Algebraic and Slide Attacks on KeeLoq

Nicolas T. Courtois[1], Gregory V. Bard[2], and David Wagner[3]

[1] University College London, Gower Street, London WC1E 6BT, UK,
[2] Fordham University, NY, USA
[3] University of California - Berkeley, Berkeley CA 94720, USA

Abstract. KeeLoq is a block cipher used in wireless devices that unlock the doors and alarms in cars manufactured by Chrysler, Daewoo, Fiat, GM, Honda, Jaguar, Toyota, Volvo, Volkswagen, etc [8, 9, 33, 34]. KeeLoq is inexpensive to implement and economical in gate count, yet according to Microchip [33] it should have "a level of security comparable to DES". In this paper we present several distinct attacks on KeeLoq, each of them is interesting for different reasons. First we show that when about 2^{32} known plaintexts are available, KeeLoq is very weak and for example for 30 % of all keys the full key can be recovered with complexity of 2^{28} KeeLoq encryptions. Then we turn our attention to algebraic attacks with the major challenge of breaking KeeLoq given potentially a very small number of known plaintexts.

Our best "direct" algebraic attack can break up to 160 rounds of KeeLoq. Much better results are achieved in combination with slide attacks. Given about 2^{16} known plaintexts, we present a slide-algebraic attack that uses a SAT solver with the complexity equivalent to about 2^{53} KeeLoq encryptions. To the best of our knowledge, this is the first time that a full-round real-life block cipher is broken using an algebraic attack.

Keywords: block ciphers, unbalanced Feistel ciphers, slide attacks, algebraic cryptanalysis, Gröbner bases, SAT solvers, KeeLoq.

1 Introduction

KeeLoq is a lightweight block cipher designed in the 1980's and in 1995 it was sold to Microchip Technology Inc for more than 10 million US dollars as documented in [8]. Following [35], the specification of KeeLoq that can be found in [34] is "not secret" but is patented and was released only under license. In 2007, a Microchip document with the specification of KeeLoq has been made public on a Russian web site [34].

KeeLoq operates with 32-bit blocks and 64-bit keys. Compared to typical block ciphers that have a few carefully-designed rounds, this cipher has 528 extremely simple rounds. KeeLoq is not a stream cipher, it does not actually use any LFSR, and the construction only resembles an NLFSR. Therefore it is not trivial to see whether KeeLoq will be vulnerable to algebraic attacks. KeeLoq is a full-fledged "unbalanced Feistel" block cipher of compressing type, and we anticipate from the Luby-Rackoff theory, that such ciphers are secure if the number of rounds is sufficient. Is 528 rounds sufficient? As it turns out, KeeLoq has been designed to be fast, and requires a low number of gates to be implemented. This is quite interesting as it has been sometimes conjectured that ciphers which require a small number of gates should be vulnerable to algebraic cryptanalysis, see [19, 13, 18, 15]. Currently algebraic attacks on block ciphers are not very powerful, for example for DES [19] and a toy cipher called CTC [17], only respectively 6 and 10 rounds can be broken, and further progress seems difficult. According to [33], KeeLoq should have "a level of security comparable to DES". In this paper we will see that the simplicity of KeeLoq makes it directly breakable by simple algebraic attacks for up to 160 rounds out of 528.

The key property that allows further more efficient attacks on KeeLoq is a sliding property: KeeLoq has a periodic structure with a period of 64 rounds. This in combination with an algebraic attack will allow us to recover the complete key for the full 528-round cipher given 2^{16} known plaintexts.

KeeLoq has unusually small block length: 32 bits. Thus, in theory the attacker can expect to recover and store the whole code-book of 2^{32} known plaintexts. Then one may wonder whether it is really useful to recover the key, as the code-book allows one to encrypt and decrypt any message. However, there are many cases in which it remains interesting for example if a master key can be recovered. We will see that given the whole code-book KeeLoq is spectacularly weak and the key will be recovered in time equivalent to about 2^{28} KeeLoq encryptions.

This paper is organised as follows: in Section 2 we describe the cipher and its usage. In Section 3, we discuss the unusual properties of block ciphers with small blocks and discuss the practical interest of key recovery attacks in this case. In Section 4 we do some preliminary analysis of KeeLoq and recall useful results about random functions. In Section 5 we describe a very fast attack that recovers the key for full KeeLoq given the knowledge of slightly less than the whole code-book. In Section 6 we demonstrate several simple algebraic attacks that work given very small quantity of known/chosen plaintexts and for a reduced number of rounds of KeeLoq. In Section 7 we study combined slide and algebraic attacks that work given about 2^{16} known plaintexts for the full 528-round cipher. In Appendix A we discuss strong keys in KeeLoq. In Appendix B we study the algebraic immunity of the Boolean function used in KeeLoq.

1.1 Notation

We will use the following notation for functional iteration:

$$f^{(n)}(x) = \underbrace{f(f(\cdots f(}_{n \text{ times}}x)\cdots))$$

2 Cipher Description

The specification of KeeLoq can be found in the Microchip product specification document [34], which actually specifies KeeLoq decryption, that can be converted to a description of the encryption, see [8, 9, 4]. Initially there were mistakes in [8, 9] as opposed to [34, 4] but they are now corrected.

The KeeLoq cipher is a strongly unbalanced Feistel construction in which the round function has one bit of output, and consequently in one round only one bit in the "state" of the cipher will be changed. Alternatively it can viewed as a modified shift register with non-linear feedback, in which the fresh bit computed by the Boolean function is XORed with one key bit.

The cipher has the total of 528 rounds, and it makes sense to view that as $528 = 512+16 = 64\times 8+16$. The encryption procedure is periodic with a period of 64 and it has been "cut" at 528 rounds, because 528 is not a multiple of 64, in order to prevent obvious slide attacks (but more advanced slide attacks remain possible as will become clear later). Let $k_{63},\ldots,k_0$ be the key. In each round, it is bitwise rotated to the right, with wrap around. Therefore, during rounds $i, i+64, i+128,\ldots$, the key register is the same. If one imagines the 64 rounds as some $f_k(x)$, then KeeLoq is

$$E_k(x) = g_k(f_k^{(8)}(x))$$

with $g(x)$ being a 16-round final step, and $E_k(x)$ being all 528 rounds. The last "surplus" 16 rounds of the cipher use the first 16 bits of the key (by which we mean $k_{15},\ldots,k_0$) and g_k is a functional"prefix" of f_k (which is also repeated at the end of the whole encryption process). In addition to the simplicity of the key schedule, each round of the cipher uses *only one* bit of the key. From this we see that each bit of the key is used exactly 8 times, except the first 16 bits, $k_{15},\ldots,k_0$, which are used 9 times.

