KeeLoq and Side-Channel Analysis — Evolution of an Attack

Christof Paar, Thomas Eisenbarth, Markus Kasper, Timo Kasper and Amir Moradi
Chair for Embedded Security
Electrical Engineering and Information Sciences Dept.
Ruhr-Universität Bochum
www.crypto.rub.de

***Abstract*—Last year we were able to break KeeLoq, which is a 64 bit block cipher that is popular for remote keyless entry (RKE) systems. KeeLoq RKEs are widely used for access control purposes such as garage openers or car door systems. Even though the attack seems almost straightforward in hindsight, there where many practical and theoretical problems to overcome. In this talk I want to describe the evolution of the attack over about two years. Also, some possible future improvements using fault-injection will be mentioned.**

During the first phase of breaking KeeLoq, a surprisingly long time was spent on analyzing the target hardware, taking measurements and wondering why we did not succeed. In the second phase, we were able to use differential power analysis attacks successfully on numerous commercially available products employing KeeLoq code hopping. Our techniques 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 a few minutes. With similar techniques but with considerably more measurements (typically on the order of 10,000) we can extract the manufacturer key which is stored in every receiver device, e.g., a garage door opener unit. In the third phase, and most recent phase, we were able to come up with several improvements. Most notably, we found that an SPA (simple power analysis) attack allows to recover the manufacturer key with one measurement. In the talk, we will also speculate about extensions to fault-injection and timing attacks.

It is important to note that most of our findings are not specific to KeeLoq but are — in principle — applicable to any symmetric cipher with an implementation that is not side-channel resistant.

I. Background

KeeLoq is a block cipher with a 64 bit key and a block size of 32 bits. 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 [1], [4], [6]. Prior to an encryption, the secret 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 the shifts and the taps for the NLF and the XOR function.

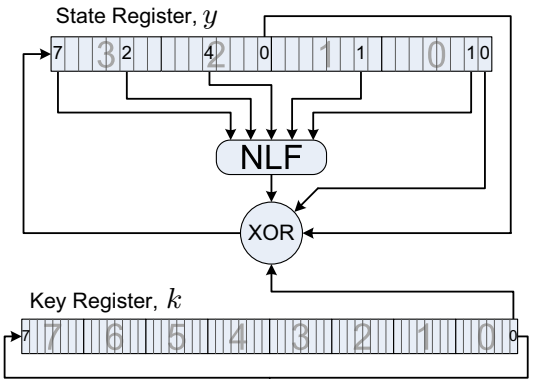

Figure 1. Block diagram of the KeeLoq encryption

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 [8]. 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 discrimination value, $disc$, with 10 or 12 bits. In addition, they are synchronized with a 16 bit or 18 bit synchronization counter, cnt, which is incremented in the encoder each time a hopping code is transmitted. The transmitter constructs a hopping code by encrypting a 32 bit message formed of $disc$, cnt and a 4 bit function information. The latter determines the task desired by a remote control, for instance, it enables 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 $disc$ and the current cnt. The transmitter is authenticated if $disc$ is identical to the shared one and cnt fits in a window of valid values. Three windows are defined for the counter. If the difference between a received cnt 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[1] 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 cnt value within this window will be ignored, to exclude the repetition of a previous code and thus prevent replay attacks.

II. DPA Attack

We summarize in the following our attack which is described in more details in [5]. When we started to analyze the targets using KeeLoq, we were exposed to a "classical" situation for physical attacks: even though the algorithm was known, hardly anything was known about the implementation. We found that the transmitters usually employ HCSXXX modules of Microchip, featuring a hardware implementation of the cipher. The receivers we looked at are typically equipped with a read-protected PIC microcontroller on which a KeeLoq decryption routine is implemented in software. This section explains the details of DPA-attacking transmitters and receivers, starting with a general approach that is appropriate for both types of realizations.

