Random Permutation Statistics and An Improved Slide-Determine Attack on KeeLoq

Nicolas T. Courtois[1] and Gregory V. Bard[2]

[1]University College London, Gower Street, London WC1E 6BT, UK
[2]Fordham University, Bronx, NY, 10458, USA

Abstract. KeeLoq is a lightweight block cipher which is extensively used in the automotive industry [7, 8, 14, 15]. Its periodic structure, and overall simplicity makes it vulnerable to many different attacks. Only certain attacks are considered as really "practical" attacks on KeeLoq: the brute force, and several other attacks which require up to 2^{16} known plaintexts and are then much faster than brute force, developed by Courtois *et al.*, [10] and (faster attack) by Dunkelman *et al.* [1].
On the other hand, due to the unusually small block size, there are yet many other attacks on KeeLoq, which require the knowledge of as much as about 2^{32} known plaintexts but are much faster still. There are many scenarios in which such attacks are of practical interest, for example if a master key can be recovered, see Section 2 in [11] for a detailed discussion. The fastest of these attacks is an attack by Courtois, Bard and Wagner from [10] that has a very low complexity of about 2^{28} KeeLoq encryptions on average. In this paper we will propose an improved and refined attack which is faster both on average and in the best case.
We also present an exact mathematical analysis of probabilities that arise in these attacks using the methods of modern analytic combinatorics.
Keywords: block ciphers, highly-unbalanced compressing Feistel ciphers, random permutation statistics, slide attacks, KeeLoq, automobile locks

1 Introduction

KeeLoq is a lightweight block cipher designed in the 1980's and licensed to Microchip [7]. It is used in automobile door locks and alarms, garage door opening systems etc. For many years there was no attack on KeeLoq, because the specification was kept confidential. In 2007, a document with the full specification of KeeLoq was made public on a Russian web site [15]. Many different cryptographic attacks on KeeLoq were published since then [1, 4–6, 10, 11]. Some of these attacks are considered as "practical" attacks [1] though brute force is really the simplest attack to handle in practice.

KeeLoq has unusually small block length: 32 bits. Thus, in theory, the attacker can expect to recover and store the entire code-book of 2^{32} known plaintexts. Then one may wonder whether it is really interesting 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 in the system can be thus recovered, or if very few plaintext/ciphertext pairs are missing and yet are very valuable to the attacker, or if the code-book is noisy and contains errors. There are many other scenarios as well, see Section 2 in [11] for examples and a detailed discussion. One of the two attacks described in

[11] will work as soon as some 60 % of the code-book is available, therefore 40 % of the code-book can still be determined by the attacker. In this paper, for simplicity we will assume that the whole code-book is available.

The starting point in this paper is the Slide-Determine attack by Courtois, Bard and Wagner [10]. This is the fastest attack known when the attacker is in possession of the entire code-book. We will propose an improved and refined version of this attack, that is faster than the original attack, both on average and in the best case.

This paper is organised as follows: in Section 2 we describe the cipher. In Section 3, we establish a number of basic lemmas and observations on KeeLoq. In Section 4 we study various statistics on fixed points and random permutations. In Section 5 we describe our main attack, and we conclude in Section 6.

1.1 Notation

We will use the following notation for functional iteration:

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

If $h(x) = x$ for some function h, then x is a fixed point of h. If $h(h(x)) = x$ but $h(x) \neq x$ then x is a "point of order 2" of h. In like manner, if $h^{(i)}(x) = x$ but $h^{(j)}(x) \neq x$ for all $j < i$, then x is a "point of order i" of h. Obviously if x is a point of order i of h, then

$$h^{(j)}(x) = x \text{ if and only if } i|j$$

2 Cipher Description

The specification of KeeLoq can be found in the Microchip product specification document [15], which actually specifies KeeLoq decryption, that can be converted to a description of the encryption, see [7, 8, 4]. Initially there were mistakes in [7, 8] as opposed to [15, 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 bit of the rotating key.

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 $g_k(x)$ 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 [7], 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 "3A5C742E" when 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:

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

where $(k_{63}, k_{62}, \ldots, k_1, k_0)$ is the original key and $(L_{31}, \ldots, L_0)$ is the plaintext.

3 Preliminary Analysis of KeeLoq

3.1 Brute Force Attack on KeeLoq

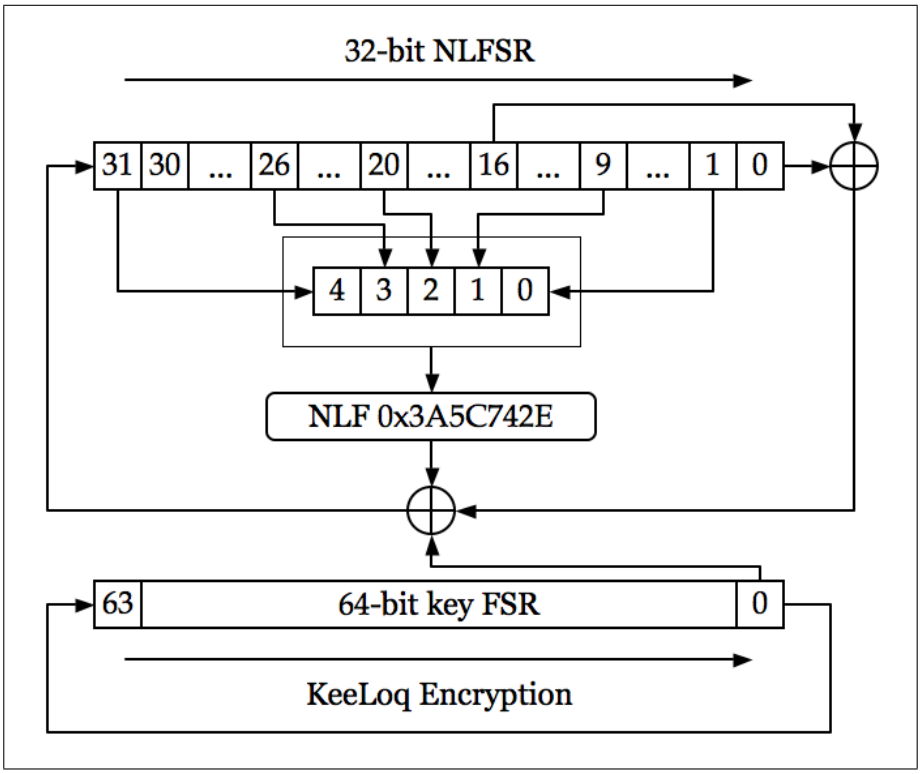
As in [10], in this paper we assume that:

Fact 3.1. 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 528-round KeeLoq, the complexity of the exhaustive key search is about 2^{75} CPU clocks.

