# Fuming Acid and Cryptanalysis: Handy Tools for Overcoming a Digital Locking and Access Control System

**(Full Version)**

Daehyun Strobel, Benedikt Driessen, Timo Kasper, Gregor Leander, David Oswald, Falk Schellenberg, and Christof Paar

Horst Görtz Institute for IT-Security
Ruhr-University Bochum, Germany

**Abstract.** We examine the widespread SimonsVoss digital locking system 3060 G2 that relies on an undisclosed, proprietary protocol to mutually authenticate transponders and locks. For assessing the security of the system, several tasks have to be performed: By decapsulating the used microcontrollers with acid and circumventing their read-out protection with UV-C light, the complete program code and data contained in door lock and transponder are extracted. As a second major step, the multi-pass challenge-response protocol and corresponding cryptographic primitives are recovered via low-level reverse-engineering. The primitives turn out to be based on DES in combination with a proprietary construction.

Our analysis pinpoints various security vulnerabilities that enable practical key-recovery attacks. We present two different approaches for unauthorizedly gaining access to installations. Firstly, an attacker having physical access to a door lock can extract a master key, allowing to mimic transponders, in altogether 30 minutes. A second, purely logical attack exploits an implementation flaw in the protocol and works solely via the wireless interface. As the only prerequisite, a valid ID of a transponder needs to be known (or guessed). After executing a few (partial) protocol runs in the vicinity of a door lock, and some seconds of computation, an adversary obtains all of the transponder's access rights.

**Key words:** Access control, electronic lock, reverse-engineering, real-world attack, hardware attack, cryptanalysis, wireless door openers

## 1 Introduction

Despite the fact that nowadays strong and well-analyzed cryptographic primitives are available for a large variety of applications, very weak cryptographic algorithms are still widely deployed in real products all over the world. Examples include algorithms like Keeloq or the Crypto1 cipher used in the Mifare Classic cards. It is very surprising how big a gap between cryptographic theory on the one hand and cryptographic protocols in real products on the other hand exists and how real-life products have their security mainly based on obscurity and not on cryptographically sound protocols and primitives.

In this paper, we add one more interesting example to the list of widely deployed ciphers that have severe design flaws. Our hope is that the presented findings contribute to the science of building more secure wireless systems.

## 1.1 (Digital) Locking Systems and Wireless Technology

For many decades, purely mechanical keys and locks were the only means for securing the access to buildings, rooms, cars, and other property. Starting in the 1950s, the first Remote Keyless Entry (RKE) systems were available on the market to open doors from a distance via a Radio Frequency (RF) interface. These "fixed-code" systems provided no cryptographic protection and could easily be circumvented by means of a replay attack. In the 1980s, manufacturers started to equip cars with this type of wireless door opener on a large scale. After the number of stolen cars rose, it became clear that the new wireless comfort came at the price of reduced security and that more elaborate authentication schemes were required to prevent theft. Combining the benefits of modern wireless technology and cryptography, a new era of access control systems began: Immobilizers and more secure RKE systems were invented and today, all new cars are furnished with remote controls incorporating cryptography, while purely mechanical keys have almost vanished from the market.

Recently, a similar trend can be observed for the access control to buildings. While mechanical keys and locks are still widespread, they suffer from certain disadvantages. For example, keys can often easily be copied and if a key gets lost or stolen, all affected door locks have to be replaced. Also, mechanical locks only allow a rudimentary type of access management. The demand for a flexible assignment of keys to locks and vice versa paved the way for augmenting mechanical locks with wireless technology and replacing mechanical keys by electronic counterparts, e. g., transponders or smartcards. In case of loss or theft, administrators can simply block affected transponders in a database.

Despite all these comforts and benefits, wireless communication implies an increase in attack surface: A transponder residing in a pocket or wallet could be read out or modified without the owner taking note of it. Moreover, the transmission of data via the RF interface can be monitored from a distance. Hence, wireless access control systems require protecting the over-the-air interface with additional security measures.

## 1.2 Related Work

In the world of access control by electronic means, various manufacturers have been inventing their own cryptographic primitives and protocols, often with low-cost properties and established "security" by keeping the details secret. The past decade has shown that the vast majority of these schemes is flawed and that once the ciphers have been reverse-engineered and become public, they can be broken with low to modest efforts.

One of the first examples is the DST40 cipher. It is used in Texas Instrument's Digital Signature Transponder (DST) and has been reverse-engineered in 2005 [1]: Knowing at least two challenge/response pairs, the 40-bit secret key of a corresponding transponder can be revealed by means of a brute-force attack in less than one day. Likewise, following the reverse-engineering of NXP's Mifare Classic cards [2] through analyzing the silicon die, the used Crypto1 cipher was found to be weak, relying on a state of only 48 bits. Further mathematical weaknesses of the cipher and implementations flaws, e. g., a weak random number generator, enable to reveal all secret keys and practically circumvent the protection mechanisms with a card-only attack in minutes [3–5]. The Hitag 2 transponders of the same manufacturer, widely used for car immobilizers—but also for RKE systems—were found to be flawed after the cipher became public [6]. Based on the latest results [7], their secret keys can be extracted in six minutes. Further insecure products for access control include HID Global iClass [8] and Legic Prime cards, both based on highly ineffective cryptographic measures [9].

Practically exploiting the vulnerabilities of the above products typically requires to be at least in the vicinity of the targets (cars, cards, card readers in the buildings, etc.). In contrast, attacking the RKE system KeeLoq is feasible from a larger distance: After the cipher became public, mathematical weaknesses were found [10, 11] and—after performing a side-channel attack to obtain the master key of the system—duplicating remote controls is feasible by means of eavesdropping from several hundred meters [12].

### 1.3 SimonsVoss Digital Locking and Access Control System 3060

One large manufacturer of digital locking systems for buildings is the Germany-based company SimonsVoss Technologies AG. SimonsVoss, the European market leader for electronic locking and access control systems [13], installed its one millionth digital locking cylinder in April 2012 and has sold more than three million corresponding transponders. The list of customers and objects secured with this technology in Europe, USA, and Asia, as listed on the official website [14], is very impressive: It includes residential buildings, tourist apartments, hospitals, universities, embassies, major banks, airports, buildings of the German armed forces and the US army, factory sites of well-known brands, police stations, stadiums, town halls, prisons, insurances, and many others.

One part of the system, termed transponder, serves as a substitute for a mechanical key. It is a battery-powered remote control that, upon pressing a button, activates the second part of the system, an electronically enhanced cylinder. The cylinder that is integrated into the door has the same dimensions as a standardized mechanical locking cylinder. If successfully activated, the door cylinder beeps twice, indicating that the lock can be opened or closed during the next few seconds (with manual force, by turning a knob that is attached to the cylinder).