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]. For every DPA, a model for estimating the power consumption is needed. Compared to the two shift registers, 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. 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. Hence, we focus on the state register $\vec{y}$. We execute a correlation DPA attack (CPA) [2] based on the following hypothetical power model

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

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, $\vec{y}^{(i)}$ indicates the content of the state register in the i^{th} round, and $\oplus$ is a 32 bit XOR function. As mentioned before, the known ciphertext attack on the encryption is identical to the known plaintext attack on the decryption[2]. We describe the known ciphertext attack on the encryption. Starting from the 528^{th} round, 32 bits of the final state $\vec{y}^{(528)} = \left(y_0^{(528)}, \ldots, y_{31}^{(528)}\right)$, are known. Furthermore, 31 bits of $\vec{y}^{(527)}$, 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

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

where $k_0^{(i)}$ is the rightmost bit of the key register in the i^{th} round. Knowing that $k_j^{(i)} = k_{(i+j) \bmod 64}$, we can rewrite Eq. (2) as

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

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), 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[3]. 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 $\vec{y}^{(526)}$. Accordingly, the nonlinearity for one key bit k_j increases in each round after it was clocked into the state.

Algorithm 1 A Scalable DPA for KeeLoq

Require: m : length of key guess, n: number of surviving key guesses, k: known previous key bits
Ensure: SurvivingKeys
1: KeyHyp $\leftarrow$ **all** $\{0,1\}^m$
2: **for all** KeyHyp$_i; 0 \le i < 2^m$ **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

Taking the increased nonlinearity in the successive rounds into account, we developed a scalable DPA, as described in Alg. 1, that allows for finding a subset n of surviving key candidates by guessing m bits of the key in an instant. Note that in step 3 of the algorithm the CPA is performed on round $(528 - m)$, hence taking advantage of a key bit passing the NLF m times. The significance of the known previous bits

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

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

[3]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.

k will become clear below in the extended attack (Alg. 2), where Alg. 1 is executed repeatedly.

We performed simulations of the attack described in Alg. 1, assuming a Hamming distance leakage model. The simulated traces allow for testing our attacks and also to evaluate how well an attack would work under "perfect" conditions. We generated a set of encryption traces with

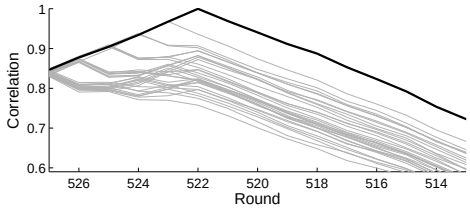

Figure 2. Simulated correlation of key hypotheses as a function of KEELOQ rounds. Correct key guess (black solid line) vs. wrong key guesses (thin gray lines).

random plaintext input and computed the Hamming distance of all registers for each round. We performed a correlation DPA where we predicted the Hamming distance of the state register of round 522, $\mathrm{P}_{Hyp} = \mathrm{HD}\left(\vec{y}^{(522)}\right)$. Fig. 2 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. Unfortunately, some of the wrong key guesses (thin gray lines) also yield a high correlation. This is due to the high linearity between both 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 LSB. This underlines our expectation that the linearity increases the error probability of guessing the less significant key bits.

Algorithm 2 Pruning for the Best Key Hypothesis

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

To improve the strength of our attack and to take care of the misleading high correlations, we added another attack step. Alg. 1 can be repeated to guess all partial keys, one after the other. These iterations of the attack need to be done one after another, because we require the previous key bits and thus the state $\vec{y}$ as a known input for each execution of the algorithm. Since some of the bits of the previous key guess might be faulty, we keep a number n of the most probable partial key guesses as survivors. Wrong surviving candidates of the previous round will result in a misleading initial state $\vec{y}$ 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 guesses, but also for detecting faulty previous keys. Hence, the attack has an error-correcting property. If all key guesses of one round show a low correlation, we can go one step back and broaden the number of surviving key guesses n. Alg. 2 describes this procedure, which is similar to the "pruning process" described by Chari *et al.* in [3]. In the last round ($i = \lceil \frac{64}{m} \rceil$) the program verifies whether an error occurred and the key with the highest correlation coefficient is selected out of the n surviving keys. It will be shown in the following subsections that Alg. 2 results in a quite strong attack.