We note that computing the entire code-book of KeeLoq requires 2^{32} KeeLoq computations, which is is about 2^{43} CPU clocks. Interestingly, the attacks we study in this paper are always faster than this, as they only have to read the KeeLoq code-book, and can rapidly discard large portions of it.



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

Fig. 1. KeeLoq Encryption

3.2 Useful Lemmas on KeeLoq

In this section we recall some basic facts from [10], with full justification.

Fact 3.2. 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. Therefore using the following formula we learn the key bit in question: $k_{i-32 \ mod \ 64} = L_i \oplus L_{i-32} \oplus L_{i-16} \oplus NLF\left(L_{i-1}, L_{i-6}, L_{i-12}, L_{i-23}, L_{i-31}\right)$. Furthermore, this also shows that there will be exactly one possible key.

Remark: For more rounds it is much less simple, yet as shown in [10], algebraic attacks allow to efficiently recover the key for up to 160 rounds given a very small number of known plaintexts.

Fact 3.3. Given (x, y), one can quickly test whether it is possible that $y = g_k(x)$ for 16 rounds of KeeLoq. The probability that a random (x, y) will pass this test is $1/2^{16}$.

Justification: This test can be implemented with a SAT solver, to find that most systems of equations of this kind are not satisfiable which takes negligible time. However it is in fact much easier.

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 3.4. 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: This is to be done by trying exhaustively all possibilities for the first 16 key bits and applying Fact 3.2.

Fact 3.5. 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 3.3 to test whether it is possible that $c = g_k(p)$. We can notice that because we exploit the periodic structure of KeeLoq, the complexity of this test does not depend on the number of rounds in KeeLoq.

4 Statistics on Cycles in Random Permutations

We start with the following well known fact: which is very frequently used in cryptanalysis:

Proposition 4.1. [cf. [10]] Given a random function from n-bits to n-bits, the probability that a given point y has i pre-images is $\frac{1}{i!e}$, when $n \to \infty$.

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

Justification: We can look at pre-images of 0 with $f_k(x) \oplus x$ that is assumed to behave as a random function. In Appendix A we show that this argument is **not** correct (but it can be repaired and one can show that it will hold for any block cipher that is not weak). The result is true for a random permutation and this is a direct consequence of Proposition B.3 that we will prove in Appendix B.1.

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. All points of order 1 (i.e. fixed points) for π come from points of order 1, 2, 4, and 8, of π. And conversely, all points of order 1, 2, 4, and 8, of π are points of order 1 (i.e. fixed points) for π. For large permutations, all these points are with very high probability distinct fixed points. For independent random events we can just add the expected values.

For another proof of this fact, see Corollary B.7.1 in Appendix B.1.

Proposition 4.4. Let π be a random permutation and $j, k \in \mathbb{N}_+$. The probability that π^k has exactly j fixed points and π has at least 1 fixed point is:

$$e^{-\sum_{i|k} \frac{1}{i}} \cdot S'(j) \quad \text{when} \quad N \to \infty$$
$$\text{where} \quad S'(j) = \left[t^j\right] exp\left(\sum_{i|k} \frac{t^i}{i}\right) - \left[t^j\right] exp\left(\sum_{\substack{i|k \\ i \neq 0}} \frac{t^i}{i}\right)$$

Where t is a formal variable, and $[t^j]f(t)$ denotes the coefficient of t^j in an [infinite] formal series expansion of the function $f(t)$.

Justification: The full proof of this fact is given in Appendix B.2. The proof is based on Proposition II.4. on page 132 in [12]. The last line in Table 1 below gives the numerical values for the probability that π has at least one fixed point and π^8 has at least j cycles, for $j \leq 7$, with π being a random permutation. We have also checked and confirmed all these results by computer simulations.

Table 1. The first values of $S'(j)$ for $k = 8$ and resulting cumulated probabilities

j	0	1	2	3	4	5	6	7
$S'(j)$	0	1	$\frac{1}{2}$	$\frac{2}{3}$	$\frac{7}{24}$	$\frac{7}{15}$	$\frac{151}{720}$	$\frac{67}{315}$
$e^{-15/8} \cdot S'(j)$	0.000	0.153	0.077	0.102	0.045	0.071	0.032	0.032
$\sum_{i=j}^{\infty} e^{-15/8} \cdot S'(i)$	0.632	0.632	0.479	0.402	0.300	0.255	0.184	0.151

5 Our Improved Slide-Determine Attack on KeeLoq

The cipher KeeLoq is an extremely weak cipher when approximately the entire code-book is known. We will now present our attack, that improves the Slide-Determine Attack from [10], both on average and in specific cases.

5.1 Setup and Assumptions

We assume that approximately the entire code-book is known: all possible 2^{32} plaintext-ciphertext pairs. As in [10] we assume that all the 2^{32} 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 estimate. In fact our attack spends most of the time in (consecutive) reading of this memory, its complexity is directly proportional to t_r and as we will see later, the complexity of the attack can be further improved if faster memory is available.

In our Slide-Determine Attack we make the following assumption. We assume that there is at least one fixed point for f_k where f_k represents the first 64 rounds of the cipher. As shown in Section 4, this happens with probability 0.63. We recall that if x is a fixed point of $f_k(\cdot)$ then 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 faster will be the overall attack. Overall, our attack works for 63 % of all keys (cf. Proposition 4.2) but for a smaller fraction of 30 % of all keys (this figure comes from Proposition 4.4 and Table 1) it will be particularly fast, and for about 15 % of keys, even faster.