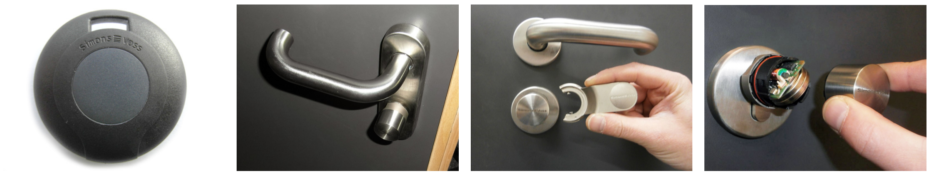

(a) Transponder 3064 (b) Door lock cylinder 3061 (c) Battery replacement tool (d) Opened lock knob

**Fig. 1.** Components of the digital locking system

The digital cylinder is also powered from batteries. In case of worn-out batteries after a few years of operating time or exceptional operating conditions[1], they can be replaced after dismantling the knob of the cylinder with the commercially available "battery replacement tool" (cf. Fig. 1c). The electronics are contained inside the knob of one side of the door cylinder, as illustrated in Fig. 1d, while the other knob is usually empty. The correct side for a battery change can be visually identified by a small, black plastic ring that is visible between the cap of the knob and the metal parts of the cylinder. This knob should be installed inside the buildings or inside the offices to prevent unauthorized access to the electronics.

The widespread digital locking system 3060 analyzed in this paper is based on a so far undisclosed, proprietary cryptographic protocol. The latest revision is termed "Generation 2" or "G2-based" system by the manufacturer. It supports up to 64,000 digital locking cylinders 3061 (cf. Fig. 1b) per installation, up to 64,000 transponders 3064 (cf. Fig. 1a) per lock, and the storage of up to 1,000 access instances on the transponder. The cost of a transponder 3064 is approx. \$ 40 and that of a locking cylinder 3061 approx. \$ 440.

The back-end of larger installations is realized as a software running on a standard PC, allowing to configure all door locks of an installation via a wireless link, e. g., in order to re-program the door locks. Likewise, transponders can be programmed to enable access to certain doors by means of the functionality of the back-end. In general, the "Generation 2" system enables to form a network of door cylinders, transponders, and the back-end by means of various communication techniques, e. g., through Ethernet or through multiple routers and nodes (cf. [15]). Small installations can also be configured offline, using the programming transponder 3067.

The door locks can have a permanent connection to a central server through multiple wireless access points ("WaveNet router nodes") at 868 MHz. These router nodes connect to so-called lock nodes [16] placed within the door knob. Lock nodes in turn communicate over a single wire with the circuitry responsible for opening the lock and the 25 kHz connection to the transponders. The (successful or unsuccessful) opening attempt of any transponder at any door lock in

[1] The batteries need to be replaced in intervals of up to 10 years or 150,000 door openings according to the information of the manufacturer.

the system can be monitored and stored in logfiles. In case of an electrical power outage or when communication between door locks and back-end is interrupted, doors remain fully functional [17, p.5]. The security of the back-end and the wireless link are explicitly not analyzed in this paper. All our findings in the following are solely based on analyzing the door locks and transponders.

### 1.4 Contribution and Outline

The security level of access control systems relying on the obsolete ciphers mentioned in Sect. 1.2 has already been evaluated and in most cases has been found to be very low. However, to the best of our knowledge, the security of the widespread SimonsVoss digital locking system 3060 "Generation 2" and its proprietary, undisclosed schemes for encryption, authentication, and key derivation has not been publicly evaluated yet. The aim of this paper is to close this gap and analyze the security of this system.

By eavesdropping the communication between transponder and lock, it quickly became clear that the protocol is rather involved, with each protocol run consisting of 11 messages being exchanged. Moreover, as non-trivial computations are executed, extracting the details of the protocol by eavesdropping only seemed out of reach. Thus, more invasive methods were needed to advance at this step: In Sect. 2, we reverse-engineer the hardware and software of transponders and digital cylinders, by means of decapsulating chips with acid, circumventing read-out protections with UV-C light, and analyzing the internals with a microscope and disassembler. As a result, the proprietary authentication scheme is disclosed in Sect. 3 and all details about the proprietary cryptographic primitives and the key derivation mechanism are given in Sect. 4. Compared to various antiquated access control systems (cf. Sect. 1.2), the SimonsVoss realization at a first glance appeared to provide an adequate security level, since a slightly changed version of the Data Encryption Standard (DES) combined with a proprietary obscurity function is applied.

The next step of our work thus consists in cryptanalyzing the cipher and the protocol (cf. Sect. 5). Most surprisingly, due to a crucial flaw in the protocol, the (modified) DES can be circumvented completely. This allowed us to mainly focus on the proprietary obscurity function. After our detailed analysis it turns out that this function can be seen as a (generalized) T-function (cf. [18]), which is the key to invert (parts of) the obscurity function very efficiently.

Our work finally resulted in very practical attacks which enable opening the doors secured by the analyzed system, as illustrated in Sect. 5: An adversary possessing an ID of an valid transponder (for instance obtained by eavesdropping, exhaustive search, or reading it out from a transponder or door lock) simply has to execute a few (partial) protocol runs in the vicinity of a door lock to obtain all access rights the respective genuine transponder possesses. The attack works solely via the wireless interface, thus leaves no traces, and does not require physical access to a valid transponder or lock.

## 2 Reverse-Engineering

When initially analyzing the SimonsVoss System 3060, we were facing a complete black-box, i. e., had no information on the inner workings of the system. From publicly available information, little can be learned about the actual implementation. Hence, we decided to obtain the necessary knowledge for our security analysis by reverse-engineering the involved components. The Printed Circuit Boards (PCBs) of transponder and door (cf. Fig. 2a) have a similar layout containing three main components that are involved in the authentication process: A SimonsVoss-proprietary Application Specific Integrated Circuit (ASIC) is connected to a Microchip PIC16F886 Microcontroller ($\mu$C) [19]. The third component is an external Electrically Erasable Programmable Read-Only Memory (EEPROM) controlled by the $\mu$C over an Inter-Integrated Circuit ($I^2C$) bus [20]. In this section, we summarize the results of reverse-engineering the functionality of these components.

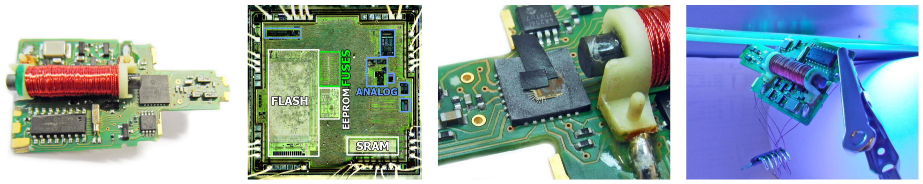
### 2.1 Reverse-Engineering the Proprietary ASIC

Since we could not obtain any documents regarding the functionality of the ASIC, we decided to analyze the device on the level of the silicon die. To this end, we decapsulated several ASICs using White Fuming Nitric Acid (WFNA) according to the two-step procedure as described in [21, p.10] and took high-resolution pictures of the die with an optical microscope. We found that the ASIC employs a $2\,\mu$m gate array design with a total number of 2320 transistors available for CMOS logic. In consequence, the amount of logic that can be implemented is rather limited and insufficient to, e. g., realize cryptographic algorithms. After reverse-engineering most of the digital circuits of the ASIC we found out that the main functions of the ASIC are (1) to implement functions to wake up the $\mu$C periodically and (2) to work as a (de)modulator for the RF transmission.

### 2.2 Reading out the Firmware of the PIC16F886

Having found that the ASIC is not related to the security-relevant parts of the system, it can be assumed that all (cryptographic) functionality is implemented in the firmware of the Microchip PIC16F886 [19] $\mu$C. Like many common $\mu$Cs, this PIC stores its firmware in an internal flash memory. Moreover, the $\mu$C contains an internal EEPROM for storing 256 bytes of user-defined data. In order to protect the firmware and the content of the EEPROM, SimonsVoss enabled the read protection fuse.

