Physical Cryptanalysis of KeeLoq Code Hopping Applications

Thomas Eisenbarth[1], Timo Kasper[1], Amir Moradi[2,⋆], Christof Paar[1], Mahmoud Salmasizadeh[2], and Mohammad T. Manzuri Shalmani[2]

[1] Horst Görtz Institute for IT Security
Ruhr University of Bochum, Germany
[2] Department of Computer Engineering and Electronic Research Center
Sharif University of Technology, Tehran, Iran
{eisenbarth,kasper,moradi,cpaar}@crypto.rub.de
{salmasi,manzuri}@sharif.edu

Abstract. KeeLoq remote keyless entry systems are widely used for access control purposes such as garage door openers or car anti-theft systems. We present the first successful differential power analysis attacks on numerous commercially available products employing KeeLoq code hopping. Our new techniques combine side-channel cryptanalysis with specific properties of the KeeLoq algorithm. They allow for efficiently revealing both the secret key of a remote transmitter and the manufacturer key stored in a receiver. As a result, a remote control can be cloned from only ten power traces, allowing for a practical key recovery in few minutes. Once knowing the manufacturer key, we demonstrate how to disclose the secret key of a remote control and replicate it from a distance, just by eavesdropping at most two messages. This key-cloning without physical access to the device has serious real-world security implications. Finally, we mount a denial-of-service attack on a KeeLoq access control system. All the proposed attacks have been verified on several commercial KeeLoq products.

1 Motivation

The KeeLoq block cipher is widely used for security relevant applications, e.g., in the form of passive Radio Frequency Identification (RFID) transponders for car immobilizers [15] and in various access control and Remote Keyless Entry (RKE) systems. In the last months, the KeeLoq algorithm has moved into the focus of international cryptographic research community. Shortly after the first cryptanalysis of the cipher [1], more analytical attacks were proposeed [4, 5], revealing mathematical weaknesses of the cipher. The best known analytical attacks targeting Identify Friend or Foe (IFF) systems require at least 2^{16} plaintext-ciphertext pairs in order to recover the secret key of one transponder employing the KeeLoq algorithm. The described approach allows, after several

⋆ Amir Moradi performed most of the work described in this contribution as a visiting researcher at the Ruhr-University of Bochum.

days of calculations, for a simple cloning of the transponder and, only in case of a very weak key derivation method[1], for obtaining the manufacturer key that is required to generate keys for new valid transponders. Despite the impressive contribution to the cryptanalysis of the cipher, the real-world impact of the previous attacks are somewhat limited: First, they cannot be directly applied to the widespread KeeLoq code hopping applications[9] on which most RKE systems rely (which appears to be the dominant commercial application of KeeLoq). The required minimum of 2^{16} plaintext-ciphertext pairs cannot be obtained in case of a code hopping scheme, because an adversary has only access to the ciphertexts that are transmitted by a remote control, while the corresponding plaintexts remain a secret stored in the device. Second, physical access to the transmitter device is needed in order to obtain the 2^{16} plaintext-ciphertext pairs. Third, and perhaps most importantly, the manufacturer key which is located in the receiver can only be recovered if a weak key derivation function is being used which is not the case in many commercial systems. An overview on the previous work is given in Sect. 2.4.

Motivated by the ongoing research we investigate the vulnerability of real-world KeeLoq implementations with respect to side-channel analysis, in order to tackle the security of code hopping applications in a realistic manner. As a result, we present three very practical key recovery attacks and a denial-of-service attack with severe implications for RKE systems that are currently used in the field. These new attacks — which combine side-channel cryptanalysis with specific properties of the KeeLoq algorithm — can be applied to various implementations of KeeLoq. In particular, we have been able to successfully attack hardware realizations, i.e., the Microchip HCSXXX family of chips, as well as software implementations running on Microchip PIC microcontrollers. In contrast to the hitherto existing attacks, the techniques proposed by us are also applicable in case of more sophisticated key derivation schemes (c.f. Sect. 2.3) and are appropriate for the KeeLoq code hopping schemes. We elaborate in Sect. 3 how the secret key of a transmitter and the manufacturer key used in a receiver can be revealed in less than one hour and less than one day, respectively. Finally, we describe how to recover the secret key of a transmitter and hence clone it from a distance, just by eavesdropping at most two hopping code messages. We detail how an attacker can gain access to sites that are protected with KeeLoq code hopping systems with our methods and show how to put the access control out of operation, i.e., illustrate how to mount a denial-of-service attack. In contrast to the hitherto existing work, the techniques proposed by us are also applicable in case of more sophisticated key derivation schemes (c.f. Sect. 2.3) and are appropriate for the KeeLoq code hopping schemes. Note that our key recovery attacks can also be applied to IFF applications. All our attacks have been extensively tested and verified. We present various experimental results and provide ascertained figures for attacks both based on the current consumption and the electromagnetic (EM) emanation of different KeeLoq devices.

[1] If the key of the transmitter is derived from XORing a simple function of the device serial number with the manufacturer key, the latter can easily be obtained

Since the introduction of power analysis in 1999 [6], it has become an established measure to access protected information from security related systems by exploiting power consumption traces of cryptographic hardware. Almost ten years later, the most powerful attack in this area, called Differential Power Analysis (DPA), remains an attack mostly performed in smart card test labs and universities. The targets are often own or known implementations on platforms that are well-known to be vulnerable to side-channel attacks, employing no countermeasures and examined in an ideal environment [18, 8, 16], for example with an artificially generated trigger signal for the measurements. The practical relevance for real-world realizations of cryptography sometimes remains an open question. During our investigations, we were confronted with black box implementations, i.e., with no previous knowledge or information about the devices except for the known cipher and the characterization in the data sheet, which demanded for some extra efforts and reverse engineering of the unknown targets. Despite these obstructions, we were able to mount highly effective attacks with considerable implications on the security of KeeLoq code hopping systems that allow for

- recovering the secret key of a KeeLoq code hopping encoder with as few as ten power traces and only minutes of computation time,
- obtaining the manufacturer key used in a receiver in less than one day,
- cloning a remote control from the distance by eavesdropping at most two hopping codes[2], and
- putting an access control system out of service.

The described complete break of the KeeLoq code hopping mechanism was performed with no previous knowledge about the implementations and is applicable to all KeeLoq key derivation schemes we are aware of. Our DPA attacks were performed on commercial KeeLoq implementations and are highly effective with regard to complexity and computational cost — for example finding the 64 bit key of a transmitter is possible after measuring the power consumption of only ten encryptions, i.e., press the button of a transmitter ten times, with a sample rate as low as 20 MS/s. The required time for the post-processing is in the order of minutes.