At the heart of the cipher is the non-linear function with algebraic normal form (ANF) given by:

$$NLF(a,b,c,d,e) = d\oplus e\oplus ac\oplus ae\oplus bc\oplus be\oplus cd\oplus de\oplus ade\oplus ace\oplus abd\oplus abc$$

Alternatively, the specification documents available [8], say that it is "the non-linear function 3A5C742E" which means that $NLF(a,b,c,d,e)$ is equal to the i^{th} bit of that hexadecimal number, where $i = 16a+8b+4c+2d+e$. For example $0,0,0,0,1$ gives $i=1$ and the second least significant (second from the right) bit of of "3A5C742E" written in binary.

The main shift register has 32 bits, (unlike the key shift register with 64 bits), and let L_i denote the leftmost or least-significant bit at the end of round i, while denoting the initial conditions as round zero. At the end of round 528, the least significant bit is thus L_{528}, and then let $L_{529}, L_{530},\ldots,L_{559}$ denote the 31 remaining bits of the shift register, with L_{559} being the most significant. The following equation gives the shift-register's feedback:

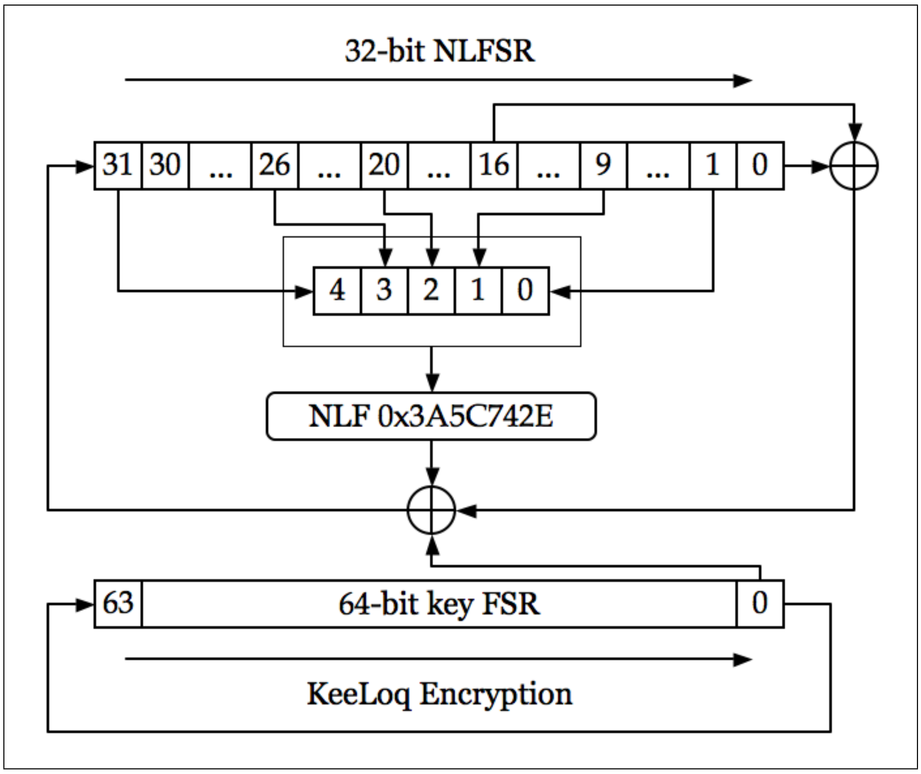
$$L_{i+32} = k_{i \ mod \ 64}\oplus L_i\oplus L_{i+16}\oplus NLF\left(L_{i+31}, L_{i+26}, L_{i+20}, L_{i+9}, L_{i+1}\right)$$

where $k_{63}, k_{62},\ldots,k_1,k_0$ is the original key.

2.1 Cipher Usage

It appears that the mode in which the cipher is used depends on the car manufacturer. One possible method is a challenge-response authentication with a fixed key and a random challenge. Another popular method is to set the plaintext to 0, and increment the key at both sides. Another important mode is a so called 'hopping' or 'rolling' method described in [4, 33]. In this case 16 bits of the plaintext are permanently fixed on both sides, and the attacker cannot hope get more than 2^{16} known plaintexts. More information can be found in [4–6].

In this paper we study the security of the KeeLoq cipher against key recovery attacks given a certain number of known or chosen plaintexts.



1. Initialize with the plaintext: $L_{31}, \ldots, L_0 = P_{31}, \ldots, P_0$
2. For $i = 0, \ldots, 528 - 1$ do
   $$L_{i+32} = k_{i \ mod \ 64} \oplus L_i \oplus L_{i+16} \oplus NLF(L_{i+31}, L_{i+26}, L_{i+20}, L_{i+9}, L_{i+1})$$
3. The ciphertext is $C_{31}, \ldots, C_0 = L_{559}, \ldots, L_{528}$.

Fig. 1. KeeLoq Encryption

3 Block Ciphers with Small Blocks and Large Key Size

Most known block ciphers operate on binary strings and consist of a function $E : \{0,1\}^{\ell_K} \times \{0,1\}^{\ell_P} \rightarrow \{0,1\}^{\ell_P}$. The stereotype is that $\ell_P = \ell_K = \ell_P$, however in practical/industrial applications, this is almost never the case, for example:

- IDEA $\ell_P = 64, \ell_K = 128$.
- DES $\ell_P = 64, \ell_K = 56$.
- Two-key triple DES $\ell_P = 64, \ell_K = 112$.
- AES $\ell_P = 128, \ell_K \in \{128, 192, 256\}$.
- KeeLoq $\ell_P = 32, \ell_K = 64$.

Another stereotype is that, when $\ell_P << \ell_K$, for example in triple-DES and AES-256, the key recovery is of purely academic interest, and an attack on AES-256 that runs in time of say 2^{240} has no practical consequences. However, for cipher such as KeeLoq, the situation is different. The block size is small enough so that the whole code-book can be stored on a PC, and several key recovery attacks are feasible in practice. In what follows we will explain that, when $\ell_P << \ell_K$, and even if more or less the whole code-book is known, recovering the key can have numerous important and practical consequences.

3.1 On Key Recovery Attacks and Ciphers with Small Blocks

The *code-book* of a cipher E under a key k is defined as the set of all 2^{ℓ_P} pairs (P, C) such that $E(k, P) = C$. If $2^{\ell_P} < 2^{\ell_K}$, a natural question is why, would one want to recover the key if it is possible to have the entire code-book? There are several answers to this question depending on the circumstances:

1. If the adversary is a powerful insider, and has an oracle chosen-plaintext access to the cipher, or the whole code-book table stored in memory, the question will be of purely academic interest. Nevertheless, in real life applications, and even for such powerful attackers, the actual key recovery can be very valuable. When there is a master key in the system – it is a common practice in the industry – a successive key recovery would allow to compromise the security of the system on a much wider scale.
2. In most practical scenarios, we rather have a known-plaintext attack, and not all plaintexts will actually arise (e.g. due to padding, specific probability distribution, some values only can appear in the future, etc.). Here the adversary can recover a number of plaintext-ciphertext pairs that can be for example 60 % of all possible pairs, but typically he cannot hope to recover all pairs. Importantly, the value of pairs he doesn't have may be very large, while the value of pairs he already has can be negligible. For example, a block cipher with small block-size can be used to anonymize records in medical and financial databases. Then, key recovery would allow the adversary to have all possible pairs, some of which may potentially be valuable. A security model for ciphers with small blocks was recently studied by Granboulan and Pornin in Section 5 of [28]: in this model, even if the adversary has the whole code-book except images of two points, his goal is to recover the images of the missing two points, which can be still very hard.

