Tatra Mt. Math. Publ. **41** (2008), 167–188

TATRA MOUNTAINS
Mathematical Publications

PERIODIC CIPHERS WITH SMALL BLOCKS AND CRYPTANALYSIS OF KEELOQ

Nicolas T. Courtois — Gregory V. Bard — Andrey Bogdanov

ABSTRACT. KeeLoq is a lightweight block cipher that is massively used in the automobile industry [12, 13, 31, 32]. KeeLoq has two remarkable properties: it is periodic and has a very short block size (32 bits). Many different attacks on KeeLoq have been published in recent years [8, 15, 9, 10, 5]. In this paper we study a unique way of attacking KeeLoq, in which the periodic property of KeeLoq is used in to distinguish 512 rounds of KeeLoq from a random permutation. Our attacks require the knowledge of the entire code-book and are not among the fastest attacks known on this cipher. However one of them works for 100 % of all keys, including so called "strong keys", see [15]. In general, it is important to show how many different attacks are possible on a weak cipher such as KeeLoq.

1. Introduction

KeeLoq was designed in the 1980's by Willem Smit from South Africa [36]. It 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... [12, 13, 31, 32]. In 1995 KeeLoq was sold to Microchip Technology Inc. for more than 10 million US dollars (which is documented in [12]). Following Microchip, [33], the specification of KeeLoq is "not secret" but is patented and was released only under license. The question whether one can really break into cars, and how, is a secondary one in a scientific paper. The main question is what ciphers could be broken, how, due to what properties, and how to design better ciphers.

KeeLoq operates with 32-bit blocks and 64-bit keys. Compared to the typical block ciphers which have a few carefully-designed rounds, this cipher has 528 extremely simple rounds with extremely few intermediate variables (in our formulation one per round). Also, only one bit of the state is modified in each round. As a result, KeeLoq can be implemented very efficiently in hardware.

2000 Mathematics Subject Classification: 11T71, 14G50, 94A60.
Keywords: block ciphers, KeeLoq, iterated permutation, slide attacks, algebraic cryptanalysis, linear cryptanalysis, Boolean functions, SAT solvers.

Sometimes is conjectured, that ciphers which require a small number of gates, will be vulnerable to algebraic cryptanalysis, see [24, 18]. Indeed, several "direct" algebraic attacks are studied in [15] and the simplicity of KeeLoq makes it breakable by simple algebraic attacks for up to 10 rounds. More generally, we wonder what key recovery attacks are possible for KeeLoq? We believe that all attacks on this unusual cipher are interesting, not merely the fastest attacks, as they should also be applicable to other similar ciphers.

An important observation allows really efficient attacks on KeeLoq: the cipher has a periodic structure with a period of 64 rounds. This allows several rather successful attacks on KeeLoq, in spite of the fact that it has 528 rounds. In general, the complexity of many attacks simply does not depend on the number of rounds of the cipher.

This paper is organised as follows. In Section 2 we discuss the unusual properties of ciphers with small blocks and the known plaintext requirements of our attacks. In Section 3 we describe the cipher. In Section 4 we introduce some useful results. In Section 5 we present a slide-algebraic attack that uses the periodicity of KeeLoq and SAT solvers. The first step of this attack is reused in Section 6, where we present a correlation attack with the same complexity, that works for more keys but requires the entire code-book (as opposed to 60 % of it). In Section 7 we discuss strong keys in KeeLoq. In Appendix we present some experimental results which justify claims made in the text.

1.1. Notation

We will use the following notation for functional iteration:

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

2. On the philosophy of block ciphers with small blocks

Abstractly, a block cipher is a function $E : K \times P \to C$, where K is the keyspace, P is the plaintext-space and C is the ciphertext-space. In most cases in practice, these are bit strings, and one can rewrite this as $E : \{0,1\}^{\ell_K} \times \{0,1\}^{\ell_P} \to \{0,1\}^{\ell_C}$. The stereotype is that $\ell_P = \ell_K = \ell_C$, but this is *almost never* the case in practice, as shown by the following examples.

- IDEA $\ell_P = \ell_C = 64$, $\ell_K = 128$.
- DES $\ell_P = \ell_C = 64$, $\ell_K = 56$.
- Two-key triple DES $\ell_P = \ell_C = 64$, $\ell_K = 112$.
- Blowfish $\ell_P = \ell_C = 64$, $\ell_K \in \{0, \ldots, 448\}$.

- RC5 $\ell_P = \ell_C \in \{32, 64, 128\}$, $\ell_K \in \{0, \ldots, 2040\}$.
- AES, Mars, RC6, Serpent, Twofish $\ell_P = \ell_C = 128$, $\ell_K \in \{128, 192, 256\}$.
- KeeLoq $\ell_P = \ell_C = 32$, $\ell_K = 64$.

The ciphers with $\ell_P < \ell_K$ have several interesting properties not shared by those with $\ell_P \geq \ell_K$. This question has not received much attention in the cryptographic community so far, and the particularities of the case $\ell_P < \ell_K$ become really very important when ℓ_P is small, for example in KeeLoq. We believe that it is important to understand this somewhat curious situation better.

Let the *code-book* of a cipher E under a key k be the set of all 2^{ℓ_P} pairs (P, C) such that $E(k, P) = C$. If $2^{\ell_P} \ll 2^{\ell_K}$, it takes less time to compute the entire code-book than to do the exhaustive key search. Therefore, a natural question would be why, precisely, would one want to recover the key if it is possible to have the entire code-book? From the point of view of theory and security model, this question was recently studied by P o r n i n and G r a n b o u l a n in Section 5 of [27]. In this paper we look at it in a similar way but from the point of view of practical real-life applications and their security. We will give several examples of such applications.