2 Background

2.1 Description of KeeLoq

KeeLoq is a block cipher with a 64-bit key and 32-bit plaintext and ciphertext. As illustrated in Fig. 1, it can be viewed as a non-linear feedback shift register (NLFSR) where the feedback depends linearly on two register bits, one key bit, and a non-linear function (NLF). The NLF maps five other register bits to a single bit with the output vector $3A5C742E_x$[3]. Prior to an encryption, the

[2] The manufacturer key is assumed to be known, e.g., from a previous attack

[3] NLF (i) is the i^{th} bit of this hexadecimal constant, where bit 0 is the least significant bit [1, 4, 5]

given key and plaintext are loaded in the key register and the state register, respectively. In each clock cycle, the key register is rotated to the right and the state register is shifted to the right so that the fresh bit prepared by the XOR function becomes part of the state. After 528 clock cycles, the state register contains the ciphertext. The decryption process is similar to the encryption, except for the direction of shifts and the taps for the NLF and the XOR function.

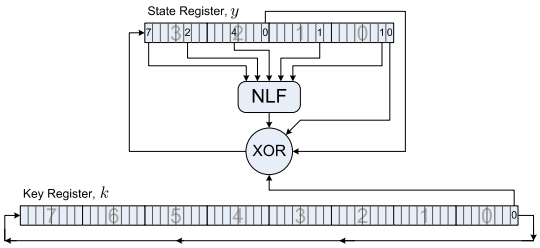


Fig. 1. Block diagram of the KeeLoq encryption

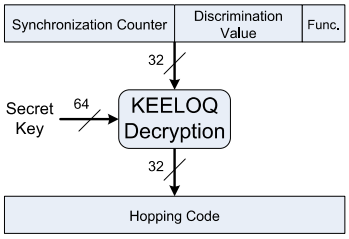


Fig. 2. Structure of KeeLoq hopping codes

2.2 Code Hopping Protocol

In addition to KeeLoq IFF systems which provide authentication of a transmitter to the main system using a simple challenge-response protocol, KeeLoq is used in code hopping (or rolling code) applications [9]. In this mechanism, which is widely used, e.g. in car anti-theft systems and garage door openers, the transmitter is equipped with an encoder and the receiver with a decoder. Both share a secret key and a fixed value with 10 or 12 bits, referred to as *discrimination value*. In addition, they are synchronized with a 16- or 18-bit *synchronization counter*. According to Fig. 2, the transmitter constructs a hopping code by encrypting a 32-bit message formed of the discrimination value, the counter and a 4-bit *function information*. The synchronization counter is incremented in the encoder each time a hopping code is transmitted. It is used in the decoder to determine whether a transmission is a repetition of a previous transmission. Previous codes are rejected to safeguard against replay attacks. The function information determines which task is desired by the transmitter, for instance, it enables a remote control to open or close more than one door in a garage opener system.

One message sent via the radio frequency (RF) interface consists of a hopping code followed by the serial number of the transmitter. The receiver decrypts the hopping code using the shared secret key to obtain the discrimination value and the current counter value. The transmitter is authenticated if the received discrimination value is identical to the one stored in the receiver and the counter fits in a window of valid values. Three windows are defined for the counter. If the difference between a received counter value and the last stored value is within the first window, i.e., 16 codes, the intended function will be executed after a single button press. Otherwise, the second window containing up to 2^{15} codes[4] is examined. In this so-called resynchronization window, the desired function is carried out only if two consecutive counter values are within it, i.e., after pressing the button twice. The third window contains the rest of the counter space. Any transmission with a counter value within this window will be ignored, to exclude the repetition of a previous code and thus prevent replay attacks.

2.3 Key Derivation Schemes

There are two types of keys involved in a typical KeeLoq application. The device key is unique to each transmitter and is shared by the transmitter and the receiver. The device key is established during a learning phase. The other type of key is the manufacturer key which is only stored in the receiver, and which is (to our knowledge) identical in all receivers of a given manufacturer. Its main use is the derivation of the device key.

In order to prevent a leakage of the device key during the learning phase (during which the transmitter and the receiver exchange the secret key, discrimination value and synchronization counter), several key derivation schemes are defined. The use of a manufacturer's key allows for a unique relationship between a specification of the transmitter, e.g., the serial number, and the secret key. This enables each manufacturer to produce transmitters that cannot be cloned by competitors. Since the manufacturer's key is critical for the product security, it is stored in a read protected memory in the microcontroller of the receivers.

The known key derivation schemes are reviewed in the following:

(a) According to Fig. 3.a, the device key is obtained by two KeeLoq decryptions. The two functions F_1 and F_2 (which are usually simple paddings) are applied to the transmitter's serial number, which then form the plaintexts for the decryptions. This scheme is proposed by Microchip as 'Normal Key Generation' method.

(b) The key derivation scheme shown in Fig. 3.b is similar to the previous one, except for a randomly generated seed value which is stored in the transmitter and is used to generate the device key. During the learning phase, a transmitter can be forced to send its seed value in some way, e.g., by pressing all of its buttons at the same time.

[4] These window sizes are recommended by Microchip, but they can be altered to fit the needs of a particular system

(c) Sometimes, the device key is generated straightforwardly from an XOR of a simple function of the device serial number with the manufacturer key. This scheme is presented in Fig. 3.c.

(d) The last scheme, illustrated in Fig. 3.d, is similar to the third one. The device key is derived from an XOR of the manufacturer key and a simple function of the seed value of the device. This scheme is referred to as 'Secure Key Generation' method by Microchip.

Due to their simplicity, the last two key derivation schemes, including the 'Secure Key Generation' method, allow for directly concluding to the manufacturer key if the adversary has access to the transmitter. In contrast to the first two (more sophisticated) derivation schemes, they can be broken by the existing attacks that are described below. Note that a manufacturer may develop a proprietary key derivation scheme not included in the above list.