5.2 Strong Keys in KeeLoq

In contrast, for 37 % of keys for which f_k has no fixed point whatsoever, our Slide-Determine Attack (as well as the more basic version of it in [10]) fails completely. Following [10, 11], they can be used in practice to prevent this attack: the manufacturer or the programmer of a device that contains KeeLoq can check each potential key for fixed points for f_k (2^{32} plaintexts have to be checked). If it has any, that key can be declared "weak" and never used. A proportion of 37 % of all the keys will be strong, and following [10], this changes the effective key space from 64 bits to 62.56 bits. This is somewhat similar to a strong-key solution that was in 2002 patented and commercialized by Gemplus corporation (currently Gemalto) to prevent GSM SIM cards from being cloned, see [13].

Overall, we get a small loss, in many scenarios perfectly acceptable, that makes the cipher more secure. Using these strong keys does totally prevent the attack we study in the present paper. However the Attack B described in [11] still works and KeeLoq is still broken but with a higher complexity: of about 2^{40} KeeLoq encryptions.

5.3 Our Slide-Determine Attack

We consider all 2^{32} plaintext-ciphertext pairs (p, c). We assumed that at least one plaintext 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 1A - Batch Guessing Fixed Points. Following our assumption there is at least one p that is a fixed point for f_k^8. For each pair (p, c), we use Fact 3.5 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 3.2 we can at the same time compute 16 bits of the key with time of about $4 \cdot 16$ CPU clocks (cf. Fact 3.2 and 3.1). To summarize, given the entire 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 2^{16} triples $p, c, (k_{15}, \ldots, k_0)$.

Stage 1B - Sorting/Ranking (Optional). This stage is optional, 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 can make it up to 2^{11} times slower, depending on the values of t_r and j_8. To bridge this gap we wish to exclude a proportion of up to $(1 - 1/2^{11})$ of all the pairs. In practice, we will not exclude any case, but rather order them in such a way that those that appear at the end are unlikely to be ever used, and those at the front of the list are much more likely. The higher is the value of j_8, the faster will be the attack and both the best-case and average complexity of the whole attack will be improved. This is achieved 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. This gives us a ranking procedure: the more triples associated with a partial-key value, the higher it should be ranked. Now we sort them using these counts f, and we enumerate the suggested values of $(k_{15}, \ldots, k_0)$ in order of descending values of f. When we are considering a fixed value of $(k_{15}, \ldots, k_0)$, we examine all of the triples $p, c, (k_{15}, \ldots, k_0)$ associated with that value of $(k_{15}, \ldots, k_0)$, and we apply later Stages 2 and 3 of our Attack. When at the Stage 3 the correct key is found, we will stop and never try the remaining triples.

The key remark is that, if $j_8 > 0$, then the real 16 bits of the actual key $(k_{15}, \ldots, k_0)$ must appear at least j_8 times in our list. This means that, if j_8 is large, our attack will terminate earlier. How earlier exactly depends in how many other keys appear at least j_8 times in our list. This number 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 value of j_8 is unknown to the attacker and we simply try all values f in our ranking in decreasing order. Overall, after Stage 1A we have less than 2^{16} 16-bit keys, but still 2^{16} triples to test in later stages of our attack. Let $U(j_8)$ be the total number of new keys that need to be examined at this stage and let $T'(j_8) = j_8 \cdot U(j_8)$ be the total number of triples that exist at this stage. For simplicity, we consider all cases $j_8 \geq 7$ together in one step and therefore we put $T'(7) = 2^{16} \cdot \sum_{i \geq 7} i \cdot \frac{1}{ei!}$ and $T'(j_8) = 2^{16} \cdot i \cdot \frac{1}{ei!}$ for $j_8 \in \{1, 2, 3, 4, 5, 6\}$. Let $T(j_8)$ be the total cumulated number of triples $p, c, (k_{15}, \ldots, k_0)$ defined as $T(j_8) = \sum_{i \geq j_8} T'(i) = 2^{16} \cdot \sum_{i \geq j_8} i \cdot \frac{1}{ei!}$. This gives depending on j_8:

Table 2. The number of $(k_{15}, \ldots, k_0)$ that have to be tested on average in our attack

j_8	≥ 7	6	5	4	3	2	1
$T'(j_8)$	$2^{5.3}$	$2^{7.7}$	$2^{10.0}$	$2^{12.0}$	$2^{13.6}$	$2^{14.6}$	$2^{14.6}$
$T(j_8)$	$2^{5.3}$	$2^{7.9}$	$2^{10.3}$	$2^{12.4}$	$2^{14.1}$	$2^{15.3}$	$2^{16.0}$

Overall, for a fixed value of j_8, in our attack we need to test all the triples counted by $T'(j)$ in columns $j > j_8$, and half of the triples (on average) in the column corresponding to j_8.

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 that less than 2 are valid on average, as we expect on average between 1 and 2 fixed points for f_k (we assumed at least one). In fact, our attack with ranking is executed 'in chunks' and will complete and stop as soon as we encounter one of them, but we are conservative and will assume that all the j_8 must be tested. 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, and if we guess additional 16 bits of the key, it takes less than 0.1 s, see [10]. But there is a simpler and direct method to get the same result. We use Fact 3.4 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. Without ranking and early abort (cf. Step 1B), this takes time equivalent to 2^{32} computations of $h_k(\cdot)$, which is about 2^{40} CPU clocks. When we execute our attack with Step 1B, we only compute these keys as needed, in order of decreasing j_8, and we expect to compute only at most $2^{16} \cdot T(j_8)$ of these full keys on 64 bits, before the attack succeeds. This gives a time of about $2^{16} \cdot T(j_8) \cdot 4 \cdot 48$ CPU clocks.

Stage 3 - Verification. Finally, we test each of these up to 2^{32} complete keys 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.