3. In many real-life situations, the code-book can be noisy, and contain errors. This can be because of transmission errors, human errors such as selecting the wrong encryption key, inadvertent interference with another system or another (active) attacker, or a defensive voluntary injection of dummy messages to frustrate the attackers. Then again, the key recovery may be the only way to know which messages were genuine.

In an extreme scenario, the whole code-book is known, but not with certainty, and a confirmation is sought. If the reader doubts the practicality of this scenario, consider the following. In 1942, the United States decrypted many messages encrypted with the famous Japanese cipher known as "Purple", forecasting an attack at "AF." Making sure that "AF" was in fact the Midway Island (which was anticipated but there was no certitude) had a pivotal impact on winning the World War II. For more details we refer to David Kahn [31].

4 Preliminary Analysis and Useful Combinatorial Facts

4.1 Preliminary Analysis of KeeLoq

Fact 4.1. Given (x, y) with $y = h_k(x)$, where h_k represents up to 32 rounds of KeeLoq, one can find the part of the key used in h_k in as much time as it takes to compute h_k.

Justification: This is because for up to 32 rounds, all state bits between round i and round $i-1$ are directly known. More precisely, after the round i, $32-i$ bits are known from the plaintext, and i bits are known from the ciphertext, for all $i = 1, 2, \ldots, 32$. Then the key bits are obtained directly: we know all the inputs of each NLF, and we know the output of it XORed with the corresponding key bit. We simply have $k_{i-32} = L_i \oplus L_{i-32} \oplus L_{i-16} \oplus NLF\,(L_{i-1}, L_{i-6}, L_{i-12}, L_{i-23}, L_{i-30})$. This also shows that there will be exactly one possible key.

Remark: For more rounds it is much less simple, yet as shown in Section 6, direct algebraic attacks allow to efficiently recover the key for up to 160 rounds.

Fact 4.2. Given (x, y), one can quickly test whether it is possible that $y = g_k(x)$ for 16 rounds. The probability that a random (x, y) will pass this test is $1/2^{16}$.

Justification: After 16 rounds of KeeLoq, only 16 bits of x are changed, and 16 bits of x are just shifted. If data is properly aligned this requires a 16-bit equality test that should take only 1-2 CPU clocks.

Fact 4.3. Given (x, y) with $y = h_k(x)$, where h_k represents 48 rounds of KeeLoq, one can find all 2^{16} possible keys for h_k in as much time as 2^{16} times the time to compute h_k.

Justification: Try exhaustively all possibilities for the first 16 key bits and apply Fact 4.1.

Fact 4.4. For full KeeLoq, given a pair (p, c) with $c = E_k(p)$, it is possible to very quickly test whether p is a possible fixed point of f_k^8. All fixed points will be accepted; all but $1/2^{16}$ of the non-fixed points will be rejected.

Justification: If p is a fixed point of f^8, then $c = g_k(p)$. We simply use Fact 4.2 to test whether it is possible that $c = g_k(p)$.

4.2 Useful facts About Fixed Points and Random Permutations

Proposition 4.1. Given a random function from n-bits to n-bits, the probability that a given point y has i pre-images is $\frac{1}{ei!}$, when $n \to \infty$.
Justification: Let y be fixed, the probability that $f(x) = y$ is $1/N$ where $N = 2^n$ is the size of the space. We get $\binom{N}{i}(1 - 1/N)^{N-i} 1/N^i \approx \frac{1}{ei!}$ when $N \to \infty$. This is a Poisson distribution with the average number of pre-images being $\lambda = 1$.

This fact can be applied to derive statistics on the expected number of fixed points of permutations tat we encounter in cryptanalysis. In particular let $f_k(x)$ be the first 64 rounds of KeeLoq. Assuming that $f_k(x) \oplus x$ is a pseudo-random function, we look at the number of pre-images of 0 with this function. This gives immediately:

Proposition 4.2. The first 64 rounds f_k of KeeLoq have 1 or more fixed points with probability $1 - 1/e \approx 0.63$.

Proposition 4.3. Assuming f_k behaves as a random permutation, the expected number of fixed points of f_k^8 is exactly 4.
Justification: A random permutation π has 1 fixed point on average. Then in addition to possible "natural" fixed points (1 on average) the π^8 will also "inherit" all fixed points of π^2, π^4 and π^8, that for large permutations are with very high probability all distinct fixed points. A rigourous proof of this fact can be obtained from the authors, see [22].

Proposition 4.4. The probability that i) first 64 rounds f_k of KeeLoq have 1 or more fixed points, and simultaneously ii) the 512 rounds f_k^8 have $j = 4$ or more fixed points, is $e^{-1} - \frac{13}{6}e^{-15/8}$ which is about 0.29985.
Justification: Roughly, from Proposition 4.3, we expect about half of 0.63. In [22] we show how one compute such probabilities exactly with the methods of modern analytic combinatorics, see [25, 38, 22].

5 Attacks on KeeLoq that Use the Whole Dictionary

We will now present an attack that is extremely fast and shows that KeeLoq is very weak when (about) the whole code-book is known. In this paper we present two relatively simple versions of this attack. Additional improved versions are described in [22].

5.1 Setup and Assumptions

We assume that one can iterate through all possible 2^{32} plaintexts. This can either be obtained from a remote encryption oracle, or simply harnessing the circuitry without being able to read the key in order to clone the device. While this may sound like a practical attack scenario, it is hard to imagine a hacker patient enough to get 2^{32} known plaintext-ciphertext pairs from the device knowing that brute force is actually feasible. For simplicity we will assume that all the plaintext-ciphertext pairs are stored in a table. This requires 16 Gigabytes of RAM which is now available on a high-end PC. We also assume that the time to get one pair is about $t_r = 16$ CPU clocks. This is a realistic and conservative estimation.

In our Slide-Determine Attack we make the following assumption.

Fixed Point Assumption. We assume that there is at least one fixed point for $f_k(x)$ where $f_k(x)$ represents the first 64 rounds of the cipher. As shown in Section 4.2, this happens with probability 0.63. We recall that if x is a fixed point of $f_k(\cdot)$ than x is a fixed point of $f_k^{(8)}(\cdot)$, which are the first 512 rounds of KeeLoq. In fact several additional fixed points for f_k^8 are expected to exist, as on average f_k^8 has 4 fixed points (cf. Proposition 4.3).

