Algebraic and Slide Attacks on KeeLoq

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

[1] University College of London, Gower Street, London WC1E 6BT, UK,
[2] University of Maryland, College Park, MD 20742, USA
[3] University of California - Berkeley, Berkeley CA 94720, USA

Abstract. KeeLoq is a block cipher used in wireless devices that unlock the doors of cars manufactured by Chrysler, Daewoo, Fiat, GM, Honda, Jaguar, Toyota, Volvo, Volkswagen, etc [6, 7, 23, 24]. KeeLoq is very simple in implementation and economical in gate count, yet according to Microchip [23] it should have "a level of security comparable to DES".
Until now, algebraic attacks on block ciphers did not yield interesting results and most researchers seriously doubted if any block cipher would ever be broken by such attacks. It turns out that direct algebraic attacks can break up to 128 rounds of KeeLoq. Much better results are achieved in combination with slide attacks. Given about 2^{16} known plaintexts, we give an algebraic attack that uses a SAT solver and allows one to recover the key in 2^{64} CPU clocks which is 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 by an algebraic attack. Our attacks are easy to implement, have been tested experimentally, and the full key can be recovered on a PC.
In addition, if about 2^{32} known plaintexts are available, we present an attack equivalent to about 2^{29} KeeLoq encryptions.

1 Introduction

KeeLoq was designed in the 1980's by Willem Smit from South Africa. Following [25], the specification of KeeLoq is "not secret" but is patented and was released only under license. The specification of KeeLoq can be found in [6, 7, 24, 4]. In 1995 KeeLoq was sold to Microchip Technology Inc for more than 10 million US dollars [6].

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. In each round, only one bit of the state is modified. More importantly, KeeLoq requires a very low number of gates to be implemented. This is quite interesting and challenging, as it has been sometimes conjectured, that ciphers which require a small number of gates should be vulnerable to algebraic cryptanalysis, see [15, 10]. In this paper we will see that the simplicity of KeeLoq makes it directly breakable by simple algebraic attacks for up to 128 rounds.

KeeLoq also has a periodic structure with period of 64 rounds. This will allow us to propose much better attacks, and in particular a practical algebraic attack

that recovers the full key for the full cipher and the complexity of which does not depend on the number of rounds of the cipher.

Very little of previous work on algebraic cryptanalysis was very successful. The vulnerability of ciphers in general against algebraic cryptanalysis is motivated by the existence of algebraic I/O relations, see [10, 12]. However, though such equations allow one to break many LFSR-based stream ciphers quite badly [11] and certain block ciphers with strong special structure [13], so far it appeared infeasible to use algebraic cryptanalysis for cryptanalysis of common types of block ciphers. In 2006 Courtois published an attack that breaks 6 rounds of a toy block cipher called CTC, see [14], however CTC is not a very strong design, see [28]. Finally, it was shown that for DES, which is known to be very robust, algebraic attacks allow one to break up to 6 rounds, see [15]. According to [23], KeeLoq should have "a level of security comparable to DES". Yet, when we take into account its periodicity (a slide property, cf. Section 2), given 2^{16} known plaintexts we are able to break the full 528 rounds of KeeLoq.

KeeLoq has unusually small 32-bit blocks. Thus, in theory the attacker can expect to recover the whole code-book of 2^{32} known plaintexts. In practice there is no hope for such attacks, the devices are simply too slow to obtain this. At the same time, the 64-bit key size implies that the exhaustive search is actually feasible in practice, and hackers and car thieves implement it with FPGA's [6]. It requires only 2 known plaintexts (one known plaintext does not alone allow one to uniquely determine the key).

In two of the attacks we present in this paper we assume that the whole code-book is known. 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, from the point of view of the cryptographic research, the question remains very interesting. Very little is known about how such a key can be recovered, with what complexity, and what is the most efficient method. More importantly, our attacks work when only 60% or less of the code-book is available.

