Breaking KeeLoq in a Flash*
-On Extracting Keys at Lightning Speed-

Markus Kasper, Timo Kasper, Amir Moradi, and Christof Paar

Horst Görtz Institute for IT Security, Ruhr University Bochum, Germany
{mkasper, tkasper, moradi, cpaar}@crypto.rub.de

Abstract. We present the first simple power analysis (SPA) of software implementations of KeeLoq. Our attack drastically reduces the efforts required for a complete break of remote keyless entry (RKE) systems based on KeeLoq. We analyze implementations of KeeLoq on microcontrollers and exploit timing vulnerabilities to develop an attack that allows for a practical key recovery within seconds of computation time, thereby significantly outperforming all existing attacks: Only one single measurement of a section of a KeeLoq decryption is sufficient to extract the 64 bit master key of commercial products, without the prior knowledge of neither plaintext nor ciphertext. We further introduce techniques for effectively realizing an automatic SPA and a method for circumventing a simple countermeasure, that can also be applied for analyzing other implementations of cryptography on microcontrollers.

1 Motivation

Due to its wide deployment in RKE systems, the KeeLoq cipher has come to the attention of cryptographers in 2007 [1]. Several improved cryptanalytical attacks followed, but still, their complexity and other requirements make them impractical for real-world products.

This situation extremely changed with the first differential power analysis (DPA) of KeeLoq as presented on CRYPTO 2008 [5]. The paper describes how secret keys can be revealed in practice from the power consumption of KeeLoq implementations in hardware and software. In Sect. 3.3 we reflect, how especially knowing his master key allows for devastating attacks on all systems of a manufacturer. Unfortunately - from the attacker's point of view - the extraction of the master key remains difficult and requires some efforts, because the software implementations programmed into the receivers are very hard to analyze using DPA, as discussed in Sect. 4.

We illustrate in the following, that in some cases performing a key recovery by SPA is much easier and much more efficient than by DPA, and demonstrate that SPA constitutes a remedy for the open problem of extracting the master key from KeeLoq software implementations. Starting from a specific unprotected software implementation of the algorithm - as recommended by Microchip - we

* The work described in this paper has been supported in part by the European Commission through the ICT programme under contract ICT-2007-216676 ECRYPT II.

develop a highly effective SPA attack in Sect. 5. Usually, an SPA is performed based on tedious visual inspection, as detailed in Sect. 5.2, or by massive profiling of a similar device, which takes a lot of efforts and time. In Sect. 5.3, a non-heuristic method to avoid the visual inspection in some types of SPA attacks is presented, enabling a full key recovery from just a single measurement of the power consumption. We practically verify our findings by attacking some commercial KeeLoq implementations on PIC 8-bit microcontrollers and proof the effectiveness of our methods, even in the presence of a simple countermeasure. Removing the effect of reoccurring disturbing patterns in the traces, that hinder DPA and SPA in the first place, is detailed in Sect. 6. Before developing our new attack, we give some necessary background information about power analysis in Sect. 2 and briefly introduce KeeLoq RKE systems in Sect. 3. Finally, the effectiveness of DPA and SPA in the case of KeeLoq is discussed in Sect. 7.

This article meliorates the CRYPTO 2008 attacks in terms of a great reduction of the required time and computations to recover secret master keys of different manufacturers and hence allows to completely circumvent many KeeLoq systems in the field with almost no effort.

2 Power Analysis in a Nutshell

In contrast to a mathematical cryptanalysis which requires pairs of plain- and ciphertexts, in the context of power analysis knowing either the input or the output of the cipher is sufficient to mount a key-recovery attack. By measuring and evaluating the power consumption of a cryptographic device, information-dependent leakage is exploited and combined with the knowledge about the plaintext or ciphertext in order to extract, e.g., a secret key. Since intermediate results of the computations can be derived from the leakage, e.g., from the Hamming weight of the data processed in a software implementation, a divide-and-conquer strategy becomes possible, i.e., the secret key could be recovered bit by bit.

2.1 Preprocessing

For unknown implementations, it is often difficult to find an appropriate trigger point for starting the oscilloscope, e.g., a special feature in the traces, that reoccurs at the same instant in each measurement. Accordingly, the alignment of the measurements typically needs to be improved as a first preprocessing step after the acquisition. Furthermore, traces can be very large or too noisy for an effective evaluation – thus they might need to be compressed or averaged prior to statistical analysis.

Peak Extraction The dynamic power consumption is the dominant factor disclosing the processed data of complementary metal oxide semiconductor (CMOS) circuits. The corresponding peaks appearing in the measurements on each edge of the clock hence play a prominent role for power analysis. Processing only the amplitudes of these peaks - instead of all acquired data points - allows for a great

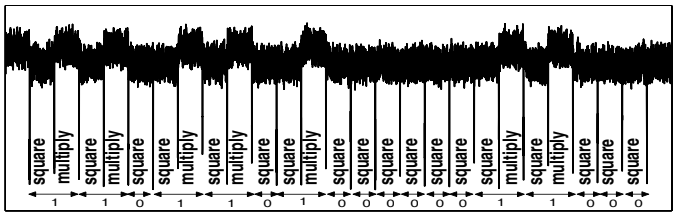


Fig. 1. SPA of an implementation of RSA.

reduction of computations and memory required during the analysis. Moreover, misalignments arising from a clock jitter due to an unstable oscillator in the cryptographic device are eliminated by peak extraction.

Averaging In case of a bad quality of the acquired power consumption, e.g., due to a noisy environment, bad measurement setup or cheap equipment, averaging can be applied by decrypting the same ciphertext repeatedly and calculating the mean of the corresponding traces. This method reduces the noise floor and can enormously increase the signal-to-noise ratio of the power traces. As exactly the same input data is processed, the measurements can be accurately aligned using a cross-correlation between two full traces. Comparing (averaged) traces for different ciphertexts can help to find the time-window in the traces, where a data-dependent behavior occurs and hence the decryption takes place.