The complexity of nearly all known attacks on KeeLoq greatly depends on the number of fixed points for f_k and f_k^8. In our attack that will follow, the more fixed points exist for f_k^8, the fastest can be the overall attack. The version A of our attack works for 63 % of all keys (cf. Proposition 4.2). The version B is faster but works for a smaller fraction of 30 % of all keys (this figure comes from Proposition 4.4). Other versions of this attack can be designed and are described in [22]. In contrast, for 37 % of keys for which f_k has no fixed point whatsoever, all versions of our Slide-Determine attack fail completely. In Appendix A we discuss this situation: there is a large class of "strong keys" for which the cipher is more secure.

5.2 Our Slide-Determine Attack

This attack requires 2^{32} plaintext-ciphertext pairs (p, c). We assume that (at least) one p is a fixed point of f_k. Then, slightly more than 4 of them on average are fixed points for f_k^8 (cf. Proposition 4.3). This attack occurs in three stages.

Stage 1 - Batch Guessing Fixed Points. Following our assumption there is (at least) one p is a fixed point for f_k^8. For each pair (p, c), we use Fact 4.4 to test whether it is possible that $f_k^8(p) = p$; if not, discard that pair. The complexity so far is about $t_r \cdot 2^{32}$ CPU clocks (mostly spent accessing the memory). Only about $2^{16} + 4$ pairs will survive, and all these where p is a fixed point to f_k^8.

Then following Fact 4.1 we can at the same time compute 16 bits of the key with time of about $4 \cdot 16$ CPU clocks (cf. Fact 4.1 and 6.1). To summarize, given the whole code-book and in time of about $t_r \cdot 2^{32}$ CPU clocks by assuming that p is a fixed point to f_k^8, we produce a list of about $\tau_A = 2^{16}$ triples $p, c, (k_{15}, \ldots, k_0)$. Assuming that $t_r = 16$ CPU clocks, the complexity of Stage 1 is about 2^{36} CPU clocks which is about 2^{25} KeeLoq encryptions.

Stage 1B - Filtering (Optional). This stage is optional, it is omitted in version A of our attack, and necessary in version B. We wish to be able to filter out a certain fixed proportion of these 2^{16} cases, so that the complexity of Stage 1 will dominate attack. Let j_8 be the number of fixed points for f_k^8. If $j_8 > 1$, our attack can be improved. If we omit Stage 1B, or if $j_8 = 1$ (which is quite infrequent), then the Stage 3 will dominate the attack, which as we will see later will make it noticeably slower. To bridge this gap we wish to exclude a proportion of all the 2^{16} pairs. The filtering is done as follows.

We store the triples $p, c, (k_{15}, \ldots, k_0)$ in a data structure keyed by $(k_{15}, \ldots, k_0)$. (For instance, we can have an array of size 2^{16}, where A[i] points to a linked list containing all triples such that $(k_{15}, \ldots, k_0) = i$.) It allows us to count, for each value $(k_{15}, \ldots, k_0)$, the number f of triples associated with that partial-key value. In version B, we assume that $j_8 \geq 4$ which occurs for 30 % of all keys (cf. Proposition 4.4). We will then discard all triples such that $(k_{15}, \dots, k_0)$ does not repeat 4 or more times in our list.

The worst-case complexity of Step 2 and Step 3 of our attack will be proportional to the size τ of our list (if all cases are tried). The expected size of the filtered list can be computed as follows: we assume that the keys that appear in this table are the 2^{16} outputs of a random function on 16 bits, that takes as input any of the 2^{16} pairs (p, c). Then following Proposition 4.1, the proportion of $\frac{1}{ei!}$ of keys will appear i times. The total number of keys that will appear 4 or more times is therefore equal to $2^{16} \cdot \sum_{i \geq 4} \frac{1}{ei!}$. However, we have to check all the triples which is more than all the keys. In our list of triples, each of these keys will appear i times (in some triple). In our attack, it is not merely sufficient to find a triple in our list having the correct 16 bits of the key: this is because our list contains several fixed points for f_k, but only about one fixed point for f_k^8 which is necessary to complete further stages of our attack. Accordingly, the expected number of elements to be checked (the size of our list) is $\tau_B = 2^{16} \cdot \sum_{i \geq 4} i \cdot \frac{1}{ei!} \approx 2^{12.4}$. This is the worst-case estimate for the attack version B (which works for 30 % of all keys). On average we only need about half of this number.

Stage 2 - Batch Solving. If we assume that p is a fixed point of f_k, at least one triple in our list is valid. Moreover we expect than less than 2 are valid on average, as we expect on average between 1 and 2 fixed points for f_k (we assumed there is at least one). For each surviving triple, assume that p is a fixed point, so that $c = E_k(p) = g_k(f_k^{(8)}(p)) = g_k(p)$. Note that if $f_k(p) = p$, then $p = h_k(c)$, where h_k represents the 48 rounds of KeeLoq using the last 48 key bits. Then an algebraic attack can be applied to suggest possible keys for this part. If we guess additional 16 bits of the key, such an attack with a SAT solver takes less than 0.1 s. We have found a simpler and faster method to get the same result. We use Fact 4.3 to recover 2^{16} possibilities for the last 48 key bits from the assumption that $p = h(c)$. Combined with the 16 bits pre-computed above for each triple, we get a list of at most 2^{32} possible full keys on 64 bits. We expect to compute only at most $2^{16} \cdot \tau$ of these full keys on 64 bits, before the attack succeeds. This takes time of at most $2^{16} \cdot \tau$ computations of $h_k(\cdot)$ (cf. Fact 4.3). And we have $\tau_A = 2^{16}$ and $\tau_B = 2^{12.4}$ in versions A and B of our attack respectively. Overall Step 2 requires $2^{16} \cdot \tau \cdot 4 \cdot 48$ CPU clocks, which is approximatively $2^{28.6}$ and $2^{25.0}$ KeeLoq encryptions for respective versions A and B.

Early Abort: About half of this number is needed on average, with an early abort strategy as follows: for each of the τ triples we will execute Step 2 and Step 3. If Step 3 recovers and confirms the full key of KeeLoq, we abort the attack.

Stage 3 - Verification. Finally, we test each of these $2^{16} \cdot \tau$ complete keys (which is less or equal to 2^{32}) on one other plaintext-ciphertext pair p', c'. Most incorrect key guesses will be discarded, and only 1 or 2 keys will survive, one of them being correct. With additional few pairs we get the right key with certainty.

In version A, this stage requires up to $2^{16} \cdot \tau_A = 2^{32}$ full 528-round KeeLoq encryptions. which dominates the complexity of the whole attack. In version B, we need at most $2^{16} \cdot \tau_B$ full KeeLoq encryptions.

Complexity Analysis.

The total complexity of the full attack is as follows:

Version A: The worst-case complexities of stages 1, 2 and 3 are 2^{25}, $2^{28.6}$ and $2^{32.0}$ KeeLoq encryptions respectively. The total is is $2^{32.1}$ KeeLoq encryptions.

On average, with early abort of Stages 2 and 3, after trying on average half of τ_A triples, as described above (Step 1 has to be executed in entirety), we get an average complexity of about $2^{31.1}$ KeeLoq encryptions.