This paper is organised as follows: in Section 2 we describe the cipher and its usage. In Section 3 we do a preliminary analysis and recall some useful results about random functions and permutations. In Section 4 we demonstrate several algebraic attacks that work given very small quantity of known/chosen plaintexts and for a reduced number of rounds of KeeLoq. In Section 5 we study combined slide and algebraic attacks that work given about $2^{32/2}$ known plaintexts for the full 528-round cipher. Finally, in Section 6 and Appendix C we describe two even faster attacks that recover the key for full KeeLoq, requiring however the knowledge of (about) the whole code-book. In Appendix A we propose a method to avoid attacks on KeeLoq without modifying the cipher. In Appendix B we study the algebraic immunity of the Boolean function used in KeeLoq. In Appendix D, we present some experimental results which justify claims made in the text.

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 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:

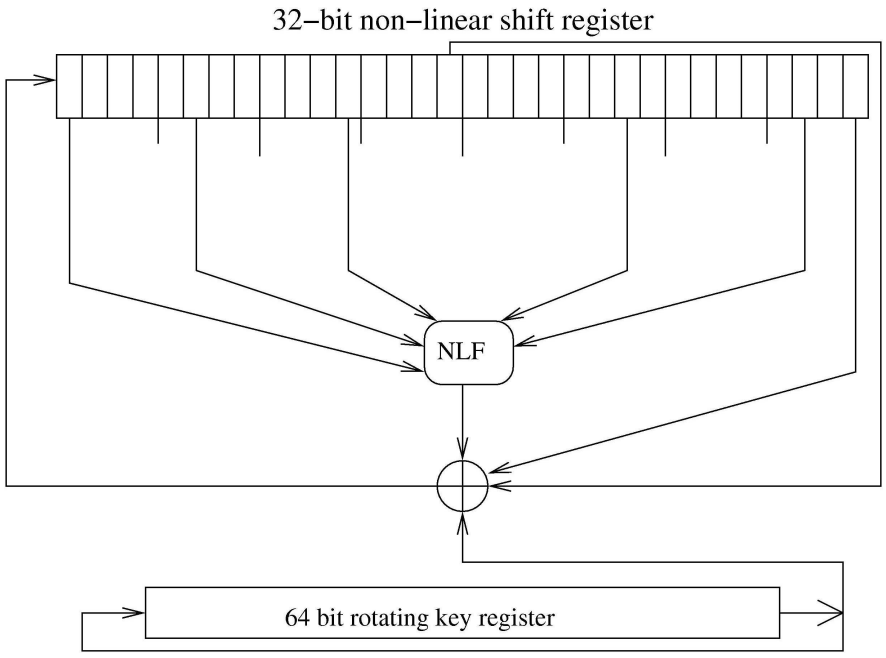
$$NLF(a,b,c,d,e) = d \oplus e \oplus ac \oplus ae \oplus bc \oplus be \oplus cd \oplus de \oplus ade \oplus ace \oplus abd \oplus abc$$

Alternatively, the specification documents available [6], say that it is "the non-linear function 3A5C742E" which means that $NLF(i)$ is the i^{th} bit of that hexadecimal number, counting 0 as the least significant and 31 as the most significant.

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:

$$L_i = k_{i-32 \ mod \ 64} \oplus L_{i-32} \oplus L_{i-16} \oplus NLF\left(L_{i-1}, L_{i-6}, L_{i-12}, L_{i-23}, L_{i-30}\right)$$

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



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

Fig. 1. KeeLoq Encryption

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, 23]. 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].

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

3 Preliminary Analysis and Useful Combinatorial Facts

3.1 The Simplicity of KeeLoq

Fact 3.1. 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 as we will see later. Algebraic attacks allow one to efficiently recover the key of such a simple cipher for up to 128 rounds.

Fact 3.2. 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 3.3. 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 3.1.

Fact 3.4. 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 3.2 to test whether it is possible that $c = g_k(p)$.

These facts are later used in Attack 4.