2.2 Simple Power Analysis

An SPA attack, as introduced in [7], relies on visual inspection of power traces, e.g., measured from an embedded microcontroller of a smartcard. The aim of an SPA is to reveal details about the execution path of a software implementation, like the detection of conditional branches depending on secret information. At first, implementations of RSA were in the focus of the attackers, because an SPA of them is rather straightforward. A typical modular exponentiation comprises two main function calls, i.e., "square" and "multiply". The execution time for processing a zero or one can often be distinguished visually from the power traces, as illustrated for an 8051-based microprocessor in Fig. 1. Obviously, an attacker can directly recover the secret exponent of the RSA encryption from the sequence of instructions visible in the measurements.

2.3 Differential Power Analysis

Contrary to SPA, DPA takes many traces with often uniformly distributed known plaintexts or known ciphertexts into account and evaluates them with statistical methods. A DPA requires no knowledge about the concrete implementation of the cipher and can hence be applied to any unprotected black box

implementation. The points in time where secret information leaks during the execution of the cipher are an outcome of a DPA [7]. The traces are divided into sets according to intermediate values depending on key hypotheses and then statistically evaluated, e.g., by calculating the mean for each point in time of all traces of each set. The probability for a zero or one being processed should be uniformly distributed in each set, and thus the difference of the means will vanish, except for the set belonging to the correct key hypothesis.

In a correlation power analysis (CPA), each point in time for all measurements is compared to a theoretical model of the implementation by calculating the correlation coefficient. A maximum correlation between the hypothetical power consumption and actually measured power values indicates the correct key hypothesis [2].

3 KeeLoq RKE Systems

An RKE system consists of one receiver in the secured object and one or more remote controls that can send transmissions to the receiver and thereby control the access to the object. The early fixed-code or multi-code systems[1] were developed soon after digital circuitry became available. They rely on sending a fixed sequence of binary data when pressing the remote, and permit access in case the code is correctly identified. The obvious need for a protection against replay attacks, with only an unidirectional channel available, brought the invention and wide deployment of so-called hopping code systems. KeeLoq RKE systems typically employ hardware implementations of the cipher, such as HCSXXX [10], for generating hopping codes in the remote controls and a software implementation running on an 8-Bit PIC microcontroller [11] in the receiver to decrypt the transmissions.

3.1 Hopping Code Scheme

The remote control possesses an internal counter that is increased each time one of its buttons is pressed. The increased value is then encrypted and transmitted as a hopping code. For each remote, the receiver stores the counter value of the last valid transmission and updates the counter only upon decryption of a valid hopping code with a moderately increased counter value. The receiver is thus capable of rejecting repetitious codes and can thereby prevent replay attacks (except if combined with jamming, see Sect. 3.3). Extra remotes can usually be made known to the receiver by putting it into a learning mode in which the key of the extra remote is derived and stored.

Key Management Microchip suggests several key derivation schemes for generating a unique device key K_{dev} for each remote control. All of these schemes involve a secret manufacturer key K_{man} that is used once in the factory for a

[1] Note that even these outdated systems are still available on the market

freshly produced remote control, and later in the receiver when the key derivation takes place. This global master key for the RKE system is of course stored securely in the memory of the microcontroller.

For the most widespread key derivation in practice, the device key is a function f of the identifier ID (serial-number) of the remote control. The ID is no secret, because it is transmitted unencryptedly with every hopping code. Any party knowing the manufacturer key K_{man}, e.g., the receiver, is hence capable of calculating $K_{dev} = \mathrm{f}(K_{man}, ID)$. For the key-derivation function f, Microchip proposes KeeLoq decryptions with K_{man} [9], as described in Sect. 3.2. Even if a K_{dev} and the corresponding ID are known, a straightforward inversion of f is impossible. The described scheme for the key-derivation enables different business models, as the receivers of one manufacturer will only cooperate with remotes of the same manufacturer and thus prohibit a competitor from selling spare remotes.

3.2 KeeLoq Decryption

The decryption algorithm described in the following is used both for deciphering the hopping codes and during the key-learning phase – note that the software in a receiver never encrypts data. Prior to a decryption employing the KeeLoq block cipher, a 32-bit state $Y = \{y_0, \ldots, y_{31}\}$ is initialized with the ciphertext C. After 528 rounds of the decryption involving a secret key $K = \{k_0, \ldots, k_{63}\}$ of length 64 bits, Y contains the plaintext P.

Details of the Cipher In each round i, one key bit $k_{(15-i) \bmod 64}$ is XORed with two bits of the state and the output bit of a non-linear function (NLF) that combines five other bits of the state. Afterwards, the state is shifted left, such that the most significant bit (MSB) y_{31} is dropped, and the output of the XOR becomes the new least significant bit (LSB) y_0. The details of the cipher

Algorithm 1 KeeLoq Decryption (Pseudo Code)

Input: ciphertext $C = \{c_0, \ldots, c_{31}\}$, key $K = \{k_0, \ldots, k_{63}\}$
Output: plaintext $P = dec_K(C)$, where dec denotes KeeLoq decryption with K

1. Load ciphertext: $Y = C$
2. For i = 0 to 527 do
   2.1. Output bit of NLF: $OUT = \mathrm{NLF}(y_{30}, y_{25}, y_{19}, y_8, y_0)$
   2.2. Output bit of XOR: $XOR = k_{(15-i) \bmod 64} \oplus y_{31} \oplus y_{15} \oplus OUT$
   2.3. Update state
      2.3.1. left-shift state: $Y = (Y << 1)$
      2.3.2. assign LSB: $y_0 = XOR$
3. RETURN Y

are given in Alg. 1, where $\oplus$ denotes a bitwise XOR. Note that each key bit is reused at least eight times, i.e., every 64 rounds of the decryption.

The Non-Linear Function While the NLF could also be realized by means of Boolean functions, performing table-look-ups as described in the following is common practice. Defining a look-up table by the hexadecimal constant $LUT =$ 0x3A5C742E, its j-th bit is equivalent to one output bit OUT of the non-linear function $\mathrm{NLF}(x_4, x_3, x_2, x_1, x_0)$. The index $j \in \{0, 1, \ldots, 31\}$ thereby equals the decimal representation of the input bits x_4 to x_0, i.e., $j = 2^4 \cdot x_4 + 2^3 \cdot x_3 + 2^2 \cdot x_2 + 2^1 \cdot x_1 + 2^0 \cdot x_0$. The implementation of the NLF can be crucial for the susceptibility to SPA, as will be shown in Sect. 5.