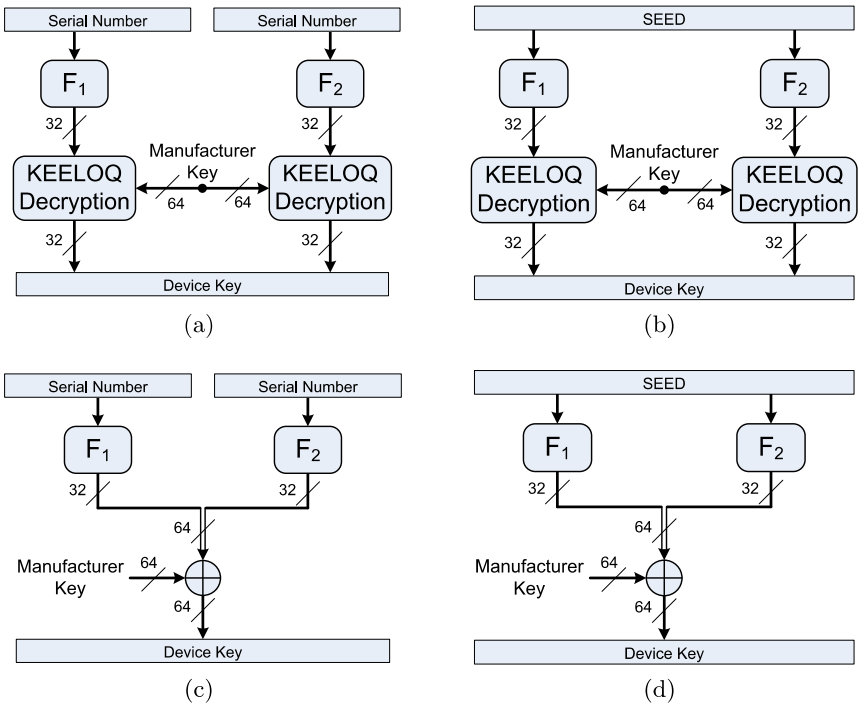


Fig. 3. Key derivation schemes

2.4 Previous Work

The first two attacks on the KeeLoq algorithm were published by Bogdanov [1]. One attack is based on slide and guess-and-determine techniques and needs about $2^{50.6}$ KeeLoq encryptions. The other one additionally uses a cycle structure analysis technique and requires 2^{37} encryptions. However, both attacks require the entire codebook, i.e. all 2^{32} plaintext-ciphertext pairs.

Afterwards, Courtois *et. al* [4] proposed two attacks. One is a slide-algebraic attack demanding for $2^{51.4}$ KeeLoq encryptions and 2^{16} known plaintext-ciphertext pairs. The second slide attack can be carried out knowing almost the entire codebook. It reveals the secret key with a complexity of approximately 2^{27} KeeLoq encryptions.

Recently, Indesteege *et. al* improved the existing work significantly and presented more practical attacks on the KeeLoq algorithm [5]. All of them are also based on slide and meet-in-the-middle attacks. The best one requires 2^{16} known plaintext-ciphertext pairs and has a complexity of $2^{44.5}$ KeeLoq encryptions. It allows for finding the secret key of the transmitter (and thus the manufacturer key for the weak key derivation schemes) in two days using 50 Dual Core machines.

The above attacks are appropriate for KeeLoq IFF systems because it is possible to collect 2^{16} plaintext-ciphertext pairs in about one hour [5] from a commercial KeeLoq IFF system. However, none of these attacks works on applications employing the KeeLoq code hopping technique, because the plaintext of the hopping codes is not known to an attacker. It is mentioned in [5] that knowing the sequence of 2^{16} ciphertexts of a code hopping application is sufficient to perform their attack as this sequence is simply repeated. However, just two products of Microchip (HCS200 and HCS201) use a 16-bit synchronization counter [10, 11] and the other ones (such as HCS300, HCS301, HCS361, ...) employ two overflow bits to not repeat a hopping code message for more than 64K transmissions [12–14], i.e. extend the number of unique transmissions to more than 192K. Even if the sequence of the synchronization counter was known, the discrimination value would still be required to perform the mentioned attack. The commercial products employing the KeeLoq code hopping protocol, i.e., HCS modules, do not allow an attacker to access this information. To our knowledge, most of the commercial applications using KeeLoq as a remote entry system employ the code hopping mechanism[5], and the attacks described above are not considered a major threat to their security.

3 DPA on KeeLoq

When we started to analyze the targets using KeeLoq, we were exposed to a "classical" situation in implementation attacks: even though the algorithm is

[5] It can be verified by comparing the number of different products of Microchip as KeeLoq code hopping encoder with the number of products as KeeLoq transponder

known, hardly anything was known about the implementation. We had to develop DPA attacks to allow for an efficient key recovery for the different KeeLoq implementations, i.e., both transmitter and receiver devices. We found that the transmitters usually employ HCSXXX modules of Microchip, featuring a hardware implementation of the cipher. The receivers we looked at do not employ a dedicated KeeLoq hardware module. Instead, they are equipped with a (typically read-protected) microcontroller on which a KeeLoq decryption routine is implemented in software. This section explains the details of attacking transmitters and receivers, starting with a general approach that is appropriate for both types of realizations.

We will first give a short outline of the single steps for performing the key recovery, as illustrated in Fig. 4, before taking a closer look at the different targets.

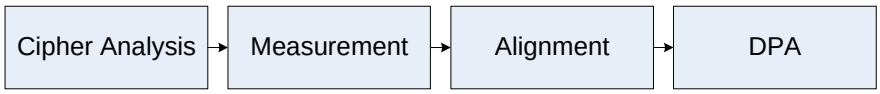


Fig. 4. Steps taken for key recovery

Initial Cipher Analysis Before being able to actually perform a DPA on a particular implementation of a cipher, one needs to make certain assumptions about the leakage produced by it. Then, a DPA scheme for exploiting that leakage must be developed, which depends on the cipher structure as well as on the particularities of the given implementation.

Even though we were unaware of the implementation details, we were able to make a few assumptions about the leakage, e.g., a software executed on a PIC microcontroller often exhibits leakage of the Hamming weight, while a hardware realization is more likely to leak the Hamming distance. Measuring the power consumption can expose the clock frequency of the device and often allows for roughly distinguishing between different parts of operation. Since hopping codes employ short messages there is no need for a high throughput, hence we assume straightforward implementations in both hardware and software.

Measurement The power traces are gathered by measuring the current via a shunt resistor connected to the ground pin of the target chip. In addition, we acquire the EM radiation of the device by means of near field probes[6]. For convenience, we have built a special printed circuit board (PCB) that allows for emulating KeeLoq chips and for controlling the transmitter or receiver from a PC so that a measurement sequence can be executed automatically. The power traces were acquired using an Agilent Infiniium 54832D digital oscilloscope with a maximum sampling rate of 4 GS/s.

[6] RF U 5-2 from `www.langer-emv.de`

Data Pre-Processing and Alignment One problem of aligning the power traces of an unknown implementation is the absence of a suitable trigger signal. The solution for this is target-specific and detailed in Sect. 3.2 and Sect. 3.3 for transmitters and receivers, respectively. Another problem is that all of the target devices are clocked by a noisy internal RC-oscillator. Hence we had to find a way to remove the clock jitter. We know that most of the data-dependent leakage occurs in the instant when the registers are clocked, a point in time which the power consumption peaks within each clock period. These peaks directly correspond to the dynamic power consumption of the target circuit and thus hold most of the information we are looking for. We developed an application to extract the peaks from the power consumption, and to base our DPA attack solely on the amplitude of the peaks. This peak extraction step has two advantages for the subsequent analysis: (*i*) the amount of data is greatly reduced, which facilitates the post-processing, the data storage and furthermore speeds up the subsequent steps significantly. (*ii*) more importantly, the peak extraction allows for an accurate alignment of the traces. Other methods for removing the clock jitter, such as Fourier transform, filtering, etc., turned out to be much more complicated and less effective.

Developing and Performing the DPA After the peak extraction and alignment steps have been performed, the traces can be processed by the DPA algorithm. For the transmitter modules we only knew the ciphertext and hence had to perform our attacks starting from the last round of the encryption. For the software implementation of the PICs we knew the plaintexts and started the attack of the first round of the decryption. The algorithms for a known plaintext attack on the decryption and for a known ciphertext attack on the encryption are the same, due to the simple structure and key management of the KeeLoq cipher.