After unsuccessful attempts to clear the respective configuration bits using power glitches during the programming operation, we considered a different method: In [22], the author successfully cleared the configuration bits of a PIC18F1330, i. e., changed the state from `0` to `1` by applying Ultraviolet-C (UV-C) light in a certain angle to the decapsulated chip. The idea is that even if the fuses are covered with a (small) metal plate as a shield, UV-C light will bounce off from various parts around the metal plate and the plate itself, eventually hitting the cells storing the fuse bit. In [23], the author confirmed that the attack worked for a PIC12F683 as well.



(a) PCB (front) of the door part (b) PIC16F886 annotated (analog to [23]) (c) PCB after decapsulation, memories covered with tape (d) Exposure to UV-C light of an EPROM erasure tool

**Fig. 2.** PCB of the door circuitry, location of the security bits (fuses) and erasing fuses of the PIC16F886

Although the PIC16F886 comes in a relatively new type of package (QFN), Microchip did not address this issue and leaves the UV-C attack still possible. We decapsulated the $\mu$C with WFNA and used an EPROM erasure tool [24] as our UV-C source (cf. Fig. 2d). After testing various positions and angles, we found that all non-volatile memories and the configuration bits were erased after an exposure of about 20 min. By applying this technique exclusively to the configuration bits, i. e., by covering the non-volatile memories with electrical isolation tape (cf. Fig. 2b and 2c), we were able to recover the complete firmware (stored in the internal Flash memory) and the contents of the internal EEPROM of several transponders and door lock $\mu$Cs.

In order to disassemble and understand the code running on the $\mu$C of both the transponder and the door lock, we utilized IDA Pro [25], a tool often employed for the analysis of regular PC-software. Nevertheless, IDA Pro also includes a module for PIC $\mu$Cs which greatly aided in the reverse-engineering process. In addition to performing a static analysis of the program code with IDA Pro, we also inserted debug routines into the disassembled code. This routine allows to dump the registers and the SRAM during the execution of the program on the original transponder or door PCB over an unused pin of the $\mu$C. Thus, being able to dump the memory contents, e. g., during the execution of a successful authentication protocol run, we were able to verify the results of the static analysis and to understand parts of the code that heavily depend on external input (e. g., from the external flash).

## 3 Authentication Keys & Protocol

In the following, we present the essential results of reverse-engineering the software running on the transponder and the lock. We focus on the keys, protocol, and the cryptographic primitives used to mutually authenticate transponders and locks.

For a successful execution of the authentication protocol, transponder and lock must be in possession of a shared secret. For this purpose, each transponder has a (unique) 128-bit long-term secret $K_T$. This key is computed from a 128-bit value $K_{T,\mathrm{ext}}$ stored in the external EEPROM and a 128-bit key $K_{T,\mathrm{int}}$ stored in the PIC's internal EEPROM as

$$K_T = K_{T,\mathrm{ext}} \oplus K_{T,\mathrm{int}}.$$

On the other hand, each lock stores a set of four 128-bit keys $K_{L,j}$ that are *identical for every lock in the entire installation.* Analogous to the transponder's key, one of these keys $K_{L,j}$ is the XOR of a key $K_{L,j,\mathrm{ext}}$ stored in the external EEPROM with one 128-bit internal key $K_{L,\mathrm{int}}$ stored in the internal EEPROM, i. e.,

$$K_{L,j} = K_{L,j,\mathrm{ext}} \oplus K_{L,\mathrm{int}}$$

with $0 \leq j \leq 3$. In the following, we refer to the set of the keys $K_{L,j}$ as the *system key.* Based on this key, the lock can derive any transponder key, as will be explained in the course of the protocol.

The system uses an 11-step challenge-response protocol to achieve mutual authentication between transponder and lock. A protocol run is initiated by the transponder when the central button is pressed in proximity of a lock. In the course of this authentication step, a multitude of messages is exchanged, cf. Fig. 3.

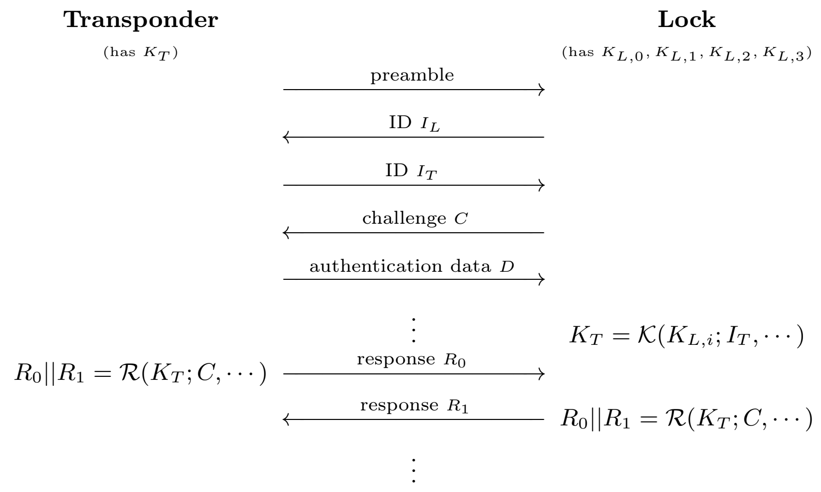

**Transponder**
(has $K_T$)
**Lock**
(has $K_{L,0}, K_{L,1}, K_{L,2}, K_{L,3}$)
preamble
ID $I_L$
ID $I_T$
challenge $C$
authentication data $D$
$K_T = \mathcal{K}(K_{L,i}; I_T, \cdots)$
$R_0 || R_1 = \mathcal{R}(K_T; C, \cdots)$
response $R_0$
response $R_1$
$R_0 || R_1 = \mathcal{R}(K_T; C, \cdots)$

**Fig. 3.** Protocol for the mutual authentication between a transponder and a lock

Here we only focus on a subset of these messages that we identified to be relevant for our security analysis:

- $I_L$ Each lock has a 24-bit ID that is transmitted to the transponder.
- $I_T$ Each transponder has a 32-bit ID that is transmitted to the door.
- $C$ After the ID exchange, the lock sends an 88-bit challenge $C$ to the transponder.
- $D$ The transponder sends 80 bits of authentication data to the lock. This includes the result of its ID verification and transponder-specific data.

In the first four steps of the protocol, most of the messages are fixed for a transponder/lock combination. Only $C$ (and conversely the responses $R_0$ and $R_1$) change between protocol runs—and of this 88-bit value, only 40-bit are actually random. The remaining bits are either fixed or change infrequently. In answer to such a challenge, both transponder and lock derive the same 64-bit response $R$ using the data exchanged in previous messages and the 128-bit long-term secret $K_T$ of the transponder. We denote the function to compute the response as $\mathcal{R}$, with

$$R = R_0||R_1 = \mathcal{R}(K_T; I_L, I_T, C, D).$$

The main part of the authentication is then accomplished by exchanging the following two messages:

- $R_0$ The transponder sends the first 32-bit half of $R$ as the first response to the lock.
- $R_1$ If $R_0$ matches the first half of $R$ computed by the lock, it sends the second 32-bit half of $R$ to the transponder.