3.3 History of Attacks on KeeLoq

A common method for electronically breaking into cars secured with hopping code systems is a combined eavesdropping-and-jamming attack: While the legitimate owner tries to lock his car with a remote control, the transmission is monitored and at the same time the frequency of the transmission is jammed, with the effect that the car won't be locked and the attacker possesses a temporarily valid hopping code. There are devices that automatically perform the described process, but in practice they are rather unreliable. One successful transmission of a new hopping code from the original remote to the car invalidates all previously eavesdropped hopping codes.

Mathematical Analysis Recently, several cryptanalytic attacks on the KeeLoq cipher have been published [3, 4, 6]. Without taking precomputed tables into account, the most efficient attack has a complexity of 2^{48} and requires 2^{16} plain- and ciphertext pairs - hence KeeLoq has to be regarded as insecure from the cryptographic point of view. Still, for a practical RKE system using hopping codes the plaintext remains secret in the remote control, rendering the mathematical attacks impractical.

Power Analysis and Eavesdropping Attack On CRYPTO 2008, a paper demonstrates the *Power of Power Analysis* [5] by describing how the K_{dev} and the master key K_{man} of commercial RKE systems based on KeeLoq can be extracted by means of DPA.

Hardware implementations of KeeLoq, such as HCS301 [10] application-specific integrated circuits (ASICs), are an ideal platform for conducting DPA attacks. The timing behavior of the chip can be foreseen very precisely, as it always performs exactly the same digital operations independent of the secret key. This implies that the power consumption at each point in time of the acquired traces is always related to the same step of the KeeLoq cipher, and extracting device keys K_{dev} with DPA is relatively straightforward. The authors of [5] report a full key recovery of K_{dev} from less than ten measurements, in the best case.

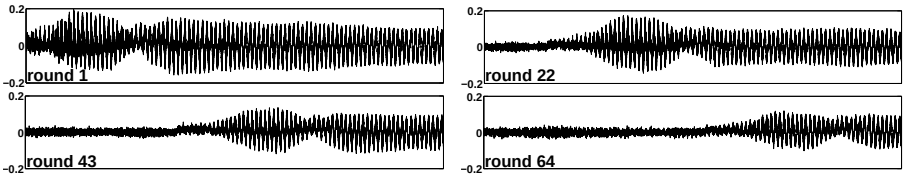


Fig. 2. Correlation coefficient of the correct key in a CPA attack on the software implementation of the KeeLoq decryption running in a PIC microcontroller.

Extracting the manufacturer key K_{man} from software implementations turned out to be orders of magnitude harder, as explained below in Sect. 4. When the secret master key K_{man} gets into the hands of an attacker, two main implications arise. Firstly, the attacker can produce fake products that are compatible with those of that manufacturer - the monopoly of the manufacturer, e.g., him being the only supplier of spare remote controls, collapses. Secondly, a remote control of this manufacturer - including its secret device key K_{dev} - can be cloned by monitoring a transmission from a distance, even without ever seeing the original. With this powerful eavesdropping approach, even a low-skilled intruder can spoof a KeeLoq receiver with technical equipment for less than US$ 50 and take over control of an RKE system, or deactivate an alarm system, leaving no physical traces.

4 Open Problem

The extraction of K_{man} from a software implementation of the KeeLoq decryption during the key-derivation mode of the receiver with DPA is much harder than a DPA attack on a hardware implementation of the cipher - mainly for two reasons. Firstly, lack of a suitable trigger point in the power consumption of the microcontroller leads to extra steps required for a proper alignment when preprocessing the traces. Secondly, as shown in Fig. 2, the correlation coefficient of the correct key continuously decreases with an increasing number of rounds, such that roughly 10 000 power traces need to be evaluated in order to fully recover the 64-bit K_{man} - a huge effort compared to 5-30 traces for extracting K_{dev} from hardware implementations. The authors of [5] predict that the cause is a data-dependent execution time for each round of a KeeLoq decryption in the program code.

4.1 Software Implementations of KeeLoq

Meanwhile, source code as proposed by Microchip for a PIC 8-bit microcontroller has become available on the Internet [12]. Appendix A shows an excerpt of the program code, revealing that the execution time of each round in the code example varies depending on the processed data. In fact, most of the program code takes the same amount of clock cycles, except for the specific implementation of

the look-up table to build the NLF (compare with Sect. 3.2). As a result, the execution time of a decryption varies for different ciphertexts - a typical indicator for a susceptibility towards an SPA.

5 SPA-Attacking KeeLoq

In this section first the mathematical aspects of our proposed SPA on KeeLoq are illustrated; then, the effectiveness of visual inspection in practice is investigated for different platforms. Finally, a new method for performing an SPA devoid of visual inspection, and empirical results from attacking commercial products, are presented.

5.1 Mathematical Background

Let us denote the content of the state register during a KeeLoq decryption by a bitstream $S = \{s_i\,; -31 \leq i \leq 528\}$. When the first 32 bits $\{s_{-31}, s_{-30}, \ldots, s_0\}$ of the bitstream are initialized with the corresponding ciphertext bits $\{c_{31}, c_{30}, \ldots, c_0\}$, the bits with indices $1 \leq i \leq 528$ can be computed according to step 2 of Alg. 1 using the iterative equation

$$s_{j+1} = k_{(15-j) \bmod 64} \oplus s_{j-31} \oplus s_{j-15} \oplus \mathrm{NLF}\left(s_{j-30}, s_{j-25}, s_{j-19}, s_{j-8}, s_j\right). \quad (1)$$

According to Eq. (1), one bit of the secret key $k_{(15-j) \bmod 64}$ can be revealed from the knowledge of eight bits of the stream S. For extracting all 64 bits of the key a consecutive section of the stream with $32 + 64 = 96$ bits is sufficient to recover all keybits. Note that in a typical known-plaintext or known-ciphertext scenario up to 32 bit of the required stream might be known a priori. The following sections will describe how to determine the required consecutive bitstream by SPA.

