Attacks on the KeeLoq Block Cipher and Authentication Systems

Andrey Bogdanov

Chair for Communication Security
Ruhr University Bochum, Germany
abogdanov@crypto.rub.de
www.crypto.rub.de

Abstract. KeeLoq is a block cipher used in numerous widespread passive entry and remote keyless entry systems as well as in various component identification applications. The KeeLoq algorithm has a 64-bit key and operates on 32-bit blocks. It is based on an NLFSR with a nonlinear feedback function of 5 variables.

In this paper new key recovery attacks on KeeLoq are proposed. The first one has a complexity of about $2^{50.6}$ KeeLoq encryptions. The second attack finds the key in 2^{37} encryptions and works for the whole key space. In our attacks we use the techniques of guess-and-determine, slide, and linear attacks as well as cycle structure analysis. Both attacks need 2^{32} known plaintext-ciphertext pairs.

We also analyze the KeeLoq key management and authentication protocols applied in rolling-code and IFF access systems widely used in real-world applications. We demonstrate several practical vulnerabilities.

Key words: KeeLoq, cryptanalysis, slide attacks, linear cryptanalysis, hopping codes, rolling codes

1 Introduction

A number of NLFSR-based stream ciphers have been recently proposed (e.g. Achterbahn [9] and [8], Grain [10]) and successfully cryptanalyzed using linear [2] and nonlinear [12], [23] approximations of nonlinear feedback functions.

KeeLoq is a block cipher based on an NLFSR with a nonlinear boolean feedback function of 5 variables. The algorithm uses a 64-bit key and operates on 32-bit blocks. Its architecture consists of two registers (a 32-bit text register and a 64-bit key register), which are rotated in each of 528 encryption cycles, and of a nonlinear function (NLF) providing nonlinear feedback. One bit of the key is added to the output of the NLF modulo 2 in each cycle.

The light-weight architecture of the KeeLoq cipher allows for an extremely low-cost and efficient hardware implementation (about 700 GE and 528 clock cycles per block). This contributed to the popularity of the KeeLoq cipher among designers of remote keyless entry systems, automotive and burglar alarm systems, automotive immobilizers, gate and garage door openers, identity tokens, component identification systems. For instance, the KeeLoq block cipher is used by such automotive OEMs as Chrysler, Daewoo, Fiat, GM, Honda, Toyota, Volvo, VW, Jaguar [26] and in the HomeLink wireless control systems to secure communication with garage door openers [11]. The KeeLoq technology supplied by Microchip Technology Inc. includes the KeeLoq cipher and a number of authentication protocols as well as key management schemes. Our description of KeeLoq is based on the newly published article [26], [15] and a number of the manufacturer's documents [14], [16], [20], [21].

Our contribution. The contribution of the paper is many-fold. First, a new technique to perform recovery attacks on KeeLoq is proposed. Our direct attack recovers the key in $2^{50.6}$. Second, the techniques allow us to propose an extended attack of complexity 2^{37} working for the whole key space. Then severe vulnerabilities of the KeeLoq protocols and key management systems are demonstrated.

Our cryptanalysis of the KeeLoq algorithm is based on the following weaknesses of the KeeLoq structure: The key schedule is self-similar, which allows us to mount a slide attack [4], [5], [3]. It is supported by the existence of an efficient linear approximation of the NLF used to recover a part of the key. Then the remainder of the key bits is obtained using other linear relations within KeeLoq.

The key recovery complexity of our first attack is $2^{50.6}$. The attack requires 2^{32} plaintext-ciphertext pairs and a memory of 2^{32} 32-bit words. Several computing devices can share the memory during the attack. All computations are perfectly parallelizable. The property inherited from the slide attacks [4], [5] is that the complexity of our attack is independent of the number of encryption cycles, which is as a rule not the case for linear or differential cryptanalysis, where the complexity often grows exponentially with the number of iterations.

The second attack is an extension of our first attack by using the cycle structure analysis introduced in [7]. Our attack finds the key in 2^{37} steps. It also requires 2^{32} known plaintext-ciphertext pairs and a memory to hold 2^{32} 32-bit words. Additionally, 2^{32} bits of memory are needed for exploring the cycle structure of the KeeLoq permutation. Our techniques work for all keys, unlike those in [7] which are applicable to 26% of all keys only. Our second attack is the best known attack on the KeeLoq block cipher working for the whole key space.

We also show how the cryptanalytic attacks on the KeeLoq block cipher apply to the KeeLoq hopping codes and IFF (Identify Friend or Foe) systems supplied by Microchip which are based on this algorithm. It is demonstrated that the attacks pose a real threat for a number of applied key management schemes: After attacking one instance of KeeLoq using attacks presented in this paper, one can reduce the effective key length of all other KeeLoq systems of the same series to 32, 48, or 60 bits depending on the key management scheme applied. In some attack models, the attacker is even able to retrieve the individual encryption key instantly.

The paper is organized as follows. Section 2 describes the KeeLoq algorithm. In Section 3 our basic sliding linear key recovery attack on KeeLoq and its extension with a reduced complexity are presented. In Section 4 we discuss the impact of attacks on the KeeLoq algorithm with respect to the standard KeeLoq real-word applications supplied by Microchip. We conclude in Section 5.

2 Description of the KeeLoq Algorithm

KeeLoq is a block cipher with a 64-bit key which operates on 32-bit words [26], [15]. Its design is based on a nonlinear feedback shift register (NLFSR) of length 32 bits with a nonlinear feedback function of 5 variables. The feedback depends linearly on two other register bits and on the next key bit taken from the rotated key register of length 64 bits.

Let $V_n = \mathrm{GF}(2)^n$ be the set of all n-bit words and $Y^{(i)} = (y_{31}^{(i)}, \ldots, y_0^{(i)}) \in V_{32}$, $y_j^{(i)} \in \mathrm{GF}(2)$, describe the state of the text register in cycle i for $j = 0, \ldots, 31$ and $i = 0, 1, \ldots$ Let also $K^{(i)} = (k_{63}^{(i)}, \ldots, k_0^{(i)}) \in V_{64}$, $k_j^{(i)} \in \mathrm{GF}(2)$, denote the state of the key register in cycle i for $j = 0, \ldots, 63$ and $i = 0, 1, \ldots$ Then each cycle of encryption can be described using the following algorithm (see Figure 1):

Compute the feedback bit: $\varphi = NLF(y_{31}^{(i)}, y_{26}^{(i)}, y_{20}^{(i)}, y_9^{(i)}, y_1^{(i)}) \oplus y_{16}^{(i)} \oplus y_0^{(i)} \oplus k_0^{(i)}$
Rotate text and insert feedback: $R^{(i+1)} = (\varphi, y_{31}^{(i)}, \ldots, y_1^{(i)})$
Rotate key: $K^{(i+1)} = (k_0^{(i)}, k_{63}^{(i)}, \ldots, k_1^{(i)})$.