3.1 Building a Powerful DPA for KeeLoq

It is known that for successfully performing a DPA attack, some intermediate value of the cipher has to be identified that (*i*) depends on known data (like the plaintext or the ciphertext), (*ii*) depends on the key bits and (*iii*) is easy to predict. Furthermore, it is advisable to choose a value that has a high degree of nonlinearity with respect to the key, to avoid so-called 'ghost peaks' for 'similar' keys [2]. The latter one can be easily achieved by predicting as many registers as possible, since registers have a strong leakage. This is especially true for KeeLoq, since it basically consists of 98 registers, a few XORs and a 5x1 non-linear function. As mentioned in Sect. 2, KeeLoq consists of two shift registers, *i.e.* the key register and the state register. Compared to these, the power consumption of the combinational part, *i.e.* a few XORs and the 5×1 non-linear function, is small and can be neglected.

Finally, a model for estimating the power consumption is needed. As mentioned above, classical approaches for modelling it are the Hamming weight and

Hamming distance models. For our attack it is important to note that the Hamming distance of the key register does not change, since the key is simply rotated. This leads to a theoretically constant power consumption of the key register in each clock cycle. Therefore it is not possible to find a correlation between key register bits and power traces using Hamming distance model. Hence, we focus on the state register $\boldsymbol{y}$. We execute a correlation DPA attack (CPA) [2] based on the following hypothetical power model

$$\mathrm{P}_{Hyp}^{(i)} = \mathrm{HD}\left(\boldsymbol{y}^{(i)}\right) = \mathrm{HW}\left(\boldsymbol{y}^{(i)} \oplus \boldsymbol{y}^{(i-1)}\right) \tag{1}$$

where $\mathrm{P}_{Hyp}^{(i)}$ denotes the hypothetical power consumption in the i^{th} round, HD and HW are Hamming distance and Hamming weight, respectively, $\boldsymbol{y}^{(i)}$ indicates the content of the state register in the i^{th} round, and $\oplus$ is a 32-bit bitwise XOR function. As mentioned before, the known ciphertext attack on the encryption is identical to the known plaintext attack on the decryption[7]. We describe the known ciphertext attack on the encryption. Starting from the 528^{th} round, 32 bits of the final state $\boldsymbol{y}^{(528)} = \left(y_0^{(528)}, \ldots, y_{31}^{(528)}\right)$, are known. Furthermore, 31 bits of $\boldsymbol{y}^{(527)}$, i.e., $\left(y_1^{(527)}, \ldots, y_{31}^{(527)}\right)$, are known because they are identical to $\left(y_0^{(528)}, \ldots, y_{30}^{(528)}\right)$. Therefore, just $y_0^{(527)}$ is unknown. According to Fig. 1, we can write

$$y_{31}^{(i+1)} = k_0^{(i)} \oplus y_{16}^{(i)} \oplus y_0^{(i)} \oplus \mathrm{NLF}\left(y_{31}^{(i)}, y_{26}^{(i)}, y_{20}^{(i)}, y_9^{(i)}, y_1^{(i)}\right) \tag{2}$$

where $k_0^{(i)}$ is the rightmost bit of the key register in the i^{th} round. Resolving Eq. (2) to $y_0^{(i)}$ and knowing that $k_j^{(i)} = k_{(i+j) \bmod 64}$, we obtain

$$y_0^{(527)} = k_{15} \oplus y_{16}^{(527)} \oplus y_{31}^{(528)} \oplus \mathrm{NLF}\left(y_{31}^{(527)}, y_{26}^{(527)}, y_{20}^{(527)}, y_9^{(527)}, y_1^{(527)}\right) \tag{3}$$

Thus, recovering $y_0^{(527)}$ directly reveals one bit of the key register. This process is the same for recovering the LSB of the state register of the previous rounds, i.e., $y_0^{(i)},\ i = (526, 525, \ldots)$. However, Eq. (3), and hence the whole state $\boldsymbol{y}^{(527)}$ depends linearly on the key bit k_{15}. Above we stated that nonlinearity helps distinguishing correct key hypotheses from wrong ones. Hence, recovering the key bit by bit might not be the best choice. Simulations show that an attack recovering the key bit by bit is much weaker than an attack that recovers several key bits at a time. Still, the key can also be recovered for single bit key guesses – in other words even a classical DPA on the LSB of the state register is feasible. Fortunately, according to Fig. 1, the LSB of the round state, $y_0^{(i)}$, enters the NLF leading to a nonlinear relation between the key bit k_{15} and the state $\boldsymbol{y}^{(526)}$. Accordingly, the nonlinearity for one key bit k_j increases in each round after it was clocked into the state.

[7] both attacks target state $\boldsymbol{y}^{(l)}$ of the decryption, which is the same as state $\boldsymbol{y}^{(528-l)}$ of the encryption.

Algorithm 1 A Scalable DPA for KeeLoq

Input: m : length of key guess, n: number of surviving key guesses, k: known previous key bits
Output: SurvivingKeys
1: KeyHyp $\leftarrow \{0,1\}^m$
2: **for all** KeyHyp$_i; 0 \leq i < 2^m$ **do**
3: Perform CPA on round $(528 - m)$ using P_{Hyp} and k
4: **end for**
5: SurvivingKeys $\leftarrow n$ most probable partial keys of KeyHyp

Prior to the practical execution, we modelled the attack described in Algorithm 1, assuming a Hamming distance leakage model for all involved registers, and simulated it. Based on this knowledge we predicted the leakage of the cipher. The generated traces allow for testing our attacks and also to evaluate how well an attack would work under 'perfect' conditions.

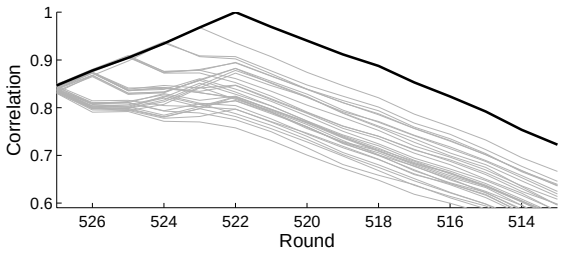


Fig. 5. Simulated correlation of key hypotheses over the KeeLoq rounds. Correct key guess (black solid line) vs. wrong key guesses (thin gray lines).

We generated a set of encryption traces with random plaintext input and computed the Hamming distance of all involved registers for each round. We performed a correlation DPA where we predicted the Hamming distance of the state register of round 522, $\mathrm{P}_{Hyp} = \mathrm{HD}(\boldsymbol{y}^{(522)})$. Fig. 5 shows the correlation for the $2^6 = 64$ key hypotheses over the first few rounds. Of course the correlation is 1 for the right key (thick solid line) in round 522. One can further see that some of the wrong key guesses (thin gray lines), that are sometimes called ghost peaks, also yield a high correlation. This is due to the high linearity between the state and the key guesses and between the different states. Furthermore we get a high correlation in the rounds before and after the predicted round. This is because most of the bits of the shift register remain unchanged in the nearby rounds. The most probable wrong key guess is always the one that differs only in the least significant bit. This underlines our expectation that the linearity increases the error probability of guessing the less significant key bits.

