Algebraic and Slide Attacks on KeeLoq

Nicolas T. Courtois[1] and Gregory V. Bard[2]
[1] University College of London, Gower Street, London, UK,
[2] University of Maryland, College Park, USA

Abstract. KeeLoq is a block cipher used in wireless devices that unlock doors in cars manufactured by Chrysler, Daewoo, Fiat, GM, Honda, Jaguar, Toyota, Volvo, Volkswagen, etc [4]. It was designed in the 80's by Willem Smit from South Africa and in 1995 was sold to Microchip Technology Inc for more than 10 million USD. Though no attack on this cipher have been found thus far, the 64-bit key size makes it no longer secure. Hackers and car thieves exploit this, to recover the key by brute force with FPGA's.
From our point of view however, this cipher is interesting for other reasons. Compared to typical block ciphers that have a few carefully designed rounds, this cipher has 528 extremely simple rounds with extremely few intermediate variables (one per round). This seems a perfect target to study algebraic attacks on block ciphers. The cipher also has a periodic structure with period of 64 rounds, and an unusually small block size of 32 bits.
We present several slide-algebraic attacks on KeeLoq that can break KeeLoq in practice, the best of which allows one to recover the full key for the full cipher within 2^{48} CPU clocks. One of the attack is practical and easy to implement.

1 Introduction

KeyLoq operates with 32-bit blocks and 64-bit keys. Though it has 528 rounds, which is a lot, it remains one of the simplest block ciphers known, and requires a very low number of gates to be implemented. In each round, only one bit of the state is modified. This is quite interesting and challenging, as it has been sometimes conjectured, ciphers that require a small number of gates should be vulnerable to algebraic cryptanalysis, see [6, 10]. In practice however, we will see that, given a very large number of rounds, the cryptanalysis of KeeLoq is still not that easy.

With 32-bit blocks, 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, and the best practical attack remains the exhaustive search with 2 known plaintexts (one known plaintext does not allow one to uniquely determine the key). In this paper we present several attacks, and the two fastest attacks assume that the whole code-book is known. At this moment 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. Little is known about how such a key can be recovered, with what complexity, and what is the most efficient method.

In this paper we try to break KeeLoq with algebraic cryptanalysis. For other ciphers, for example DES, the algebraic approach does only allow one to break a limited number of rounds (e.g. 6 for DES, see [6]). For KeeLoq, here more rounds, (for example 128), can be broken directly given the description of the cipher and very few known plaintexts, as we will see below. In addition to the simplicity of the cipher we will also exploit its periodicity (a slide property). This will allow us to break the full 528 rounds of KeeLoq,

This paper is organised as follows: in Section 2 we describe the cipher and its usage. In Section 3 we recall some useful results about random functions and permutations. In Section 4 we study algebraic attacks that work given very small quantity of known/chosen plaintext and for reduced number of rounds of KeeLoq. In Section 5 we study slide attacks that work given about $2^{n/2}$ known plaintexts where $n = 32$, for full 528-round KeeLoq cipher. Our second attack is about 2^{11} times faster than brute force. Finally, in Section 6 we show two even faster slide-algebraic attacks that recover the key for full KeeLoq with complexities being the order of 2^{48}, requiring however the knowledge of the whole code-book.

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 that is not a multiple of 64, in order to prevent obvious slide attacks (but more advanced slide attacks remain possible as it will become clear later).

Let $k_{63}, \ldots, k_0$ be the key. In each round, it is rotated to the right, with wrap around. Therefore, during rounds $i, i+64, i+128, \ldots$, the key 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 prefix" of f_k. 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 given by:

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

Instead, the specification documents available [4], 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.

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.