For encryption the key register is filled with the 64 key bits $K = (k_{63}, \ldots k_0) \in V_{64}$, $k_j \in \mathrm{GF}(2)$, $j = 0, \ldots, 63$, in the straightforward way: $K^{(0)} = K$. If $X = (x_{31}, \ldots, x_0) \in V_{32}$, $x_j \in \mathrm{GF}(2)$, $j = 0, \ldots, 31$, is a block of plaintext, the initial state of the text register is $Y^{(0)} = (x_{31}, \ldots, x_0)$. The output of the algorithm is the ciphertext $Z = (z_{31}, \ldots, z_0) = Y^{(528)} \in V_{32}$, $z_j \in \mathrm{GF}(2)$, $j = 0, \ldots, 31$.

For decryption the key register is filled in the same way: $K^{(0)} = K = (k_{63}, \ldots k_0) \in V_{64}$. But the decryption procedure complements the encryption. One decryption cycle can be defined by the following sequence of operations:

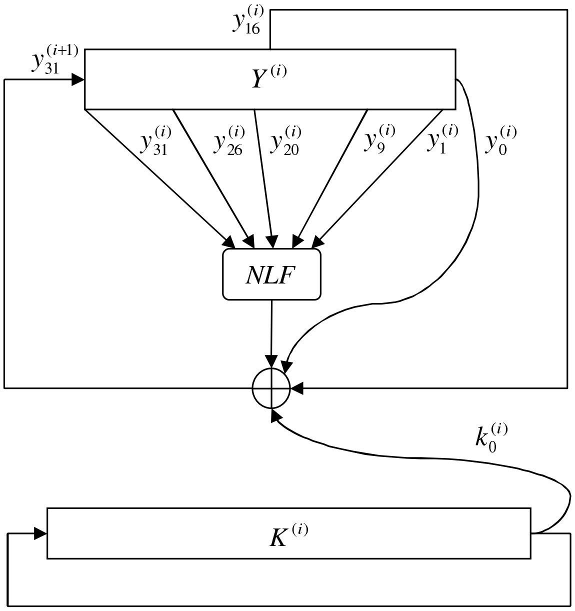

Fig. 1. The i-th KeeLoq encryption cycle

Compute the feedback bit: $\varphi = NLF(y_{30}^{(i)}, y_{25}^{(i)}, y_{19}^{(i)}, y_{8}^{(i)}, y_{0}^{(i)}) \oplus y_{15}^{(i)} \oplus y_{31}^{(i)} \oplus k_{15}^{(i)}$
Rotate text and insert feedback: $R^{(i+1)} = (y_{30}^{(i)}, \dots, y_{0}^{(i)}, \varphi)$
Rotate key: $K^{(i+1)} = (k_{62}^{(i)}, \dots, k_{0}^{(i)}, k_{63}^{(i)})$.

The ciphertext and plaintext are input/output in a similar way: The ciphertext is input into the text register before decryption, $Y^{(0)} = Z$, and the plaintext can be read out after 528 decryption cycles, $Y^{(528)} = X$.

The NLF is a boolean function of 5 variables and is of degree 3. In the specification [15] the NLF is assigned using a table. This corresponds to the following ANF:

$$\begin{aligned} NLF(x_4, x_3, x_2, x_1, x_0) = {} & x_0 \oplus x_1 \oplus \\ & x_0x_1 \oplus x_1x_2 \oplus x_2x_3 \oplus x_0x_4 \oplus x_0x_3 \oplus x_2x_4 \oplus \\ & x_0x_1x_4 \oplus x_0x_2x_4 \oplus x_1x_3x_4 \oplus x_2x_3x_4. \end{aligned} \tag{1}$$

The NLF is balanced and its correlation immunity order is 1, $\mathrm{cor}(NLF) = 1$ [24], [25]. This means that the NLF is 1-resilient [6], which is the maximum for a function of 5 variables with $\deg(NLF) = 3$ due to Siegenthaler's inequality [24]:

$$\deg(NLF) + \mathrm{cor}(NLF) \leq 4.$$

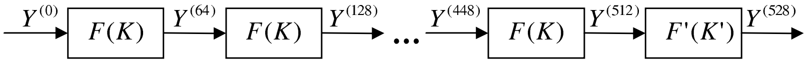

Fig. 2. Round structure of KeeLoq encryption

The KeeLoq algorithm has the following round structure. We define a KeeLoq round as the permutation $F(K) : V_{32} \to V_{32}$ depending on the key $K \in V_{64}$. A KeeLoq quarter round is defined as the permutation $F'(K') : V_{32} \to V_{32}$ depending on the subkey $K' = (k_{15}, \dots, k_0) \in V_{16}$. Then the whole KeeLoq encryption mapping consists of successively computing 8 full round permutations

$F(K)$ and consequently applying the last quarter round permutation $F'(K')$, see Figure 2. Note that the first 8 full rounds are identical. The decryption can be represented in a similar way using inverse permutations $F'(K')^{-1}$ and $F(K)^{-1}$.

The algorithm allows for an extremely simple hardware implementation comprised of a 32-bit shift register with taps on fixed positions, a 64-bit shift resister with a single tap and a 32-bit (5×1) look-up table (LUT) for the NLF. The LUT can be replaced with the corresponding logical elements according to (1). In this case the hardware implementation of KeeLoq requires about 700 GE and 528 clock cycles per block, which is to be compared to about 1500-2000 GE needed for modern light-weight hardware-oriented stream ciphers such as Grain or Trivium.

3 Attacks on the KeeLoq Algorithm

3.1 Basic Sliding Linear Attack on KeeLoq

Our basic attack is based on the following weaknesses of the algorithm: self-similar key schedule scheme, relatively short blocks of 32 bits, and existence of an efficient linear approximation of the NLF.