To improve the strength of our attack and to take care of the misleading high correlations, we added another attack step. Algorithm 1 can be repeated to guess all partial keys, one after the other. These iterations of the attack need to be done serially, because we require the previous key bits and thus the state $\boldsymbol{y}$ as a known input for each execution of the algorithm. Since some of the bits of the previous key guess might be faulty, we want keep a number n of the most probable partial key guesses as survivors. The wrong surviving candidates of the previous round will result in a wrong initial state $\boldsymbol{y}$ for the following attack round and hence strongly decrease the correlation of subsequent key guesses. This does not only allow for an assertion of the correct previous key guess, but also allows for detecting faulty previous keys. Hence the attack has an error-correcting property, since the correlation becomes worse as soon as a prediction is wrong. In case that all key guesses of one round show a low correlation, we can take one step back and broaden the number of possible key guesses.

Algorithm 2 Pruning for the Best Key Hypothesis

Input: m : length of key guess, n: number of surviving key guesses
Output: K: recovered key

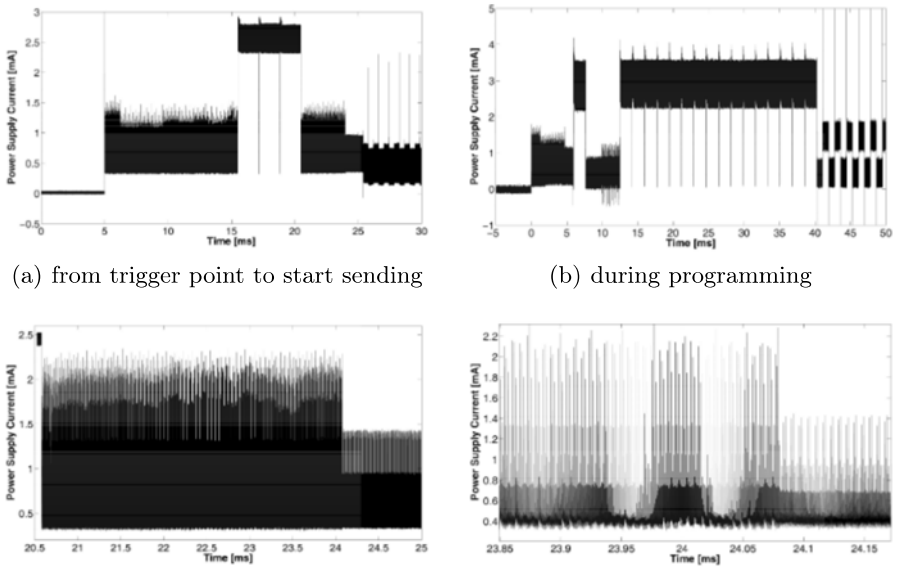
1: $K \leftarrow$ Algorithm 1$(m, n, \emptyset)$
2: **for** $round = 1$ to $\lceil \frac{64}{m} \rceil$ **do**
3: $\quad K' \leftarrow \emptyset$
4: $\quad$ **for all** $k_i \in K, 0 \leq i < n$ **do**
5: $\quad\quad K' \leftarrow K' \cup$ Algorithm 1(m, n, k_i)
6: $\quad$ **end for**
7: $\quad K \leftarrow n$ most probable keys of K'
8: **end for**
9: **return** K

Algorithm 2 describes this procedure, which is similar to the 'pruning process' described by Chari *et al.* in [3]. The last round ($i = \lceil \frac{64}{m} \rceil$) chooses the best of the last n surviving keys and verifies if an error occurred. If an error occurred, the attack can be repeated with an increased n. It will be shown in the following subsections that Algorithm 2 results in a quite strong attack.

3.2 Details of the Hardware Attack

Since we did not have any information about the architecture and design details of commercial KeeLoq code hopping encoders such as HCS200 [10], HCS201 [11], HCS300 [12], HCS301 [13], the first problem we encountered was finding the points in time in the power consumption traces that correspond to the encryption function. As depicted in Fig. 6.a, the highest amplitude is for the period of between 15 ms to 20 ms. Comparing this period with the high amplitude periods of Fig. 6.b (a power trace of the programming mode) indicates that this part corresponds to writing in the internal EEPROM cells. After several thousand measurements we found by trial and error that the encryption happens after

writing to the EEPROM, *i.e.* between 20.5 ms to 24 ms (see Fig. 6.c). The power traces furthermore reveal that the internal operating frequency of the chips is approximately 1.25 MHz.



(a) from trigger point to start sending

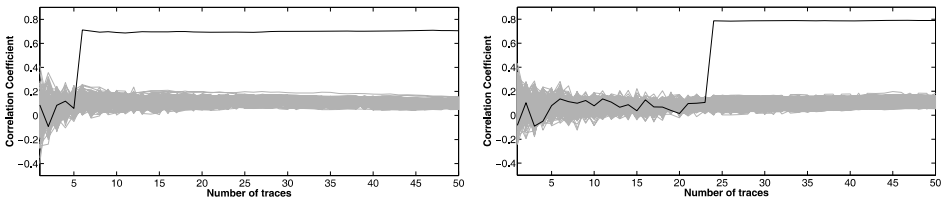
(b) during programming

(c) encryption part

(d) end of encryption part

Fig. 6. Power consumption traces of a HCS module

We performed the attack described in Sect. 3.1 on HCS200, HCS201, HCS300, HCS301, HCS361, HCS362, and HCS410 in both DIP and SOIC packages. Since it was obvious that each power value is relevant to which round of the algorithm, we modified the attack algorithm to contribute the power values of all known rounds in the CPA. The power traces were collected by measuring the voltage of a small resistor in the GND path with a sampling rate of 200 MS/s. With this sample rate, we were able to recover the secret key of DIP packages from only 10 power traces. However, the power consumption of devices in SOIC packages is smaller compared to those in DIP packages. Hence, the decreased signal-to-noise ratio (SNR) of the attack demands for more power traces. At most 30 power traces are sufficient to reveal the secret key of an HCS module in an SOIC package. Fig. 7 compares the correlation coefficient of the correct key of HCS201 chips in DIP and SOIC packages for a different number of traces. The sudden increase of the correlation is due to the error correcting property of our attack and also due to the fact that we repeated the attack for all 528 rounds of the algorithm in order to verify the revealed key.



(a) DIP package (b) SOIC package

Fig. 7. Correlation coefficients of the correct key of HCS201 chips

To estimate how costly the attack is for differently equipped adversaries, we performed experiments with varying sampling rates and compared the number of traces required in order to recover the correct key. Fig. 8 shows the results for a HCS201 chip in a DIP package.