Version B: In this version, the worst-case complexities of stages 1, 2 and 3 are 2^{25}, 2^{25} and $2^{28.4}$ KeeLoq encryptions. The total is $2^{28.7}$ KeeLoq encryptions.

On average, we get about $2^{27.7}$ KeeLoq encryptions.

Summary of Our Slide-Determine Attack. To summarize, for 30 % of all keys our Slide-Determine Attack version B allows to recover the key in average time equivalent to about 2^{28} KeeLoq encryptions. Overall, for all 63 % of keys our Slide-Determine Attack version A requires about 2^{31} KeeLoq encryptions on average. No attack of comparable efficiency is known for the remaining 37 % of keys.

6 Direct Algebraic Attacks on KeeLoq

Our goal is to recover the key of the cipher by solving a system of multivariate equations given a small quantity of known, chosen or random plaintexts, as in [13]. Very few such attacks are really efficient on block ciphers. For example DES can be broken for up to 6 rounds by such attacks, see [19]. For KeeLoq, due to its simplicity, up to 160 rounds can be directly attacked, without (not yet) exploiting the sliding properties of the cipher. This is in particular interesting for the 37 % of keys for which all our sliding attacks fail.

6.1 How to Write the Equations

We write equations in a straightforward way: namely by following directly the description of Fig 1. One new variable represents the output of the NLF in the current round. In addition, in order to decrease the degree, we add two additional variables per round, to represent the monomials $\alpha = ab$ and $\beta = ae$, and add equations of the form $\alpha_i = a_i b_i$ and $\beta_i = a_i e_i$.

This means we have:

$$y = NLF(a,b,c,d,e) = d \oplus e \oplus ac \oplus \beta \oplus bc \oplus be \oplus cd \oplus de \oplus \beta d \oplus \beta c \oplus \alpha d \oplus \alpha c$$

which permits us to write

$$\begin{aligned} L_{i+32} &= k_{i \bmod 64} \oplus L_i \oplus L_{i+16} \oplus L_{i+9} \oplus L_{i+1} \\ &\quad \oplus L_{i+31}L_{i+20} \oplus \beta_i \oplus L_{i+26}L_{i+20} \oplus L_{i+26}L_{i+1} \oplus L_{i+20}L_{i+9} \\ &\quad \oplus L_{i+9}L_{i+1} \oplus \beta_i L_{i+9} \oplus \beta_i L_{i+20} \oplus \alpha_i L_{i+9} \oplus \alpha_i L_{i+20} \\ \alpha_i &= L_{i+31}L_{i+26} \\ \beta_i &= L_{i+31}L_{i+1} \end{aligned}$$

These three equations need merely be repeated for each round.

The values of the plaintext, the ciphertext, and a certain number of key bits that we may fix (i.e. guess, cf. Section 6.2) during the attack are written as separate equations (for example we write that $L_31 = 1$ for the leftmost bit of the plaintext). Thus, given r rounds of the cipher, and for each known plaintext, assuming that F bits of the key are known, we will get a system of $3r+32+32+F$ multivariate quadratic equations with $3r + 64 + 32$ variables: these are all the $L_0, \ldots, L_{r+31}$, the key variables k_i, the α_i and the β_i. Out of these the values of $32 + 32 + F$ variables are already known. It should be noted that these equation and monomial counts are exact, and that this system is overdefined. The total number of distinct monomials that appear in these equations is roughly $12r$.

The equations are written for one or several known plaintexts. This will be our known-plaintext attack. In another version, we consider that the cipher is used in the counter mode, i.e. the set of plaintexts forms a set of consecutive integers encoded on 32 bits. This will be called a counter mode attack. Several complete and working examples of equations can be downloaded from [10].

6.2 Direct Algebraic Attacks on KeeLoq vs. Brute Force

The equations of KeeLoq are of very low degree (i.e. 2), and very sparse. One can try to solve with an off-the-shelf computer algebra system such as Magma's implementation of F4 algorithm [24] or Singular's slimgb() algorithm [39]. We have also tried a much simpler method called ElimLin and described in [19]. Another family of techniques are SAT solvers. Any system of multivariate equations is amenable for transformation into a CNF-SAT problem, using the methods of [20].

Fact 6.1. An optimised assembly language implementation of r rounds of KeeLoq is expected to take only about $4r$ CPU clocks.

Justification: See footnote 4 in [4].

Thus, the complexity of an attack on r rounds of KeeLoq with k bits of the key should be compared to $4r \times 2^{k-1}$ which is the expected complexity of the brute force key search. For example, for full KeeLoq, the reference complexity for the exhaustive key search is about 2^{75} CPU clocks. Assuming that the CPU runs at 2.5 GHz, one can execute about 2^{43} CPU clocks per hour. Consider the following example. Suppose we guess 32 key bits for example $k_1 = 0, k_2 = 1, \ldots$. Suppose that the remaining key bits are found on a PC in less than an hour, or $< 2^{43}$ CPU clocks. In reality, the attacker is not given 32 bits of the key. Instead one can guess them and on average 2^{31} such guesses must be made. With early abort of unsuccessful tries after for example 1.5 hours, the expected running time is $< 2^{43}2^{31+1}$ or $< 2^{75}$, which is faster than brute force.

Note: In the real life hackers recover the KeeLoq key by brute force with FPGAs which takes about two weeks, see [8].

6.3 Frontal Assault – Elimination and Gröbner Bases Attacks

Example 1. For example, we consider 64 rounds of KeeLoq and 2 known plaintexts, and we run ElimLin as described in [19]. In 5 seconds, the program manages to eliminate all but 130 variables out of the initial 512 variables. Moreover, in the linear span of the equations after ElimLin, the program is able to find one equation of degree 2, that involves *only* the 64 key variables and in which all the internal variables of the cipher are eliminated. This is sufficient to show that 64 rounds are very easy to break by Gröbner bases. For example, we may proceed as follows: for each new pair of known plaintexts, we get a new equation of this type. Given a sufficient number of known plaintexts (a small multiple of 64 will be sufficient), we will get a very overdefined system of equations with 64 variables. Such systems are known to be easily solvable by the XL algorithm and Gröbner bases, see [12, 11, 1].

Example 2. Here also, we consider 64 rounds of KeeLoq and 4 known plaintexts, and we run ElimLin as described in [19]. We fix 10 key bits to their true values. Then the remaining 54 key bits are recovered by ElimLin alone in 8 seconds. With Singular slimgb() function [39] the same computation takes 1 minute.

Example 3. With 64 rounds, 2 plaintexts that differ only in 1 bit, (it is no longer a known plaintext attack, but rather a chosen plaintext attack), and with 14 key bits fixed, the key is computed by ElimLin in 7 seconds and by Singular in 10 seconds.

Example 4. With 128 rounds and 128 plaintexts in the counter mode (the plaintexts are consecutive integers on 32-bits), and 30 bits fixed, the remaining 34 bits are recovered by ElimLin in 3 hours. This is slightly faster than brute force.