Without the ranking method (Stage 1B), this stage would require up to 2^{32} full 528-round KeeLoq encryptions, which would dominate the complexity of the whole attack. With Stage 1B, we need at most $2^{16} \cdot T(j_8)$ full KeeLoq encryptions.

5.4 Complexity Analysis

In the following table we compute the complexity of Stages 1, 2, 3, assuming successively that $j_8 \geq 7$, then $j_8 = 6, j_8 = 5, \ldots, j_8 = 1$. Since the complexity of Stage 1 greatly depends on how fast is the speed of (sustained consecutive) RAM access compared to the speed of the CPU, we give two versions. In very realistic "Stage 1" we assume that $t_r = 16$ CPU clocks. In the very optimistic "Stage 1*" we assume that $t_r = 1$ CPU clocks, and the comparison of 16 bits that is executed in Stage 1A in order to check whether p is likely to be a fixed point, is executed in parallel by the same CPU.

The attack has to be executed in the order of columns in the following table. The probabilities of each case are computed following Proposition 4.4 and Table 1. The complexities are converted to full KeeLoq computations, which means that the unit is 2^{11} CPU clocks.

Table 3. The complexity of each stage of our slide-determine attack

j_8	≥ 7	6	5	4	3	2	1
probability/keys	0.151	0.032	0.072	0.045	0.102	0.077	0.153
cumulated keys	0.151	0.184	0.255	0.300	0.402	0.479	0.632
$T'(j_8)$	$2^{5.3}$	$2^{7.7}$	$2^{10.0}$	$2^{12.0}$	$2^{13.6}$	$2^{14.6}$	$2^{14.6}$
Stage 1	2^{27}	–	–	–	–	–	–
Stage 1*	2^{23*}	–	–	–	–	–	–
Stage 2	$2^{17.7}$	$2^{20.2}$	$2^{22.6}$	$2^{24.6}$	$2^{26.1}$	$2^{27.1}$	$2^{27.1}$
Stage 3	$2^{21.1}$	$2^{23.7}$	$2^{26.0}$	$2^{28.0}$	$2^{29.6}$	$2^{30.6}$	$2^{30.6}$
Stage 2+3	$2^{21.2}$	$2^{23.8}$	$2^{26.1}$	$2^{28.1}$	$2^{29.7}$	$2^{30.7}$	$2^{30.7}$

Now we will compute the complexity of the full attack.

5.5 Results

We will distinguish 3 versions of the attack.

Optimistic Version 1: In the most optimistic scenario we can think of, assuming $t_r = 1$ and $j_8 \geq 7$ which is the case for 15 % of all keys (cf. Table 1), the total complexity is dominated by Stage 1* and the whole attack requires only 2^{23} KeeLoq encryptions.

Restricted Realistic Version 2: In a fully realistic scenario, with $t_r = 16$, assuming $j_8 \geq 4$ which happens for 30 % of all keys (cf. Table 1), the total complexity is dominated by Stage 1, and the whole attack requires on average $2^{27.4}$ KeeLoq encryptions. This number is obtained from Table 3 and Table 1 and we compute an approximation of the average complexity as follows: $2^{27} + 0.072/0.3 \cdot 1/2 \cdot 2^{26.1} + 0.045/0.3 \cdot 2^{26.1} + 0.045/0.3 \cdot 1/2 \cdot 2^{28.1}$. This is because for these 30 % of the keys, when $j_8 > 5$, the Stages 2+3 take negligible time compared to 2^{27}, and we require $2^{26.1}$ and $2^{28.1}$ steps only for a fraction of all keys. For example only for about 0.045/0.3 of these keys with $j_8 \geq 4$, we need $2^{28.1}$ operations, and half of this number on average for this interval.

General Version 3: In order to calculate the expected (average) complexity of our attack (in the realistic scenario) we add the complexity of the Stage 1

that is always executed, and for Stage 2+3 we need to compute an integral of an increasing function that is approximated by a straight line in each of the 7 intervals. This is done as follows. We add the full cost of all previously considered valued j_8, and for the corresponding fraction of keys corresponding to the current j_8, the actual number of steps in Stages 2+3 that has to be executed is on average half of the figure given in the last row of Table 3. Let $C'_{2+3}(j)$ be the figure given in the last row, and let $p'(j)$ be the probability from the first row, and $p(j)$ be the cumulative probability from the second row. The expected total complexity of our attack is about

$$2^{27} + \sum_{j=1}^{7} C'_{2+3}(j) \cdot \frac{p'(j)/2 + 0.632 - p(j)}{0.632} \approx 2^{29.6}.$$

Therefore our combined attack with ranking (Stage 1B) runs for 63 % of keys (cf. Proposition 4.2), the worst-case running time is 2^{32} KeeLoq encryptions (in which case all the figures in the last columns have to be added), and the average running time is about $2^{29.6}$ KeeLoq encryptions.

Summary of Our Slide-Determine Attack.

To summarize, for 15 % of the keys the complexity can be a low as 2^{23} KeeLoq encryptions, but only if the memory is very fast. For 30 % of all keys the complexity is 2^{27} KeeLoq encryptions with realistic assumptions on the speed of memory, and overall for all 63 % of keys for which f_k has a fixed point, the average complexity is $2^{29.6}$ KeeLoq encryptions.

6 Conclusion

KeeLoq is a block cipher that is in widespread use throughout the automobile industry and is used by millions of people every day. KeeLoq is a weak and simple cipher, and has several vulnerabilities. Its greatest weakness is clearly the periodic property of KeeLoq that allows many sliding attacks which are very efficient [1, 4–6, 10, 11]. Their overall complexity does not depend on the number of rounds of the cipher and in spite of having 528-rounds KeeLoq is insecure.