Moreover, we repeated the experiments with an EM probe to directly measure the electromagnetic emanation instead of power consumption values. Surprisingly, the results and the number of the traces needed are similar to the case of power traces based on the current consumption. This implies that in practice our attacks can be carried out non-invasively from a distance, i.e., requiring no modification such as placing a resistor in the power supply of the remote controllers. Note that collecting the power (or EM) traces and finding the secret key is performed in less than one hour.

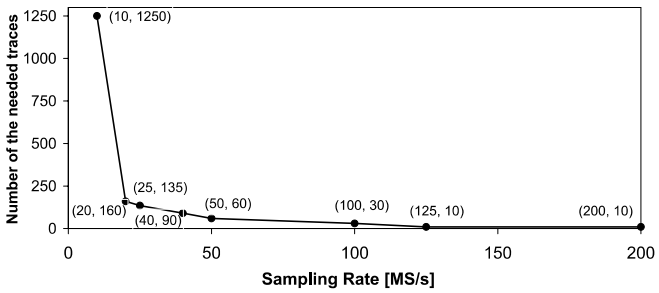


Fig. 8. The number of the needed measurements for different sampling rates. The numbers in parentheses give the exact coordinates of the points.

3.3 Details of the Software Attack

The next target of our attack is the code hopping decoder implemented in the receiver. We recall that the receiver contains the manufacturer key, which is

an attractive target for a complete break of the system. The receiver does not employ a dedicated KeeLoq hardware implementation. Instead the receiver is equipped with a standard PIC microcontroller. The microcontroller handles the key management, controls *e.g.* the motor of the garage door or the locking system of the car and performs the KeeLoq decryption in software.

Receivers usually offer a so-called 'learning mode'. In this learning mode the user can register new transmitters to cooperate with the receiver. As shown in Sect. 2.3 we know that the receiver uses the serial number or the seed of the new transmitter together with the manufacturer key to derive the device key. Our goal is to target the KeeLoq algorithm during the 'learning mode' to directly identify the manufacturer key k_M of that receiver, which is identical for all receivers of one product line.

From recovering two keys of transmitters of the same product line, as described above, we knew that neither key derivation scheme c) nor d) is used[8]. Hence we expected the receiver to perform a KeeLoq decryption using the manufacturer key during the 'learning mode'.

The receiver we looked at uses a PIC16F687 microcontroller running at a clock rate of 4 MHz. We decided to keep the receiver box fully operational and perform only minimal invasive changes. Hence we disconnected the GND-pin of the microcontroller from the PCB and inserted a shunt resistor in series in order to measure the power consumption. Finally the receiver was put into learning mode and we sent hopping code messages with random serial numbers to the receiver, emulating a transmitter by connecting the RF interface of a transmitter to the parallel port of a PC with a PCB tailored for this purpose. Our receiver can be set to training mode by pressing one button (We later just connected this button to the parallel port to perform this automatically). If the receiver obtains a message of an unknown device in the learning mode, it performs the following steps:

1. Derive the device key k_{Dev} using the received serial number of the transmitter and the stored manufacturer key k_M (by applying one of the key derivation schemes from Sect. 2.3).
2. Decrypt the hopping portion of the message using k_{Dev}.
3. Check the discrimination value. If it is valid, the transmitter is obviously of the same manufacturer and may be used with the receiver. Hence it stores (*i*) the serial number, (*ii*) the counter value and possibly (*iii*) the device key (to avoid the need to derive it each time).
4. If the discrimination value of the message is not correct, the message is simply ignored.
5. Approx. 90 seconds after pressing the button or, respectively, after the last correct transmitter was registered, the receiver leaves the 'learning mode' and only accepts messages of transmitters with known serial numbers.

Most of this procedure was explained in the manual of the receiver. Of course it was not stated which key derivation method was used, but obviously it did

[8] Reference [5] can only recover manufacturer keys if one of these key derivation schemes is employed.

not involve the seed value, since we did not need to transmit it in learning mode (it is actually possible to do that with the transmitters we used). The choices of possibly used key derivation schemes are now reduced to method (a) or an unknown other scheme. Hence we knew, if the manufacturer did not come up with an own key derivation scheme that we do not know, they used the one involving KeeLoq decryption with the manufacturer key during 'learning mode'.

We set up a simple communication channel to the receiver by directly connecting the parallel port of a PC to the communication pin of a transmitter with a removed HCS module. To verify the functionality we simply sent a valid message using a known serial number, key pair and discrimination value to register the PC to our receiver. It worked fine. The probability to accidentally generate a correct discrimination value is 2^{-12} and hence rather unlikely.

We could identify the key derivation scheme of the target receiver as scheme (a) of Sect. 2.3 by trial-and-error. As stated in Sect. 2.3, scheme a) performs a decryption of the received serial number using the manufacturer key k_M as decryption key. Hence we can recover the manufacturer key by performing a DPA key recovery on the KeeLoq decryption that is performed during 'learning mode'.

Although we do not know the exact code of the KeeLoq software implementation, one is able to make a few assumptions about how a straightforward implementation of the KeeLoq algorithm in software would look like. Before performing the DPA, we adapted the power model of the attack of Sect. 3.1 to a PIC software implementation. Typically, PIC microcontrollers leak the Hamming weight of the processed data [17].

Furthermore, one can assume that the state is stored in the 8-bit registers of the PIC microcontroller, since the content of the state is constantly accessed and changed. While the NLF is probably implemented as some kind of look-up table, we can be almost certain that the shifting of the registers is implemented using the rotate operation of the PIC. Hence, instead of predicting the Hamming distance of the whole state ($\mathrm{HD}(\boldsymbol{y})$) – as we did for the hardware attack in Sect. 3.2 – we now predict the Hamming weight of the least significant byte (LSB) of the KeeLoq state register.

$$P_{Hyp}^{(i)} = \mathrm{HW}(\boldsymbol{y}_{\mathrm{LSB}}^{(i)}) = \sum_{k=0}^{7} y_k^{(i)}$$

For the hardware attack it was very likely that one KeeLoq round was performed in each clock cycle. This is not true for the software implementation. Accordingly, we correlated the power traces for a single intermediate state only for each iteration of the attack.

We performed the attack by putting the receiver into learning mode and sending hopping code messages with random serial numbers to the receiver. Lacking any special features in the power consumption of the PIC that could have been used as trigger, we triggered the oscilloscope directly after transmitting the last bit via the RF interface. This results in our power traces not being well-aligned, leading to a high number of power samples needed to perform

a successful DPA attack. We then generated an arbitrary number of random ciphertexts (the serial number) to perform a successful DPA on the decryption. Since we did not know how much time the KeeLoq decryption needed, we chose a low sampling frequency (125MS/s) to be able to gather a long time span with each measurement.