6.4 Cryptanalysis of KeeLoq with SAT Solvers

From [19], one may expect that better results will be obtained with SAT solvers. Given some number of pairs of plaintext and ciphertexts, over the whole 528 rounds, we rewrite the equations as a SAT problem and try to solve them. We write equations as polynomials (cf. previous section) and use the simplest version of the ANF to CNF conversion method described in [20].

Example 5. For full 528 rounds of KeeLoq, these attacks remain much slower than exhaustive search. For example with 2 plaintexts in counter mode (two consecutive integers on 32-bits) and 48 bits fixed, the remaining 16 key bits are recovered in 30 seconds with our conversion to CNF and MiniSat 2.0., done as described in [19, 20]. This is much slower than brute force. However, with a reduced number of rounds, the results are more interesting.

Example 6. For 64 rounds of KeeLoq and 2 known plaintexts, the key is recovered by MiniSat 2.0. in 0.3 s.

Example 7. For 96 rounds of KeeLoq, 4 known plaintexts, and when 20 key bits are guessed, the key is recovered by MiniSat 2.0. in 0.4 s.

Example 8. With 128 rounds and 2 known plaintexts, and 30 bits guessed, the remaining 34 bits are recovered in 150 s by MiniSat 2.0. This is about 80x faster than brute force.

Example 9. With 160 rounds, 2 plaintexts in counter mode, and 30 bits guessed, the remaining 34 bits are recovered in 233 s by MiniSat 2.0. This is clearly faster than brute force.

We note that the maximum number of rounds that we can break faster than by exhaustive search by our best algebraic attack with SAT solvers is 160 rounds. This attack does not exploit the periodicity of the cipher and uses an extremely low number of know plaintext-ciphertext pairs. In comparison up to 32 rounds can be broken directly "by hand" (cf. Fact 4.1). We also note that when the number of rounds is reduced to 64, the full key can be obtained almost instantly. This fact gave inspiration to design Slide-Algebraic attacks.

7 Combining Slide and Algebraic Attacks on KeeLoq

If the number of rounds were 512, and not 528, then it would be easy to analyse KeeLoq as an 8-fold iteration of 64 rounds. The last 16 rounds are a "barrier", which we can remove by guessing the 16 bits of the key used in those 16 rounds. These are the first 16 key bits, or $k_0, \ldots, k_{15}$, and the guess is correct with probability 2^{-16}. This is what we will do in our Slide-Determine Attack and our Slide-Algebraic Attack 1. Alternatively (as we will see in our Slide-Algebraic Attack 2), we may assume/guess some particular property of the 512 rounds of the cipher and try to recover the 16 (or more) bits that confirm this property.

Classical sliding attacks [3, 26, 30] exploit pairs of plaintext that have the following property:

Definition 7.1. Given a block cipher with periodic structure of the form $E_k(x) = g_k(f_k^{(m)}(x))$, $m > 1$, we call a "slid pair" any pair of plaintexts (P_i, P_j) such that $f_k(P_i) = P_j$.

7.1 Slide-Algebraic Attack 1

A simple sliding attack on KeeLoq would proceed as follows.

1. We guess the 16 key bits of g_k which gives us "oracle access" to 512 rounds of KeeLoq that we denote by $O = f_k^{(8)}$.
2. We consider 2^{16} known plaintexts (P_i, C_i).
3. By birthday paradox, one pair (P_i, P_j) is a "slid pair" for 64 rounds.
4. From this, one can derive as many known plaintexts for 64 rounds of KeeLoq as desired. For example, if $f_k(P_i) = P_j$ then $f_k(O(P_i)) = O(P_j)$. Additional "slid pairs" are obtained by iterating O twice, three times etc..
5. The whole attack has to be run about 2^{32} times, to find the correct "slid pair" (P_i, P_j).

In all with guessing the key of g_k there are 2^{48} possibilities to check. For each potential value for the first 16 bits of the key, and for each couple (P_i, P_j) we compute some 4 plaintext-ciphertext pairs for 64 rounds and then the key is recovered by MiniSat (cf. above) in 0.4 s which is about 2^{30} CPU clocks. The total complexity of the attack is about 2^{78} CPU clocks which is more than the exhaustive search.

7.2 Slide-Algebraic Attack 2

Another, better sliding attack proceeds as follows.

1. We do *not* guess 16 key bits, they will be determined later.
2. We consider 2^{16} known plaintexts (P_i, C_i).
3. By birthday paradox, one pair (P_i, P_j) is a "slid pair": $f_k(P_i) = P_j$.
4. Then the pair (C_i, C_j) is a plaintext-ciphertext pair for a "slided" version of the same cipher: starting at round 16 and finishing before round 80. This is to say a cipher with absolutely identical equations in every respect except for the (permuted) subscripts of the k_i.
5. From the point of view of multivariate equations and algebraic cryptanalysis, this situation is **not** much different than in Example 6 above solved in 0.3 seconds. We have one system of equations with the pair (P_i, P_j) for the first 64 rounds, and the same system of equations with the pair (C_i, C_j) and the key bits that are rotated by 16 positions.
6. We did write this system of equations and try ElimLin and MiniSat. For example with 15 first bits of the key fixed, ElimLin solves the system in 8 seconds. Better results are obtained with MiniSat, and without guessing any key variables, the key is computed in typically [1] about 2 seconds. Thus, with ElimLin, we can recover the key in about 2^{49} CPU clocks, and with MiniSat, we can do it in about 2^{32} CPU clocks.
7. There are about 2^{32} pairs (P_i, P_j) to be tried.

The total complexity of the attack, in the version with MiniSat is about $2^{32+32} = 2^{64}$ CPU clocks which is much faster than exhaustive search that requires about 2^{75} CPU clocks.

Summary. Our Slide-Algebraic Attack 2 can break KeeLoq within 2^{64} CPU clocks given 2^{16} known plaintexts. This is about 2^{53} KeeLoq encryptions. The attack is realistic, practical and has been fully implemented.

8 Conclusions

In this paper we described several key recovery attacks on KeeLoq, a block cipher with a very small block size and a simple periodic structure. KeeLoq is widespread in the automobile industry and is used by millions of people every day. Recently it has been shown that for a more complex cipher such as DES, up to 6 rounds can be broken by an algebraic attack given only one known plaintext [19]. In this paper we showed that up to 160 rounds of KeeLoq can be broken directly using MiniSat 2.0. algorithm with only 2 chosen plaintexts.

In combination with sliding attacks, an algebraic attack one the full 528-round KeeLoq is possible. Given about 2^{16} known plaintexts, we have proposed a working slide-algebraic attack equivalent to 2^{53} KeeLoq encryptions. In particular, in the so called 'hopping' or 'rolling' mode described in [4–6, 33], one cannot

[1] We have written these equations for 10 different random keys with randomly chosen plaintexts, and the timings we obtained were: 2.3, 9.1, 1.5, 0.5, 4.4, 0.3, 8.1, 1.8, 0.4, 0.6 seconds. Here the average time is 2.9 s and the median time is 1.65 s.

obtain more than 2^{16} known plaintexts. We are the first to have proposed such an attack on KeeLoq (all previous attacks required 2^{32} known plaintexts). Our attack is practical and was implemented with little programming effort.