2.1. Brute-force generic attacks on ciphers with small blocks

Key recovery attack on block ciphers with very small blocks are more or less interesting depending on the circumstances. We see three distinct scenarios.

Scenario 1—Theoretical. From a theoretical perspective, we can assume that the adversary is very powerful and has chosen-plaintext oracle access to the cipher and a very large (usually unrealistic) quantity of memory. Then if the block size is small, one can judge that the security of block cipher is $2^{\min(\ell_K, \ell_P)}$, and once the adversary recovers and is able to store the entire code-book, one can consider that the adversary has no interest in actually recovering the original key. From a scientific point of view, of course, the key-recovery process remains interesting in its own right. Moreover, in practice, even in this extreme scenario, the actual key can be very valuable because it may lead to a master key—having one is a very common practice in the industry—which key would compromise the security on a much wider scale.

Scenario 2—Practical. This is a known-plaintext attack, and even if the block size is very small the known-plaintext attack is *not* equivalent to a chosen-plaintext attack, not only because storage may be limited, but more importantly because not all plaintexts actually arise in real life (there is some padding and a specific probability distribution of possible data). Here the adversary can recover a number of plaintext-ciphertext pairs that can be, for example up to 50 % of all possible pairs, but he cannot hope to recover all pairs. Importantly, the value of pairs he does not have may be very large, while the value of pairs he

already has, is (by definition) very small. Here the key recovery allows the adversary to have all possible pairs, some of which potentially very valuable, or to recover a master key, even more valuable.

To summarize, in the first (theoretical) scenario the security of the block cipher is $2^{\min(\ell_K,\ell_P)}$, while in the second more practical scenario, the security is 2^{ℓ_K} whatever is the block size. In the next section we present several practical application scenarios which illustrate the importance of key recovery for ciphers with small blocks and a larger key size. This is meant to motivate further detailed study of key recovery in ciphers such as KeeLoq.

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

2.2. Key recovery vs. applications of ciphers with small blocks

Scenario 1—Military code book. Ciphers with small blocks can be used to generate code-books for old-style but very practical military or diplomatic communication methods that do not require any machine to encrypt messages. We can note that the question how these code-books are generated is generally ignored, yet humans cannot be trusted to produce randomness "off-the-cuff", an the traditional military solution of using octahedral dice to produce bits, three at a time is too slow to be practical for code-books beyond a certain size. Therefore, using a block cipher with small blocks seems to be a default and very sensible solution to this problem. In particular, a good code-book will be also a polymorphic cipher, one with several ciphertexts per each plaintext. Then, it can be used in such a way that the same code-words are rarely or never re-used. Then even if we know 99.9 % of the code-book, and only two values are not known, the practical "value" of the missing information can be very, very high.

A practical scenario is as follows: imagine that the CIA has reconstructed 60 % of the code-book of the most dangerous terrorists on the planet. The code-book is short and used to encrypt short messages over the phone and very few messages are ever sent. In theory, for a short random message, they have a 40 % chance to understand nothing. In addition it could be a polymorphic code-book so that every message has several versions. With such a system, the terrorist can communicate with his sergeants with the security of a one-time pad, if he thinks about never re-using the same code-word twice to send the same message, knowing that after-the-fact a detailed enquiry about the terrorist attack will **always** allow one to determine both the plaintext and the ciphertext,

each used only once. We can imagine that the code-book was generated using a cipher like KeeLoq and the source code is known.[1] Then no new message can ever be decrypted, and key recovery is the only option. This holds even if the block size is very small, for example one can use 8-bit blocks to command a series of attacks (e.g., in ASCII). We have here an example of a cipher, with its prescribed usage mode, that is in fact a perfectly secure system (in the sense of information-security) except if the recovery of the master key can be done.

A similar method can be used to design computer viruses that spread unnoticed and later use a perfectly secure communication method to make a coordinated world-wide large scale cyber-attack that can hardly be detected by looking at communications on the network (messages are random strings). Here finding the initial source code and then recovering the master key would be, perhaps, the only way to prove the origin of the attack.

Scenario 2—LORI-KPA/LORI-CPA. Consider the notion of Left-or-Right-Indistinguishability in either Known-Plaintext Attack, or Chosen-Plaintext Attack [4]. There are two plaintexts, either known to the attacker, or chosen by the attacker, which we will denote as "active". The attacker can then make "polynomially many" queries, and submit plaintexts of his choice for encryption. We can translate this definition to a "concrete security" treatment when the security parameter (key length) is fixed, and allow the attacker to request, in fact, the encryption of any plaintext, except the two which are active. Therefore one can consider that the code-book is actually known to the adversary, for all but two values. Such a scenario is also explicitly considered in Section 5 of [27].

We note that if a message has been transmitted and it is not found in the code-book, then it is clearly one of the remaining two. This message can be of vital importance, yet it might not be possible to determine which of the remaining two it is. Key recovery would accomplish this.