As each party computes the full 64-bit output of $\mathcal{R}$, both can verify the response of the other party and mutually authenticate each other on the basis of $K_T$. Instead of storing the key $K_T$ for each transponder, a lock is able to derive $K_T$ from a transponder's ID $I_T$, the authentication data $D$, and part of the long-term system key. A key derivation function $\mathcal{K}$ is used for this purpose, i. e.,

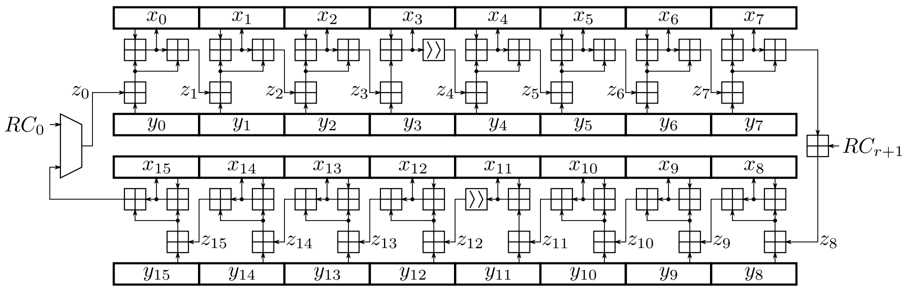
$$K_T = \mathcal{K}(K_{L,j}; I_T, D)$$

with $0 \leq j \leq 3$.

## 4 Cryptographic Primitives

In the authentication protocol, the two basic functions $\mathcal{K}$ (for key derivation on the door's side) and $\mathcal{R}$ (for response computation) are used. These functions are proprietary constructions and share two building blocks we denote as $\mathcal{O}$ and $\mathcal{D}$. While it turned out that $\mathcal{D}$ is simply a modified DES [26], what we call "the obscurity function" $\mathcal{O}$ is a more intricate design, which we are describing in the following.

The $\mathcal{O}$ function takes two 128-bit inputs (a plaintext and a key) and returns a 128-bit output. Figure 4 shows the internal structure of $\mathcal{O}$. This function operates byte-wise on two registers with 16 8-bit cells. The upper registers are continuously updated while the lower registers remain constant.



$x_0$ $x_1$ $x_2$ $x_3$ $x_4$ $x_5$ $x_6$ $x_7$
$z_0$ $z_1$ $z_2$ $z_3$ $z_4$ $z_5$ $z_6$ $z_7$
$RC_0$
$y_0$ $y_1$ $y_2$ $y_3$ $y_4$ $y_5$ $y_6$ $y_7$
$RC_{r+1}$
$x_{15}$ $x_{14}$ $x_{13}$ $x_{12}$ $x_{11}$ $x_{10}$ $x_9$ $x_8$
$z_{15}$ $z_{14}$ $z_{13}$ $z_{12}$ $z_{11}$ $z_{10}$ $z_9$ $z_8$
$y_{15}$ $y_{14}$ $y_{13}$ $y_{12}$ $y_{11}$ $y_{10}$ $y_9$ $y_8$

**Fig. 4.** Structure of the obscurity function

To compute the output of $\mathcal{O}$ the upper $x$ registers are initialized with the plaintext, while the lower $y$ registers are set to the key. After that, the registers are updated successively, all in all each of the $x$ registers is updated for 8 times, according to the following scheme: Updates start with $x_0$, then $x_1$, etc., and are computed mostly as sums of 8-bit values modulo 256. Additionally, each cell update incorporates an 8-bit chaining value $z_i$, which is the result of the update of the preceding cell. The update equation for the successive state $x'_i$ from $x_i$ is given as $x'_i = y_i + z_i + x_i \bmod 2^8$. There are basically three ways the chaining value is computed for any round $r$ with $0 \leq r \leq 7$:

$$z_{i+1} = \begin{cases} (x_i + 2(y_i + z_i)) \bmod 2^8 & \text{if } i \in \{0, 1, ..., 14\} \setminus \{3, 7, 11\}, \\ (x_i + 2(y_i + z_i) + RC_{r+1}) \bmod 2^8 & \text{if } i = 7, \\ \big((y_i + z_i \bmod 2^8) + x_i >> 1\big) \bmod 2^8 & \text{if } i \in \{3, 11\}. \end{cases}$$

The computation of the first value $z_0$ is different: Initially, it is set to $RC_0$ and then, for the next round of cell updates it is computed as

$$z_0 = (x_{15} + 2(y_{15} + z_{15})) \bmod 2^8.$$

Here, all $RC_r$ with $0 \leq r \leq 8$ are 8-bit round constants, which we do not disclose at this time. The function's output is given by the contents of the $x$ cells after 8 rounds.

## 4.1 $\mathcal{K}$: Key Derivation Function

In the following we describe how $\mathcal{D}$ and $\mathcal{O}$ are combined to construct the key derivation function, which is used only in the door. This function can be decomposed into three blocks, $\mathcal{D}$ and two instances of the aforementioned obscurity function $\mathcal{O}$, as illustrated in Fig. 5.

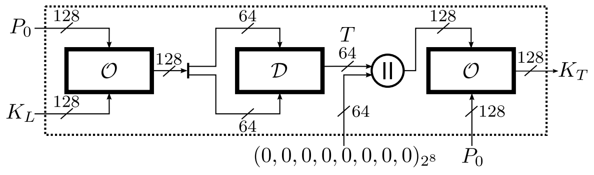


**Fig. 5.** Construction of the key derivation function

The construction has four inputs, one fixed to 64 zero bits and two other 128-bit inputs that are exchanged in the authentication protocol: The value $P_0$, used twice during key derivation, is composed of the first three bytes $I_{T,0}, I_{T,1}, I_{T,2}$ of the transponder ID $I_T$ and the first three bytes $D_0, D_1, D_2$ of the authentication data $D$. The last of each of these three bytes is masked by a Boolean AND-operation with the fixed constant `0xC7` or `0x3F`, respectively, thus selecting only certain bits. All other bytes are filled with zeros, i. e.,

$$P_0 = (I_{T,0}, I_{T,1}, I_{T,2} \,\&\, \texttt{0xC7}, D_0, D_1, D_2 \,\&\, \texttt{0x3F}, 0, \ldots, 0)_{2^8}.$$

Only one input of $\mathcal{K}$ is secret: One of the four 128-bit keys of the system key set is selected according to the two most significant bits of the third byte of $I_T$. This 128-bit key is used as key $K_L$ for the first instance of $\mathcal{O}$ to encrypt $P_0$. The output of this operation is split into two 64-bit halves which are used as plaintext and key for $\mathcal{D}$. The output of $\mathcal{D}$, denoted by $T$, is then concatenated with 64 zero bits, and the result is encrypted with $\mathcal{O}$—using $P_0$ as the key. The resulting 128-bit value is the transponder's key $K_T$, i. e.,

$$\mathcal{K}(K_L; P_0) = \mathcal{O}\Big(P_0; \mathcal{D}\Big(\mathcal{O}(K_L; P_0)_{64..127}; \mathcal{O}(K_L; P_0)_{0..63}\Big)||0\ldots0\Big).$$

## 4.2 $\mathcal{R}$: Response Computation Function

The structure of the response computation $\mathcal{R}$ is very similar to the key derivation function $\mathcal{K}$. However, the way the building blocks are combined is different. Figure 6 shows the internal structure of $\mathcal{R}$.