5.2 Visual Inspection

Visual inspection and its utilization in an SPA attack on RSA are presented in Sect. 2.2. However, the KeeLoq algorithm is extremely different from RSA and there are no distinguishable functions called during the encryption or decryption routines. As illustrated in Sect. 4.1, there are conditional branches depending on the values of the state register in the software implementation of the KeeLoq decryption recommended by Microchip. Typically, no difference in the power patterns for taking or not taking these branches can be observed for a PIC microcontroller, as it mostly leaks the operands being processed, not the operations. Hence, pinpointing the small variations of two or three instructions between two rounds of the algorithm by visual inspection is a very challenging and sometimes impossible task.

Fig. 3 shows power traces measured from commercial implementations of the KeeLoq decryption on different PIC microcontrollers. Spending a lot of time

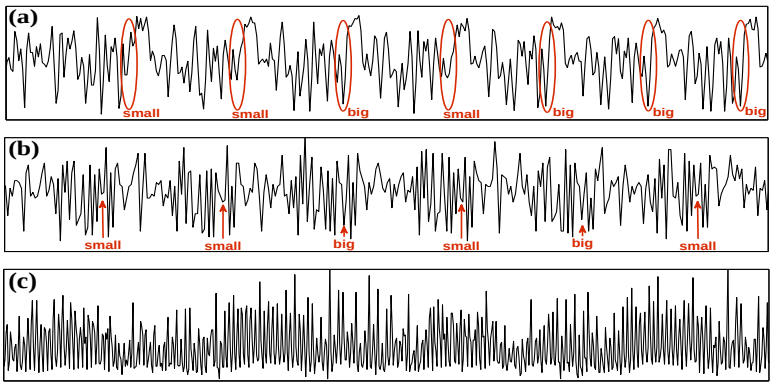


Fig. 3. Visual inspection of power traces of the KeeLoq cipher.

and efforts with manually analyzing the details of the power consumption, in few cases a distinguishable behavior can be spotted in the periodic power patterns of the microcontroller, as illustrated in Fig.3(a) and Fig.3(b). If the difference in these patterns would furthermore directly depend on the values of the status register, a key recovery according to Sect. 5.1 could be possible. However, Fig. 3(c) illustrates that in some cases no difference between the periodic patterns can be detected by means of heuristic methods, even by averaging the traces as detailed in Sect. 2.1. Note that Fig. 3 shows pure measurements without averaging, directly sampled by the oscilloscope.

5.3 A Non-Heuristic Method for SPA

In the following, we will develop a non-heuristic method that allows for automatically identifying differences in power traces - as required for an SPA - even for those implementations in which a visual inspection is not effectual. First, we examine the time-variations occurring during a KeeLoq decryption more precisely, using the example of a program code proposed by Microchip. It will turn out that the conclusions drawn from analyzing this code can be applied to many different, unknown implementations of KeeLoq decryptions[2] on PIC microcontrollers.

Investigating the Code The duration of all conditional branches that take place during the decryption is examined in Fig. 4 by comparing the number of clock-cycles required for each instruction of a PIC microcontroller [11]. While the decisions taken in the code excerpts (a), (c), (d), (e), (g), and (h) shown in Fig. 4 do not affect the execution time, the number of cycles required for (b), (f),

[2] In fact, the described method allows for extracting K_{man} from all implementations we are aware of

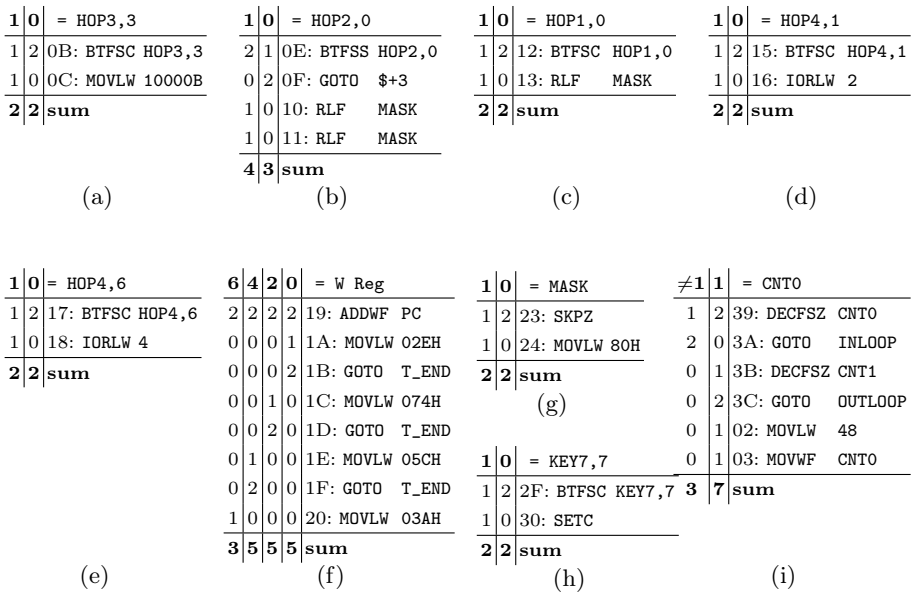


Fig. 4. Number of cycles required for the execution of an exemplary implementation of the KeeLoq decryption, depending on conditional branches.

and (i) varies with the respective condition being fulfilled or not - hence the time variations in different rounds are due to these three conditional branches. Table 1 summarizes the effect of the conditional branches on the difference in clock-cycles for the execution of one KeeLoq round. The duration of the program code in (f) and (i) of Fig. 4 can increase by a multiple of two cycles, depending on the state of the checked register, and (b) can likewise increase the length of a round by one cycle. As a consequence, taking the execution time of each round modulo 2 can reveal the result of the decision taken in (b), where the value of `HOP2,0`, i.e., the 9th bit of the status register y_8, is tested. It is hence possible to deduce one bit of the status register from the duration of each round and, as described in Sect. 5.1, recover the whole 64-bit secret key from the execution time of at least 96 consecutive rounds[3].