3.2 Random Functions, Random Permutations and Fixed Points

Given a random function from n-bits to n-bits, the probability that a given point has i pre-images is $\frac{1}{i!e}$, when $n \to \infty$. (This is a Poisson distribution with the average number of pre-images being $\lambda = 1$).

This distribution can be applied to derive statistics on the expected number of fixed points of a (random) permutation. It is also expected to work for (not exactly random) permutations that we encounter in cryptanalysis of KeeLoq. 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 3.1. The first 64 rounds of KeeLoq have 1 or more fixed points with probability $1 - 1/e \approx 0.63$.

Proposition 3.2. The first 64 rounds of KeeLoq have 2 or more fixed points with probability of $1 - 2/e \approx 0.26$.

3.3 On the Expected Number of Cycles in a Random Permutation

It is well known (see [29] for example) that:

Proposition 3.3. The expected number of cycles in a permutation on n bits is equal to H_{2^n} where $H_k = \sum_{i=0}^{k} 1/i$ is the k-th Harmonic number. We have $H_k \approx \ln k + \gamma$ where $\gamma \approx 0.58$ is the Euler-Mascheroni constant.

For example, when $n = 8$ we expect to have 6 cycles on average, and when $n = 32$ we expect to have 23 cycles on average.

4 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 [10]. 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 [15]. For KeeLoq, due to its simplicity, many more rounds can be directly attacked.

4.1 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 $\alpha_i = a_i \cdot b_i$ and $\beta_i = a_i \cdot e_i$. The values of the plaintext, the ciphertext, and a certain number of key bits that we may fix (i.e. guess, cf. Section 4.2) during the attack are written as separate equations. 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. Out of these the values of $32 + 32 + F$ variables are already known. The total number of monomials that appear in these equations is about $12r$.

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.

4.2 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 [17] or Singular's slimgb() algorithm [30]. We have also tried a much simpler method called ElimLin and described in [15]. 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 [16].

Fact 4.1. 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 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 then 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 [6].

4.3 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 [15]. The program manages to eliminate all but 137 variables out of the initial 372 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 can be solved very easily by the XL algorithm or Gröbner bases, see [9, 8, 1].

Example 2. Here also, we consider 64 rounds of KeeLoq and 4 known plaintexts, and we run ElimLin as described in [15]. We fix 10 key bits to their values. Then the remaining 54 key bits are recovered by ElimLin alone in 10 seconds. The same result is obtained by using Singular slimgb() function [30] in 70 seconds.

Example 3. With 64 rounds, 2 plaintexts that differ only in 1 bit, (it is no longer a known plaintext attack), and with 10 key bits fixed, the key is computed by ElimLin in 20 seconds and by Singular in 5 seconds (here Singular is faster).

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.

4.4 Cryptanalysis of KeeLoq with SAT Solvers

From [15], 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 [16].

Example 5. For full 528 rounds of KeeLoq, these attacks remain much slower than exhaustive search. For example with 8 plaintexts in counter mode (consecutive integers on 32-bits) and 44 bits fixed, the remaining 20 key bits are recovered in 7 hours with a conversion to CNF and MiniSat 2.0., done as described in [15, 16]. This is much slower than brute force. However, with a reduced number of rounds, the results are quite interesting.

Example 6. For 64 rounds of KeeLoq and 2 known plaintexts, the key is recovered by MiniSat 2.0. in 0.19 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 0.3 s.

Example 8. With 128 rounds, 2 plaintexts in counter mode, and 30 bits guessed, the remaining 34 bits are recovered in 2 hours by MiniSat 2.0. This is only slightly faster than brute force.

Future Work. So far we are not aware of an attack that would break more than 128 rounds of KeeLoq faster than the exhaustive search given a small number of known or chosen plaintexts.

5 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 Attacks 1 and 3. Alternatively (as we will see in Attacks 2 and 4), 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, 19, 21] exploit pairs of plaintext that have the following property:

Definition 5.1. 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 $f_k(P_i) = P_j$.

5.1 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 an unlimited number of known plaintexts for 64 rounds of KeeLoq. 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.