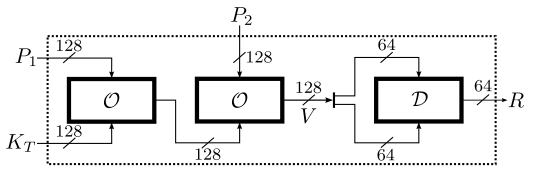


**Fig. 6.** Construction of the response computation function

Again two instances of the proprietary obscurity function $\mathcal{O}$ are used along with the modified DES $\mathcal{D}$. The 128-bit input $P_1$ to $\mathcal{R}$ is the concatenation of the challenge $C$ and part of the authentication data $D$, i. e.,

$$P_1 = (C_0, C_1, ..., C_{10}, D_6, D_7, D_8, D_9, 0)_{2^8}.$$

The output of the first instance of $\mathcal{O}$ is used as key for the second iteration of $\mathcal{O}$. The 128-bit input $P_2$ is fixed for every transponder/lock combination and is composed of more bytes taken from the IDs of lock and door, i. e.,

$$P_2 = (I_{L,2}, I_{T,2}, I_{T,3}, D_3, D_4, D_5, 0, ..., 0)_{2^8}.$$

The output of this operation is split into two 64-bit halves, whereas the first half is used as plaintext for $\mathcal{D}$ and the second as the respective key. The two halves of the 64-bit result $R$ form the responses $R_0$ and $R_1$ used in the protocol, i. e.,

$$\mathcal{R}(K_T; P_1, P_2) = \mathcal{D}\Big(\mathcal{O}\Big(\mathcal{O}(K_T; P_1); P_2\Big)_{64..127}; \mathcal{O}\Big(\mathcal{O}(K_T; P_1); P_2\Big)_{0..63}\Big)$$

## 5 Attacks

Having revealed the inner structure of the authentication protocol, we now introduce two types of attacks. The physical attacks exploit weaknesses of the platform that is used for cryptographic purposes and the basic system design. The cryptanalytical attack goes more into details of the implemented cryptographic structure, especially of the obscurity function $\mathcal{O}$.

All presented attacks have been verified at several installations of the SimonsVoss 3060 G2 system.

### 5.1 Physical Attacks

The attacks presented in the following are of invasive nature and assume physical access to either a transponder or a digital cylinder.

**Attack 1: Invasive Cloning of a Transponder** As presented in Sect. 2, the program and internal EEPROM memories of the $\mu$C can be read after clearing the respective fuse bit with UV-C light after decapsulation. The external EEPROM is unprotected and can thus be read out trivially. Since the proprietary ASIC does not contain, e. g., a unique identifier, it is possible to copy one transponder to another by writing the respective memories. The whole process of duplicating a transponder takes less than 30 minutes, including the decapsulation of the $\mu$C. Note that the memory protection fuse bits can also be reset by performing a full chip-erase of the PIC16F886. Of course, this also deletes the program memory and the internal EEPROM and hence cannot be used for reading out the respective contents. However, this characteristic is handy for an adversary when cloning a transponder. Instead of building a custom “emulation” device, an original SimonsVoss transponder can be fully erased and then reprogrammed with the contents of the transponder to be duplicated. It should be noted that this attack does not require an adversary to understand the program code or the contents of the EEPROMs at all. In particular, this means that no knowledge of the cryptographic details of the protocol is necessary.

**Attack 2: Cloning using the System Key** Analyzing the key derivation scheme described in Sect. 4, it is obvious that the system key is a single-point-of-failure. Given the system key together with a valid transponder ID, the key for the respective transponder can be generated. Apart from the system key, all other inputs to the key derivation are—by design—publicly known. Since the system key is stored in every door lock, only one lock PCB has to be in the hands of the adversary temporarily. The PCB could be for instance removed from a door that is rarely locked or that is accessible from the outside, e. g., a door of a main entrance[2]. A "battery replacement" tool to remove the metal casing of the lock is publicly available. Note that these doors are very likely to contain the valid IDs of transponders with a system-wide validity, as required for emergencies. By invasively reading out the $\mu$C, an adversary obtains the system key and IDs of valid transponders. Similar to the previous invasive cloning of a transponder, the data can now be programmed onto an original SimonsVoss transponder that optically appears genuine. She can furthermore attempt to cover the tracks of the attack, e. g., the decapsulated $\mu$C could be replaced with a new, re-programmed PIC. Again, the complete attack can be carried out in less than 30 minutes.

Compared to the attack cloning a single transponder, the consequences of obtaining the system key are far more severe. Given the ID, any transponder in the system can be cloned without physical access to the actual transponder hardware. Since a list of IDs can be extracted from a door lock together with the system key, the problem for the adversary reduces to obtaining the PCB of one single lock.

### 5.2 Cryptanalytical Attack

In the following we present a non-invasive attack that allows to recover $K_T$ in a very practical setting within a few seconds resp. minutes. The attack exploits the following properties of the system:

1. When the door computes $R_0^{(t)}||R_1^{(t)} = \mathcal{R}(K_T; C^{(t)}, \cdots)$ to verify the transponder's response, 40 bits of the internally computed DES key $V^{(t)}$ are used as part of the next challenge, i. e.,
$$\left(C_2^{(t+1)}, C_3^{(t+1)}, C_4^{(t+1)}, C_5^{(t+1)}, C_6^{(t+1)}\right)_{2^8} = \left(V_8^{(t)}, V_9^{(t)} V_{10}^{(t)}, V_{11}^{(t)}, V_{12}^{(t)}\right)_{2^8}.$$
2. Looking at one instance of $\mathcal{O}$ and the equations describing the Least Significant Bits (LSBs) of each $x$ cell after 8 rounds, these bits are only dependent on 32 bits of the key. More specifically, the equations reveal that the LSBs of the $y$ cells occur in non-linear combinations, while the bits next to the LSBs occur only in linear combinations.
This observation can be generalized for any bit $b$ in the 16 output bytes, for cases where $M$ instances of $\mathcal{O}$ are chained (like in $\mathcal{R}$). Here, the $M + b$ lower

[2] In various real-world systems we found that the electronics of the main entrance's doors are placed facing *outwards* the building and can thus be easily accessed.

bits per byte of the key are found in non-linear combinations in the output bits at position $b$, while bits at position $M + b + 1$ are only appearing in linear combinations.

   Thus, in a sense even multiple instances of $\mathcal{O}$ resemble a $T$ function, making it significantly easier to invert them.

3. The output of $\mathcal{K}$, and thus every transponder key, has actually only 64 bits of entropy (the output of $\mathcal{D}$). We denote this 64-bit value as $T$, which—if recovered—allows to compute the full 128-bit key $K_T$ if the corresponding $P_0$ is known. Note that this fact alone allows to break the scheme in practice using dedicated hardware.

Especially the first item is a weakness in the protocol. However, it is a well-known fact that obtaining "good" random numbers in (constrained) embedded systems is hard, therefore it is not entirely surprising that these seemingly random looking bits are re-used as challenge.