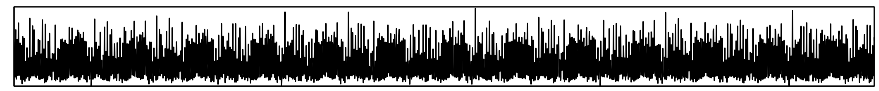
As shown in Sect. 5.2, visual inspection is not feasible for some implementations - even less can the length of each round be precisely detected. In the following, we thus introduce a non-heuristic technique for determining the number of cycles in each round.

Power Leakage of PIC microcontrollers Each execution cycle of a PIC microcontroller lasts four clock cycles [11], hence four peaks in a power trace relate to one execution cycle. Fig. 5 shows peaks extracted from power traces

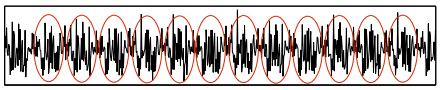
[3] 64 consecutive rounds may suffice if the ciphertext or the plaintext is known prior to the attack

Table 1. Difference of the number of cycles depending on the conditional branches.

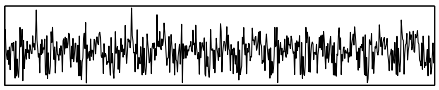
CNT0=1	HOP4,1=1 and HOP4,6=1	HOP2,0=1	Diff. of no. of cycles	
			#	# mod 2
✗	✗	✗	**2**	**0**
✗	✗	✓	**3**	**1**
✗	✓	✗	**0**	**0**
✗	✓	✓	**1**	**1**
✓	✗	✗	**6**	**0**
✓	✗	✓	**7**	**1**
✓	✓	✗	**4**	**0**
✓	✓	✓	**5**	**1**



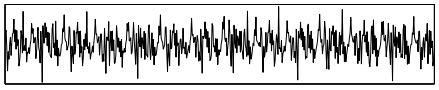
(a) four peaks per execution cycle



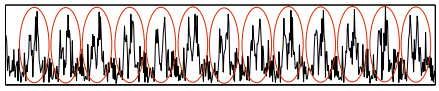
(b) first of each four peaks



(c) second of each four peaks



(d) third of each four peaks



(e) fourth of each four peaks

Fig. 5. Peaks of a power consumption trace of a PIC microcontroller running KeeLoq.

of a PIC microcontroller running a KeeLoq decryption. Extracting all four peaks of an execution cycle, as illustrated in Fig. 5(a), does not allow to locate the rounds of the decryption algorithm. In an attempt to facilitate the round-detection, only the first, second, third or fourth peak of each execution cycle are taken into account to yield Fig. 5(b), (c), (d) and (e), respectively. While focusing on the second or third peak does not improve the noticeability of the KeeLoq rounds, confining the analysis to the first or the fourth peak of each execution cycle, as shown in Fig. 5(b) and Fig. 5(e), allows for accurately distinguishing the successive rounds.

Scrutinizing the Timing In order to pinpoint the duration of each round of the algorithm, the cross-correlation between periodic patterns in the traces and a reference pattern is computed similarly to [8]. Suppose the reference pattern $\mathcal{R} = (r_1, r_2, \ldots, r_l)$ with a length of l which consists of the power-peaks of one

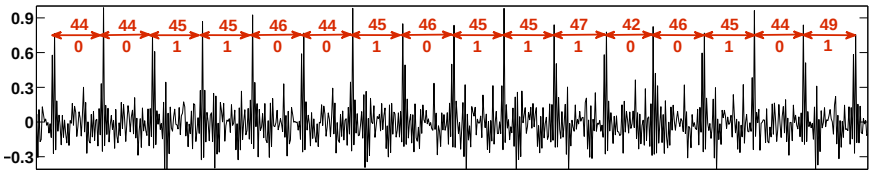


Fig. 6. An example for the correlation coefficients in vector $\mathcal{C}$.

particular round of the KeeLoq decryption. Furthermore, the vector containing the power-peaks of a whole KeeLoq decryption, with a length of n, is denoted by $\mathcal{P} = (p_1, p_2, \ldots, p_n)$. Then, a vector $\mathcal{C}$ showing the linear dependency between $\mathcal{R}$ and each section of $\mathcal{P}$ can be computed as

$$\mathcal{C} = (c_1, c_2, \ldots, c_{n-l+1}) \quad ; \quad c_i = \text{Correlation}\left(\mathcal{R}, (p_i, p_{i+1}, \ldots, p_{i+l-1})\right). \qquad (2)$$

As illustrated in Fig. 6, the rounds can be clearly identified by consecutive maxima of $\mathcal{C}$. The locations of these maxima reveal the exact length of each round and hence, taking the length of each round modulo 2 discloses the content of the state register and consequently the bits of the secret key, as described in Sect. 5.1.

5.4 Attack

Following the above described approach all 64 bits of the secret key can be recovered, but still there are three remaining problems:

i) Due to noise in the traces of the power-consumption the detection of the length of individual rounds may fail, leading to an incorrect detection of bits. Thus, a method to verify the recovered key bits would be convenient.
ii) The efficiency of the illustrated method depends strongly on the accuracy and correctness of the reference pattern $\mathcal{R}$. This demands for an in-depth study on choosing an accurate reference pattern.
iii) Suppose that all 64 bits of the key are recovered correctly. Since the key-bits are used periodically during the KeeLoq decryption and the attack can be started at any point in time with respect to the beginning of the decryption, the correct position of the recovered key bits in the secret key is not clear. Thus, the correct order of the bits needs to found out of 64 different alternatives.