5.2 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 0.2 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 2.3 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 exactly $2^{32+32} = 2^{64}$ CPU clocks which is much faster than exhaustive search that requires about 2^{75} CPU clocks.

Summary. Our 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.

6 Attacks that Use the Whole Dictionary

We will now present two attacks that are faster than our Attack 2, but require the attacker to know about 2^{32} known plaintexts. In this case the question of key recovery is still an important question that deserves attention.

In our Attack 3 we will construct a distinguisher that allows one to distinguish 512 rounds of KeeLoq from a random permutation and thus recovers 16 bits of the key. The final key will be recovered by a pure algebraic attack – solving a system of multivariate equations. In our Attack 4, we will compute a list of pairs that, for each possible value of the first 16 bits of the key, will contain several plausible fixed points for 64 rounds of KeeLoq.

Both attacks 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 plaintexts 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 and the time to get one pair is about 16 CPU clocks. This would require 16 Gigabytes of RAM which is now available on a high-end PC.

6.1 Preliminary Analysis and Working Conditions

In Attacks 3 and 4 we make the following observations. Recall, $f_k(x)$ represents the first 64 rounds of the cipher. First we assume that there are at least one and

two fixed points, respectively, for $f_k(x)$. As shown in Section 3.2, these events happen with probability 0.63 and 0.26, respectively. For the sake of simplicity, we will assume that Attacks 3 and 4 will fail when the required number of fixed points is not present.

Secondly, we observe that if x is a fixed point of $f_k(\cdot)$, or in an orbit of size 2, 4, or 8, then x is a fixed point of $f_k^{(8)}(\cdot)$. We estimate that $f_k^{(8)}(\cdot)$ – the first 512 rounds of KeeLoq – will on average have about 4 fixed points. In Attack 4, the attacker determines possible fixed points for $f_k^{(8)}$ and then assumes that it is also a fixed point for $f_k(\cdot)$. This guess will be correct with probability about 1/4. (For Attack 3 see Appendix C and D.)

6.2 Slide-Algebraic Attack 3

This part has been moved to Appendix C. The following Attack 4 is much faster and works for a larger proportion of keys.

6.3 Slide-and-Determine Attack 4

This attack occurs in two stages.

Stage 1 - Batch Guessing Fixed Points. This attack requires 2^{32} plaintext/ciphertext pairs (p, c). We expect that one p is a fixed point of f (about 4 of them are fixed points for f^8). For each such pair (p, c), test whether it is possible that p is a fixed point of f (and thus also for f^8) if not, discard that pair. We use Fact 3.4 and the complexity so far is about $16 \cdot 2^{32} = 2^{36}$ CPU clocks (mostly spent accessing the memory). Only about $2^{16} + 4$ pairs will survive.

Then following Fact 3.1 we can at the same time compute 16 bits of the key with time of about $4 \cdot 16$ CPU clocks (cf. Fact 3.1 and 4.1). To summarize, given the whole code-book and in time of about 2^{38} CPU clocks we produce a list of 2^{16} triples $P, C, (k_{15}, \ldots, k_0)$. Since we assumed that f_k has a fixed point, at least one triple is valid (moreover we expect that less than 2 are valid).

Stage 2 - Batch Solving and Verification. 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. There is a simpler and direct method to get the same result. We use Fact 3.3 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 2^{32} possible full keys of 64 bits. This takes time equivalent to 2^{32} computations of $h_k(\cdot)$, which is about 2^{40} CPU clocks.

Finally, test each of these 2^{32} complete keys on one other plaintext/ciphertext pair. This is again only about 2^{36} CPU clocks with our precomputed table. 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.

Summary of Attack 4. This attack succeeds for 63 % of keys (cf. Proposition 3.1). The running time is about 2^{40} CPU clocks which is only about 2^{29} KeeLoq encryptions. This is if we ignore the time to get the 2^{32} plaintext/ciphertext pairs. If we added it, the complexity of the attack would be 2^{32} KeeLoq encryptions.