**Lock-Only Attack** For our attack we merge three instances of $\mathcal{O}$; we focus on the part of the computation that maps the output $T$ of the DES function in the key-derivation phase, along with the (known) values of $P_0$,$P_1$ and $P_2$, to the leaked input bits of the final (modified) DES function in the response computation. As mentioned above, in order to obtain the leaked data, the protocol has to be triggered once more, as those bits leak as part of the challenge in the following protocol run.

Attacking this part allows to circumvent the (except for the small key-size) cryptographically strong, modified DES and focus on the rather weak obscurity function $\mathcal{O}$ only.

We denote by

$$F : \mathbb{F}_2^{128} \times \mathbb{F}_2^{128} \times \mathbb{F}_2^{128} \times \mathbb{F}_2^{64} \to \mathbb{F}_2^{40}$$

$$F(P_0, P_1, P_2, T) \to \mathcal{O}\Big(\mathcal{O}\Big(\mathcal{O}(P_0; T||0\dots0); P_1\Big); P_2\Big)_{64..103} = V_L$$

the corresponding part of the commutation where $V_L$ corresponds (up to a permutation of bits) to the leaked bits. In order to simplify the following description of our attack, we reordered the bits in such a way that the 5 first output-bits of $F$ correspond to the 5 least significant bits of $V_i^{(t)}$, $8 \leq i \leq 12$. Similarly, the next 5 output-bits of $F$ correspond to the second least significant bits, etc.

As mentioned above the function $F$ inhibits a structure resembling a (slight generalization of a) T-function. More precisely, splitting the output of $F$ in eight 5-bit chunks (with $F_0$ being the LSBs, $F_1$ the next bits per output byte, etc.), i. e.,

$$F(P_0, P_1, P_2, T) = \begin{pmatrix} F_7(P_0, P_1, P_2, T) \\ \vdots \\ F_0(P_0, P_1, P_2, T) \end{pmatrix}$$

it turns out that not all $F_i$ depend on all 64 bits of $T$. In fact, $F_0$ depends only on 30 bits of $T$, $F_1$ on 38, $F_2$ on 46, $F_3$ on 54 and $F_4$ on 62 bits. Even more, out of the 30 bits influencing $F_0$ seven bits of $T$ enter linearly in $F_0$.

Making this more precise, denote for a subset $S \subset \{0, \ldots, 63\}$ by $T_{\mathcal{S}}$, the projection of $T$ to $S$, i. e., $(T_{\mathcal{S}})_i = T_i$ if $i \in S$ and $(T_{\mathcal{S}})_i = 0$ otherwise. We have

$$F_0(P_0, P_1, P_2, T) = F_0(P_0, P_1, P_2, T_{\mathcal{S}_0^{(0)}}) + L_0(T_{\mathcal{S}_0^{(1)}})$$

where $\mathcal{S}_0 = \mathcal{S}_0^{(0)} \cup \mathcal{S}_0^{(1)}$ with

$$\mathcal{S}_0^{(0)} = \{0, 1, 2, 8, 9, 10, 16, 17, 18, 24, 25, 26, 32, 33, 40, 41, 42, 48, 49, 50, 56, 57, 58\}$$

and

$$\mathcal{S}_0^{(1)} = \{3, 11, 19, 27, 34, 51, 59\}.$$

and the mapping $L_0$ is linear with rank 4.

For $i = 1..7$ we get

$$F_i(P_0, P_1, P_2, T) = F_i(P_0, P_1, P_2, T_{\mathcal{S}_{i-1}}) + L_i(T_{\mathcal{S}_i \setminus \mathcal{S}_{i-1}})$$

where

$$\begin{aligned} \mathcal{S}_1 &= \mathcal{S}_0 \cup \{4, 12, 20, 28, 35, 43, 52, 60\} & \mathcal{S}_2 &= \mathcal{S}_1 \cup \{5, 13, 21, 29, 36, 44, 53, 61\} \\ \mathcal{S}_3 &= \mathcal{S}_2 \cup \{6, 14, 22, 30, 37, 45, 54, 62\} & \mathcal{S}_4 &= \mathcal{S}_3 \cup \{7, 15, 23, 31, 38, 46, 55, 63\} \\ \mathcal{S}_5 &= \mathcal{S}_4 \cup \{39, 47\} & \mathcal{S}_6 &= \mathcal{S}_7 = \mathcal{S}_5 \cup \{\} = \{0, \ldots, 63\} \end{aligned}$$

and the mappings $L_i$ are linear with rank 5 for $i \in \{1, 2, 3, 4\}$, rank 2 for $i = 5$ and rank zero for $i = 6$ and $i = 7$.

Given a set of values $\left(P_0^{(i)}, P_1^{(i)}, P_2^{(i)}\right)$ along with the leaked values $V_L^{(i)} = F\left(P_0^{(i)}, P_1^{(i)}, P_2^{(i)}, T\right)$ this structure of $F$ immediately leads to a recursive attack procedure: One first guesses the 23 key bits in $\mathcal{S}_0^{(0)}$ and sets up a system of linear equations for the bits in $\mathcal{S}_0^{(1)}$ using $F_0$. For each solution, one recursively sets up and solves the corresponding linear system for the bits in $\mathcal{S}_i$ using the values of $F_i$ for $1 \leq i \leq 5$. All remaining solutions are finally validated against the values of $F_6$ and $F_7$ and a key-candidate is output when all values match known data.

**Practical Results** The attack assumes that the attacker is able to obtain (or guess) a transponder's ID $I_T$, thus being able to construct valid $P_0, P_1$ and $P_2$. Additionally, a handful of random, consecutive, challenges $C$ are required, which can be obtained from partial consecutive protocol runs. The following two-step procedure will reveal all relevant data:

1. Temporarily obtain a transponder that has access to a desired door, press its button and record $I_T$. Alternatively, since certain keys (such as emergency keys) are assigned very low IDs (e. g., `0x00000005`)—which might be true across installations—it also seems very likely that IDs with sufficient privileges can be guessed.

2. Find the most convenient door the transponder has access to—this does not have to be the targeted door—and, using some hardware which is able to communicate on the desired frequency, run these steps of the protocol:
   (a) Send the preamble, receive $I_L$ of the door.
   (b) Send the previously obtained $I_T$ of the transponder and receive a challenge $C$.
   (c) Choose a suitable 80-bit string as $D$ and send it to the door.

   All interactions are recorded, then the protocol is terminated and the procedure is repeated until a handful of challenges have been collected.

As this attack does not make use of the responses $R_0$ and $R_1$ in the protocol, the challenges can be obtained by communication with the lock only (i. e., without a valid transponder). We implemented the attack in C and tested it against real data. The correct key can be recovered on a standard PC in less then one second on average.

Table 1 summarizes the running time of the attack and the number of key candidates in relation to the number of known input output-pairs $\left(P_0^{(i)}, P_1^{(i)}, P_2^{(i)}\right)$, $V_L^{(i)}$. Most importantly, all attack complexities are clearly practical and when using more than two pairs, in all our 1000 tries no false positives were detected as key candidates.

| # Pairs Used | Aver. Running Time | Aver. # Key Candidates |
|---|---|---|
| 2 | 3.36 min. | 21.34 |
| 3 | 11.5 sec. | 1 |
| 4 | 1.2 sec. | 1 |
| 5 | 0.65 sec. | 1 |

**Table 1.** Performance of our attack on an Intel(R) Xeon(R) CPU E5540. Note that the number of (partial) protocol runs is the number of pairs used plus one.