If the reader doubts the practicality of this scenario, where most of a code-book is known and only a few values remain, consider the following. According to David K a h n [29], in 1942, the United States decrypted many messages encrypted with the famous "Purple" cipher, forecasting an attack at "AF." There were only a few possible targets, and so a very short list of candidates was made and Midway Island seemed the most reasonable choice. The Americans needed however a confirmation to be 100 % confident, because they planned to strike with every available aircraft carrier, and a mistake would be a tremendous waste of scarce resources. The US Navy decided to send a message about water supply on Midway, using their own code that they knew to be broken be the Japanese. Very soon another message about "AF" was sent over Japanese channels, describing the problem with the water. Consequently, overwhelming force was sent

[1] Perhaps, it was generated using a commercial implementation of a cipher, or somebody found the source code on an old hard drive, and either way of everything is known except the key.

Midway and Japan's offensive power at sea was castrated, which had a pivotal impact on winning the World War II.

Scenario 3—Manufacturer sub-keys. One usage of KeeLoq in automobiles could be to take a 32-bit string called a "manufacturer key", and a 32-bit string called a "per-automobile" key, and concatenate them to form a key for each automobile. This means that the automobile manufacturer can produce a machine to recover the key for this particular vehicle in 2^{32} operations, but all other attackers cannot, if the key remains unknown for every automobile. If the codebook is known for one automobile, then that specific automobile can be stolen. But if a key recovery is then performed, both keys are recovered and thus every automobile of that manufacturer could then be much more easily stolen, using 2^{32} rather than 2^{64} test encryptions.

Incidentally, a more secure way of accomplishing the above is to generate a "manufacturer key" k_M randomly, and let the per automobile key be $k_s = E(k_M, s)$, where s is the serial number of the car. There would be no obvious attack.

Scenario 4—Short but private data. Suppose short strings must be encrypted with high security. In the USA, social security numbers (SSN's) are 9 digits, and this can be encoded in Binary Coded Decimal (BCD) with 36 bits. Of course, one can use AES (E with $\ell_K = \ell_P = \ell_C = 128$) and encrypt the 36 bits padded with 92 bits of zeroes or a fixed padding, or even with a padding that is a function of the SSN. But then this defines an induced E' with $\ell_K = 128$, $\ell_P = 36$ and $\ell_C = 128$. This is related to the idea of "nuggets" as presented in [1].

Scenario 5—Fast shuffing and anonymity. Given a random permutation σ on the set of n elements, one can trivially shuffle a list of n objects. This is needed in many areas, most notably in scrambling data to preserve the privacy of patients in medical research. Note that sending each item i to the spot $\sigma(i)$ is sufficient for a random shuffle and takes $\Theta(n)$ time total; for a large n this is much better than assigning a random number to each item and then sorting, which would take $\Theta(n \log_2 n)$ time. One can do this by using $\sigma(i) = E(i, k) \bmod n$. But, especially if n is a power of 2, this induces a block cipher with high ℓ_K (to protect anonymity) but with small block size $\ell_p = \ell_c = \log_2 n$.

Scenario 6—Assigning account numbers to people. A bank or a stockbroker can assign random-looking account numbers to their unique identifiers such as their name plus date of birth or their social security number. In these applications every single new plaintext-ciphertext pair is valuable to the attacker, and one single pair can be worth much more than any other pair (a particularly wealthy customer can be targeted).

Scenario 7—Scratch cards and software serial numbers. Block ciphers with small blocks are used by the industry to generate so called scratch cards, that are used for example to obtain calling credit on a mobile phone. The permutation is used to associate random-looking and unique (hard to forge) numbers on scratch cards, to unique account identifiers that are typically the numbers $0, 1, 2, 3, 4, \ldots$ The same method is sometimes used to obtain unique serial numbers for software. This avoids keeping a database of all existing serial numbers which can be replaced by a short piece of code (not very secure) or a secure cryptographic hardware or token with embedded key (much more secure).

2.3. KeeLoq code-book—practical considerations

We have not touched upon the issue of how the code-book can be obtained in the case of Keeloq and automobile applications. Either it can 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, in practice the devices are simply too slow to obtain this. It is also noteworthy that since each plaintext is 2^5 bits long, and there are 2^{32} of them, the entire code-book is 2^{37} bits or 16 Gigabytes. This amount of RAM is already available on high-end PC's at the time of writing.

Oddly, 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 [12]. Such an attack requires only 2 known plaintexts (one known plaintext alone does not allow one to uniquely determine the key, which is another consequence of the unusually small block size). We note that while 2^{32} encryptions is difficult to obtain with the original chips that are quite inexpensive and slow, with FPGA's as much as 2^{64} encryptions is feasible. This is because the FPGA's are faster and do a great deal of parallel processing.

3. The cipher description

The specification of KeeLoq can be found in [12, 13, 32, 8, 2, 15]. Initially, the specification found in [12, 13] was mistaken, as opposed to [32, 8], but now all available sources agree on the updated specification.

The KeeLoq cipher is a strongly unbalanced Feistel construction in which the round function is "compressing" and has only one bit of output. Consequently, in one round, only one bit in the "state" of the cipher is 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 additionally XORed with (only) one key bit in each round. The cipher has a total of 528 rounds. The encryption procedure is periodic with a period of 64 and it has been "cut" at 528 rounds,

with $528 = 512 + 16 = 64 \times 8 + 16$. The fact that 528 is not a multiple of 64 prevents a direct application of "slide attacks" [28, 7, 6]. However more advanced slide attacks remain possible as we will see in this paper and in other known attacks on KeeLoq [8, 15, 9, 10, 5].

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 64-bit key register is the same. If we denote the first 64 rounds by $f_k(x)$, then KeeLoq is

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

with $g(x)$ being a 16-round final step, and $E_k(x)$ being all 528 rounds. The last "extra" 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.