Error Correction Suppose all bits of the secret key are deduced with the described attack and let $\widehat{S} = \{\widehat{s}_i \ ; \ 1 \leq i \leq 528\}$ be the resulting bitstream, containing a part of the stream S of a decryption. The corresponding $\widehat{K} = \{\widehat{k}_i \ ; \ 0 \leq i \leq 527\}$ contains the key-bits computed from $\widehat{S}$. As each key bit is used at least eight times - every 64 rounds of the decryption - the correct key

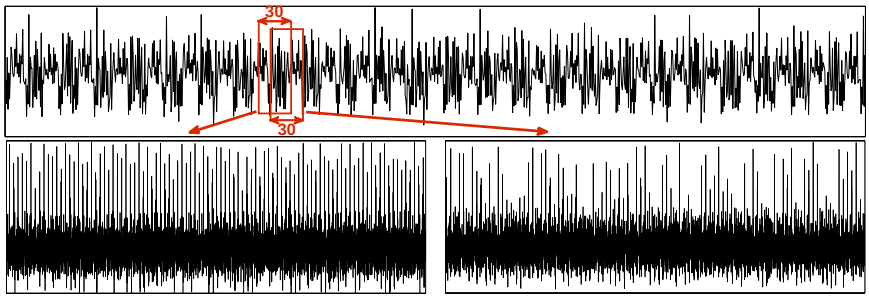


Fig. 7. Two exemplary reference patterns resulting in different correlation vectors $\mathcal{C}$.

bits reappear in stream $\widehat{K}$. Let $\widetilde{K}_i = (\widehat{k}_i, \widehat{k}_{i+1}, \ldots, \widehat{k}_{i+63})$, $0 \leq i \leq 464$, be a part of $\widehat{K}$ with a length of 64 bits, then

$$\exists i, j \ ; \ i \neq j \ , \ i = j \ (\text{mod } 64) \ , \ \widetilde{K}_i = \widetilde{K}_j.$$

Errors in the detection of the correct key-bits due to noise can hence be corrected by a majority decision.

Generation of a Reference Pattern Since the characteristics of the power consumption strongly depend on the device under test (DUT), the best basis for the reference pattern is a part of the power peaks produced by the DUT itself. As Fig. 5(b) illustrates, the durations of the rounds can be estimated by visual inspection of a decryption. Comparing with the source code described in Sect. 4.1 one can estimate that each round takes between 42 and 49 execution cycles - a reference pattern with a length of approximately 30 cycles is hence adequate. However, as the beginning and the end of a round can only be guessed, the best position of the reference pattern in the power-peaks has to be found by moving the window until $\mathcal{C}$ contains regular maximums with a similar amplitude. In Fig. 7, two $\mathcal{C}$ vectors of the same power-peaks are plotted for two different reference patterns - the vector on the left-hand side is more appropriate.

As detailed in Sect. 3.1, a device key K_{dev} is obtained by a KeeLoq decryption of the corresponding ID of a remote control. Suppose that K_{dev} is already known from performing a DPA attack on the remote [5] and that $\widetilde{K}_k$ contains the correct bits of the secret master key. With $\widetilde{K}_k^{(i)}$ denoting a rotation of the bits[4] of $\widetilde{K}_k$ by i times, where $0 \leq i < 64$, the correct secret key is found if

$$\exists i \ ; \ \mathrm{f}\left(\widetilde{K}_k^{(i)}, ID\right) = K_{dev},$$

where f denotes the key derivation function as detailed in Sect. 3.1. Hence, a known device key K_{dev} can be used to verify the correctness of the revealed key bits and furthermore simplifies to find the correct number of rotations i of $\widetilde{K}_k$.

[4] The direction of the rotations is not important, as long as it remains the same.

Attack Results The power traces of several PIC microcontrollers, such as PIC16C56 and PIC16F84A, were acquired using an Agilent Infiniium 54832D digital oscilloscope with a sampling rate of 125 MS/s by measuring the differential voltage of a 100 Ω resistor inserted in the ground path. Using the presented techniques we are able to extract the secret master key K_{man} of commercial KeeLoq code hopping receivers from only one single power trace. The efficiency of our attack is due to a software implementation leaking various key dependent information, and due to the nature of the KeeLoq cipher, i.e., using the key bits more than once.

6 Dealing with Interrupts

Most real-world implementations of the KeeLoq decryption algorithm are running on microcontrollers that are also responsible for other controlling tasks. In garage door systems this could be controlling the motor of the garage door or safety algorithms protecting users from injuries. These co-existing tasks of access control and other functionality may interfere by means of interrupt calls leading to unforeseen program flows. The resulting power traces prohibit averaging over multiple measurements and hinder straightforward CPA and SPA of the implementation. In this section we describe how power traces can be preprocessed in order to remove the power consumption of irrelevant program code inserted during the execution of the algorithm to still ensure the feasibility of side channel attacks.

Profiling A recent implementation of the KeeLoq decryption, running on the 8-bit PIC microcontroller of a commercial product, proved to be resistant to both CPA and the SPA attack detailed above. Further investigations confirmed the existence of a periodic pattern in the power consumption that appeared at unpredictable positions, independent of the start of the decryption algorithm. In order to remove the pattern, it was necessary to identify its exact length and position. Alg. 2 allows to extract the required information.

Practical results for the profiling are depicted in Fig. 8. The given power-, mean- and variance traces show the end of the KeeLoq decryption, which can be identified as a fast changing pattern on the left of the top picture. The right parts of the traces illustrate the situation after the microcontroller has finished

Algorithm 2 Profiling of Interrupted Traces

1. Measure a reasonable amount of power traces (100 traces were sufficient)
2. Identify prominent parts of the pattern to be removed by visual inspection and select one occurrence as a template
3. Align all power traces on the first match of the template since the beginning of the decryption, e.g., using least square comparison
4. Calculate mean and variance of each data point over all aligned traces