While performing the attack we noticed that the correlation of the correct key became continuously worse with an increasing number of rounds. For the first few key bits 1000 traces sampled at 125 MS/s are roughly sufficient to find the key. Surprisingly, we need roughly ten times as many to be able to recover the full 64-bit key. This gradual decrease of the correlation is due to a misalignment that occurs during the execution of the KeeLoq algorithm. It is not due to the bad trigger condition, since the trigger affects all time instances in the same way. Also we were pretty confident that it is not due to the clock jitter, since this was removed easily using the peak detection. We conclude that the code is likely to have a data-dependent execution time, resulting in an increasing misalignment with an increasing number of rounds.

4 Attack Scenarios

In the previous section we showed how the keys of hardware and software implementations of KeeLoq can be recovered. We will now evaluate the vulnerability of real-world systems to our attacks and illustrate the implications, assuming a moderately skilled adversary. We detail four different attack scenarios, which allow for breaking basically any system using KeeLoq with modest efforts. We focus on code hopping applications, since they are more commonly used and, due to the lack of known plaintexts, harder to cryptanalyze than IFF systems. Still, IFF systems are just as vulnerable to our DPA attacks as the code hopping devices. Some of the transmitters we analyzed even offer both operating modes.

The success of some of our attacks depends on the knowledge about the particular key derivation scheme, as described in Sect. 2.3. However, the attacks are appropriate for all the key derivation schemes we are aware of.

4.1 Cloning a Transmitter

For cloning a transmitter using power analysis, an adversary needs physical access to it to acquire at least 10 to 30 power traces. Hence, the button of the remote control has to be pressed several times, while measuring the power consumption and monitoring the transmitted hopping code messages. After recovering the device key k_{Dev} with the side-channel attack described in Sect. 3.2, the recorded messages can be decrypted, disclosing the discrimination and counter values of the original transmitter at the time of the attack. Now, the HCS module of a spare remote control can be programmed with the serial number, counter value and discrimination value of the master. Consequently, the freshly produced transmitter appears to be genuine to a receiver and allows for accessing the same target as the original.

4.2 Recovering a Manufacturer Key

The actual key recovery of the manufacturer key k_M depends on the applied key derivation scheme.

If scheme (c) or (d) of Sect. 2.3 is used, i.e., an XOR of a known input and the manufacturer key k_M, disclosing the latter is trivial. After a successful key recovery attack on one transmitter of the same brand, k_M is found by reversing the XOR function. The known input is either part of each hopping code message, in case of the serial number, or can be extracted from the remote control, in case of a seed. The derived manufacturer key can be verified with a second transmitter.

An adversary targeting the manufacturer key for scheme (a) or (b) of Sect. 2.3 requires physical access to one receiver of the same brand and model as is used at the destination site. Obtaining these is a realistic assumption, as the number of manufacturers of access systems is small. The different brands can even be distinguished from only viewing a remote control or receiver from the distance, as each manufacturer likes to invent its own special design for the product. Prior to attacking the receiver, an adversary performs the above attack from Sect. 4.1 on one transmitter, to get known to its device key K_{Dev}. Then, the key of the KeeLoq decryption performed inside the receiver during the key derivation step can be recovered, according to the attack described in Sect. 3.3. The adversary now possesses the manufacturer key k_M of that device family and can hence generate an arbitrary number of new valid remote controls with chosen serial numbers and counter values. In this case, all the prior cryptanalyses [5, 1, 4] of KeeLoq will fail, unless they recovered the key of at least 2^{16} different transmitter devices.

4.3 Cloning any Transmitter Without Physical Access

Assuming an adversary that knows the result of the previous attack, namely the manufacturer key k_M, and the key derivation method of a target device family, a remote control can be cloned by eavesdropping. The attacker has to intercept at most two hopping code messages, c_1 and c_2, sent by an unknown transmitter of the same brand. The process of finding the secret key of the eavesdropped transmitter and copying it depends on the key derivation scheme.

If the key is derived from the serial number of the transmitter, finding its device key is straightforward, since the intercepted messages contain the serial number. Knowing the manufacturer key, the attacker can simply perform the key derivation process to obtain the device key.

$$k_{Dev} = k_M \oplus [\mathrm{F}_1\,(\mathrm{SerialNumber})\,, \mathrm{F}_2\,(\mathrm{SerialNumber})]$$

Afterwards, the adversary decrypts one of the messages, c_1 or c_2 in order to disclose the current counter value. It is now possible to generate valid hopping code messages to spoof the receiver and gain access to a protected site. Note that for this scenario a single intercepted message is sufficient — the other one

can be used for a verification of the found key. The computational complexity of this attack is two KeeLoq decryptions, in the worst case.

However, if a seed value plays a role in the key derivation scheme, ,i.e., schemes (b) and (d) of of Sect. 2.3, recovering of the secret key of the eavesdropped transmitter is more difficult. During the attack, an exhaustive search needs to be performed to find the seed value. For recovering k_{Dev}, the adversary decrypts the two intercepted messages using the known manufacturer key k_M and a guessed seed value. This has to be repeated for all possible values of the seed.

$$k_{Dev}^{(i)} = KeyDerivation\left(k_M, seed^{(i)}\right)$$

$$\left(Counter_1^{(i)}, Disc_1^{(i)}\right) = \mathrm{DEC}\left(c_1, k_{Dev}^{(i)}\right)$$

$$\left(Counter_2^{(i)}, Disc_2^{(i)}\right) = \mathrm{DEC}\left(c_2, k_{Dev}^{(i)}\right)$$

Once both messages show the same discrimination value, i.e., $Disc_1^{(i)} = Disc_2^{(i)}$, and similar counter values[9], the correct device key is found with a high probability.

There are three different seed sizes used for KeeLoq systems in the field. If a 32-bit seed value is used, for example in HCS200, HCS201, HCS300, HCS301 and HCS320 KeeLoq code hopping encoders, the adversary has to run in average 2^{32} KeeLoq decryptions to find the correct seed. According to our practical implementations, this number of KeeLoq decryptions takes less than three hours on a 2.4 GHz Quad-core PC. On a special-purpose computing machine such as COPACOBANA [7], the correct 32-bit seed value and hence the key can be recovered in one second. In case of a 48-bit seed value, as used in HCS360 and HCS361 modules, it is not promising to recover the correct seed value using standard PCs. Still, it is possible to perform the 2^{48} required KeeLoq decryptions in average in about 9 hours using COPACOBANA. However, chips like the HCS410, using a 60-bit seed, are not vulnerable to this attack. Running 2^{60} KeeLoq decryptions is not feasible in a reasonable time with currently existing equipment. Note that, if physical access to the transmitter is given, even 60-bit seed values are obtained by pressing one button.