The main (and only) non-linear component is a Boolean function with the truth table given in Table 11 of [32]. This truth table is encoded by "3A5C742E" in [12] which should be read as follows: $operatornameNLF(a, b, c, d, e)$ is equal to the i^{th} bit of that hexadecimal number, where $i = 16a + 8b + 4c + 2d + e$. Thus $(a, b, c, d, e) = (0, 0, 0, 0, 0)$ gives $i = 0$ and the function outputs the least significant (rightmost) bit of of "3A5C742E". With $(1, 1, 1, 1, 1)$ we get the most significant leftmost) bit number of "3A5C742E", i.e., $i = 31$. The corresponding algebraic normal form (ANF) of this function is given by [8]:

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

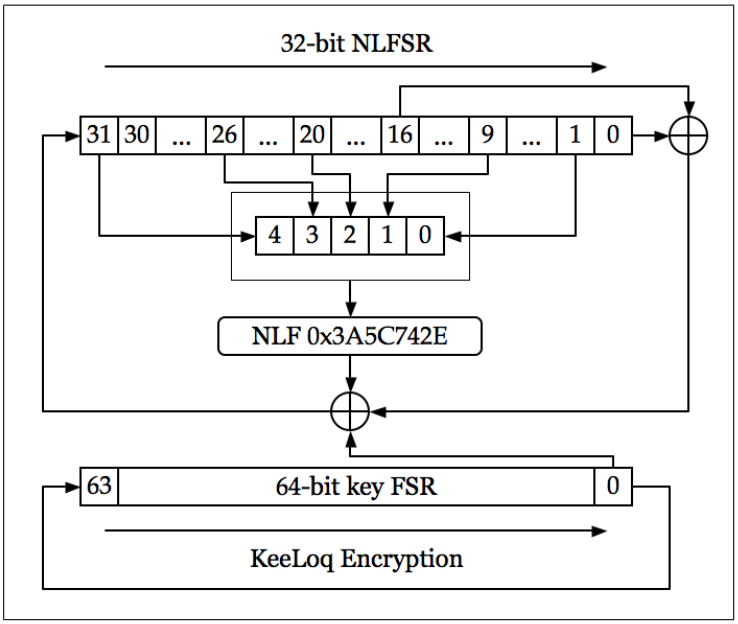
The main shift register has 32 bits, (unlike the key shift register with 64 bits), and let L_i denote the leftmost or least-significant bit at the end of round i, with L_0 being its initial value. At the end of round 528, the least significant bit is thus L_{528}, and then let $L_{529}, L_{530}, \ldots, L_{559}$ denote the 31 remaining bits of the shift register, with L_{559} being the most significant. Let $k_{63}, k_{62}, \ldots, k_1, k_0$ be the key and the initial content of the key register. The complete KeeLoq encryption process is fully described on Fig. 1 (this corresponds to the decryption process that is described by Fig. 12 in [32]).

4. Useful combinatorial facts

4.1. 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 $\sim 1/(i!e)$, in the limit when $n \to \infty$. (Furthermore, as a random variable, the number of pre-images has 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



(1) Initialize with the plaintext: $L_{31}, \ldots, L_0 = P_{31}, \ldots, P_0$.

(2) For $i = 0, \ldots, 528 - 1$ do
$L_{i+32} = k_{i \bmod 64} \oplus L_i \oplus L_{i+16} \oplus \mathrm{NLF}(L_{i+31}, L_{i+26}, L_{i+20}, L_{i+9}, L_{i+1})$.

(3) The ciphertext is $C_{31}, \ldots, C_0 = L_{559}, \ldots, L_{528}$.

FIGURE 1. KeeLoq encryption.

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

PROPOSITION 4.2. *The first 64 rounds of KeeLoq have 2 or more fixed points with probability of* $1 - 2/e \approx 0.26$.

Experiments to verify this were performed, some of which are described in Appendix.

4.2. On the expected number of cycles in a random permutation

It is well known (for example, see [36]) that:

Proposition 4.3. *The expected number of cycles in a decomposition of permutation on n bits into disjoint cycles is equal to H_{2^n}, where $H_k = \sum_{i=0}^{k} 1/i$ is the kth Harmonic Number. We have $H_k \approx \ln k + \gamma$, where $\gamma \approx 0.577216\ldots$ is the Euler-Mascheroni constant.*

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

5. Algebraic and slide attack on KeeLoq

Several direct algebraic attacks on KeeLoq have been proposed and studied in [15], in this paper we will recall and re-use essentially one of them.

5.1. Pure algebraic attacks

The goal of an algebraic attack, 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 [18]. Unlike for stream ciphers [19], very few such attacks are actually very efficient on real-world block ciphers. For example, DES can be broken for up to 6 rounds out of 16, see [24]. For KeeLoq, on the other hand, many more rounds can be directly attacked, due to its simplicity.

One method is to write KeeLoq as a system of equations, and is described in [15]. The same paper studies to what extent these equations can be solved in practice by two families of methods. With Gröbner bases (and similar but simpler) techniques, it is possible to break up to 128 rounds of KeeLoq. Another family of techniques are attacks with conversion of the MQ problem to a well-known logical CNF-SAT problem using the methods of [25], and solving the resulting satisfiability problem by one of the existing SAT solver packages, for example, MiniSat 2.0. [34]. In this way, and with modern SAT solvers, it is possible to break as many as 160 rounds of KeeLoq, we refer to [15] for more details and in [2] there is a chapter that explains how SAT solvers actually work.