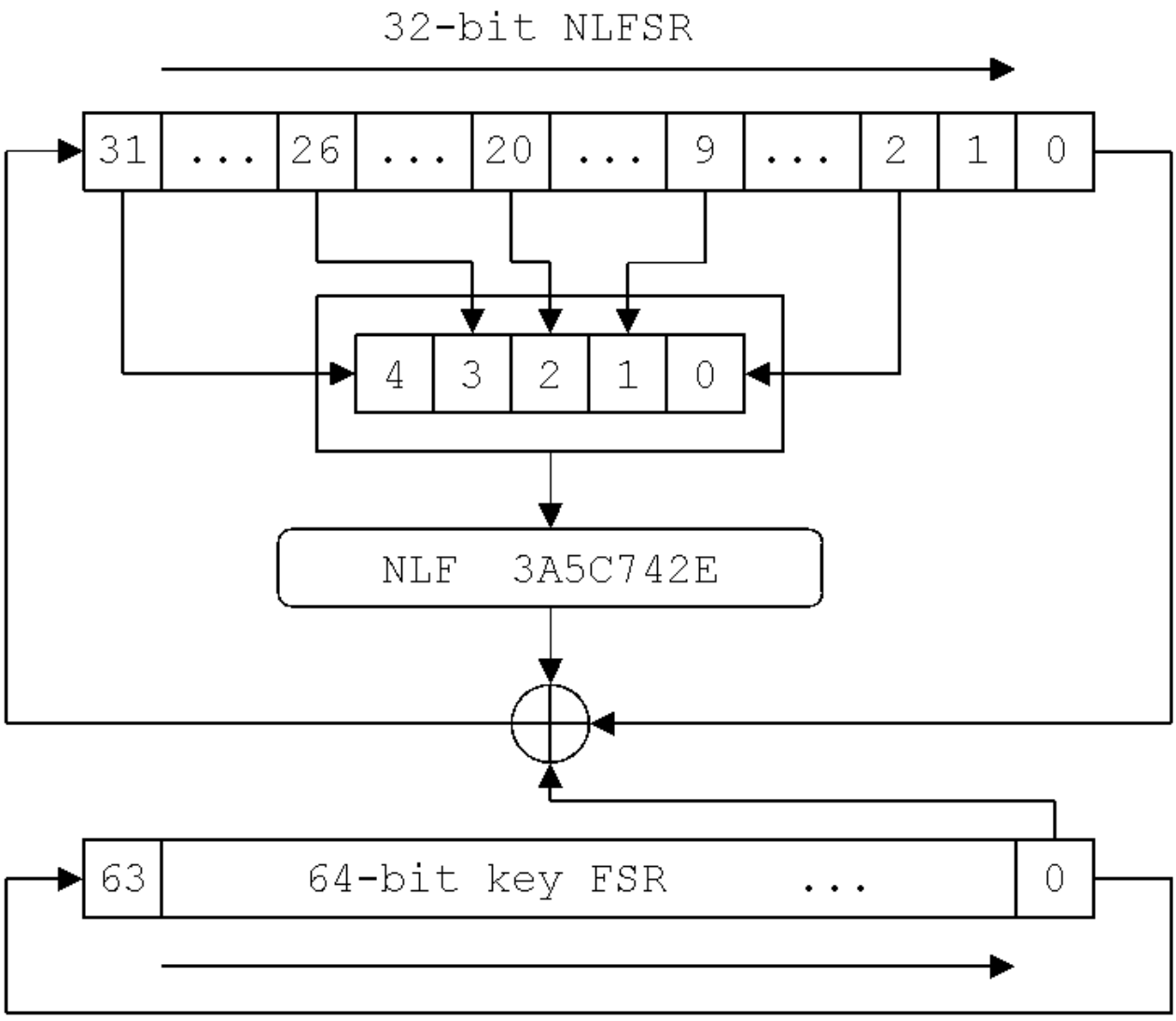
In this paper we study mostly the general security of the KeeLoq cipher against key recovery attacks. Like almost always in cryptanalysis of block ciphers, our attacks do not constitue a practical threat for real-life applications where KeeLoq is typically cracked by brute force with FPGAs in about two weeks [4].

3 Preliminary Analysis

3.1 Random Functions, Random Permutations and Fixed Points

Given a random function on n bits, we expect that that the probability that a given point has i pre-images to be $\frac{1}{ei!}$ (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 that are expected to work also for (not exactly random) permutations that we encounter in cryptanalysis of KeeLoq. In particular let $f_k(x)$ be the 64 rounds of KeeLoq. By assuming that $f_k(x) \oplus x$ is a pseudo-random function, we obtain that with probability of $1 - \frac{2}{e}$, zero has two or more pre-images. Thus, with probability about 0.82 the first 64 rounds of KeeLoq have 2 or more fixed points.