## 6 Conclusion

Our work shows one reason why not more existing products with weak proprietary solutions are broken: It is a challenging, time-consuming task that requires a large variety of skills; from reverse-engineering hardware and software to cryptanalytical abilities. Drawing upon *all* of these skills, we were able to perform a thorough analysis of the widely used SimonsVoss 3060 G2 access control system. We detailed methods, procedures, and results of our work to reveal all relevant physical and logical properties.

Based on the recovered details of the system, we presented attacks exploiting the found weaknesses on the hardware level. The fact that fuse bits can be erased allows to dump secret internal EEPROM contents and the firmware of the used $\mu$C. This enables the straightforward invasive cloning of *one* specific transponder. Utilizing this flaw to read out the system key from one lock compromises the long-term secret of *arbitrary* transponders. However, these attacks are heavily invasive and require access to the hardware of the system, time, and special equipment.

More powerful, logical attacks were enabled by further analysis; we found that the locking system actually uses a cryptographically strong primitive (DES) which, especially when compared to KeeLoq or Crypto1, does provide some resistance against attacks. However, here DES is used in such a way that it can be circumvented, resulting in non-invasive attacks that are even more practical than the known (non-invasive) attacks against KeeLoq or DST40. Our presented attack is able to retrieve an arbitrary transponder key after obtaining its ID and partially running the authentication protocol with a lock only a few times. A second, more passive attack (see Appendix A) is even able to obtain the key with only one full authentication (and the following challenge). In contrast to the invasive attacks, this class of attacks requires basic RF knowledge (and less time) to be executed successfully.

In conclusion it must be said that the attacks we have presented and executed are devastating and—although initially facilitated by the ability to easily bypass fuse bits (which can be attributed to Microchip Technology)—ultimately enabled by a faulty system design. Consequently, the security of *any installation* based on the analyzed system is questionable.

We are in close contact with SimonsVoss and discussed the found vulnerabilities. As a first response, SimonsVoss is currently developing a patch for new and existing G2 systems that prohibits the here presented mathematical attacks; by changing the source of the randomness in the cylinder. Therefor, it is *not* required to modify any transponder. Further, this patch does not require a change of the hardware and can be deployed over-the-air via WaveNet (if the system is online, cf. Sect. 1.3) or by a programming transponder. We assume that these mathematical attacks do not apply anymore at the time this paper gets publicly available. However, we have intentionally left out certain details of the cryptographic primitives, i.e., the round constants $RC_i$ and the modifications[3] of the DES. Since the physical attacks are not affected by this patch and are not fixable without changing the hardware, SimonsVoss is planning on a long-term basis—but as fast as possible—to move to hardware specially designed for security applications.

## References

1. S. C. Bono, M. Green, A. Stubblefield, A. Juels, A. D. Rubin, and M. Szydlo, "Security analysis of a cryptographically-enabled RFID device," in *Proceedings of the 14th conference on USENIX Security Symposium - Volume 14*. USENIX Association, 2005, http://www.usenix.org/events/sec05/tech/bono/bono.pdf.
2. K. Nohl, D. Evans, Starbug, and H. Plötz, "Reverse-Engineering a Cryptographic RFID Tag," in *USENIX Security Symposium*, P. C. van Oorschot, Ed., 2008, pp. 185–194, http://www.usenix.org/events/sec08/tech/full_papers/nohl/nohl.pdf.

[3] Note that these modifications have no effect on any results presented here.

3. F. D. Garcia, P. van Rossum, R. Verdult, and R. W. Schreur, "Wirelessly Pickpocketing a Mifare Classic Card," in *IEEE Symposium on Security and Privacy*. IEEE, 2009, pp. 3–15.
4. N. Courtois, "The Dark Side of Security by Obscurity - and Cloning MiFare Classic Rail and Building Passes, Anywhere, Anytime," in *SECRYPT*. INSTICC, 2009, pp. 331–338.
5. T. Kasper, M. Silbermann, and C. Paar, "All You Can Eat or Breaking a Real-World Contactless Payment System," in *Financial Cryptography 2010*, ser. Lecture Notes in Computer Science, vol. 6052. Springer, 2010, pp. 343–350.
6. N. T. Courtois, S. O'Neil, and J.-J. Quisquater, "Practical Algebraic Attacks on the Hitag2 Stream Cipher," in *12th International Conference on Information Security (ISC '09)*. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 167–176.
7. R. Verdult, F. D. Garcia, and J. Balasch, "Gone in 360 seconds: Hijacking with Hitag2," in *USENIX Security Symposium*. USENIX Association, August 2012, pp. 237–252, https://www.usenix.org/system/files/conference/usenixsecurity12/sec12-final95.pdf.
8. F. D. Garcia, G. de Koning Gans, R. Verdult, and M. Meriac, "Dismantling iClass and iClass Elite," in *ESORICS*, ser. Lecture Notes in Computer Science, vol. 7459. Springer, 2012, pp. 697–715.
9. H. Plötz and K. Nohl, "Legic Prime: Obscurity in Depth," 2009, http://events.ccc.de/congress/2009/Fahrplan/attachments/1506_legic-slides.pdf.
10. A. Bogdanov, "Attacks on the KeeLoq Block Cipher and Authentication Systems," in *Workshop on RFID Security (RFIDSec'08)*, 2007, rfidsec07.etsit.uma.es/slides/papers/paper-22.pdf.
11. W. Aerts, E. Biham, D. De Moitie, E. De Mulder, O. Dunkelman, S. Indesteege, N. Keller, B. Preneel, G. Vandenbosch, and I. Verbauwhede, "A Practical Attack on KeeLoq." Springer New York, 2010, pp. 1–22.
12. T. Eisenbarth, T. Kasper, A. Moradi, C. Paar, M. Salmasizadeh, and M. T. M. Shalmani, "On the Power of Power Analysis in the Real World: A Complete Break of the KeeLoq Code Hopping Scheme." in *Advances in Cryptology - CRYPTO 2008*, ser. Lecture Notes in Computer Science, vol. 5157. Springer, 2008, pp. 203–220.
13. SimonsVoss Technologies AG, "SimonsVoss posts record sales yet again in 2011," available at http://www.simons-voss.us/Record-sales-in-2011.1112.0.html?&L=6, as of September 16, 2013.
14. ——, "References," 2012, available at http://www.simons-voss.com/References.1163.0.html?&L=1, as ofSeptember 16, 2013.
15. ——, "Manual for WAVENET- FUNKNETZWERK 3065," 2011, available at http://www.simons-voss.de/fileadmin/media/produkte/Handbuch_WaveNet_Funknetzwerk_3065_D.pdf.
16. ——, "Direct Networking – WaveNet network knob cap," 2012, available at http://www.simons-voss.com/Direct-networking.631.0.html?&L=1, as of September 16, 2013.
17. ——, "Digital Locking System 3060," 2006, available at http://www.simons-voss.com/fileadmin/media/produkte/english/Manual_digital-locking-system_overview_GB.pdf.
18. A. Klimov and A. Shamir, "A new class of invertible mappings," in *CHES*, ser. Lecture Notes in Computer Science, B. S. K. Jr., Çetin Kaya Koç, and C. Paar, Eds., vol. 2523. Springer, 2002, pp. 470–483.
19. Microchip Technology Inc., "PIC16F882/883/884/886/887 Data Sheet," 2009, available at http://ww1.microchip.com/downloads/en/devicedoc/41291f.pdf.