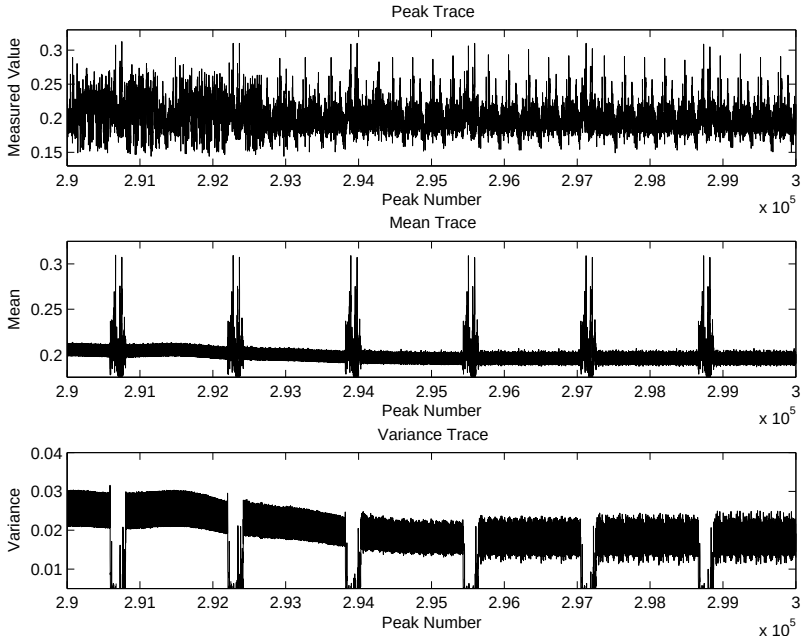


Fig. 8. A power trace with a periodic pattern (top), the mean of the aligned traces (middle), and their variance (bottom).

the decryption. The mean and variance traces reveal the pattern contained in all traces that is independent of the KeeLoq algorithm. The variance allows to identify the extent and the position of the pattern, while the mean trace shows an averaged instance of the pattern that can be used as template to identify it. Note that the pattern occurs even after the KeeLoq algorithm has finished, indicating its independency from the execution of the cipher.

Preprocessing For periodically occurring patterns, Alg. 3 provides a method to clean the traces. A similar preprocessing can be applied in case of non-periodic patterns in the power consumption, as long as they can be identified and characterized during profiling. The exact position and length of the unwanted pattern can again be found via the variance of adequately aligned traces.

Practical Results While an improved CPA on the clean traces now succeeds with around 5000 power traces, we are again able to extract the master key from a single trace using SPA. The methods described in this section can generally be used to remove the effect of timer-based interrupts and inserted dummy operations from power traces, as long as their patterns are prominent enough to allow identification of their rough position.

Algorithm 3 Preprocessing of Interrupted Traces

1. Find the first occurrence of the pattern using least squares.
2. Jump to the end of the pattern, whose relative position is known from the profiling.
3. Save its absolute position in the data point index to *Start*.
4. From the beginning of the trace to its end calculate for each data point index:
   $$RelPos = CurrentDataPointIndex - Start \bmod PeriodLength$$
5. For each point decide:
   if $RelPos \leq (PeriodLength - PatternLength)$, append data point to *NewTrace*
   if $RelPos > (PeriodLength - PatternLength)$, discard data point

The *PeriodLength* and the *PatternLength* denote the least separation between identical points of different instances of the pattern and the length of the pattern, respectively.

7 Comparison of DPA and SPA

The efforts for performing an SPA are significantly smaller than those for a DPA, because the latter naturally requires acquiring many traces and a lot of memory for storing and evaluating them. Analyzing commercial black-box implementations with DPA moreover poses the in practice sometimes difficult tasks of triggering the oscilloscope and aligning the measurements accurately. The SPA described in Sect. 5 requires neither alignment nor memory, as one measurement starting at an almost arbitrary point[5] during a decryption is sufficient for a full key recovery. Furthermore, our proposed SPA requires no knowledge about neither the plaintext, nor the ciphertext of the attacked decryption, as all necessary parameters for the SPA can be derived solely from the power measurements. A DPA is clearly impossible under these premises.

The outcome, that conducting a DPA is difficult for an unknown implementation does not imply that the implementation is more secure. In the contrary, it may turn out - as demonstrated in this paper - that an even simpler and much more effective attack is applicable, due to data-dependent execution times in the algorithm.

Implementing the cipher such that the duration of a table look-up takes equally long for any input will most likely prevent from a key recovery with the SPA as described in this paper. However, this approach cannot be recommended, because it would simultaneously facilitate an extraction of the secret key via DPA of the - now well aligned - measurements.

8 Conclusion

Obtaining the device key K_{dev} of a remote control by DPA of the hardware implementation of KeeLoq is straightforward. However, recovering the manufacturer key K_{man} from a software implementation of the cipher was still a challenging task. In this paper, we developed an SPA targeting KeeLoq software implementations on 8-bit PIC microcontrollers, making an extraction of

[5] Any starting point that captures ≥ 96 rounds of KeeLoq is appropriate

K_{man} from commercial KeeLoq systems much more feasible: where thousands of power traces were originally required to mount a successful DPA, now one single measurement suffices to recover the secret key.

After an in-depth analysis of a reference implementation of KeeLoq, we pinpointed a fatal vulnerability to SPA and exploited it to develop a very efficient key-recovery attack that requires no prior knowledge about neither the plaintext nor the ciphertext. The described approach includes a non-heuristic method for automatically extracting the parameters required for the SPA from power traces, and thus avoids tedious visual inspection. Our attack neither requires a sophisticated measurement setup, nor any preprocessing steps to align or average traces. We further detailed techniques for correcting errors, e.g., due to noisy measurements, and how irrelevant program code inserted during the execution of an algorithm can be removed a priori.

The feasibility of our attacks was demonstrated by successfully attacking several commercial products based on different PIC microcontrollers. In all cases, the efforts for extracting the correct K_{man} were reduced to evaluating one measurement. To our knowledge, and without naming any manufacturers, the described SPA can be applied to the vast majority of KeeLoq receivers in the field. Therefore, it becomes practical for criminals to extract and collect master keys of many manufacturers, and perform devastating attacks on KeeLoq RKE systems.

The assumption that extracting the manufacturer key from the software running in a receiver is very demanding and it thus could be regarded as being stored more securely than a device key of a remote control, does no longer hold. With the developed SPA attack, the manufacturer key can be extracted even much simpler than the device keys - a tragedy for the security of all owners of KeeLoq-based RKE systems.