The attack can be outlined in the following way. For each subkey $K' = (k_{15}, \ldots, k_0)$ and for a random 32-bit input $I_0 \in V_{32}$ guess the corresponding output $O_0 \in V_{32}$ after the 64 clock cycles which depends on the other 48 key bits $(k_{63}, \ldots, k_{16})$. Using the periodic structure of the KeeLoq key schedule generate several other pairs $(I_i, O_i) \in (V_{32})^2$, $i = 1, \ldots, N-1$ (*sliding step*). For a successful attack N has to be about 2^8. For each number of such pairs we mount a distinguishing attack to obtain linear relations on some unknown key bits with a high probability due to the fact that the KeeLoq NLF is not 2-resilient (*linear correlation step*). In this way it is possible to determine $(k_{47}, \ldots, k_{16})$ bit by bit. After this an input/output pair for 16 encryption cycles can be represented as a triangular system of linear equations with the remaining bits $(k_{63}, \ldots, k_{48})$ of K as variables. It can be solved using 16 simple computational operations (*linear step*).

Sliding step. Using a single input/output pair (I_0, O_0) for the full round of 64 cycles and knowing the first 16 key bits $K' = (k_{15}, \ldots, k_0)$ one can produce an arbitrary number of other input/output pairs for this round. This is possible due to the fact that (almost) all rounds in KeeLoq are identical permutations which is the property on which the slide attacks by Biryukov and Wagner are substantially based [4], [5]. Once a pair (I_0, O_0) is known, the next input/output pair is produced by encrypting I_0 and O_0 with the key to be recovered (it is a chosen plaintext attack) and obtaining (I'_1, O'_1) as ciphertext. Then I'_1 and O'_1 are decrypted using the guessed partial key $K' = (k_{15}, \ldots, k_0)$. The resulting plaintexts form the needed pair (I_1, O_1), see Figure 3. From one pair (I_i, O_i), $i = 0, 1, 2, \ldots$, an arbitrary number of pairs (I_j, O_j), $j > i$ can be derived for a certain value of K' by iteratively encrypting I_i, O_i using KeeLoq and decrypting them with K' (thus, obtaining (I_{j+1}, O_{j+1})). We call the set of pairs (I_i, O_i) needed for determining the key a *pseudo-slide group*.

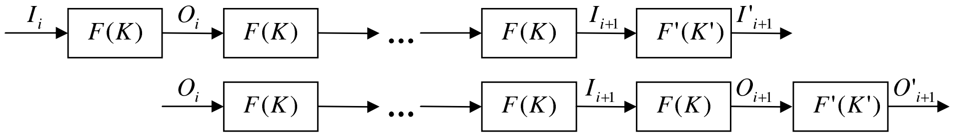

Fig. 3. Generating input/output pairs using sliding techniques

As the sliding has to be performed for each guess of $K' = (k_{15}, \ldots, k_0)$ and each round output (2^{47} times on average) in our basic attack, its complexity is crucial for the efficiency of the whole attack.

Correlation step. Once the pseudo-slide group was generated in the sliding step, the following weakness of the KeeLoq NLF with respect to correlation attacks is used due to the fact that the NLF is 1-resilient, but not 2-resilient.

Lemma 1. *For uniformly distributed $x_4, x_3, x_2 \in GF(2)$ the following holds:*

- $\Pr\{NLF(x_4, x_3, x_2, x_1, x_0) = 0 \mid x_0 \oplus x_1 = 0\} = \frac{5}{8}$,
- $\Pr\{NLF(x_4, x_3, x_2, x_1, x_0) = 1 \mid x_0 \oplus x_1 = 1\} = \frac{5}{8}$.

This means that the NLF can be efficiently approximated by $x_0 \oplus x_1$. So, if x_0, x_1 are known and x_4, x_3, x_2 are random and unknown, we can determine $f(K)$ by statistically filtering out the contribution of $NLF(x_4, x_3, x_2, x_1, x_0)$ to the equation

$$NLF(x_4, x_3, x_2, x_1, x_0) \oplus f(K) = 0$$

using a very limited number of such samples. $f(K)$ is a key-dependent boolean function remaining constant for all samples.

Here we show how to obtain k_{16} and k_{32} from I_i and O_i. The remaining key bits $(k_{47}, \ldots, k_{33})$ and $(k_{31}, \ldots, k_{17})$ can be obtain in the same way by using k_{32}, k_{16} and shifting input and output bits.

We denote $I_i = Y^{(0)}$ and $O_i = Y^{(64)}$ for each i. The idea is to make use of the correlation weakness of the dependency between the output bits $y_0^{(64)}$, $y_1^{(64)}$ and the input bits $Y^{(0)}$. One can compute $Y^{(16)}$ from $Y^{(0)}$, since $K' = (k_{15}, \ldots, k_0)$ is known. For the next bit $y_{31}^{(17)}$, which is the first key-dependent bit, one has the following equation:

$$\begin{aligned} y_{16}^{(32)} = y_{31}^{(17)} &= NLF(y_{31}^{(16)}, y_{26}^{(16)}, y_{20}^{(16)}, y_9^{(16)}, y_1^{(16)}) \oplus y_0^{(16)} \oplus y_{16}^{(16)} \oplus k_{16} = \\ &= c_0 \oplus k_{16}, \end{aligned} \tag{2}$$

where $c_0 \in \mathrm{GF}(2)$ denotes the key-independent part of (2).

After 32 encryption cycles the following holds:

$$(y_{15}^{(32)}, y_{14}^{(32)}, \ldots, y_0^{(32)}) = (y_{31}^{(16)}, y_{30}^{(16)}, \ldots, y_{16}^{(16)}) \in V_{16}.$$