KeeLoq Encryption

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

Fig. 1. KeeLoq Encryption

3.2 On the Expected Number of Cycles in a Random Permutation

It is well known that, see for example [17] that the number permutations with a given number of cycles is equal to the unsigned Stirling numbers of the first kind. The expected number of cycles in a permutation on n bits is equal to H_{2^n} where $H_k = \sum_{i=0}^{k} 1/i \approx \ln k$ is the k-th Harmonic number. 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 (or chosen) plaintexts, see [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 [6]. For KeeLoq, due to its simplicity, many more rounds can be directly attacked.

4.1 How to Write 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 ab and ae, and add equations that define these new variables that express fact that $ab = a \cdot b$ and $ae = a \cdot e$. The values of the plaintext, the ciphertext, and a certain number of key bits that we may fix (i.e. guess) 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 $12k$.

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. In this case, we will speak about counter mode attack.

The complexity of an attack on r rounds of KeeLoq with k bits of the key should be compared to $2^k \cdot r$ which is the complexity of the brute force key search in which we assume that an optimised assembly language implementation of r rounds KeeLoq should take about $4r$ CPU clocks.

Thus, for example, for full KeeLoq, the reference complexity for the exhaustive key search is 2^{75} CPU clocks. Assuming that the CPU runs at 2 GHz, in 1 hour one can execute about 2^{43} CPU clocks. From here, as a rule of thumb, we get that for example, if we fix 32 key bits to their actual values (the real attacker should guess these bits), and then the attack runs on our PC in less than 1 hour, the full attack complexity will be less than 2^{32+43}. Such an attack will already be faster than brute force.

4.2 Direct Algebraic Attacks on KeeLoq

One can try to solve the equations of KeeLoq with a ready computer algebra system such as Magma F4 algorithm [11] or Singular slimgb() algorithm [18]. We have also tried a much simpler method called ElimLin and described in [6]. 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 [7]. Here the equations are of very low degree, and very sparse.

Frontal Assault – Elimination and Gröbner Bases Attacks on KeeLoq

Example 1. For example, we consider 64 rounds of KeeLoq and 2 known plaintexts, and we run ElimLin as described in [6]. The program manages to eliminate all except 137 variables out of initial 372 variables. Moreover, the program is able to find in the linear span of the equations after ElimLin, 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 couple 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 [6]. We fix 10 key bits to their values. Then the remaining 54 key bits are recovered by ElimLin alone in 10 seconds. The solution can also found by running Singular slimgb() function [18] in 70 seconds.

Example 3. With 64 rounds, 2 plaintexts that differ only in 1 bit, and 10 key bits fixed, The key is computed by ElimLin in 20 seconds and by Singular in 5 seconds (in this case Singular is faster).

Example 4. With 128 rounds and 128 plaintexts in the counter mode (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.

Cryptanalysis of KeeLoq with SAT Solvers

From [6], 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. For full 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 [6, 7]. This is much slower than brute force. However, with a reduced number of rounds, the results are quite interesting.

Example 5. For 64 rounds of KeeLoq and 2 known plaintexts, the key is recovered by MiniSat 2.0. in 0.19 s.

Example 6. 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 7. 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 again is 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. We are running additional simulations and in the future we will report more results on breaking reduced-round KeeLoq with Gröbner bases algorithms and SAT solvers.

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.

5.1 Slide-Algebraic Attack 1

A simple sliding attack [12, 3] on KeeLoq would proceed as follows. This attack is slower than brute force and we describe it for completeness.

1. We guess 16 key bits 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, i.e. $f_k(P_i) = P_j$.

4. From this one can derive an unlimited number of known plaintexts for 64 rounds of KeeLoq: if $f_k(P_i) = P_j$ then $f_k(O(P_i)) = O(P_j)$. More "slid pairs" are obtained by iterating O twice, three times etc..
5. There are about 2^{32} pairs (P_i, P_j) to be tried.

In all there are 2^{48} tries. For each candidate 16 bits of the key, 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" for 64 rounds, i.e. $f_k(P_i) = P_j$.
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.
5. From the point of view of multivariate equations and algebraic cryptanalysis, this situation is **not** much different than in Example 5 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. We did write this system of equations and try ElimLin and MiniSat. With 15 first key variables fixed ElimLin solves the system in 8 seconds. With 0 key variables fixed, MiniSat solves the system 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.
6. 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 second attack can break KeeLoq within 2^{64} CPU clocks given 2^{16} known plaintexts. This is about 2^{53} KeeLoq encryptions. The attack is practical and have been fully implemented and tested.