In addition, KeeLoq has a very small block size of 32 bits which makes it feasible for an attacker to know and store more or less the whole code-book. But when the attacker is given the whole code book of 2^{32} plaintexts-ciphertext pairs, KeeLoq becomes extremely weak. In [10] Courtois Bard and Wagner showed that it can be broken in time which is faster than the time needed to compute the code-book by successive application of the cipher. This is the Slide-Determine attack from [10] which is also the fastest key recovery attack currently known on KeeLoq. In this paper we have described and analysed a refined and improved Slide-Determine attack that is faster both on average and in specific sub-cases (cf. Table 4 below). For example, for 30 % of all keys, we can recover the key of the full cipher with complexity equivalent to 2^{27} KeeLoq encryptions. This attack does not work at all for a proportion of 37 % so called strong keys [10, 11]. However the Attack B described in [11] still works.

In Table 4 we compare our results to previously published attacks on KeeLoq [1, 4–6, 10, 11], We can observe that the fastest attacks do unfortunately require the knowledge of the entire code-book. Then, there are attacks that require 2^{16} known plaintexts but are slower. Finally, there is brute force attacks that are the slowest. In practice however, since they require only 2 known plaintexts and are actually feasible, brute force will be maybe the only attack that will be executed in practice by hackers. 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 and in many different attack scenarios.

Table 4. Comparison of our attacks to other attacks reported on KeeLoq.

Type of attack	Data	Time	Memory	Reference
Brute Force	2 KP	2^{63}	tiny	
Slide-Algebraic	2^{16}KP	$\mathbf{2^{53}}$	small	Slide-Algebraic Attack 2 of [10]
Slide-Meet-in-the-Middle	2^{16}KP	$\mathbf{2^{46}}$	small	Preneel *et al*[1]
Slide-Meet-in-the-Middle	2^{16}CP	$\mathbf{2^{45}}$	small	Preneel *et al*[1]
Slide-Correlation	2^{32}KP	2^{51}	16 Gb	Bogdanov[4, 5]
Slide-Cycle-Algebraic	2^{32}KP	2^{40}	18 Gb	Attack A in [11],
Slide-Cycle-Correlation	2^{32}KP	2^{40}	18 Gb	Attack B in [11], cf. also [5]
				Two previous attacks in [10]
Slide-Determine	2^{32}KP	2^{31}	16 Gb	A: for 63 % of all keys
Slide-Determine	2^{32}KP	2^{28}	16 Gb	B: for 30 % of all keys
				THIS PAPER:
				Our three new refined attacks
Slide-Determine	2^{32}KP	$\mathbf{2^{30}}$	16 Gb	Average time, 63 % of keys
Slide-Determine	2^{32}KP	$\mathbf{2^{27}}$	16 Gb	Realistic, 30 % of keys
Slide-Determine	2^{32}KP	$\mathbf{2^{23}}$	16 Gb	Optimistic, 15 % of keys

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

Acknowledgments We are greatly indebted to Sebastiaan Indesteege, David Wagner and Sean O'Neil, without whom this paper would never have been written.

References

1. Eli Biham, Orr Dunkelman, Sebastiaan Indesteege, Nathan Keller, Bart Preneel: How to Steal Cars A Practical Attack on KeeLoq, in Eurocrypt 2008, LNCS 4965, pp. 1-18, Springer, 2008.
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. Keeloq wikipedia article. On 25 January 2007 the specification given here was incorrect but was updated since. See http://en.wikipedia.org/wiki/KeeLoq.
8. Keeloq C source code by Ruptor. See http://cryptolib.com/ciphers/
9. Nicolas Courtois: Examples of equations generated for experiments with algebraic cryptanalysis of KeeLoq, http://www.cryptosystem.net/aes/toyciphers.html.
10. Nicolas Courtois, Gregory V. Bard, David Wagner: *Algebraic and Slide Attacks on KeeLoq,* In FSE 2008, pp. 97-115, LNCS 5086, Springer. Older (partly out of date) preprint available at eprint.iacr.org/2007/062/.
11. Nicolas Courtois, Gregory V. Bard and Andrey Bogdanov: *Periodic Ciphers with Small Blocks and Cryptanalysis of KeeLoq,* In Tatra Mountains Mathematic Publications, post-proceedings of Tatracrypt 2007 conference, The 7th Central European Conference on Cryptology, June 22-24, 2007, Smolenice, Slovakia.
12. Philippe Flajolet and Robert Sedgewick; *Analytic Combinatorics,* book of 807 pages, to apear in Cambridge University Press in the first half of 2008. Available **in full** on the Internet, see http://algo.inria.fr/flajolet/Publications/book.pdf
13. 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.
14. Microchip. An Introduction to KeeLoq Code Hopping. Available from http://ww1.microchip.com/downloads/en/AppNotes/91002a.pdf, 1996.
15. Microchip. Hopping Code Decoder using a PIC16C56, AN642. Available from http://www.keeloq.boom.ru/decryption.pdf, 1998.
16. Marko R. Riedel, *Random Permutation Statistics, paper available on the internet, at http://www.geocities.com/markoriedelde/papers/randperms.pdf.*
17. *Serge Vaudenay: Communication Security: An Introduction to Cryptography, textbok, 2004 edition, Ecole Polytechnique Fédérale de Lausanne, Swiss Federal Institute of Technology.*
18. *Thomas Eisenbarth, Timo Kasper, Amir Moradi, Christof Paar, Mahmoud Salmasizadeh, Mohammad T. Manzuri Shalmani: On the Power of Power Analysis in the RealWorld: A Complete Break of the KeeLoq Code Hopping Scheme,* In Crypto 2008, LNCS 5157, pp. 203-220, Springer.

A About a Frequently Used Argument on Random Permutations and Random Functions

This section expands one point that arises in Section 4. It is totally secondary in this paper, except that it discussed the validity of Proposition 4.2 in both theory and practice (the numerical precision we can expect for KeeLoq). We recall that:
Proposition 4.1.[cf. Section 4] Given a random function $GF(2)^n \rightarrow GF(2)^n$ the probability that a given point y has i pre-images is $\frac{1}{ei!}$, when $n \rightarrow \infty$.
Justification: Let y be fixed, the probability that $f(x) = y$ is $1/N$ where $N = 2^n$ is the size of the space. There are N^N possible random functions. Each possible subset of i out of exactly N pre-images $x_1, \ldots, x_n$ constitutes an independent disjoint event the probability of which is $(N-1)^{N-i}/N^N$. We multiply by $\binom{N}{i}$ and get: $\binom{N}{i}(1-1/N)^{N-i}1/N^i \approx \frac{1}{ei!}$ when $N \rightarrow \infty$. This is a Poisson distribution with the average number of pre-images being $\lambda = 1$.