The attack is parallelizable and can be performed online, as known texts are received. It can be performed with less than the full code-book, at a proportional decrease in the success probability.

7 Conclusions

In this paper we described four 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 [15]. In this paper we showed that, for example up to 128 rounds of KeeLoq can be broken using MiniSat algorithm, given up to 4 known plaintexts.

For the full 528-round KeeLoq cipher and given about 2^{16} known plaintexts, we have proposed a working slide-algebraic attack equivalent to 2^{53} KeeLoq encryptions. This attack is practical and was implemented with little programming effort. It appears that it is actually the first cryptanalytic attack on KeeLoq that can work in practice. In particular, in the so called 'hopping' or 'rolling' mode described in [4, 23], it is impossible to get more than 2^{16} known plaintexts.

A faster attack can be found if nearly 2^{32} known plaintexts are available. For about 63 % of all keys, we can recover the key of the full cipher with complexity equivalent to about 2^{29} KeeLoq encryptions.

It is interesting to note that attacks that use sliding properties can be quite powerful because typically (as in Attacks 1,2 and 4) their complexity simply does not depend on the number of rounds of the cipher. The results of this paper can be compared to [4], another very recent work on KeeLoq. It appears that algebraic cryptanalysis gives better results than traditional methods such as Linear Cryptanalysis, and our algebraic attack will be the fastest attack we know requiring a reasonable quantity of known plaintexts.

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. 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.
6. Keeloq wikipedia article. 25 January 2007. See http://en.wikipedia.org/wiki/KeeLoq.
7. Keeloq C source code by Ruptor. See http://cryptolib.com/ciphers/
8. Nicolas Courtois and Jacques Patarin, *About the XL Algorithm over GF*(2)*,* Cryptographers' Track RSA 2003, LNCS 2612, pp. 141-157, Springer 2003.
9. 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.
10. Nicolas Courtois and Josef Pieprzyk: *Cryptanalysis of Block Ciphers with Overdefined Systems of Equations,* Asiacrypt 2002, LNCS 2501, pp.267-287, Springer.
11. Nicolas Courtois and Willi Meier: *Algebraic Attacks on Stream Ciphers with Linear Feedback,* Eurocrypt 2003, Warsaw, Poland, LNCS 2656, pp. 345-359, Springer.
12. 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.
13. 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.
14. Nicolas T. Courtois *How Fast can be Algebraic Attacks on Block Ciphers?* Available at http://eprint.iacr.org/2006/168/.
15. Nicolas T. Courtois and Gregory V. Bard: *Algebraic Cryptanalysis of the Data Encryption Standard,* Available at http://eprint.iacr.org/2006/402/.
16. 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/.
17. 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
18. 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.
19. Soichi Furuya: *Slide Attacks with a Known-Plaintext Cryptanalysis.* In ICISC 2001, LNCS 2288, pp. 214-225, Springer, 2002.
20. 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.
21. 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.
22. 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.
23. Microchip. An Introduction to KeeLoq Code Hopping. Available from http://ww1.microchip.com/downloads/en/AppNotes/91002a.pdf, 1996.
24. Microchip. Hopping Code Decoder using a PIC16C56, AN642. Available from http://www.keeloq.boom.ru/decryption.pdf, 1998.

25. Microchip. Using KeeLoq to Validate Subsystem Compatibility, AN827. Available from http://ww1.microchip.com/downloads/en/AppNotes/00827a.pdf, 2002.
26. 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/
27. 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.
28. Orr Dunkelman and Nathan Keller: *Linear Cryptanalysis of CTC,* Available at http://eprint.iacr.org/2006/250/.
29. Random Permutation Statistics – wikipedia article, 25 January 2007, available at http://en.wikipedia.org/wiki/Random_permutation_statistics
30. 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 removes all attacks described in this paper (unhappily brute force attacks will remain feasible).

This solution can be used in practice, and is very similar to a known solution that was in 2002 patented and commercialized by Gemplus (currently Gemalto) to prevent GSM SIM cards from being cloned, see [20].