Thus, the least significant half of $Y^{(32)}$ is known. Then $y_0^{(64)}$ can be represented as:

$$\begin{aligned} y_0^{(64)} &= NLF(y_{31}^{(32)}, y_{26}^{(32)}, y_{20}^{(32)}, y_9^{(32)}, y_1^{(32)}) \oplus y_0^{(32)} \oplus y_{16}^{(32)} \oplus k_{32} = \\ &= NLF(y_{31}^{(32)}, y_{26}^{(32)}, y_{20}^{(32)}, y_9^{(32)}, y_1^{(32)}) \oplus y_0^{(32)} \oplus (c_0 \oplus k_{16}) \oplus k_{32}, \end{aligned} \tag{3}$$

where $y_0^{(64)}$, $y_0^{(32)}$, $y_1^{(32)}$, $y_9^{(32)}$, c_0 are known and $y_{31}^{(32)}$, $y_{26}^{(32)}$, $y_{20}^{(32)}$, k_{32}, k_{16} are unknown. As the first two inputs of the NLF are known, its contribution to (3) can be replaced with the random variate ε using Lemma 1:

$$NLF(y_{31}^{(32)}, y_{26}^{(32)}, y_{20}^{(32)}, y_9^{(32)}, y_1^{(32)}) \oplus y_9^{(32)} \oplus y_1^{(32)} = \varepsilon \tag{4}$$

with

$$\Pr\{\varepsilon = 0\} = \frac{5}{8}. \tag{5}$$

Then the following holds:

$$y_0^{(64)} \oplus y_0^{(32)} \oplus c_0 \oplus y_9^{(32)} \oplus y_1^{(32)} = \varepsilon \oplus k_{16} \oplus k_{32}. \tag{6}$$

In order to determine $k_{16} \oplus k_{32}$ one has to distinguish between the following two cases: $k_{16} \oplus k_{32} = 0$ and $k_{16} \oplus k_{32} = 1$. In the first case:

$$\Pr\{y_0^{(64)} \oplus y_0^{(32)} \oplus c_0 \oplus y_9^{(32)} \oplus y_1^{(32)} = 0\} = \frac{5}{8}.$$

Otherwise:

$$\Pr\{y_0^{(64)} \oplus y_0^{(32)} \oplus c_0 \oplus y_9^{(32)} \oplus y_1^{(32)} = 0\} = \frac{3}{8}.$$

Thus, the bias δ of the first random variable with respect to the second one is $\delta = \frac{1}{4}$. Our experiments show that about 2^7 equations (6) for different pairs (I_i, O_i), $i = 0, \dots, 2^7 - 1$, are needed to recover $\alpha = k_{16} \oplus k_{32}$ with an acceptable error probability (for all 32 key-dependent linear combinations to be determined in this way), which agrees[1] with Theorem 6 of [1].

Next we consider $y_1^{(64)}$ and its dependencies from the input and key bits. Similar to (2) one has:

$$\begin{aligned} y_{16}^{(33)} &= NLF(y_{31}^{(17)}, y_{27}^{(16)}, y_{21}^{(16)}, y_{10}^{(16)}, y_2^{(16)}) \oplus y_1^{(16)} \oplus y_{17}^{(16)} \oplus k_{17} = \\ &= NLF(c_0 \oplus k_{16}, y_{27}^{(16)}, y_{21}^{(16)}, y_{10}^{(16)}, y_2^{(16)}) \oplus y_1^{(16)} \oplus y_{17}^{(16)} \oplus k_{17} = \\ &= c_1' \oplus c_2 k_{16} \oplus y_1^{(16)} \oplus y_{17}^{(16)} \oplus k_{17} = c_1 \oplus c_2 k_{16} \oplus k_{17}, \end{aligned} \tag{7}$$

where $c_1' \in \mathrm{GF}(2)$ is the free term of NLF, $c_2 \in \mathrm{GF}(2)$ is its linear term with respect to k_{16}, and $c_1 = c_1' \oplus y_1^{(16)} \oplus y_{17}^{(16)} \in \mathrm{GF}(2)$. Here c_1 and c_2 are known and depend on $Y^{(0)}$. Then the second output bit $y_1^{(64)}$ is represented as follows:

$$\begin{aligned} y_1^{(64)} &= NLF(y_{31}^{(33)}, y_{26}^{(33)}, y_{20}^{(33)}, y_9^{(33)}, y_1^{(33)}) \oplus y_0^{(33)} \oplus y_{16}^{(33)} \oplus k_{33} = \\ &= (\varepsilon \oplus y_9^{(33)} \oplus y_1^{(33)}) \oplus y_0^{(33)} \oplus (c_1 \oplus c_2 k_{16} \oplus k_{17}) \oplus k_{33}, \end{aligned} \tag{8}$$

where the random variate ε is assigned in a way similar to (4) and c_0, c_1, c_2, $y_0^{(33)}$, $y_9^{(33)}$, $y_1^{(33)}$ are known. To determine $k_{17} \oplus k_{33}$ pairs (I_i, O_i) with $c_2 = 0$ are selected[2]. Then ε in (8) is filtered out statistically, which recovers $\beta = k_{17} \oplus k_{33}$. After this the remaining pairs (I_i, O_i) (with $c_2 = 1$) are used to obtain $\gamma = k_{16} \oplus k_{17} \oplus k_{33}$ in the same way. Thus, $k_{16} = \beta \oplus \gamma$ and $k_{32} = \alpha \oplus k_{16}$.

Now k_{16}, k_{32} and $k_{17} \oplus k_{33}$ are known. In the next step we determine $k_{18} \oplus k_{34}$ and $k_{17} \oplus k_{18} \oplus k_{34}$ using the same plaintext/ciphertext pairs (I_i, O_i) and the same statistical recovery method. In this way all 32 key bits $(k_{47}, \dots, k_{16})$ are obtained in only 16 rather simple computational steps.

Linear step and key verification. The remaining key bits $(k_{63}, \dots, k_{48}) \in V_{32}$ can be recovered as follows. As $(k_{47}, \dots, k_0)$ are known, $Y^{(48)}$ can be computed for each pair (I_i, O_i). $y_{16}^{(64)}$ can be expressed as:

$$y_{16}^{(64)} = NLF(y_{31}^{(48)}, y_{26}^{(48)}, y_{20}^{(48)}, y_9^{(48)}, y_1^{(48)}) \oplus y_{16}^{(48)} \oplus y_0^{(48)} \oplus k_{48}, \tag{9}$$

which reveals k_{48} since the entire state $Y^{(48)}$ is known. Now $Y^{(49)}$ can be completely calculated which leads to the value of k_{49} using $y_{17}^{(64)}$, and so on. In this way the rest of the key is recovered.

At the end of the key recovery procedure we expect to obtain a number of key candidates. The upper bound for their average quantity is $2^{64-32} = 2^{32}$ due to the known plaintext unicity distance [13], since the block length is 32 bit and the key length is 64 bit. Thus, we need to verify each key candidate against max. $\lceil \frac{64+4}{32} \rceil = 3$ plaintext-ciphertext pairs for all 528 encryption cycles.