20. NXP Semiconductors, “User Manual UM10204 – I$^2$C-bus specification and user manual, Rev. 5,” 2012, available at http://www.nxp.com/documents/user_manual/UM10204.pdf.
21. F. Beck, *Präparationstechniken für die Fehleranalyse an integrierten Halbleiterschaltungen*. VCH Verlagsgesellschaft, 1988.
22. A. Huang, “Hacking the PIC 18F1320,” 2005, available at http://www.bunniestudios.com/blog/?page_id=40, as of September 16, 2013.
23. A. Zonenberg, “Microchip PIC12F683 teardown,” 2011, available at siliconexposed.blogspot.de/2011/03/microchip-pic12f683-teardown.html.
24. Weltronik, “EPROM Löschgerät,” only found reference at http://www.weltronik.de/, apparently the company is out of business.
25. Hex-Rays, “IDA Starter Edition,” available at http://www.hex-rays.com/products/ida/processors.shtml, as of September 16, 2013.
26. National Bureau of Standards, “Data Encryption Standard,” in *FIPS-Pub.46, Federal Information Processing Standards Publication*, 1977.

## A A MitM-Attack with Minimal Data Complexity

The following attack is designed to work with an absolute minimum of protocol runs. Additionally, it is purely passive, i. e., instead of actively interacting with the transponder in order to obtain its ID, followed by partial protocol runs with a lock, we simply eavesdrop on two successive, genuine authentications. From these observations, we can derive a valid tuple $(P_0, P_1, P_2)$, as well as $R = R_0||R_1$ and the corresponding leak $V_L$ (as defined for the previous attack). We use the meet-in-the-middle principle and build on the same observations which are exploited in the previous attack. However, in contrast to the first attack, we use observable outputs (namely $R_0$ and $R_1$) of the first protocol run to build a table, which is then used to filter guesses for bits of $T$. This makes the requirement for more challenges (to detect false positives) obsolete. We have implemented the attack on the same machine as the first attack and validated it with real-world data. The average runtime is 17.1 minutes.

As for the notation, again, we merge the last part of $\mathcal{K}$ with $\mathcal{R}$ and subsequently denote the iterated application of $\mathcal{O}$ by

$$\begin{aligned} G : \mathbb{F}_2^{128} \times \mathbb{F}_2^{128} \times \mathbb{F}_2^{128} \times \mathbb{F}_2^{64} &\to \mathbb{F}_2^{128} \\ G(P_0, P_1, P_2, T) &\to V \end{aligned}$$

which is naturally[4] defined as $G(P_0, P_1, P_2, T) = \mathcal{O}(\mathcal{O}(\mathcal{O}(P_0; T||0 \ldots 0); P_1); P_2)$. Of the 128-bit output of $G$ we have only partial knowledge, i. e., we know

$$V_L = (V_8, V_9, V_{10}, V_{11}, V_{12})_{2^8}.$$

[4] Note that $F$ used in the previous attack corresponds to a subset of the output bits of $G$.

If we split the output of $G$ into 8 chunks of 16 bits (with $G_0$ being the LSBs, $G_1$ the next bits per output byte, etc.), i. e.,

$$G(P_0, P_1, P_2, T) = \begin{pmatrix} G_0(P_0, P_1, P_2, T) \\ \vdots \\ G_7(P_0, P_1, P_2, T) \end{pmatrix},$$

we have

$$G_i(P_0, P_1, P_2, T) = \begin{cases} G_0(P_0, P_1, P_2, T_{\mathcal{S}_0}) & \text{if } i = 0, \\ G_i(P_0, P_1, P_2, T_{\mathcal{S}_{i-1}}) + L_i(T_{\mathcal{S}_i \setminus \mathcal{S}_{i-1}}) & \text{if } 1 \le i \le 4, \\ G_i(P_0, P_1, P_2, T_{\mathcal{S}_i}) & \text{if } 5 \le i \le 7, \end{cases}$$

where $L_i(T_{\mathcal{S}_i \setminus \mathcal{S}_{i-1}})$ are linear mappings of rank 5 for these sets of key bits:

$$\begin{aligned} \mathcal{S}_0 = {} & \quad \{0, 8, 16, 24, 32, 40, 48, 56\} \\ & \cup \{1, 9, 17, 25, 33, 41, 49, 57\} \\ & \cup \{2, 10, 18, 26, 34, 42, 50, 58\} \\ & \cup \{3, 11, 19, 27, 35, 43, 51, 59\} \\ \mathcal{S}_1 = {} & \mathcal{S}_0 \cup \{4, 12, 20, 28, 36, 44, 52, 60\} \\ \mathcal{S}_2 = {} & \mathcal{S}_1 \cup \{5, 13, 21, 29, 37, 45, 53, 61\} \\ \mathcal{S}_3 = {} & \mathcal{S}_2 \cup \{6, 14, 22, 30, 38, 46, 54, 62\} \\ \mathcal{S}_4 = {} & \mathcal{S}_3 \cup \{7, 15, 23, 31, 39, 47, 55, 63\} \\ \mathcal{S}_5 = \mathcal{S}_6 = \mathcal{S}_7 = {} & \mathcal{S}_4 \cup \{\} = \{0, \dots, 63\}. \end{aligned}$$

Up to minor changes, those sets coincide with the ones used in the attack introduced in Sect. 5.2. Here, we deliberately choose $\mathcal{S}_0$ to contain 32 key bits, because guessing these actually fixes 25 bits of $G$, which are completely independent of any other key bit. Having this many fixed bits is advantageous, as will be understood in the following description of the actual attack.

We begin by building a table. Due to $V_L$ we already know 35 bits of the key used for the modified DES in $\mathcal{R}$. We invert the $\mathcal{D}$ function by decrypting $R$ for all $2^{21}$ possible keys. For each key, we get 8 different 64-bit key candidates. The $2^{24}$ results of this operation are stored in a sorted lookup table, each entry consisting of a 128-bit string, being the concatenation of DES plaintext and key and thus a potential output of $G$.

The remainder of the attack proceeds in a manner quite similar to the first attack. One starts by guessing all 32 bits of $T$ in $\mathcal{S}_0$. This allows to compute $G_0$ and parts of $G_1$ which are then matched against the lookup table to filter wrong candidates. Please note that here we have a very strong filter, i. e., 25 bits for a table with $2^{24}$ entries. For all remaining candidates one recursively sets up and solves a linear system for the bits of $T$ in $\mathcal{S}_i$ with $1 \le i \le 4$. At each step, a partial matching is used to further filter out wrong candidates. After solving equations for key bits in $\mathcal{S}_4$ the entire key is known and thus $G_5, G_6$ and $G_7$ are only used to filter further. The guessed (and derived) bits that pass all filters constitute the correct $T$, from which $K_T$ can easily be computed with the help of $P_0$.

Interestingly enough, this attack would even work (in principle) when $V_L$ is no longer leaked, which would be a trivial fix of the authentication protocol. The only difference would be that one would start by guessing the 32 bits of $S_0$ and build a table based on the resulting, fixed bits.