From this, as we already explained, it is a common practice to argue that:
Proposition 4.2.A. A random permutation $GF(2)^n \rightarrow GF(2)^n$ has 1 or more fixed points with probability $1 - 1/e \approx 0.63$, when $n \rightarrow \infty$
Proposition 4.2.[cf. Section 4] The same holds first 64 rounds f_k of KeeLoq.

Justification and Discussion: In Section 4, to justify this result, we have assumed that $f_k(x) \oplus x$ is a pseudo-random function, and looked at the number of pre-images of 0 with this function. Therefore we need Proposition 4.2.A. However one can **not** prove Proposition 4.2.A. in the suggested way from Proposition 4.1. This would be incorrect. Below we explain the argument in details, why it is incorrect, and we will propose two different ways to fix it.

A.1 Does Proposition 4.2.A. Hold for Random Permutations and can the Result be Applied to Block Ciphers?

The Faulty Argument. Our argument is not valid as it stands for random permutations. Let $\pi : \{0,1\}^n \rightarrow \{0,1\}^n$ be a random permutation. We cannot claim that $Q(x) = \pi(x) \oplus x$ is a random function. In fact it isn't a random function. Let $N = 2^n$ be the size of the domain. For a random function, by a birthday paradox, after $\sqrt{N}$ evaluations, we would have with probability about 0.39 (cf. page 134 in [17]) the existence of at least two points $x \neq x'$ such that $Q(x) \oplus x = Q(x') \oplus x'$. Here this would imply $\pi(x) = \pi(x')$ which is impossible for a permutation.

Thus the argument as it stands, claiming that $Q(x)$ is a random function is not a valid mathematical argument for random permutations. A rigourous proof of Proposition 4.2.A. will be given in Appendix B. We will in fact show a more general fact, see Proposition B.3. Interestingly, we can also 'repair' our failed proof, through a cryptographic argument about indistinguishability. More precisely, we will show below that Proposition 4.2.A. holds with (at least) a very good precision for random permutations. Importantly, by the same method we will later see that this can also work for a block cipher such as KeeLoq. Quite surprisingly it is possible to show that Proposition 4.2. holds with a sufficient precision for a real-life cipher such as KeeLoq, if we make a cryptographic assumption about f_k.

A.2 How to Repair the Faulty Argument

Does Proposition 4.2.A. Give a Good Approximation of the Result for Random Permutations? We want to show that for a random permutation the predicted probability $1 - 1/e$ is a very good approximation. Roughly speaking this can be shown as follows. We consider an adversary that can do only K computations, where $K \ll \sqrt{N}$. It should be then then possible to show, by the methods from the Luby-Rackoff theory, that the probability advantage for distinguishing $Q(x) = \pi(x) \oplus x$ from a random function is negligible. Indeed, for K queries to a random function, the probability that there is a collision on $x \mapsto Q(x) \oplus x$ is about $1 - e^{-K^2/2N}$, see page 134 in [17]. This probability will be negligible as long as $K \ll \sqrt{N}$. So though, $Q(x) = \pi(x) \oplus x$ is not a random function, the difference cannot be observed by our adversary. Then, any non negligible difference between he probability that π has a fixed point, and 0.63 would mean that $Q(x) = \pi(x) \oplus x$ can be distinguished from a random function, which is impossible when $K \ll \sqrt{N}$. We need to note here that we didn't really give here a complete proof, just explained the main idea. Instead we will prove Proposition 4.2.A. mathematically in Appendix B.

Is This Approximation Good for KeeLoq? In real life, we are not dealing with random permutations, but with real-life objects such as KeeLoq. Can Proposition 4.2. be shown to hold, at least with some precision, for the 64 rounds of KeeLoq? We will show, by a straightforward black-box reduction argument, that the difference between the observed probability and $1 - 1/e$ cannot be 'too large' for KeeLoq. This is if we assume that f_k is an (even very moderately) secure block cipher and admits no very efficient distinguisher attack.

Roughly speaking the argument is as follows. Assume that $R_k(x) = f_k(x) \oplus x$ can only be distinguished from a pseudo-random function with a probability advantage bigger than say 0.1, given at least 2^{16} computations and 2^{16} queries to $R_k(x)$. We cannot expect a much stronger assumption, because here $N = 2^{32}$ and the birthday paradox allows us to realize that we will never find two points such that $R_k(x) \oplus x = R_k(x') \oplus x'$, which would otherwise happen if R_k really was a random function. We assume that there is no better distinguisher attack than this attack (with a probability advantage of about 0.39, cf. page 134 in [17]). With the same time complexity and the same number of queries to $R_k(x)$, by using the same rule as in Linear Cryptanalysis, we can measure the frequency of fixed points with precision of the standard deviation that will be about $1/\sqrt{2^{16}} = 2^{-8}$, which is a good precision. The conclusion is that, any deviation from $(1 - 1/e)$ bigger than a small multiple of 2^{-8}, will be visible to the attacker and will be detected with a high probability. It would actually give us a distinguisher attack on f_k that works with complexity being either less then 2^{16} or for a probability advantage bigger than 0.1, which we assumed impossible.

B Appendix: Fixed Points and Small Cycles in Random Permutations

B.1 On Points Lying in Small Cycles for a Random Permutation

We denote the set of all positive integers by $\mathbb{N}_+$. In this section we study general permutations $\pi : \{0, \ldots, N-1\} \to \{0, \ldots, N-1\}$ with N elements, for any $N \in \mathbb{N}_+$, i.e. N does not have to be of the form 2^n (in KeeLoq $N = 2^{32}$).