Attack complexity and experiments. The attack consists of the following stages:

- Compute all plaintext-ciphertext pairs for the whole cipher;
- Guess the partial key K' and the output O_0 after one round for some input I_0;
- For each pair of guesses (K', O_0) do the following:
  - Obtain $2^8 - 1$ other pairs (I_i, O_i); thus, the cardinality of the pseudo-slide group is 2^8 for this attack;
  - Determine $k_{16} \oplus k_{32}$ by evaluating c_0 for the first 2^7 pairs of the pseudo-slide group;
  - Determine $(k_{47}, \dots, k_{16})$ by evaluating c_1 and c_2 2^8 times;
  - Determine $(k_{63}, \dots, k_{48})$ by evaluating 2^4 nonlinear boolean functions;

[1] Strictly speaking, the mentioned Theorem 6 cannot be applied here since Assumption 4 of [1] does not hold due to the fact that the mutual bias is relatively large in our case. But this suggests that our experimental estimations are correct.

[2] Note that for random inputs I_i the probability of $c_2 = 0$ is 0.5. Therefore about $N/2$ out of N known pairs (I_i, O_i) will lead to $c_2 = 0$. This raises the required number of plaintext/ciphertext pairs to about 2^8.

- Verify max. 2^{32} candidate keys using at most 3 plaintext-ciphertext pairs for the whole cipher and 3 full encryption operations.

If one step is equivalent to a single full KeeLoq encryption (528 encryption cycles), 2^{32} steps are needed for generating 2^{32} plaintext-ciphertext pairs. Each element has to be stored in a memory of 2^{32} 32-bit words.

For each guess of (I_0, O_0) and K' operations of the following complexity have to be be performed:

– $2^9 - 2$ memory accesses for obtaining (I_i, O_i), $i = 1, \ldots, 2^8 - 1$. We assume a single memory access equivalent to 4 encryption cycles. This leads to approximately 2^2 steps required to perform the memory accesses.
– 2^7 evaluations of c_0 and $k_{16} \oplus k_{32}$, each evaluation being equivalent to one encryption cycle. The complexity of this stage is then $2^7/528 \approx 2^{-2}$ steps.
– $16 \cdot 2^8 = 2^{12}$ evaluations of c_1 and c_2 for determining $(k_{47}, \ldots, k_{16})$. Each evaluation is computationally equivalent to one encryption cycle. This stage requires about $2^{12}/528 \approx 2^3$ steps.
– 2^4 evaluations of a boolean function to determine $(k_{63}, \ldots, k_{48})$. Each evaluation is equivalent to one encryption cycle which leads to a complexity of about $2^4 \cdot 2^{-9} = 2^{-5}$ steps.

Max. 2^{32} candidate keys have to be verified using at most 3 full encryptions which requires max. 2^{34} steps. Thus, the overall computational complexity of the attack is

$$2^{32} + \frac{2^{32} \cdot 2^{16}}{2} \cdot (2^2 + 2^{-2} + 2^3 + 2^{-5}) + 2^{34} \approx 2^{50.6} \text{ steps.}$$

The memory complexity is quite reasonable and is 2^{32} 32-bit words (16 GByte). This enables an attacker to place all plaintext-ciphertext values into RAM which substantially accelerates the implementation of the attack.

Most computations in our attack are perfectly parallelizable: The attacker can distribute the 2^{48} combinations of guesses between a number of computational machines (PCs, FPGAs, ASICs, etc.) sharing the 16 GBytes of memory containing the plaintext-ciphertext pairs.

Attacks of this type are potentially applicable to all NLFSR-based block cipher constructions of this kind. To avoid slide attacks some sort of round dependency should be introduced into the key schedule. Moreover, the correlation immunity order of the nonlinear feedback function has to be increased and the non-existence of efficient nonlinear approximations should be provided.

We have implemented sliding, correlation and linear steps on a standard Pentium M PC with 1.73 GHz and 1 GB RAM for random keys. On average we were able to recover the whole key in about 45000 tact cycles[3] for the correct guesses of (I_0, O_0) and K' as well as for the corresponding slid pairs (I_i, C_i). The implementation was performed in non-optimized C using Microsoft Visual C++ 7 compiler. Moreover, our experiments show that a pseudo-slide groups of about 2^8 pairs (I_i, O_i) is required on average to recover the key correctly.

3.2 Advanced Attack on the KeeLoq Algorithm

In this subsection, we first outline some parallel work on the cryptanalysis of KeeLoq including algebraic attacks and cycle structure analysis. Then we combine our basic sliding linear attack with the cycle structure analysis and obtain the best known attack on KeeLoq, which requires 2^{37} KeeLoq encryptions and works for the whole key space.

Algebraic Attack. Courtois and Bard [7] used the idea of sliding KeeLoq, but employed algebraic techniques to obtain the key from a slid pair. The attack requires only one slid pair. This eliminates the necessity to guess the partial key K'.

The attack works as follows. By exploring 2^{16} random known plaintext-ciphertext pairs, the attacker expects to have at least one slid pair (I_i, I'_{i+1}), (O_i, O'_{i+1}) with $O_i = F(I_i)$. This forms the first

[3] A single encryption step on the same platform requires about 2200 clock cycles.

32 equations of the non-linear system, which are not sufficient for uniquely determining the key. To obtain more equations, the attacker can use the fact that the ciphertexts I'_{i+1} and O'_{i+1} are related by $F'[F[F'^{-1}[I'_{i+1}]]] = O'_{i+1}$ (see Figure 3), which gives the other 32 binary equations. That is, the ciphertexts in the slid pair are one round (64 KeeLoq cycles) apart in the same KeeLoq cipher with K rotated by 16 bits - $(k_{16}, k_{17}, \ldots, k_{63}, k_0, \ldots, k_{15})$.

After this, the 64 equations are solved using the SAT solver MiniSat. The procedure has to be performed 2^{31} times on average (for each combination of the available plaintext-ciphertext pairs). The total complexity of the attack is about 2^{53} KeeLoq encryptions and it requires only 2^{16} known plaintexts. However, our basic attack is faster than this algebraic attack, though requiring more known plaintexts.

Cycle structure attack. The second attack from [7] makes use of the specific cycle structure imposed by the fact that the first 8 KeeLoq rounds (64 clock cycles each) are exactly the same.