Further research. With current attacks, there is no need to remove all fixed points $f_k^{(8)}$. Further research might force one to do so. For example it is possible to see that Attack 3 might force one to remove all fixed points of $f_k^{(2)}$. This is because Attack 3 that is dominated by the first step, and if we consider the cipher as $4 \times 128 + 16$ rounds instead of $8 \times 64 + 16$, in the second step of the attack we might be able to break 128 rounds of KeeLoq with 2 known plaintexts, with an improved algebraic attack that remains to be found. This shows the interest of studying "pure" algebraic cryptanalysis as done in Section 4, further improvements here might force us to remove also fixed points for $f_k^{(2)}$, further reducing the "safe" key space of KeeLoq.

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.

C Slide-Algebraic Attack 3

In this attack we will guess the first 16 bits of the key namely $k_0, \ldots, k_{15}$, and construct a distinguisher between $f_k^{(8)}$ and a random permutation.

Preliminary Remarks We assume that there are at least two fixed points for $f_k(x)$, which happens with probability 0.26 (cf. Proposition 3.2). In the remaining cases the attack fails. Under this assumption, we expect that there will be about 6 fixed points for $f_k^{(8)}(\cdot)$, i.e. the first 512 rounds of KeeLoq. We did computer simulations that confirm this figure, see Appendix D. The attacker will try to guess which out of 6 are fixed points for $f_k(\cdot)$. The probability that the guess is correct is about $\binom{6}{2}^{-1} \approx 1/15$. Instead of guessing, the attacker will try all subsets of 2 out of 6 points until the right pair is used, which requires on average about 15 tries.

Stage 1 - Recover 16 Key Bits with a Distinguisher. Let B be a permutation on 32 bit words. From Proposition 3.3, assuming that it behaves as a random permutation, we expect that B has about 23 cycles. Half of them should have even sizes. When we compose B with itself, all cycles that are of even size split into two pieces, that can be of either even or odd size depending on whether the initial cycle size was congruent to 0 or 2 modulo 4. All cycles of odd size remain intact (but points are permuted). Thus, we expect that the number of even cycles will be dived by 2.

Consider what happens when this composition operation is repeated 3 times:

$$B \to B^2 \to B^4 \to B^8.$$

We expect that B^8 has $11.5 \mapsto 5.75 \mapsto 2.8 \mapsto 1.4$ which is about 1 cycle of even size left. Note that a cycle of B must be of length 0 mod 16 to be of even length for B^8. Otherwise, if it is of length $1, 2, \ldots 15$ mod 16 then it will be

of odd length for B^8. This property allows one to distinguish between $f_k^{(8)}$ and a random permutation that should have about 11–12 even length cycles. The proposed distinguisher works as follows: if there are 6 or more cycles, we say it is the wrong key. Otherwise we say that $k_0, \ldots, k_{15}$ is be correct.

The probability of a false positive is equal to the probability that some 6 cycles in B have length that are multiples of 16, as only such cycles can still be of even size after splitting into two 3 times. This probability is $p = 16^{-6} = 2^{-24}$. Our distinguisher has a very low threshold, only 6, yet the resulting probability of a false positive $p = 2^{-24}$ is clearly sufficient to be able to uniquely determine which 16-bit key is the right key. At the same time, since the expected number of even cycles in a random permutation is about 11.5, the probability of the right key being not detected – a false negative – which amounts to having only 5 or less even-size cycles for a random permutation is extremely low and will be neglected. The success rate of this part of the attack is close to 1 and we expect that exactly one key will be found.