We also showed that if as many as 2^{32} known plaintexts are available, KeeLoq is in fact extremely weak. For example, for 30 % of all keys, we can recover the key of the full cipher with complexity equivalent to 2^{28} KeeLoq encryptions. This attack can be prevented by a class of "strong keys" we defined that decreases the effective key space from 64 bits to 62.56 bits.

KeeLoq is a weak and simple cipher, and has several vulnerabilities. It is interesting to note that attacks that use sliding properties can be quite powerful because typically (in all our Slide-Determine and Slide-Algebraic Attacks) their complexity simply does not depend on the number of rounds of the cipher.

Table 1. Comparison of our attacks to other attacks reported on KeeLoq.

Type of attack	Data	Time	Memory	Reference
Pure Algebraic/SAT	2 KP	2^{73}	small	Our Example 5
Brute Force	2 KP	2^{63}	small	
Slide-Algebraic	2^{16}KP	2^{67}	small	Our Slide-Algebraic Attack 1
Slide-Algebraic	2^{16}KP	$\mathbf{2^{53}}$	small	Our Slide-Algebraic Attack 2
Slide-Meet-in-the-Middle	2^{16}KP	$\mathbf{2^{46}}$	small	Biham, Dunkelman *et al*[23]
Slide-Meet-in-the-Middle	2^{16}CP	$\mathbf{2^{45}}$	small	Biham, Dunkelman *et al*[23]
Slide-Correlation	2^{32}KP	2^{51}	16 Gb	Bogdanov[4, 5]
Slide-Cycle-Algebraic	2^{32}KP	2^{40}	18 Gb	Attack 3 in [21]
Slide-Cycle-Correlation	2^{32}KP	2^{40}	18 Gb	Bogdanov [5]
				Two versions:
Slide-Determine	2^{32}KP	$\mathbf{2^{31}}$	16 Gb	A: for 63 % of all keys
Slide-Determine	2^{32}KP	$\mathbf{2^{28}}$	16 Gb	B: for 30 % of all keys

Legend: The unit of time complexity here is one KeeLoq encryption.

The results of this paper can be compared to [4–6], other very recent work on KeeLoq. Recently, another attack with 2^{16} KP and time about 2^{45} KeeLoq encryptions was proposed by Biham, Dunkelman *et al.* [23]. Knowing which is the fastest attack on one specific cipher, and whether one can really break into cars and how, should be secondary questions in a scientific paper. Instead, in cryptanalysis we need to study a variety of attacks on a variety of ciphers. Brute force will be in fact maybe the only attack that will be executed in practice by hackers. It is precisely by attacking weak ciphers such as KeeLoq in many different ways that we discover many interesting attacks, and some important attacks such as algebraic attacks would never be discovered without extensive experimentation.

Acknowledgments. We thank Sebastiaan Indesteege and Sean O'Neil for valuable help.

References

1. Magali Bardet, Jean-Charles Faugère and Bruno Salvy, *On the complexity of Gröbner basis computation of semi-regular overdetermined algebraic equations,* in Proceedings of International Conference on Polynomial System Solving (ICPSS, Paris, France), pp. 71-75, 2004.
2. Alex Biryukov, David Wagner: *Advanced Slide Attacks,* In Eurocrypt 2000, LNCS 1807, pp. 589-606, Springer 2000.
3. Alex Biryukov, David Wagner: *Slide Attacks,* In Fast Software Encryption, 6th International Workshop, FSE '99, Springer, LNCS 1636, pp. 245-259.
4. Andrey Bogdanov: *Cryptanalysis of the KeeLoq block cipher,* http://eprint.iacr.org/2007/055.
5. Andrey Bogdanov: *Attacks on the KeeLoq Block Cipher and Authentication Systems,* 3rd Conference on RFID Security 2007, RFIDSec 2007.
6. Andrey Bogdanov: *Linear Slide Attacks on the KeeLoq Block Cipher,* The 3rd SKLOIS Conference on Information Security and Cryptology (Inscrypt 2007), LNCS, Springer-Verlag, 2007
7. C.Cid, S. Babbage, N. Pramstaller and H. Raddum: *An Analysis of the Hermes8 Stream Cipher,* In ACISP 2007, LNCS 4586, pages 1-10, Townsville, Australia, July 2007. Springer.
8. Keeloq wikipedia article. On 25 January 2007 the specification given here was incorrect and was updated since. See http://en.wikipedia.org/wiki/KeeLoq.
9. Keeloq C source code by Ruptor. See http://cryptolib.com/ciphers/
10. Nicolas Courtois: Examples of equations generated for experiments with algebraic cryptanalysis of KeeLoq, http://www.cryptosystem.net/aes/toyciphers.html.
11. Nicolas Courtois and Jacques Patarin, *About the XL Algorithm over* $GF(2)$*,* Cryptographers' Track RSA 2003, LNCS 2612, pp. 141-157, Springer 2003.
12. Nicolas Courtois, Adi Shamir, Jacques Patarin, Alexander Klimov, *Efficient Algorithms for solving Overdefined Systems of Multivariate Polynomial Equations*, In Advances in Cryptology, Eurocrypt'2000, LNCS 1807, Springer, pp. 392-407.
13. Nicolas Courtois and Josef Pieprzyk: *Cryptanalysis of Block Ciphers with Overdefined Systems of Equations,* Asiacrypt 2002, LNCS 2501, pp.267-287, Springer.
14. Nicolas Courtois and Willi Meier: *Algebraic Attacks on Stream Ciphers with Linear Feedback,* Eurocrypt 2003, Warsaw, Poland, LNCS 2656, pp. 345-359, Springer.
15. Nicolas Courtois: *General Principles of Algebraic Attacks and New Design Criteria for Components of Symmetric Ciphers,* in AES 4 Conference, Bonn May 10-12 2004, LNCS 3373, pp. 67-83, Springer, 2005.
16. Nicolas Courtois: *The Inverse S-box, Non-linear Polynomial Relations and Cryptanalysis of Block Ciphers,* in AES 4 Conference, Bonn May 10-12 2004, LNCS 3373, pp. 170-188, Springer, 2005.
17. Nicolas T. Courtois: *How Fast can be Algebraic Attacks on Block Ciphers?* In online proceedings of Dagstuhl Seminar 07021, *Symmetric Cryptography* 07-12 January 2007, E. Biham, H. Handschuh, S. Lucks, V. Rijmen (Eds.), http://drops.dagstuhl.de/portals/index.php?semnr=07021, ISSN 1862 - 4405, 2007. Also available from http://eprint.iacr.org/2006/168/.
18. Nicolas Courtois *CTC2 and Fast Algebraic Attacks on Block Ciphers Revisited* Available at http://eprint.iacr.org/2007/152/.
19. Nicolas Courtois and Gregory V. Bard: *Algebraic Cryptanalysis of the Data Encryption Standard,* In Cryptography and Coding, 11-th IMA Conference, Cirencester, UK, 18-20 December 2007. Springer LNCS. Preprint available at