A. Details of the Hardware Attack

For attacking commercial KEELOQ code hopping encoders we first had to find the points in time in the power traces (Fig. 3). that correspond to the encryption function. We found that the encryption happens after writing to the EEPROM[4], i.e., in the time interval between 20.5 ms and 24 ms The power traces reveal that the frequency of the internal oscillators of the ICs is approximately 1.25 MHz.

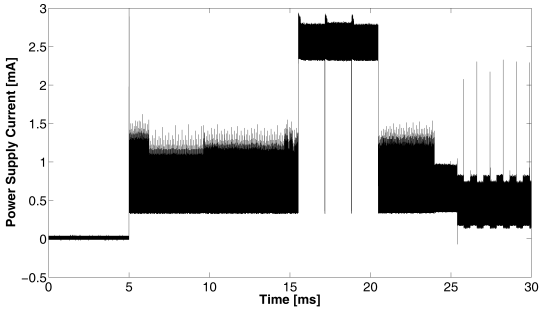

Figure 3. Power consumption traces of a HCS module

We modified the attack described above to correlate all known and predicted rounds to the corresponding power peaks. This is possible since we are able to locate the leakage of each round. The modified attack was performed on HCS200, HCS201, HCS300, HCS301, HCS361, HCS362, and HCS410 [9], [10] in both DIP and SOIC packages. In the best case we were able to recover the secret key of DIP

[4]The high amplitude periods of the power trace correspond to writing to the internal EEPROM.

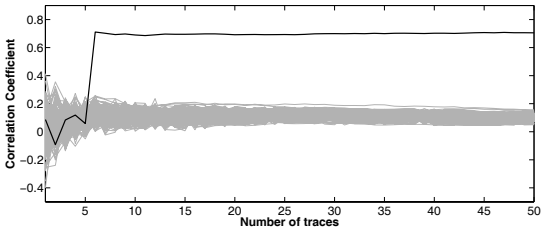

Figure 4. Correlation coefficients of key hypotheses of HCS201 ICs as a function of the number of measured traces.

package ICs from only six power traces when sampling at a rate of 200 MS/s. At most 30 power traces are sufficient to reveal the secret key of an HCS module in an SOIC package, which has a lower power consumption, resulting in a worse signal-to-noise ratio (SNR) of the measurements. Fig. 4 shows the correlation coefficients of the correct key of HCS201 chips in a DIP packages as a function of the 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.

To estimate the minimum technical requirements for the SCA, we performed experiments with varying sampling rates and evaluated the number of power traces required for recovering the correct key. Fig. 5 shows the results for attacking a HCS201 chip in a DIP package in the case of current measurements via a resistor. We conclude that our attack can be carried out effectively even with low-cost equipment, e.g., an oscilloscope with a maximum sample rate as low as 50 MS/s enables finding the secret key from only 60 power traces.

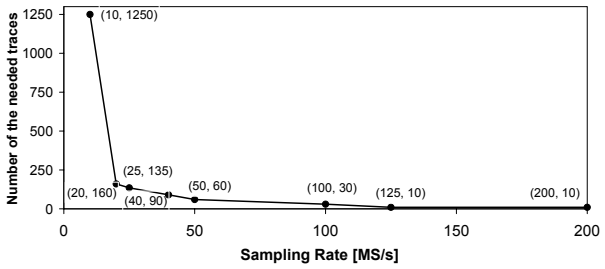

Figure 5. Number of measurements required for revealing the secret key of a HCS201 IC in a DIP package as a function of the sampling rate. The numbers in parentheses give the exact coordinates of the points.

B. 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. A PIC microcontroller handles the key management, controls for instance the motor of the garage door or the locking system of the car, and performs the KEELOQ decryption in software.