For a random permutation on n-bit words, there are on average about $\ln 2^n$ cycles. That is, for $n = 32$ the expected number of cycles is about 22, approximately half of them being even. If the same permutation is applied twice, the cycles of even size split into two classes - even and odd cycles. The odd cycles of the original permutation persist.

In KeeLoq, the same permutation F, which we consider as random, is applied 8 times (8 full rounds of KeeLoq) followed by a quarter round F'. That is, the permutation $F^8(\cdot)$ has about $22/2^{\log 8} = 2.75$ cycles of even size. At the same time, a random permutation would have about 11 cycles of even size.

Thus, one can determine the correct value of K' by decrypting all ciphertexts one quarter round and counting the number of cycles of even size. If there are more than 6 even cycles, this can be seen as a random permutation and the guess of K' is wrong. Otherwise, the guessed K' is correct. This test allows one to determine the correct value of K' with a high probability. The complexity of this step is about 2^{37} KeeLoq encryptions.

Our extended attack. Now we can determine the quarter key $K' = (k_{15}, \ldots, k_0)$ using the cycle structure attack described above. This requires 2^{32} known plaintext-ciphertext pairs and about 2^{37} KeeLoq encryptions. Then we just apply our basic attack from Section 3.1.

Actually, we do not have to perform the whole attack, as the first 16 key bits are already known. The attacker chooses two plaintext-ciphertext pairs and builds the corresponding pseudo-slide group of size 2^8. Then the correlation and linear steps of our basic attack are directly applied to determine the remaining 48 bits of the key. This operation has to be performed for approximately 2^{31} random plaintext-ciphertext pairs to achieve a high success probability. That is, the complexity of these steps is about 2^{33} KeeLoq encryptions.

Thus, we built an attack of complexity 2^{37} operations working for the whole key space and requiring 2^{32} known plaintexts. This is the best known attack on the KeeLoq block cipher.

Though it might seem that the data requirements make the attack unpractical, we show in the next section that our attacks can have practical relevance due to some severe weaknesses of the key management schemes used in real-world KeeLoq systems.

4 Attacks on KeeLoq-based Systems in Practice

4.1 KeeLoq Protocols

The typical applications of KeeLoq are the car anti-theft systems. Here the car ignition key authenticates itself to the car. For instance, the KeeLoq block cipher is used by such automotive OEMs as Chrysler, Daewoo, Fiat, GM, Honda, Toyota, Volvo, VW, Jaguar [26] and in the HomeLink wireless control systems to secure communication with garage door openers [11]. Other automotive KeeLoq applications include component authentication, vehicle-to-garage authentication, etc. KeeLoq is also used in various access control and property identification systems. Below three major types of security protocols are outlined in which the KeeLoq block cipher is involved.

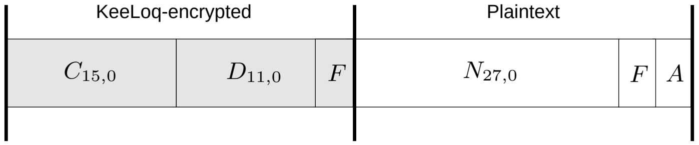

Fig. 4. Typical message structure of KeeLoq hopping codes

KeeLoq hopping codes. These are also known as rolling codes and provide authentication of an encoder to the decoder (the main system) by sending an encrypted counter value (unilateral communication) [14]. The encoder and decoder share a 64-bit symmetric key and a 16-bit synchronized counter value. To authenticate itself the encoder encrypts the next counter value and sends it to the decoder which decrypts the message and verifies whether the received counter value is within the open window of length 16. A resynchronization mechanism exists to repair communication in case the counter value received exceeds the bounds of the open window. See also [16], [17], [19].

A typical structure of the message sent by the transmitter to the main systems can be found in Figure 4. The message consists of two parts: ciphertext and plaintext. The ciphertext part is the 32-bit output of the KeeLoq encryption. The input to the encryption is a concatenation of the 16-bit counter[4] $C_{15,0}$, a constant 12-bit discrimination value $D_{11,0}$ and 4 functional bits $F = F_{3,0}$. The non-encrypted part of the message contains the 28-bit individual encoder identifier $N_{27,0}$, the 4 functional bits F and several auxiliary bits A. The decoder receives the message, invokes the encoder key corresponding to $N_{27,0}$, and decrypts the ciphertext using this key. Then it verifies the discrimination value $D_{11,0}$ and the counter $C_{15,0}$. If the values are correct, the authentication is accepted.

KeeLoq IFF. The IFF (Identify Friend or Foe) systems provide authentication of a transponder to the main systems (decoder) using a simple challenge-response protocol (bilateral communication), see [20]. The transponder and decoder share a 64-bit symmetric key K. The encoder sends its 28-bit identifier $N_{27,0}$ to the main system. To require authentication the decoder sends a 32-bit random challenge $R = R_{31,0}$ to the transponder that replies with the corresponding KeeLoq-encrypted challenge using K. The decoder encrypts the genuine challenge using K corresponding to $N_{27,0}$ and compares the message received as a reply with this value. If they coincide, the authentication is accepted. See also [18].

PIC12F635/PIC16F636/639. Proprietary protocols based on PIC-controllers PIC12F635, PIC16F636, PIC16F639 [22], [21] supplied by Microchip which are equipped with a hardware module implementing the KeeLoq block cipher.

4.2 KeeLoq Key Management Schemes

The KeeLoq systems use a number of different key management mechanisms depending on the concrete model of encoder/decoder. In all these schemes, an individual encoder key is derived from a manufacturer's key MK and some encoder-specific information in some way during the learning phase. The individual (KeeLoq encryption) key K is stored in the EEPROM of the encoder. The manufacturer's key MK is stored in the ROM and is fixed for large series of encoders. This enables each manufacturer to produce encoders that cannot be cloned by competitors. For example, MK can remain the same for all immobilizers installed in a certain car model within one production year. This fact makes the manufacturer's key a valuable attack target. It turns out that one can deduce some information about the manufacturer key from an individual encoder key which can be found using the cryptanalytic attacks

[4] Throughout this section we denote the selection of bits i to j, $i > j \geq 0$, of a number A as $A_{i,j}$. For example, $K_{63,32}$ denotes the 32 most significant bits of the individual 64-bit key K.