eprint.iacr.org/2006/402/. Also presented at ECRYPT workshop Tools for Cryptanalysis, Krakow, 24-25 September 2007.
20. Gregory V. Bard, Nicolas T. Courtois and Chris Jefferson: *Efficient Methods for Conversion and Solution of Sparse Systems of Low-Degree Multivariate Polynomials over GF(2) via SAT-Solvers,* Available at http://eprint.iacr.org/2007/024/.
21. Nicolas Courtois, Gregory V. Bard, David Wagner: *Algebraic and Slide Attacks on KeeLoq,* Older preprint with using incorrect specification of KeeLoq, available at eprint.iacr.org/2007/062/.
22. Nicolas Courtois, Gregory V. Bard, David Wagner: *An Improved Algebraic-Slide Attack on KeeLoq,* A sequel to the oresent paper, preprint available from the authors.
23. Eli Biham, Orr Dunkelman, Sebastiaan Indesteege, Nathan Keller, Bart Preneel: How to Steal Cars A Practical Attack on KeeLoq, Crypto 2007 rump session talk, http://www.cosic.esat.kuleuven.be/keeloq/keeloq-rump.pdf. Full paper will be presented at Eurocrypt 2008 and published in Springer LNCS.
24. Jean-Charles Faugère: *A new efficient algorithm for computing Gröbner bases (F_4),* Journal of Pure and Applied Algebra 139 (1999) pp. 61-88. See www.elsevier.com/locate/jpaa
25. Philippe Flajolet and Robert Sedgewick; *Analytic Combinatorics,* book of 807 pages, to apear in Cambridge University Press in the first half of 2008. The full book is available on the Internet, see http://algo.inria.fr/flajolet/Publications/book.pdf
26. Raphael Chung-Wei Phan, Soichi Furuya: *Sliding Properties of the DES Key Schedule and Potential Extensions to the Slide Attacks,* In ICISC 2002, LNCS 2587, pp. 138-148, Springer, 2003.
27. Soichi Furuya: *Slide Attacks with a Known-Plaintext Cryptanalysis.* In ICISC 2001, LNCS 2288, pp. 214-225, Springer, 2002.
28. Louis Granboulan, Thomas Pornin: *Perfect Block Ciphers with Small Blocks,* In Fast Software Encryption - FSE 2007, LNCS 4593, pp.452-465, Springer, 2007.
29. Gemplus Combats SIM Card Cloning with Strong Key Security Solution, Press release, Paris, 5 November 2002, see http://www.gemalto.com/press/gemplus/2002/r_d/strong_key_05112002.htm.
30. E.K.Grossman and B.Tuckerman: *Analysis of a Feistel-like cipher weakened by having no rotating key,* IBM Thomas J. Watson Research Report RC 6375, 1977.
31. David Kahn, *The Codebreakers,* The Comprehensive History of Secret Communication from Ancient Times to the Internet, First published in 1967, new chapter added in 1996.
32. L. Marraro, and F. Massacci. *Towards the Formal Verification of Ciphers: Logical Cryptanalysis of DES, Proc. Third LICS Workshop on Formal Methods and Security Protocols, Federated Logic Conferences (FLOC-99).* 1999.
33. Microchip. An Introduction to KeeLoq Code Hopping. Available from http://ww1.microchip.com/downloads/en/AppNotes/91002a.pdf, 1996.
34. Microchip. Hopping Code Decoder using a PIC16C56, AN642. Available from http://www.keeloq.boom.ru/decryption.pdf, 1998.
35. Microchip. Using KeeLoq to Validate Subsystem Compatibility, AN827. Available from http://ww1.microchip.com/downloads/en/AppNotes/00827a.pdf, 2002.
36. MiniSat 2.0. An open-source SAT solver package, by Niklas Eén, Niklas Sörensson, available from http://www.cs.chalmers.se/Cs/Research/FormalMethods/MiniSat/
37. Ilya Mironov and Lintao Zhang *Applications of SAT Solvers to Cryptanalysis of Hash Functions,* In Proc. Theory and Applications of Satisfiability Testing, SAT 2006, pp. 102-115, 2006. Also available at http://eprint.iacr.org/2006/254.

38. Marko R. Riedel, *Random Permutation Statistics, paper available on the internet, at http://www.geocities.com/markoriedelde/papers/randperms.pdf.*
39. *Singular: A Free Computer Algebra System for polynomial computations. http://www.singular.uni-kl.de/*

A Strong Keys in KeeLoq

It is possible to see that the manufacturer or the programmer of a device that contains KeeLoq can check each potential key for fixed points for f_k. If it has any, that key can be declared "weak" and never used. This means that 63% of keys will be weak, and changes the effective key space from 64 bits to 62.56 bits, which is in fact a small loss. This appears to be practical for KeeLoq because the size of the plaintext-space is only 2^{32} and can be checked. A similar strong-key solution was in 2002 patented and commercialized by Gemplus corporation (currently Gemalto) to prevent GSM SIM cards from being cloned, see [29]. This removes our fastest attack on KeeLoq, Slide-Determine Attack. Further research is needed to see what is the best attack on KeeLoq in this case, and whether it is also necessary to remove fixed points for $f_k^{(2)}$.

B Algebraic Immunity and Boolean Function Used in KeeLoq

The security of KeeLoq depends on the quality of KeeLoq Boolean function NLF. We have:

$$y = NLF(a, b, c, d, e) = d \oplus e \oplus ac \oplus ae \oplus bc \oplus be \oplus cd \oplus de \oplus ade \oplus ace \oplus abd \oplus abc$$

Following [4] , this function is weak with respect to correlation attacks, it is 1-resilient but it is not 2-resilient and can in fact be quite well approximated by the linear function $d \oplus e$.

From the point of view of algebraic cryptanalysis, the fundamental question to consider is to determine the "Algebraic Immunity" of the NLF, which is also known "Graph Algebraic Immunity" or "I/O degree". We found that it is only 2, and one can verify that this NLF allows one to write the following I/O equation of degree 2 with no extra variables:

$$(e + b + a + y) * (c + d + y) = 0$$

However, there is only 1 such equation, and this equation by itself does *not* give a lot of information on the NLF of KeeLoq. This equation is "naturally" true with probability 3/4 whatever is the actual NLF used. It is therefore easy to see that this equation alone does **not** fully specify the NLF, and taken alone cannot be used in algebraic cryptanalysis. When used in combination with other equations, this should allow some algebraic attacks to be faster, at least slightly. At present time we are not aware of any concrete attack on KeeLoq that is enabled or aided by using this equation.