6 Faster Combined Slide and Algebraic Attacks

In our Attack 3, we will compute a list of pairs that, for each possible 16 bits of the key, will contain several plausible fixed points for 64 rounds of KeeLoq. In the Attack 4 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. As in all our attacks, the final key will be recovered by a pure algebraic attack – solving a system of multivariate equations.

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 (this may sound like a practical attack scenario, however in fact it is hard to imagine a hacker patient enough to get 2^{32} known plaintexts from the device, the best practical attack is again brute force).

6.1 Preliminary Analysis

In order to simplify the Attacks 3 and 4, we will make several assumptions that are true with very high probability.

First, as explained in Section 3.1 we assume that there are two fixed points for $f_k(x)$, the first 64 rounds of the cipher, which happens with probability 0.82. For the sake of simplicity, in the remaining 18 % of cases, we will say that the attack fails.

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 $f_k^{(8)}(x)$. We estimate that, under the assumption that there are already two fixed points for $f_k(\cdot)$, on average there will be about 4 fixed points for $f_k^{(8)}(x)$ – the first 512 rounds of KeeLoq. In our attacks, the attacker determines fixed points for $f_k^{(8)}$ by computing the complete table for 512 rounds of the cipher, given (or assuming) the knowledge the first 16 key bits used in the last "odd" rounds, and given the code-book of the cipher. Then the attacker may try to guess which out of 4 are fixed points for $f_k(x)$. The probability that the guess is correct can be estimated to be roughly about $2/4 \cdot 1/3 \approx 1/6$. Instead of guessing, the attacker will try all subsets of 2 out of 4 points until the right pair is used, which require on average about 6 tries.

For simplicity we will also assume that all the plaintext-ciphertext pairs are stored in a table and therefore the time to get one pair is 1 CPU clock. This would require 16 Gigabytes of RAM which is now available on a PC.

6.2 Slide-Algebraic Attack 3

This attack occurs in two stages.

Stage 1 - Batch Guessing Fixed Points. For any plaintext P on 32 bits, we assume that it is a fixed point for $f_k(\cdot)$. We get the corresponding ciphertext C. Then we know that $g(P) = C$, i.e. one gets C when P is encrypted with the last 16 rounds of the cipher (which are identical to the first 16 rounds). At this stage we can use the SAT solver to find that most systems of equations of this kind are not satisfiable.

We did implement this method and it takes about 0.01 s to show that in 99.999 % of cases a given P cannot be a fixed point. However here a brute

force approach will be faster. We try all possible keys and assume that a very optimised test whether $g(P) = C$ takes about $4 \cdot 16$ CPU clocks (there are only 16 very simple rounds and 16 key bits are involved). With probability about 2^{-16} a given triple $P, C, (k_{15}, \ldots, k_0)$ works. By trying all 2^{32} values for P we get at the end about 2^{16} triples $P, C, (k_{15}, \ldots, k_0)$ that can be valid. In fact, we expect about 4 times more. This is because we assumed that $f_k^{(8)}$ has 4 fixed points on average, which makes fixed points happen 4 times more frequently than what we expect from a random permutation.

Next, we sort the list of $4 \cdot 2^{16} \approx 2^{18}$ triples obtained, and we get a list which for each 16 key bits gives us **all possible** fixed points of $f_k^{(8)}$, assuming that the 16 bits are correct. This step requires about 2^{54} CPU clocks and the whole code-book.

Stage 2 - Batch Solving and Verification. For each 16 bits of the key, out of some 4 fixed points for $f_k^{(8)}$ present in our table, we choose two that we assume to be fixed points also for f_k. The guess is correct with probability about 1/6 (as explained above). Therefore we have about 2^{18} tests to perform. For each choice of 16 key bits and two alleged fixed points, we write the usual system of polynomial equations over $GF(2)$ and use a SAT-Solver to solve it. At this stage we have 64 bits of information (two fixed points) and 16+48 bits to be determined (16 are guessed only 48 are still variables). With probability very close to 1 we expect that the system has 0 or 1 solutions. Globally, we expect that this method yields the right solution plus – on average – less than one "false" solution. We did implement this step of the attack and with 64 rounds of KeeLoq, 2 known plaintexts and 16 bits already known, one determines the key with a SAT solver in 0.19 seconds on a 2GHz Pentium-M CPU. This is about 2^{28} CPU clocks. (It is also quite easy with ElimLin, which takes 10 s, which is about 2^{34} CPU clocks.)