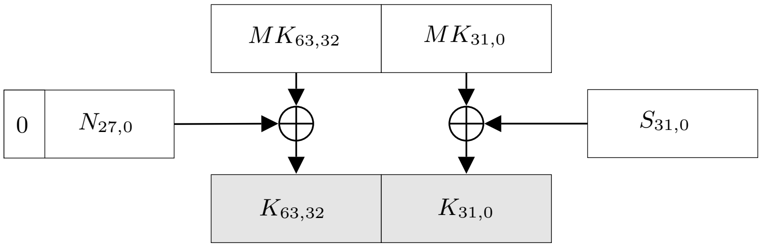

Fig. 5. XOR-based secure key generation with a 32-bit seed

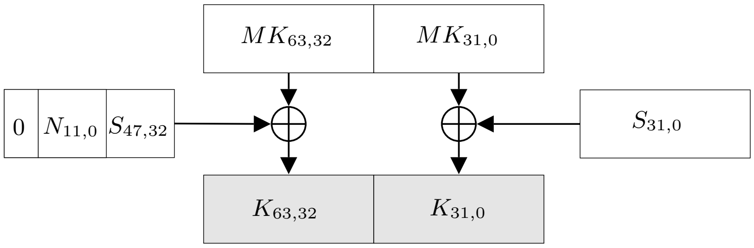

Fig. 6. XOR-based secure key generation with a 48-bit seed

on the KeeLoq block cipher described above. The concrete amount of gained information depends on the specific key derivation function.

There are two classes of key derivation functions used in KeeLoq systems: normal key generation and secure key generation. In this paper, we concentrate on the XOR-based secure key generation and do not treat the normal key generation. The secure key derivation procedure has a variety of modes. Their general feature is that the individual encoder key depends on a randomly looking but fixed seed stored in the encoder and sent to the main system at the learning stage. The modes differ with respect to the length of the seed used.

To derive an individual key according to XOR-based secure key generation method, the encoder uses a seed S, its 28-bit encoder identifier $N_{27,0}$ and the 64-bit manufacturer's key $MK = MK_{63,0} = MK_{63,32}|MK_{31,0}$. Here $MK_{63,32}$ and $MK_{31,0}$ are the most significant and least significant 32-bit parts of MK, respectively.

There are three modes of the KeeLoq XOR-based secure key generation:

- *32-bit seed*: The 64-bit individual encoder key $K = K_{63,0} = K_{63,32}|K_{31,0}$ is obtained by XORing $MK_{31,0}$ with the seed $S = S_{31,0}$ and $MK_{63,32}$ with the zero-padded $N_{27,0}$ (Figure 5).
- *48-bit seed*: The seed $S_{47,0}$ is split into two parts - $S_{47,32}$ and $S_{31,0}$. $K_{31,0}$ is obtained by XORing $MK_{31,0}$ with $S = S_{31,0}$. $K_{63,32}$ is the XOR of the zero-padded $N_{11,0}|S_{47,32}$ with $MK_{31,0}$ (Figure 6).
- *60-bit seed*: In this case, the individual encoder key K does not depend on the encoder identifier SN. K is obtained by XORing MK with the zero-padded $S = S_{59,0}$ (Figure 7).

4.3 Attacking KeeLoq Hopping Codes and IFF Systems

The protocols and key derivation schemes described above allow one to deduce some information about the manufacturer's key MK from a single individual encoder key.

Attack on KeeLoq IFF. In the case of KeeLoq IFF, the attacker can freely collect plaintext-ciphertext pairs choosing challenges and recording the responses. The PIC16 controllers run at a frequency of 4 MHz. As the KeeLoq cipher is implemented in hardware, every of its 528 clocks takes

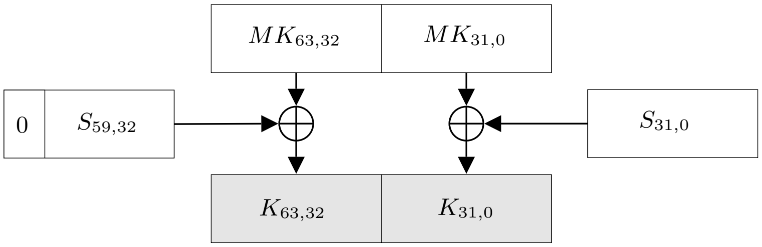

Fig. 7. XOR-based secure key generation with a 60-bit seed

about 2^{-18} s. That is, it can be theoretically possible to encrypt all 2^{32} plaintexts within about 100 days on a PIC16 controller. Other controllers using KeeLoq can be clocked faster, thus reducing the time needed to collect the plaintext-ciphertext pairs. For instance, if a KeeLoq implementation is clocked with a moderate frequency of 64 MHz, the attacker can theoretically collect all needed pairs within about 6 days. Note that the encoder identifier is known as it is transmitted in plaintext before each protocol run.

Thus, one can apply the advanced sliding linear attack with complexity 2^{37} working for the whole key space (Section 3.2) to the collected 2^{32} pairs and recover the individual encoder key K. As $N_{27,0}$ is known, the attacker obtains 32, 16, or 4 bits of MK by XORing the found K with $N_{27,0}$ or simply taking the 4 most significant bits, depending on the used key derivation function. This reduces the security of all other KeeLoq systems using the same manufacturer's key MK to 32, 48, or 60 bits, respectively.

Moreover, the attacker can have access to the seed used. For instance, he can intercept the transmission of the seed from the encoder to the decoder during the learn procedure, as the seed is sent in plaintext. Note that the attacker can also put an encoder into the learn mode by activating some of the PIC16 pins. In this case, the attacker pulls off the seed from the individual encoder key by XORing these two numbers and obtains the full manufacturer's key MK. This all other KeeLoq systems using the same manufacturer's key totally insecure, if the corresponding seeds are known.

Attack on KeeLoq hopping codes. As the discrimination value $D_{11,0}$ is a constant, there are at most 2^{16} input-output texts available, F also remaining constant and the counter $C_{15,0}$ running through $\{0,\ldots,2^{16}-1\}$. The ciphertexts can be obtained directly from the communication channel. The corresponding plaintexts can be deduced, if the discrimination value is known. Note that it takes the encoder at most one hour to generate 2^{16} ciphertexts (even including the wireless data transmission, each session of which requires at most 50 ms). Moreover, the encoder identifier is known to the attacker, since it is transmitted in plaintext with every hopping code.

Having collected all needed plaintext-ciphertext pairs, the attacker can launch the algebraic attack mentioned above in Section 3.2 that is very well parallelizable and finds K in 2^{53} KeeLoq operations.