It can be seen that, with the conversion of [25] and modern SAT solvers, when the number of rounds is not too large, the key can be obtained almost instantly. In this paper we will only use the following fact:

Proposition 5.1 (Example 6 of [15])**.** *For 64 rounds of KeeLoq and 2 known plaintexts, the full 64-bit key is recovered by conversion to SAT and MiniSat 2.0. in about 0.3 s.*

Several complete working examples of equations can be downloaded from [14] and with MiniSat 2.0. being freely available [34] it is easy to check these results.

5.2. Comparing algebraic attacks on KeeLoq to brute force

Fact 5.1. An optimised assembly language implementation of r rounds of KeeLoq is expected to take about $4r$ CPU clocks. For justification, see footnote 4 in [8].

Thus, the complexity of an attack on r rounds of KeeLoq with k bits of the key should be compared to $4r \times 2^{k-1}$, which is the expected complexity of the brute force key search. For example, for full KeeLoq, the reference complexity for the exhaustive key search is about 2^{75} CPU clocks.

Assuming that the CPU runs at 2.5 GHz, one can execute about 2^{43} CPU clocks per hour (if only one CPU core is used). Thus a brute force attack on KeeLoq requires 2^{32} hours per core, or 0.5 millions of CPU-years.

5.3. Our combined slide and algebraic attack A

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 4.2). In the remaining cases this attack fails (but one can apply the Attack B). Under this assumption, we expect that there will be about 6 fixed points for $f_k^{(8)}(\cdot)$. We did computer simulations to confirm this figure, see Appendix 8

Recall $f_k^{(8)}$ consists of the first 512 rounds of KeeLoq. The attacker will try to guess which, out of the 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 (6 plaintexts) until the right pair is used, which requires about 15 tries and about 15/2 on average.

Stage 1—recover 16 key bits with a distinguisher

Let B be a permutation on 32 bit words. From Proposition 4.3, assuming that it behaves as a random permutation, we expect that B has about 23 cycles. Half of them should have even sizes, and half odd—or 11.5 each. 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 divided by 2. In summary we would expect to see 17.75 odd cycles and 5.75 even cycles.

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

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

TABLE 1. Pseudocode for stage 1 used in attacks A and B.

(1) Assume that the entire code-book is stored in 16 Gigabytes of RAM.
(2) Guess 16 bits of the key.
  (a) Apply $g_k^{-1}(\cdot)$ to the entire code-book, get a complete table for $f^{(8)}(\cdot)$. This may require another 16 Gigabytes of RAM (or instead, one can notice that here the code-book is read sequentially so a very fast hard drive can also be used for the code-book).
  (b) Check the cycle lengths: start from a random not-yet-used point, cycle through the whole cycle and mark each point as used. This requires extra 2 Gigabytes of RAM. There is an early abort so that there is no need to compute the exact number of cycles, namely:
    (i) If there are 6 or more even length cycles, go back to Step 2. (We note that before this early abort happens, about 12 (larger) cycles are found with expected sizes decreasing roughly by a factor of (2/3) each time. So a time of about
$$\sum_{i=0}^{11}\left((1/3)\cdot\cdot(2/3)^i\right)^{-1}\approx 770$$
    is spent on random sampling through the whole space looking for a random not-yet-used point, which is truly negligible.)
    (ii) With probability 2^{-16} the guess of 16 bits of the key is good, and only in this exceptional case we do not have an early abort. If there are only 5 or fewer even length cycles, we will consider that the guessed 16 bits of the key are correct.

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 even-length cycles, we say it is the wrong key-prefix. Otherwise, we say that $k_0, \ldots, k_{15}$ is 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 requires having only 5 or fewer even-sized cycles for a random permutation is extremely low and will be neglected. But as an approximation, suppose that there were exactly 23 cycles. Then the probability of having less than six even length cycles is identical to having exactly zero, or exactly one, or exactly two, up to exactly five, and thus can be computed from the binomial distribution. These cases are mutually exclusive and collectively exhaustive. Since a cycle is expected to be even or odd with probability one-half, this comes to

$$\sum_{i=0}^{i=5} \binom{23}{i} \left(\frac{1}{2}\right)^{i} \left(\frac{1}{2}\right)^{23-i} = 2^{-23} \cdot 44552 \approx 2^{-7.56} \approx 0.0053\ldots$$

The success rate of this part of the attack is close to 1 and we expect that exactly one 16-bit 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. Since we assumed that plaintext--ciphertext pairs are stored in a table, and the access time is 16 CPU clocks, this will take time roughly equal to 2^{36} 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 total memory used is 16+2 Gigabytes with an extra 2 Gigabytes needed to remember which points were already used. The fact that we can reject a key when 6 even-size cycles are already found, avoids systematically computing all cycles, only the biggest ones, and allows for an early abort. A pseudo-code with additional explanations is given in Table 1.

The complexity of an optimised version of this attack should be 2^{36} CPU clocks per guessed 16-bit key, or 2^{16+36} total in the worst case, and 2^{15+36} total on average. To summarise, given about 60 % of the entire code-book of 2^{32} known plaintext (this is explained in Appendix), at Stage 1 of the attack gives us 16 bits of the key $k_0, \ldots, k_{15}$ with the workfactor of about 2^{51} CPU clocks on average which is about 2^{40} KeeLoq encryptions.

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 Proposition 5.1 except that we now actually know 16 bits of the key, and know the resulting approximately 4 fixed points of $f_k^{(8)}$. Here again we

TABLE 2. Pseudocode for the whole attack A.