In order to implement the distinguisher, we need to compute the sizes of all cycles for a permutation on 2^{32} elements. This is easy and takes time of roughly about 2^{36} CPU clocks, as we assumed that plaintext-ciphertext pairs are stored in a table and the time to get one pair is 16 of CPU clocks. For each point not previously used, we explore the cycle and count how many elements it has. Then we start with a random point not previously used. The additional memory required (in addition to 16 Gigabytes already used for storing the whole code-book) is only 2^{32} bits - we need to remember which points were used. The fact that we can reject a key as long as 6 even-size cycles are found, avoids systematically computing all cycles, only the biggest ones, and allows for an early abort. It is clear that the average complexity of an optimised version of this attack will be not much more than 2^{16+36} CPU clocks. To summarise, at this stage the attack gives us 16 bits of the key $k_0, \ldots, k_{15}$ with the workfactor of about 2^{52} CPU clocks.

Stage 2 - Recover the Missing 48 Bits. The first idea would be to use brute force. The complexity is however 2^{48+11} which is already too much in comparison to our Stage 1. Instead we proceed exactly as in Attack 3, except that we now actually know 16 bits of the key, and know the resulting (approximatively) 4 fixed points of $f_k^{(8)}$. Here again we will assume that there are two fixed points for f_k which works for 26 % of keys. (the complexity of this attack for other keys remains to be seen.) We need to guess which two points are fixed points of f_k and then we solve a system of equations corresponding to 64 rounds of KeeLoq and 2 known plaintexts. This takes 0.2 s $\approx 2^{28}$ CPU clocks with MiniSat 2.0.

The probability of correctly guessing which two fixed points of $f_k^{(8)}$ are fixed points for f_k is $\binom{6}{2}^{-1} = 1/15$ as explained earlier. Thus the total complexity of this stage is about $15 \cdot 2^{28} \approx 2^{32}$ CPU clocks and we expect that for the wrong pair of fixed points no solution will be found (there are 48 bits of key left to be found determined by the 64 bits of the two fixed point). The first stage that requires about 2^{52} CPU clocks dominates the attack.

Summary of Attack 3. This attack succeeds with probability 0.26 i.e; for 26 % of keys (cf. Proposition 3.2). The running time is about 2^{52} CPU clocks which is only about 2^{41} KeeLoq encryptions.

It is possible to see that this attack works and allows one to find two fixed points of f_k, already if we have about 60% of the code-book, see Appendix D. However, since we need two fixed points, if a smaller fraction $0 < \alpha \leq 1$ of the code-book is available, unlike Attack 3, the success probability of this attack declines quite quickly, decreasing as a square of this proportion α.

D Simulations on Fixed Points and Random Permutations

In this section we do some computer simulations to justify certain claims about permutations and fixed points made in text. In Attack 3, we need to know how many (on average) fixed points do we expect for f^8 when we assume that f already has at least 2 fixed points. The answer is about 6 fixed points.

It is also interesting to know what is the percentage of the plaintext space that has to be searched, to find enough fixed points of $f^{(8)}$, such that at least two of these are also fixed points for f. Our experiments show that $\eta = 60\%$ of the plaintext space must be explored on average.

In our experiment, we generated random permutations f of domain size 2^{12} through 2^{16}. We checked for fixed points by exhaustion. If that permutation indeed had zero or one fixed points, then we denoted this an abortion. Then for those permutations that did not abort (i.e. those which two or more fixed points), we iterated through the domain to see at what value the second fixed point was found. We also counted the number of fixed points of f, and the number of fixed points of $f^{(8)}$ and computed their average. For a random permutation f the number of fixed points of f is denoted n_1, and the the number of fixed points for f^8 is denoted n_8.

Table 1. Fixed points of random permutations and their 8th powers

Size	2^{12}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}
Experiments	1000	10,000	10,000	10,000	10,000	100,000
Abortions ($n_1 < 2$)	780	7781	7628	7731	7727	76,824
Good Examples ($n_1 \geq 2$)	220	2219	2372	2269	2273	23,176
Average n_1	2.445	2.447	2.436	2.422	2.425	2.440
Average n_8	4.964	5.684	5.739	5.612	5.695	5.746
Average Location	2482	2483	4918	9752	19,829	39,707
Percentage (η)	60.60%	60.62%	60.11%	59.59%	60.51%	60.59%