Random permutations are frequently studied. Nevertheless, we need to be careful, because a number of facts about random permutations are counter-intuitive, see for example Proposition B.5 below. Other intuitively obvious facts are not all obvious to prove rigourously, as illustrated in the previous section when trying to prove that the Proposition 4.2 holds for random permutations. For these reasons, in this paper we will be 'marching on the shoulders of giants' and will use some advanced tools, in order to be able to compute the exact answers to all our questions directly, without fear of making a mistake, and with a truly remarkable numerical precision. We will be using methods of the modern analytic combinatorics. It is a highly structured and very elegant theory, that stems from the method of generating series attributed to Laplace, and is greatly developed in the recent monograph book by Flajolet and Sedgewick [12].

Definition B.1 (EGF). If $A_i \in \mathbb{N}$ is the number of certain combinatorial objects of size i, the EGF (exponential generating function) of this object is defined as a formal (infinite) series:

$$a(z) = \sum_{i \geq 0} \frac{A_i}{i!} z^n.$$

The probability that a random permutation belongs to our class is $p_A = \lim_{i \to \infty} \frac{A_i}{i!}$. We assume that the combinatorial object is such that there is no sudden jumps, discontinuities, or particular cases, and this limit exists. Then the question is how to compute this limit.

Proposition B.1. With the above notations, we consider the analytical expression of the function $h(z) = a(z)(1-z)$ as a function $\mathbb{R} \to \mathbb{R}$. Then $p_A = h(1)$.

Justification: By definition of $a(z)$, this function has the following Taylor series expansion stopped at order k:

$$a_k(z) = \frac{A_0}{0!} + \sum_{1 \leq i \leq k} (\frac{A_n}{n!} - \frac{A_{n-1}}{(n-1)!}) z^i$$

and the series $a(z)$ is well defined and converges when $z = 1$ because for every k we have: $a_k(1) = A_n/n!$ and we assumed that $A_n/n!$ converges.

All non-trivial combinatorial results in this paper are direct consequences of one single fact that comes from the representation of permutations as sets of cycles, and that appears explicitly as Proposition II.4. on page 132 in [12].

Proposition B.2. Let $\mathcal{A} \subset \mathbb{N}_+$ be an arbitrary subset of cycle lengths, and let $\mathcal{B} \subset \mathbb{N}_+$ be an arbitrary subset of cycle sizes. The class $\mathcal{P}^{(\mathcal{A},\mathcal{B})}$ of permutations with cycle lengths in A and with cycle number that belongs to B has EGF as follows:

$$g(z) = \mathcal{P}^{(\mathcal{A},\mathcal{B})}(z) = \beta(\alpha(z)), \qquad \text{where } \alpha(x) = \sum_{i\in A} \frac{x^i}{i} \text{ and } \beta(x) = \sum_{i\in B} \frac{x^i}{i!}$$

This result is exploited extensively in a paper by Marko R. Riedel dedicated to random permutation statistics, see [16]. One direct consequence of it is the following fact:

Corollary B.2.1. The EGF of all permutations that do not contain cycles that belong to $\mathcal{A}$ is equal to

$$g(z) = exp(f(z))$$

where $f(z)$ is obtained by dropping all the terms in z^i for all the $i \in \mathcal{A}$, from the formal series $ln(\frac{1}{1-z}) \stackrel{def}{=} \sum_{i\geq 1} \frac{z^i}{i}$, and where $exp(x)$ can be defined as an operation on arbitrary formal series defined by $exp(x) = \sum_{i\geq 0} \frac{x^i}{i!}$.

Now we are ready to show, in a surprisingly direct and easy way, our generalised version of our Proposition 4.2.

Proposition B.3. Let π be a random permutation of size N. The probability that π has no cycles of length k is $e^{-\frac{1}{k}}$ when $N \to \infty$.

Justification: Following Corollary B.2.1, the EGF of permutations with no cycles of length k is:

$$g(z) = exp\left(log(\frac{1}{1-z}) - \frac{z^k}{k}\right) = \frac{1}{1-z} \cdot exp\left(-\frac{z^k}{k}\right)$$

The following Proposition B.1, we have $p_A = h(1)$ with $h(z) = g(z)(1-z)$ and therefore $p_A = h(1) = e^{-\frac{1}{k}}$.

On precision of these estimations: This result means that $p_A \to e^{-\frac{1}{k}}$ when $N \to \infty$. What about when $N = 2^{32}$? We can answer this question easily by observing that the Taylor expansion of the function $g(z)$ gives all the **exact** values of $A_n/n!$. For example when $k = 4$ we computed the Taylor expansion of $g(z)$ at order 201, where each coefficient is a computed as a fraction of two large integers. This takes less than 1 second with Maple computer algebra software. The results are surprisingly precise: the difference between the $A_{200}/200!$ and the limit is less than 2^{-321}. So the convergence is very fast and even for very small permutations (on 200 elements) our result is extremely accurate.

Proposition B.4. The probability that π has no cycles of length $i \in \mathcal{A}$ is:

$$p_A = \prod_{i\in\mathcal{A}} e^{-\frac{1}{i}} = e^{-\sum_{i\in\mathcal{A}} \frac{1}{i}} \quad \text{when } N \to \infty$$

Justification: The proof is the same as for Proposition B.3. The EGF is equal to:

$$g(z) = \frac{1}{1-z} \cdot \prod_{i\in\mathcal{A}} exp\left(-\frac{z^i}{i}\right)$$

Remark. This shows that for two fixed integers, $i \neq j$ the events that π has no cycle of length i and j respectively, are independent when N is large, which is

what we expect because these events concern (on average) a very small number of points as compared to N. Moreover, when N is not large compared to the numbers in $\mathcal{A}$, these events are not be independent, but we can still compute the **exact** probability for any N by taking the appropriate term in Taylor expansion of $g(z)$, and see how precise is our approximation.