4.4 Denial of Service

As mentioned in Sect. 2.2, the synchronization counter of a receiver and a transmitter is synchronized with every valid hopping code message received. As mentioned above, three windows are defined for the counter, determining whether a message is accepted by a receiver. If the difference between a received synchronization counter and the last stored value is within the first window, ,i.e., 16 codes, the intended function will be executed after a single button press. Otherwise, the second window containing up to 2^{15} codes is examined. In this window,

[9] 'Similar' counters means that the difference $Counter_2^{(i)} - Counter_1^{(i)}$ is less than a small threshold, *e.g.* 16, depending on the period between the two eavesdrops.

the function is carried out only if two consecutive counter values are within it. The third window represents the rest of the counter space. Any transmission with a synchronization counter value within his window will be ignored. This window excludes previously used, perhaps code-grabbed transmissions from accessing the system. The behaviour can be exploited for putting an access control system out of operation.

We assume the adversary has recovered the device key k_{Dev} of a target transmitter by performing one of the attacks described in Sect. 4.1 or Sect. 4.3, and thus possesses a cloned remote control. The counter value of this remote control can now be set to the maximum value inside the second window. After sending two consecutive valid hopping codes, the receiver updates its counter to the new value, which is in the third window. As a result, the original transmitter button needs to be pressed very often, namely 2^{15} times, to increase the counter value back into the first window and hence produce a valid hopping code message.

5 Conclusion

We presented the first successful cryptanalysis of a whole device family employing cryptography – namely the KeeLoq algorithm – by performing DPA and DEMA attacks against implementations that are unknown to the adversary. These attacks can be applied to both IFF and code hopping devices.

We attacked both, software and hardware implementations of KeeLoq. We revealed a manufacturer key from a receiver and recovered the device key of a remote control acquiring only 10 power traces. By doing this we showed that even an adversary with little knowledge about the target implementation and almost no special equipment is able to recover device keys and even the manufacturer key. Surprisingly, we showed that dedicated lightweight hardware implementations are even easier to break than software implementations. This is opposing the common belief that performing DPA attacks on dedicated hardware is much more difficult than DPA attacks on microcontrollers.

Analyzing real-world applications of KeeLoq and taking into account several key derivation schemes, we developed an eavesdropping attack that allows for cloning a transmitter from a distance. Just two intercepted hopping code messages suffice to open a car or a garage door protected by a system that claims to be highly secure [19]. In addition, we introduced a denial-of-service attack for the KeeLoq code hopping scheme. It ensures, that the owner of a remote control, authorized to gain access to a certain site, will be locked out and the original transmitter is rendered useless.

Summarized, we conduct a complete break of the KeeLoq code hopping scheme, with severe implications for many KeeLoq systems currently used in the field. Some manufacturers of security sensitive devices still do not seem to care about the threat of side channel cryptanalysis. This contribution shows, that widespread commercial applications, claiming to be secure, can be practically broken with modest cost and efforts.

References

1. A. Bogdanov. Attacks on the KeeLoq Block Cipher and Authentication Systems. In *3rd Conference on RFID Security 2007 (RFIDSec 2007)*, 2007. http://www.crypto.rub.de/imperia/md/content/texte/publications/conferences/keeloq_rfidsec2007.pdf.
2. E. Brier, C. Clavier, and F. Olivier. Correlation Power Analysis with a Leakage Model. In M. Joye and J.-J. Quisquater, editors, *Cryptographic Hardware and Embedded Systems - CHES 2004*, volume 3156 of *Lecture Notes in Computer Science*, pages 16–29. Springer, 2004.
3. S. Chari, J. Rao, and P. Rohatgi. Template Attacks. *Cryptographic Hardware and Embedded Systems-Ches 2002: 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002: Revised Papers*, 2002.
4. N. T. Courtois, G. V. Bard, and D. Wagner. Algebraic and Slide Attacks on KeeLoq. In *Fast Software Encryption - FSE 2008*, Lecture Notes in Computer Science. Springer, 2008. to appear. Also available in http://eprint.iacr.org/2007/062.
5. S. Indesteege, N. Keller, O. Dunkelman, E. Biham, and B. Preneel. A Practical Attack on KeeLoq. In *Advances in Cryptology - EUROCRYPT 2008*, Lecture Notes in Computer Science. Springer, 2008. to appear.
6. P. C. Kocher, J. Jaffe, and B. Jun. Differential power analysis. In *CRYPTO '99: Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology*, pages 388–397, London, UK, 1999. Springer-Verlag.
7. S. Kumar, C. Paar, J. Pelzl, G. Pfeiffer, and M. Schimmler. Breaking Ciphers with COPACOBANA - A Cost-Optimized Parallel Code Breaker. In L. Goubin and M. Matsui, editors, *Cryptographic Hardware and Embedded Systems - CHES 2006*, volume 4249 of *Lecture Notes in Computer Science*, pages 101–118. Springer, 2006.
8. S. Mangard, N. Pramstaller, and E. Oswald. Successfully Attacking Masked AES Hardware Implementations. In *Cryptographic Hardware and Embedded Systems - CHES 2005*, volume 3659 of *Lecture Notes in Computer Science*, pages 157–171. Springer, 2005.
9. Microchip. An Introduction to KeeLoq Code Hopping. Available in http://ww1.microchip.com/downloads/en/AppNotes/91002a.pdf.
10. Microchip. HCS200, KeeLoq Code Hopping Encoder. Available in http://ww1.microchip.com/downloads/en/DeviceDoc/40138c.pdf.
11. Microchip. HCS201, KeeLoq Code Hopping Encoder. Available in http://ww1.microchip.com/downloads/en/DeviceDoc/41098c.pdf.
12. Microchip. HCS300, KeeLoq Code Hopping Encoder. Available in http://ww1.microchip.com/downloads/en/DeviceDoc/21137f.pdf.
13. Microchip. HCS301, KeeLoq Code Hopping Encoder. Available in http://ww1.microchip.com/downloads/en/DeviceDoc/21143b.pdf.
14. Microchip. HCS410, KeeLoq Code Hopping Encoder and Transponder. Available in http://ww1.microchip.com/downloads/en/DeviceDoc/40158e.pdf.
15. Microchip. HCS410/WM, KeeLoq Crypto Read/Write Transponder Module. Available in http://ww1.microchip.com/downloads/en/DeviceDoc/41116b.pdf.
16. S. B. Örs, E. Oswald, and B. Preneel. Power-Analysis Attacks on an FPGA - First Experimental Results. In *CHES*, volume 2779 of *Lecture Notes in Computer Science*, pages 35–50. Springer, 2003.
17. E. Peeters, F. Standaert, and J. Quisquater. Power and electromagnetic analysis: Improved model, consequences and comparisons. *Integration, the VLSI Journal*, 40(1):52–60, 2007.

18. K. Schramm, G. Leander, P. Felke, and C. Paar. A Collision-Attack on AES: Combining Side Channel- and Differential-Attack. In *Cryptographic Hardware and Embedded Systems - CHES 2004*, volume 3156 of *Lecture Notes in Computer Science*, pages 163–175. Springer, 2004.
19. E. Sells. Lexus RX 300 Uses Keeloq Code-Hopping Technology for Highly Secure RKE System. `http://www.thefreelibrary.com/Lexus+RX+300+Uses+Keeloq+Code-Hopping+Technology+for+Highly+Secure...-a021122718`.