(1) Assume that the entire code-book is stored in 16 Gigabytes of RAM.
(2) With Stage 1 recover 16 bits of the key.
(3) Determine the fixed points of $f_k^{(8)}(\cdot)$ using the code-book, and call this set F.
(4) For each possible pair $(p_i, p_j) \in F$, with $i < j$:
  (a) Assume that $f(p_i) = p_i$ and also $f(p_j) = p_j$.
  (b) Write equations accordingly.
  (c) Solve them (this takes about 0.3 seconds, cf. Proposition 5.1).
  (d) If a key results, see if it is correct and if so, terminate.
  (e) Otherwise repeat for the next pair.

will assume that there are two fixed points for f_k. This is true for 26 % of the keys. We need to guess which (out of these approximately 4 points) are the fixed points of f_k, see pseudocode in Table 2, 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 $\binom{6}{2}^{-1} = 1/15$, it was explained earlier. Thus the total complexity of this stage is in expectation about $15/2 \cdot 2^{28} \approx 2^{31}$ 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 points). The first stage that requires about 2^{51} CPU clocks dominates the attack.

Summary of attack A

Given about 60 % of the entire code-book (see Appendix) this attack succeeds with probability 0.26, i.e., for 26 % of keys (cf. Proposition 4.2). The running time is about 2^{51} CPU clocks which is about 2^{40} KeeLoq encryptions. This attack is also described in [2].

6. Attack B: A correlation attack on KeeLoq

In this attack we will replace Stage 2 by another attack that works for all possible keys, not only 26 % of keys. However it requires the entire code-book (as opposed to 60 % of it). The attack uses the following basic facts.

The used NLF can be approximated by a linear combination of two input variables, since it is 1-resilient but not 2-resilient.

Proposition 6.1. *For the nonlinear KeeLoq function* NLF *and uniformly distributed* $x_4, x_3, x_2 \in GF(2)$*:*

$$Pr\big\{\mathrm{NLF}(x_4, x_3, x_2, x_1, x_0) = 0 \mid x_0 \oplus x_1 = 0\big\} =$$
$$Pr\big\{\mathrm{NLF}(x_4, x_3, x_2, x_1, x_0) = 1 \mid x_0 \oplus x_1 = 1\big\} = \frac{1}{2} + \frac{1}{8}.$$

If four of five input bits are known, NLF is an affine function of one variable:

Proposition 6.2. *For the nonlinear KeeLoq function* NLF *with* x_0*,* x_1*,* x_2*,* x_3 *known and* x_4 *unknown:*

$$\mathrm{NLF}(x_4, x_3, x_2, x_1, x_0) = c_1 x_4 \oplus c_0,$$

where c_1 *and* c_0 *are known constants dependent on* x_0*,* x_1*,* x_2*,* x_3*.*

Proposition 6.3. *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, by looking forwards *and* backwards, we see that all state bits inside the cipher are directly known, from either the plaintext or the ciphertext. Thus 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.

Proposition 6.4. *Given* $k_0, \ldots, k_{31}$ *and* (α, β) *with* $\beta = f_k(\alpha)$*,* $k_{32}, \ldots, k_{63}$ *can be derived with a complexity of computing* f_k *(64 rounds) with known* k*.*

Justification: Follows directly from the previous proposition, with 32 fist key bits known, we can remove the first 32 rounds.

Description of the attack B

Stage 1—recover 16 key bits

The same as in Attack A and Table 2.

Stage 2—recover extra 16 key bits with linear cryptanalysis

Now the first 16 key bits $k_0, \ldots, k_{15}$ are known. As in Attack A, by applying $g_k^{-1}(\,)$ to the entire code-book we get a complete table for $f_k^{(8)}(\,)$ that is stored in 16 Gigabytes of RAM. This is a completely periodic cipher architecture $E'_k(x) = f_k^{(8)}(x)$ which is vulnerable to slide attacks. Now we proceed as follows:

(1) First we choose and fix a random 32-bit input α_1 (can be the same in the whole attack) and guess $\beta_1 = f_k(\alpha_1)$ (average 2^{31} and max. 2^{32} possibilities to be checked). Now, as in classical slide attacks [7], we observe that one such pair allows one to compute many other such pairs as follows:

$(\alpha_{i+1}, \beta_{i+1}) = \left(f_k^{(8)}(\alpha_i), f_k^{(8)}(\beta_i)\right)$. For each guess of β_1 a "slide group" is defined as the set of couples $G_{\alpha_1,\beta_1} = \left\{(\alpha_i, \beta_i) : \ \beta_i = f_k(\alpha_i)\right\}_{i=1}^{N}$. We note that from our formula above, a "slide group" of size N is generated with $2(N-1)$ table lookups.

(2) Using G_{α_1,β_1} the next 16 key bits $k_{16}, \ldots, k_{31}$ can be obtained by applying the correlation step outlined below. For the sake of simplicity we only show how to obtain k_{16} and k_{32} here. All the further operations are very similar. The least significant output bit L_{64} at the output of f_k is equal to:

$$\begin{aligned} L_{64} &= \mathrm{NLF}(L_{63}, L_{58}, L_{52}, L_{41}, L_{33}) \oplus L_{32} \oplus L_{48} \oplus k_{32} \\ &= \mathrm{NLF}(L_{63}, L_{58}, L_{52}, L_{41}, L_{33}) \oplus \mathrm{NLF}(L_{47}, L_{42}, L_{36}, L_{25}, L_{17}) \\ &\quad \oplus L_{16} \oplus (k_{32} \oplus k_{16}), \end{aligned}$$

where L_{32} was eliminated. Here all the L_i, $i < 48$, are known (because 16 key bits are known) and the only nonlinear expression with unknown variables $\mathrm{NLF}(L_{63}, L_{58}, L_{52}, L_{41}, L_{33})$ can be efficiently approximated by the sum $L_{41} \oplus L_{33}$ of two known values using Proposition 6.1. By looking at this equation just for few pairs from the "slide group", we get $k_{16} \oplus k_{32}$ by majority voting.