The rest of the attack works as described for KeeLoq IFF above, resulting in the reduction of the security to 32, 48, 60, or 0 bits, depending on the used key generation method and assumed security model.

5 Conclusion

In this paper we proposed practical key-recovery attacks on the KeeLoq block cipher used in numerous automotive applications as well as in various property identification systems.

Our cryptanalysis uses techniques of guess-and-determine, sliding, linear and cycle structure attacks. Our basic attack works with complexity of $2^{50.6}$ KeeLoq encryptions (while KeeLoq uses a 64-bit key). It requires all 2^{32} plaintexts and a memory of 2^{32} 32-bit words. Our second attack uses the cycle structure techniques to determine the first 16 bits of the key and then applies our first attack. It has a complexity of 2^{37} encryptions and requires 2^{32} known plaintexts, while being applicable to the whole key space. This is the best known attack on the KeeLoq block cipher.

We demonstrated that several real-world applications are vulnerable to attacks on the KeeLoq block cipher, including KeeLoq IFF systems and KeeLoq hopping codes. After attacking one instance of KeeLoq using attacks presented in this paper, one can reduce the effective key length of all other KeeLoq systems of the same series to 32, 48, or 60 bits depending on the key management scheme applied. In some cases, the attacker is even able to obtain the encryption key instantly. All this imposes a real threat on the KeeLoq systems.

References

1. T. Baigneres, P. Junod, and S. Vaudenay. How Far Can We Go Beyond Linear Cryptanalysis? In *Proc. of ASIACRYPT'04*, volume 3329 of *LNCS*. Springer-Verlag, 2004.
2. C. Berbain, H. Gilbert, and A. Maximov. Cryptanalysis of Grain. In *Proc. of FSE'06*, volume 4047 of *LNCS*. Springer-Verlag, 2006.
3. E. Biham, O. Dunkelman, and N. Keller. Improved Slide Attacks. In *FSE'07*, LNCS. Springer-Verlag, 2007.
4. A. Biryukov and D. Wagner. Slide Attacks. In *Proc. of FSE'99*, volume 1636 of *LNCS*. Springer-Verlag, 1999.
5. A. Biryukov and D. Wagner. Advanced Slide Attacks. In *Proc. of EUROCRYPT'00*, volume 1807 of *LNCS*. Springer-Verlag, 2000.
6. B. Chor, O. Goldreich, J. Hastad, J. Fridman, S. Rudich, and R. Smolensky. The Bit Extraction Problem or *t*-Resilient Functions. In *26th Symposium on Foundations of Computer Science*, 1985.
7. N. Courtois and G.V. Bard. Algebraic and Slide attacks on KeeLoq. Available at `http://eprint.iacr.org/2007/062`, February 2007.
8. B. Gammel, R. Goettfert, and O. Kniffler. Achterbahn-128/80. Available from `http://www.ecrypt.eu.org/stream/p2ciphers/achterbahn/achterbahn_p2.pdf`, June 2006.
9. B. M. Gammel, R. Goettfert, and O. Kniffler. The Achterbahn Stream Cipher. Available from `http://www.ecrypt.eu.org/stream/ciphers/achterbahn/achterbahn.pdf`, April 2005.
10. M. Hell, T. Johansson, and W. Meier. Grain - A Stream Cipher for Constrained Environments. Available from `http://www.ecrypt.eu.org/stream/ciphers/grain/grain.pdf`, 2005.
11. HomeLink. Homelink and KeeLoq-based Rolling Code Garage Door Openers. Available from `http://www.homelink.com/home/keeloq.tml`, 2006.
12. T. Johansson, W. Meier, and F. Muller. Cryptanalysis of Achterbahn. In *Proc. of FSE'06*, volume 4047 of *LNCS*. Springer-Verlag, 2006.
13. A. Menezes, P. van Oorshot, and S. Vanstone. *Handbook of Applied Cryptography.* CRC Press, 1996.
14. Microchip. An Introduction to KeeLoq Code Hopping. Available from `http://ww1.microchip.com/downloads/en/AppNotes/91002a.pdf`, 1996.
15. Microchip. Hopping Code Decoder using a PIC16C56, AN642. Available from `http://en.wikipedia.org/wiki/KeeLoq` and `http://www.keeloq.boom.ru/decryption.pdf`, 1998.
16. Microchip. HCS101 Fixed Code Encoder Data Sheet. Available from `http://ww1.microchip.com/downloads/en/DeviceDoc/41115c.pdf`, 2001.
17. Microchip. HCS301 KeeLoq Code Hopping Encoder Data Sheet. Available from `http://ww1.microchip.com/downloads/en/devicedoc/21143b.pdf`, 2001.
18. Microchip. HCS410 Keeloq Code Hopping Encoder and Transponder. Available from `http://ww1.microchip.com/downloads/en/DeviceDoc/40158e.pdf`, 2001.
19. Microchip. HCS301 KeeLoq Code Hopping Encoder Data Sheet. Available from `http://ww1.microchip.com/downloads/en/devicedoc/21143b.pdf`, 2002.
20. Microchip. Using KeeLoq to Validate Subsystem Compatibility, AN827. Available from `http://ww1.microchip.com/downloads/en/AppNotes/00827a.pdf`, 2002.
21. Microchip. PIC12F635/PIC16F636/PIC16F639 Cryptographic Module General Overview, TB086. Available from `http://ww1.microchip.com/downloads/en/DeviceDoc/91086A.pdf`, 2005.
22. Microchip. PIC12F635/PIC16F636/639 Data Sheet. Available from `http://ww1.microchip.com/downloads/en/DeviceDoc/41232B.pdf`, 2005.
23. M. N. Plasencia. Cryptanalysis of Achterbahn-128/80. Available from `http://www.ecrypt.eu.org/stream/papersdir/2006/055.pdf`, November 2006.

24. T. Siegenthaler. Correlation-immunity of Nonlinear Combining Functions for Cryptographic Applications. *IEEE Trans. on Inform. Theory*, IT-30(5), 1984.
25. T. Siegenthaler. Decrypting a Class of Stream Ciphers Using Ciphertext Only. *IEEE Trans. on Computers*, 34(1), 1985.
26. Wikipedia. Keeloq algorithm. Available from `http://en.wikipedia.org/wiki/KeeLoq`, November 2006.