Before executing the DPA, we adapted the power model of the attack to a PIC software implementation. Typically, PIC microcontrollers leak the Hamming weight of the processed data [11]. Furthermore, one can assume that the state is stored in the 8 bit registers of the PIC microcontroller which are regularly accessed. Hence, instead of predicting the Hamming distance $\mathrm{HD}\left(\vec{y}^{(i)}, \vec{y}^{(i-1)}\right)$ of the whole state – as was done for the hardware attack in Sect. II-A – we predict the Hamming weight of the least significant byte (LSB) of the KEELOQ state register:

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

We performed the attack by putting the receiver into learning mode and sending hopping code messages with random serial numbers to the receiver[5]. Lacking any information in the power consumption of the PIC that could have been used as trigger, we triggered the scope directly after transmitting the last bit via the RF interface. This results in our traces not being well-aligned, leading to a high number of power samples needed to perform a successful DPA attack.

While performing the attack we noticed that the correlation coefficient of the correct key become continuously worse with an increasing number of rounds. For the first few key bits, 1000 traces sampled at 125 MS/s are sufficient to find the key. Surprisingly, we need roughly ten times as many traces for recovering 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. Hence, the problem is not a bad trigger condition, since the trigger affects all time instances in the same way. We assume that the program code is likely to have a data-dependent execution time for each round of KEELOQ, causing the increasing misalignment with an increasing number of rounds, and hence complicating the SCA.

III. SPA ATTACK

The extraction of the manufacturer key 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, the 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, 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

[5]We emulated a remote control by connecting the RF interface of a transmitter to the parallel port of a PC.

recover the 64-bit. Even though this is certainly doable, it constitutes a major effort compared to the few dozens of traces needed for extracting an individual device key from hardware implementations.

Since the DPA attacks had been developed by us, the source code as proposed by Microchip for a PIC 8-bit microcontroller has become available on the Internet [12]. 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. As a result, the execution time of a decryption varies for different ciphertexts — a typical indicator for a susceptibility towards an SPA.

Based on this observation we were recently able to develop an SPA attack which is considerably more powerful than the DPA attack. The SPA attack is described in [7].

For the attack, 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 $100\,\Omega$ resistor inserted in the ground path. Using the SPA techniques we are able to extract the secret manufacturer key 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.

IV. The Future: Fault Injection and Timing Attacks

Even though the SPA attack is extremely powerful, it is certainly possible to defeat it by using code with constant run time. Similarly, there are implementations possible which make the cipher more robust against DPA. Thus, it is worth speculating about other physical attacks which are powerful. One class of attacks which has not been investigated in the context of KeeLoq are fault injection attacks. Since KeeLoq is typically implemented on low-cost microcontrollers, it is likely that injecting faults during code execution, e.g., via voltage spikes, is not too difficult. Given that fault injection attacks often combine side-channel observation with mathematical properties of the cipher, it seems interesting to develop such attacks against KeeLoq. Especially if SPA and DPA countermeasures are implemented, fault injection attacks might be an attack evolution that should be exploited. Similarly, the fact that the run time is not constants also makes timing attacks a possibility.

References

[1] A. Bogdanov. Attacks on the KeeLoq Block Cipher and Authentication Systems. In *3rd Conference on RFID Security 2007 (RFIDSec 2007)*. http://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] S. Chari, J. R. Rao, and P. Rohatgi. Template Attacks. In *CHES 2002*, volume 2523 of *LNCS*, pages 13–28. Springer, 2002.

[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] M. Kasper, T. Kasper, A. Moradi, and C. Paar. Breaking KeeLoq in a Flash: On Extracting Keys at Lightning Speed. In *AFRICACRYPT 2009*, volume 5580 of *LNCS*, pages 403–420. Springer, 2009.

[8] Microchip. An Introduction to KeeLoq Code Hopping. http://ww1.microchip.com/downloads/en/AppNotes/91002a.pdf.

[9] Microchip. HCS200, KEELOQ Code Hopping Encoder. http://ww1.microchip.com/downloads/en/DeviceDoc/40138c.pdf.

[10] Microchip. HCS410, KEELOQ Code Hopping Encoder and Transponder. http://ww1.microchip.com/downloads/en/DeviceDoc/40158e.pdf.

[11] 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.

[12] Webpage. Program Code for KeeLoq Decryption. http://www.pic16.com/bbs/dispbbs.asp?boardID=27&ID=19437.