The total complexity of this stage of the attack is about $6 \cdot 2^{16+28} \approx 2^{47}$ CPU clocks.

Technical Note: If for a given pair of fixed points there are several solutions, current SAT solvers will find only one, and we need to run the attack again with one or two variables fixed. On average we expect only one solution and the real complexity of Stage 2 is a bit more. However, Stage 1 dominates, and the whole attack takes about 2^{54} CPU clocks.

6.3 Slide-Algebraic Attack 4

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

Stage 1 - Recover 16 Key Bits with a Distinguisher. Let B be a permutation on 32 bit words. From Section 3.1, 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 whether the initial cycle size was congruent to 0 or 2 modulo 4. All cycles of odd size remain intact (points are permuted). Thus, we expect that the number of even cycles will be dived by 2.

Now we look what happens when this operation is repeated 3 times:

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

We expect that B^8 has $23 \mapsto 11.5 \mapsto 5.75 \mapsto 2.825$ which is about 3 cycles of even size left. This property allows to distinguish between $f_k^{(8)}$ and a random permutation that should have about 11-12 even cycles. The proposed distinguisher works as follows: if there are 6 or more cycles, we say it must be the wrong key. Otherwise we say that $k_0, \ldots, k_{15}$ must 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, only such cycles can be still of even size after splitting in 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 – false positive – which amounts to having only 5 or less even-size cycles for a random permutation is 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^{32} 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 8 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 early abort. It is clear that the average complexity of an optimised version of this attack will be not much more than 2^{16+32} 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 2^{48} 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 $f_k^{(8)}$. We need to guess which points are fixed points of f_k and then we solve the (same as before) system of equations in 0.2 s $\approx 2^{28}$ CPU clocks. The complexity of this stage is about $6 \cdot 2^{28} \approx 2^{31}$ 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 2^{48} CPU clocks dominates the attack.

7 Conclusions

In this paper we work on key recovery attacks on KeeLoq, a block cipher with a very small block size and simple periodic structure, that is massively used in the automobile industry.

Recently it has been shown that for complex ciphers such as DES, up to 6 rounds can be broken by an algebraic attack given a tiny quantity of known plaintexts [6]. In this paper we show that up to 128 rounds of KeeLoq can be broken given not more than 4 known plaintexts.

For the full 528-round KeeLoq cipher and given about 2^{16} known plaintexts, we have proposed and implemented a working slide-algebraic attack equivalent to 2^{53} KeeLoq encryptions. This attack is practical and easy to implement.

In addition, we showed that, given 2^{32} known plaintexts, we can recover the key of the full cipher with complexity of about 2^{48} CPU clocks.

References

1. Magali Bardet, Jean-Charles Faugère and Bruno Salvy, *On the complexity of Gröbner basis computation of semi-regular overdetermined algebraic equations,* in Proc. International Conference on Polynomial System Solving (ICPSS,Paris,France), pp.71-75.
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. Keeloq wikipedia article. 25 January 2007. See http://en.wikipedia.org/wiki/KeeLoq.
5. Keeloq C source code by Ruptor. See http://cryptolib.com/ciphers/
6. Nicolas T. Courtois and Gregory V. Bard: *Algebraic Cryptanalysis of the Data Encryption Standard,* Available at http://eprint.iacr.org/2006/402/.
7. 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/.
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. 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

12. 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.
13. 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.
14. F. Massacci. *Using Walk-SAT and Rel-SAT for cryptographic key search, Proc. 16th International Joint Conference on Artificial Intelligence (IJCAI'99).* 1999.
15. 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/
16. 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.
17. Random Permutation Statistics – wikipedia article, 25 January 2007, available at http://en.wikipedia.org/wiki/Random_permutation_statistics
18. Singular: A Free Computer Algebra System for polynomial computations. http://www.singular.uni-kl.de/

A Algebraic Immunity and Boolean Function Used in KeeLoq

A natural question with regard to the KeeLoq Boolean function is what is its "Graph Algebraic Immunity", also known as "I/O degree".

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

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. At present time we are not aware of any attack that is helped by using this equation.