The next output bit L_{65} of f_k can be written as follows:

$$\begin{aligned} L_{65} &= \mathrm{NLF}(L_{64}, L_{59}, L_{53}, L_{42}, L_{34}) \oplus L_{33} \oplus L_{49} \oplus k_{33} \\ &= \mathrm{NLF}(L_{64}, L_{59}, L_{53}, L_{42}, L_{34}) \oplus L_{33} \oplus c_0 \oplus L_{17} \\ &\quad \oplus c_1 k_{16} \oplus k_{17} \oplus k_{33}, \end{aligned}$$

where L_{49} is expressed as an affine function on k_{16} and k_{17} using Proposition 6.2 with known constants c_0 and c_1 dependent on the input. Statistically, one half of the elements in G_{α_1,β_1} give $c_1 = 0$ which leads to the recovery of $k_{17} \oplus k_{33}$ by majority voting as outlined above. The other half of inputs gives $c_1 = 1$ and recovers $k_{16} \oplus k_{17} \oplus k_{33}$. Now k_{16} and k_{32} can be directly computed.

By proceeding iteratively for the next 14 output bits L_i, $65 < i \leq 79$, of f_k and using Propositions 6.1 and 6.2, one obtains all $k_{16}, \ldots, k_{31}$. Our experiments show that a "slide group" of size $N = 2^8$ is enough for the whole correlation step to succeed with a high probability. Thus, the complexity of Stage 2 of the attack (the first 16 key bits being known in advance) is about $2^{31}(16(2N-2)+16N)/528 \approx 2^{35.5}$ KeeLoq encryptions, if a single table lookup requires 16 CPU cycles.

Stage 3—recover remaining 32 key bits

The remaining 32 key bits are recovered using Proposition 6.4. Each 64-bit key candidate corresponding to the current guess of β_1 is tested using known plaintext-ciphertext pairs for KeeLoq.

Summary of attack B

Given 2^{32} known plaintexts, this attack succeeds for all keys with probability very close to 1. As in Attack A, the first stage still dominates the attack and the running time is about 2^{51} CPU clocks which is only about 2^{40} KeeLoq encryptions.

Remark 1. Though the Stage 1 of this attack, works given about 60 % of the code-book (as in Attack A), the whole attack really needs more or less the whole code-book. This is because we have to guess 32 bits of β_1 and generate a slide group of size 2^8 for each guess.

7. Strong keys in KeeLoq

Following [15], the manufacturer or the programmer of a device that contains KeeLoq can check each potential key for fixed points for f_k (2^{32} plaintext have to be checked). If it has any, that key can be declared "weak" and never used. A proportion of 63 % of all the keys will be weak, and following [15], removing these weak keys will change the effective key space from 64 bits to 62.56 bits. This is a small loss, in many scenarios perfectly acceptable. Unfortunately, this fix removes only our Attack A. The Attack B still works.

8. Conclusion

In this paper we presented two attacks on KeeLoq, a block cipher that is in widespread use throughout the automobile industry and that is used by millions of people every day. One particularity of this cipher is that the block size is unusually small, only 32 bits. It is therefore feasible to compute and store the entire code-book, which leads to many interesting attacks and considerations that only occur for ciphers with small blocks.

In particular, the small block size combined with the periodic structure of KeeLoq, allows one to distinguish 528 rounds of KeeLoq from a random permutation, and this independently of the strength of the cipher and its key length. This is quite interesting: iterating even an extremely strong cipher on small blocks gives a cipher that will be distinguishable from a random permutation. Starting from this fact, two cycling attacks on KeeLoq are proposed in this paper.

Our Attack A uses an algebraic cryptanalysis step with SAT solvers. It requires the knowledge of about 60 % of the entire code-book of 2^{32} known plaintexts, works for 26 % of keys and is equivalent to about 2^{40} KeeLoq encryptions.

Our attack B is a correlation attack that has the same complexity, it is better in that it works for all keys, yet it requires the whole code-book. It is interesting to note that attacks that use sliding properties can be quite powerful because typically their complexity simply does not depend on the number of rounds of the cipher. Very similar attacks can be designed for most iterated ciphers.

Nonetheless, due to the short key size in KeeLoq, and given that it is rather difficult to obtain a large quantity of plaintexts, in practice the best attack on KeeLoq remains the brute force that requires only two known plaintexts.

REFERENCES

[1] BARD, G.: *A challenging but feasible blockwise-adaptive chosen-plaintext attack on SSL,* in: Proceedings of the International Conference on Security and Cryptography——SECRYPT '06 (M. Malek, E. Fernández-Medina, J. Hernando.), INSTICC Press. Setúbal, Portugal, 2006, pp. 99–109, `http://eprint.iacr.org/2006/136.`

[2] BARD, G.: *Algorithms for Solving of Linear and Polynomial Systems of Equations over Finite Fields, with Applications to Cryptanalysis,* PhD Thesis, Department of Mathematics, University of Maryland at College Park, 2007.

