00 02 24 F7 FF 12 E0`	**`5C 0E`** `00 00 45 01 45 45`
`00 05 01 21 F7 FF 12 E0`	**`5F 0F`** `00 00 45 01 45 45`

Fig. 3.1. Chosen card identities

has recovered the first 16 bytes of *hash2*($\mathcal{K}^{cus}$). Finally, exploiting the weakness described in Section 3.2.1, the adversary is able to recover the custom master key $\mathcal{K}^{cus}$ with a total computational complexity of 2^{25} DES encryptions.

4 Conclusions

In this paper we have shown that the security of several building blocks of iClass is unsatisfactory. We have found many vulnerabilities in the cryptography and the implementation of iClass that result in two key recovery attacks. Our first attack requires one eavesdropped authentication trace with a genuine reader (which takes about 10ms). Next, the adversary needs 2^{22} authentication attempts with a card, which in practice takes approximately six hours. To conclude the attack, the adversary needs only 2^{40} off-line MAC computations to recover the card key. The whole attack can be executed within a day. For the attack against iClass Elite, an adversary only needs 15 authentication attempts with a genuine reader to recover the custom master key. The computational complexity of this attack is negligible, i.e., 2^{25} DES encryptions. This attack can be executed from beginning to end in less than five seconds. We have successfully executed both attacks in practice and verified the claimed attack times.

This paper reinforces the point that has been made many times: security by obscurity often covers up negligent designs. The built-in key diversification and especially the function *hash0* is advertised as a security feature but in fact it is a patch to circumvent weaknesses in the cipher. The cipher is a basic building block for any secure protocol. Experience shows that once a weakness in a cipher has been found, it is extremely difficult to patch it in a satisfactory manner. Using a well known and community reviewed cipher is a better alternative. The technique described in [28] could be considered as a palliating countermeasure for our first attack. More is not always better: the key diversification algorithm of iClass Elite requires fifteen DES operations more than iClass Standard while it achieves inferior security. Instead, it would have been more secure and efficient to use 3DES than computing 16 single DES operations in an ad hoc manner. NIST have proposed a statistical test suite [29] that can be used to measure the cryptographic strength of a cipher. Although, many weaknesses arise from mistakes in the implementation. Best practice in the development and implementation of security products should incorporate some form of formal verification to prevent that, see for instance [13]. Furthermore, systematic and automated model checking techniques proposed in [31] can help to detect and avoid implementation weaknesses like the privilege escalation in iClass. Alternatively, formalizing the whole design in a theorem prover [3,21] may reveal additional weaknesses. In line with the principles of responsible disclosure, we have notified the manufacturer HID Global and informed them of our findings back in November 2011. Our collaboration and communication with HID Global is ‘open and productive’. HID has established a Product Security Reporting Center to encourage and improve this type of communication.

Acknowledgments

The authors would like to thank Bart Jacobs for his firm support.

References

1. Josep Balasch, Benedikt Gierlichs, Roel Verdult, Lejla Batina, and Ingrid Verbauwhede. Power analysis of Atmel CryptoMemory - recovering keys from secure EEPROMs. In *12th Cryptographers' Track at the RSA Conference (CT-RSA 2012)*, volume 7178 of *Lecture Notes in Computer Science*, pages 19–34. Springer-Verlag, 2012.
2. Alex Biryukov, Ilya Kizhvatov, and Bin Zhang. Cryptanalysis of the Atmel cipher in SecureMemory, CryptoMemory and CryptoRF. In *9th Applied Cryptography and Network Security (ACNS 2011)*, pages 91–109. Springer-Verlag, 2011.
3. Bruno Blanchet. An efficient cryptographic protocol verifier based on prolog rules. In *14th IEEE workshop on Computer Security Foundations (CSFW 2001)*, pages 82–96. IEEE Computer Society, 2001.
4. Andrey Bogdanov. Linear slide attacks on the KeeLoq block cipher. In *Information Security and Cryptology (INSCRYPT 2007)*, volume 4990 of *Lecture Notes in Computer Science*, pages 66–80. Springer, 2007.
5. Inside Contactless. Datasheet PicoPass 2KS, November 2004.
6. Nicolas T. Courtois. The dark side of security by obscurity - and cloning MIFARE Classic rail and building passes, anywhere, anytime. In *4th International Conference on Security and Cryptography (SECRYPT 2009)*, pages 331–338. INSTICC Press, 2009.
7. Nicolas T. Courtois, Sean O'Neil, and Jean-Jacques Quisquater. Practical algebraic attacks on the Hitag2 stream cipher. In *12th Information Security Conference (ISC 2009)*, volume 5735 of *Lecture Notes in Computer Science*, pages 167–176. Springer-Verlag, 2009.
8. Nathan Cummings. iClass levels of security, April 2003.
9. Nathan Cummings. Sales training. Slides from HID Technologies, March 2006.
10. Lucas Davi, Alexandra Dmitrienko, Ahmad-Reza Sadeghi, and Marcel Winandy. Privilege escalation attacks on Android. In *13th Information Security Conference (ISC 2010)*, volume 6531, pages 346–360. Springer-Verlag, 2011.
11. Gerhard de Koning Gans, Jaap-Henk Hoepman, and Flavio D. Garcia. A practical attack on the MIFARE Classic. In *8th Smart Card Research and Advanced Applications Conference (CARDIS 2008)*, volume 5189 of *Lecture Notes in Computer Science*, pages 267–282. Springer-Verlag, 2008.
12. PUB FIPS. 46-3, Data Encryption Standard (DES). *National Institute for Standards and Technology (NIST), Gaithersburg, MD, USA*, 1999.
13. Riccardo Focardi and Flaminia L. Luccio. Secure recharge of disposable RFID tickets. In *8th International Workshop on Formal Aspects of Security and Trust (FAST 2011)*, volume 7140, pages 85–99. Springer-Verlag, 2012.
14. Flavio D. Garcia, Gerhard de Koning Gans, Ruben Muijrers, Peter van Rossum, Roel Verdult, Ronny Wichers Schreur, and Bart Jacobs. Dismantling MIFARE Classic. In *13th European Symposium on Research in Computer Security (ESORICS 2008)*, volume 5283 of *Lecture Notes in Computer Science*, pages 97–114. Springer-Verlag, 2008.
15. Flavio D. Garcia, Gerhard de Koning Gans, and Roel Verdult. Exposing iClass key diversification. In *5th USENIX Workshop on Offensive Technologies (USENIX WOOT 2011)*, pages 128–136. USENIX Association, 2011.
16. Flavio D. Garcia, Peter van Rossum, Roel Verdult, and Ronny Wichers Schreur. Wirelessly pickpocketing a MIFARE Classic card. In *30th IEEE Symposium on Security and Privacy (S&P 2009)*, pages 3–15. IEEE Computer Society, 2009.