Another result that can also be obtained as a direct consequence of Corollary B.2.1, is interesting mainly because it is quite counter-intuitive:

Proposition B.5. If we fix x and choose a random permutation π on n elements, the probability that x lives in cycle of length i is exactly $1/n$, for all $1 \leq i \leq n$, i.e. this probability does NOT depend on the value of i.

Justification: This result can very easily be shown by extending $g(z)$ to a bivariate generating function and formal derivation w.r.t. one variable, see [16].

B.2 On Cycles in Iterated Permutations

Proposition B.6. A point x is a fixed point for π^k if and only if it lives in a cycle of length i for π for some positive integer $i|k$.

Justification: Let i be the cycle size for π of a given fixed point x of π^k. So $\pi^i(x) = x$ and it is the smallest such integer $i > 0$, i.e. $\pi^j(x) \neq x$ for any $0 < j \leq i-1$. Then for any $k > i$, $\pi^k(x) = \pi^{k \bmod i}(x)$. Now if x is a fixed point for π^k, then $x = \pi^k(x) = \pi^{k \bmod i}(x)$ and this can only be x when $k \bmod i = 0$. Conversely, any point in a cycle of positive length $i|k$ gives a fixed point for π^k.

Proposition B.7. Let $k \in \mathbb{N}_+$. The expected number of fixed points of π^k is exactly $\tau(k)$ where $\tau(k)$ is the number of all positive divisors of k.

Justification: A random permutation π has 1 fixed point on average. Then in addition to possible "natural" fixed points (1 on average) the π^k will also "inherit" all fixed points of π^i, $i|k$ that will be with high probability all distinct fixed points. A rigourous proof goes as follows. Following Proposition B.6, we need to count all points that live in a cycle of length $\in \mathcal{A}$ for π itself. We define an indicator variable I_x as equal to 1 if x lives in a cycle of length $\in \mathcal{A}$, and 0 otherwise. Then following Proposition B.5, $I_x = 1$ with probability $|\mathcal{A}|/n$ and by summation over all possible values of x we get that the expected number of x living in such a cycle is exactly $\tau(k) = |\mathcal{A}|$.

Corollary B.7.1. The expected number of fixed points for the 512 rounds f_k^8 of KeeLoq is exactly $\tau(8) = 4$.

Proposition B.8. Let π be a random permutation and $k > 0$. The probability that π^k has no fixed points is:

$$e^{-\sum_{i|k} \frac{1}{i}} \quad \text{when} \quad N \to \infty$$

Justification: We combine Proposition B.4 and Proposition B.6.

Proposition B.9. Let π be a random permutation and $j, k \in \mathbb{N}_+$. The probability that π^k has exactly j fixed points is:

$$e^{-\sum_{i|k} \frac{1}{i}} \cdot S(j) \quad \text{when} \quad N \to \infty \quad \text{where} \quad S(j) = \left[t^j\right] exp\left(\sum_{i|k} \frac{t^i}{i}\right)$$

Justification: Let $\mathcal{A} = \{i \in \mathbb{N}_+ \text{ such that } i|k\}$. Any permutation can be split in a unique way as follows: we have one permutation $\pi_{\mathcal{A}}$ that has only cycles of type $i \in \mathcal{A}$, and another permutation $\pi_{\mathcal{A}^c}$ that has no cycles of length $i \in \mathcal{A}$. This, following the terminology of [12] can be seen as a "labelled product" of two classes of permutations. Following Section II.2. in [12]), the corresponding EGF are simply multiplied. We need to compute them first.

Following Proposition B.6 the number of fixed points for π^k is exactly the size of $\pi_{\mathcal{A}}$. which must be equal to exactly j for all the permutations in the set we consider here. We compute the EGF of permutations $\pi_{\mathcal{A}}$ of size exactly j and that have only cycles of size $i \in \mathcal{A}$. This EGF is simply obtained by taking the part of degree exactly j in the appropriate EGF written directly following Proposition B.2 as follows:

$$g_{\mathcal{A},j}(t) = z^j \cdot \left[t^j\right] exp\left(\sum_{i|k} \frac{t^i}{i}\right)$$

Finally, we multiply the result by the EGF from Proposition B.4:

$$g(z) = \frac{1}{1-z} \cdot exp\left(-\sum_{i\in\mathcal{A}} \frac{z^i}{i}\right) \cdot z^j \cdot \left[t^j\right] exp\left(\sum_{i|k} \frac{t^i}{i}\right)$$

It remains to apply the Proposition B.1 and we get the result.

Finally we will prove the following result, already announced in Section 3:

Proposition 4.4 Let π be a random permutation and $j, k \in \mathbb{N}_+$. The probability that π^k has exactly j fixed points and π has at least 1 fixed point is:

$$e^{-\sum_{i|k} \frac{1}{i}} \cdot S'(j) \quad \text{when } N \to \infty$$
$$\text{where } S'(j) = \left[t^j\right] exp\left(\sum_{i|k} \frac{t^i}{i}\right) - \left[t^j\right] exp\left(\sum_{\substack{i|k \\ i\neq 0}} \frac{t^i}{i}\right)$$

Justification: As in the previous proposition, our permutations can be constructed as a "labelled product" of permutations that have only cycles of type $i \in \mathcal{A}$ and at least 1 fixed point, and permutations with no cycles of length $i \in \mathcal{A}$. The first EGF is then derived exactly as in the previous proposition except that, from $g_{\mathcal{A},j}(t)$ we have to remove the part of degree exactly j in the EGF of all permutations that have no fixed points, and therefore have cycle lengths in $\mathcal{A} - \{1\}$. Thus we replace $g_{\mathcal{A},j}(t)$ by:

$$z^j \cdot \left[t^j\right] exp\left(\sum_{i|k} \frac{t^i}{i}\right) - z^j \cdot \left[t^j\right] exp\left(\sum_{\substack{i|k \\ i\neq 0}} \frac{t^i}{i}\right)$$

Corollary B.9.1. In Table 1 on page 6 we computed the numerical values for $k = 8$ needed in this paper.