References

1. A. Bogdanov. Attacks on the KeeLoq Block Cipher and Authentication Systems. In *RFIDSec 2007*. `rfidsec07.etsit.uma.es/slides/papers/paper-22.pdf`.
2. E. Brier, C. Clavier, and F. Olivier. Correlation Power Analysis with a Leakage Model. In *CHES 2004*, volume 3156 of *LNCS*, pages 16–29. Springer, 2004.
3. N. T. Courtois, G. V. Bard, and A. Bogdanov. Periodic ciphers with small blocks and cryptanalysis of keeloq. In *Tatra Mountains Mathematical Publications*, 2008.
4. N. T. Courtois, G. V. Bard, and D. Wagner. Algebraic and Slide Attacks on KeeLoq. In *FSE 2008*, volume 5086 of *LNCS*, pages 97–115. Springer, 2008.
5. 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 *CRYPTO 2008*, volume 5157 of *LNCS*, pages 203–220. Springer, 2008.
6. S. Indesteege, N. Keller, O. Dunkelman, E. Biham, and B. Preneel. A Practical Attack on KeeLoq. In *EUROCRYPT 2008*, volume 4965 of *LNCS*, pages 1–18. Springer, 2008.
7. P. C. Kocher, J. Jaffe, and B. Jun. Differential Power Analysis. In *CRYPTO 1999*, volume 1666 of *LNCS*, pages 388–397. Springer, 1999.

8. T. S. Messerges, E. A. Dabbish, and R. H. Sloan. Power Analysis Attacks of Modular Exponentiation in Smartcards. In *CHES 1999*, volume 1717 of *LNCS*, pages 144–157. Springer, 1999.
9. Microchip. AN642: Code Hopping Decoder using a PIC16C56. `http://www.keeloq.boom.ru/decryption.pdf`.
10. Microchip. HCS301 KeeLoq Code Hopping Encoder Data sheet. `http://ww1.microchip.com/downloads/en/DeviceDoc/21143b.pdf`.
11. Microchip. PIC16C5X Data Sheet. `http://ww1.microchip.com/downloads/en/DeviceDoc/30453d.pdf`.
12. Webpage. Program Code for KeeLoq Decryption. `http://www.pic16.com/bbs/dispbbs.asp?boardID=27&ID=19437`.

Appendix A: The KeeLoq Decryption Program Code

```
; DECRYPT   using [Key7 . . . Key0]
; | HOP4 | HOP3 | HOP2 | HOP1 |<-- Feed

DECRYPT
00: MOVLW   11+1     ; OUTLOOP COUNTER
01: MOVWF   CNT1     ; 11+1 TIMES

OUTLOOP
02: MOVLW   48       ; INLOOP COUNTER
03: MOVWF   CNT0     ; 48 TIMES

INLOOP
04: CLRWDT           ;
05: MOVFW   CNT1     ;
06: XORLW   1        ;
07: SKPNZ            ; LAST 48 LOOPS
08: GOTO    ROT_KEY  ; RESTORE THE KEY

09: CLRC             ; CLEAR CARRY
0A: MOVLW   1        ; MASK = 1
0B: BTFSC   HOP3,3   ; SHIFT MASK 4X
0C: MOVLW   10000B   ; IF BIT 2 SET
0D: MOVWF   MASK     ;

0E: BTFSS   HOP2,0   ; SHIFT MASK
0F: GOTO    $+3      ; ANOTHER 2X
10: RLF     MASK     ; IF BIT 1 SET
11: RLF     MASK     ;

12: BTFSC   HOP1,0   ; SHIFT MASK
13: RLF     MASK     ; 1X MORE IF BIT 0

14: MOVLW   0        ; TABLE INDEX = 0
15: BTFSC   HOP4,1   ; IF BIT 3 SET
16: IORLW   2        ; TABLE INDEX += 2
17: BTFSC   HOP4,6   ; IF BIT 4 SET
18: IORLW   4        ; TABLE INDEX += 4

19: ADDWF   PC       ; PC += TABLE INDEX

TABLE
1A: MOVLW   02EH     ; BITS 4:3 WERE 00
1B: GOTO    T_END    ; END OF TABLE

1C: MOVLW   074H     ; BITS 4:3 WERE 01
1D: GOTO    T_END    ; END OF TABLE

1E: MOVLW   05CH     ; BITS 4:3 WERE 10
1F: GOTO    T_END    ; END OF TABLE

20: MOVLW   03AH     ; BITS 4:3 WERE 11

T_END
21: ANDWF   MASK     ; ISOLATE THE
22: MOVLW   0        ; CORRECT BIT
23: SKPZ             ;
24: MOVLW   80H      ; W = NLF OUTPUT

25: XORWF   HOP2,W   ; W XOR= HOP2,7
26: XORWF   HOP4,W   ; W XOR= HOP4,7
27: XORWF   KEY1,W   ; W XOR= KEYREG1,7

28: MOVWF   MASK     ; FEEDBACK = BIT 7
29: RLF     MASK     ; CARRY = BIT 7

2A: RLF     HOP1     ; SHIFT IN
2B: RLF     HOP2     ; THE NEW BIT
2C: RLF     HOP3     ;
2D: RLF     HOP4     ;

ROT_KEY
2E: CLRC             ; CLEAR CARRY
2F: BTFSC   KEY7,7   ; IF BIT 7 SET
30: SETC             ; SET CARRY

31: RLF     KEY0     ; LEFT-ROTATE
32: RLF     KEY1     ; THE 64-BIT KEY
33: RLF     KEY2     ;
34: RLF     KEY3     ;
35: RLF     KEY4     ;
36: RLF     KEY5     ;
37: RLF     KEY6     ;
38: RLF     KEY7     ;

39: DECFSZ  CNT0     ;
3A: GOTO    INLOOP   ; INLOOP 48 TIMES

3B: DECFSZ  CNT1     ;
3C: GOTO    OUTLOOP  ; OUTLOOP 12 TIMES
3D: RETLW   0        ; RETURN
```