17. Flavio D. Garcia, Peter van Rossum, Roel Verdult, and Ronny Wichers Schreur. Dismantling SecureMemory, CryptoMemory and CryptoRF. In *17th ACM Conference on Computer and Communications Security (CCS 2010)*, pages 250–259. ACM/SIGSAC, 2010.
18. HID Global. HID management key letter, November 2006.
19. HID Global. iClass RW100, RW150, RW300, RW400 readers, 2009.
20. Identification cards – contactless integrated circuit(s) cards – vicinity cards (ISO/IEC 15693), 2000. International Organization for Standardization (ISO).
21. Bart Jacobs and Ronny Wichers Schreur. Logical formalisation and analysis of the MIFARE Classic card in PVS. In *2nd International Conference on Interactive Theorem Proving*, volume 6898 of *Lecture Notes in Computer Science*, pages 3–17. Springer-Verlag, 2011.
22. Markus Kasper, Timo Kasper, Amir Moradi, and Christof Paar. Breaking KeeLoq in a flash: on extracting keys at lightning speed. In *2nd International Conference on Cryptology in Africa, Progress in Cryptology (AFRICACRYPT 2009)*, volume 5580 of *Lecture Notes in Computer Science*, pages 403–420. Springer-Verlag, 2009.
23. ChangKyun Kim, Eun-Gu Jung, Dong Hoon Lee, Chang-Ho Jung, and Daewan Han. Cryptanalysis of INCrypt32 in HID's iClass systems. Cryptology ePrint Archive, Report 2011/469, 2011.
24. Tadayoshi Kohno, Adam Stubblefield, Aviel D. Rubin, and Dan S. Wallach. Analysis of an electronic voting system. In *25th IEEE Symposium on Security and Privacy (S&P 2004)*, pages 27–40. IEEE Computer Society, 2004.
25. Milosch Meriac. Heart of darkness - exploring the uncharted backwaters of HID iClass security. Technical report, Bitmanufaktur GmbH, December 2010. Presentation at the 27th Chaos Computer Congress (27C3).
26. Karsten Nohl, David Evans, Starbug, and Henryk Plötz. Reverse engineering a cryptographic RFID tag. In *17th USENIX Security Symposium (USENIX Security 2008)*, pages 185–193. USENIX Association, 2008.
27. Henryk Plötz and Karsten Nohl. Peeling away layers of an RFID security system. In *Financial Cryptography and Data Security (FC 2012)*, volume 7035 of *Lecture Notes in Computer Science*, pages 205–219. Springer-Verlag, 2012.
28. Amir Rahmati, Mastooreh Salajegheh, Dan Holcomb, Jacob Sorber, Wayne P. Burleson, and Kevin Fu. TARDIS: Time and remanence decay in SRAM to implement secure protocols on embedded devices without clocks. In *21st USENIX Security Symposium (USENIX Security 2012)*. USENIX Association, 2012.
29. Andrew Rukhin, Juan Soto, James Nechvatal, Miles Smid, Elaine Barker, Stefan Leigh, Mark Levenson, Mark Vangel, David Banks, Alan Heckert, James Dray, and San Vo. A statistical test suite for the validation of random number generators and pseudo random number generators for cryptographic applications. *NIST Special Publication*, pages 800–822, 2001.
30. Mate Soos, Karsten Nohl, and Claude Castelluccia. Extending SAT solvers to cryptographic problems. In *12th International Conference on Theory and Applications of Satisfiability Testing (SAT 2009)*, volume 5584 of *Lecture Notes in Computer Science*, pages 244–257. Springer-Verlag, 2009.
31. Jan Tretmans. Model based testing with labelled transition systems. In *Formal Methods and Testing*, volume 4949 of *Lecture Notes in Computer Science*, pages 1–38. Springer-Verlag, 2008.
32. Roel Verdult, Flavio D. Garcia, and Josep Balasch. Gone in 360 seconds: Hijacking with Hitag2. In *21st USENIX Security Symposium (USENIX Security 2012)*. USENIX Association, 2012.