[3] BARDET, M.—FAUGÈRE, J.-CH.—SALVY, B.: *On the complexity of Gröbner basis computation of semi-regular overdetermined algebraic equations*, in: Proceedings of International Conference on Polynomial System Solving—ICPSS '04 (J. C. Faugère, F. Rouillier, eds.), Paris, France, November 24–26, 2004, pp. 71–75.

[4] BELLARE, M.—DESAI, A.—JOKIPII, E.—ROGAWAY, P.: *A concrete security treatment of symmetric encryption: Analysis of the DES modes of operation*, in: Proceedings of the 38th Annual Symposium on Foundations of Computer Science—FOCS '97, IEEE Computer Society, Washington DC, USA, 1997, pp. 394–403.

[5] INDESTEEGE, S.—KELLER, N.—DUNKELMAN, O.—BIHAM, E.—PRENEEL, B.: *A practical attack on KeeLoq*, in: Advances in Cryptology—EUROCRYPT '08 (N. P. Smart, ed.), Lecture Notes in Comput. Sci., Vol. 4965, Springer-Verlag, Berlin, 2008, pp. 1–18.

[6] BIRYUKOV, A.—WAGNER, D.: *Advanced slide attacks*, in: Advances in Cryptology——EUROCRYPT 2000 (M. Ballare, ed.), Lecture Notes in Comput. Sci., Vol. 1807, Springer-Verlag, Berlin, 2000, pp. 598–606.

[7] BIRYUKOV, A.—WAGNER, D.: *Slide attacks*, in: Fast Software Encryption—FSE '99 (L. R. Knudsen, ed.), Lecture Notes in Comput. Sci., Vol. 1636, Springer-Verlag, 1999, Berlin, pp. 245–259.

[8] BOGDANOV, A.: *Cryptanalysis of the KeeLoq block cipher*. Cryptology ePrint Archive, Report 2007/055, February 16, 2007, `http://eprint.iacr.org/2007/055.`

[9] BOGDANOV, A.: *Attacks on the KeeLoq block cipher and authentication systems,* in: The 3rd Conference on RFID Security—RFIDSec '07, (V. Rijmen ed.), Malaga, Spain, July 11–13, 2007.

[10] BOGDANOV, A.: *Linear slide attacks on the KeeLoq block cipher,* in: INSCRYPT '07 (D. Pei, M. Yung, D. Lin, Ch. Wu. eds.), Lecture Notes in Comput. Sci., Vol. 4990, Springer-Verlag, 2008, pp. 66–80.

[11] BABBAGE, S.—CID, C.—PRAMSTALLER, N.—RADDUM, H.: *An analysis of the Hermes8 stream cipher*, in: ACISP '07, The 12th Australasian Conference—ACISP '07 (J. Pieprzyk H. Ghodosi, E. Dawson., eds.), Lecture Notes in Comput. Sci., Vol. 4586, Springer-Verlag, Berlin, 2007, pp. 1–10.

[12] *Keeloq*, Wikipedia, January 25, 2007, `http://en.wikipedia.org/wiki/KeeLoq.`

[13] *Keeloq C source code by Ruptor*, `http://cryptolib.com/ciphers.`

[14] COURTOIS, N. T.: *Examples of equations generated for experiments with algebraic cryptanalysis of KeeLoq,* `http://www.cryptosystem.net/aes/toyciphers.html.`

[15] COURTOIS, N. T.—BARD, G. V.—WAGNER, D.: *Algebraic and slide attacks on KeeLoq*, in: Proceeding of Fast Software Encryption—FSE '08 (K. Nyberg, ed.), Lausanne, Switzerland, February 10–13, 2008, Lecture Notes in Comput. Sci., Vol. 5086, Springer-Verlag, Berlin, 2008, pp. 97–115.

[16] COURTOIS, N. T.—PATARIN, J.: *About the XL algorithm over $GF(2)$*, in: in Proceedings of The Cryptographers' Track at the RSA Conference 2003—CT-RSA '03 (M. Joye, ed.), Lecture Notes in Comput. Sci., Vol. 2612, Springer-Verlag, Berlin, pp. 141–157.

[17] COURTOIS, N. T.—KLIMOV, A.—PATARIN, J.—SHAMIR, A.: *Efficient algorithms for solving overdefined systems of multivariate polynomial equations*, in: Advances in Cryptology—EUROCRYPT 2000 (M. Ballare, ed.), Lecture Notes in Comput. Sci., Vol. 1807, Springer-Verlag, Berlin, 2000, pp. 392–407.

[18] COURTOIS, N. T.—PIEPRZYK, J.: *Cryptanalysis of block ciphers with overdefined systems of equations*, in: Advances in Cryptology—ASIACRYPT '02, (Y. Zheng, ed.), Lecture Notes in Comput. Sci., Vol. 2501, Springer-Verlag, Berlin, 2002, pp. 267–287.

[19] COURTOIS, N. T.—MEIER, W.: *Algebraic attacks on stream ciphers with linear feedback*, in: Advances in Cryptology—EUROCRYPT '03, (E. Biham, ed.), Lecture Notes in Comput. Sci., Vol. 2656, Springer-Verlag, Berlin, 2003, pp. 345–359, `http://www.nicolascourtois.me.uk.`

[20] COURTOIS, N. T.: *General principles of algebraic attacks and new design criteria for components of symmetric ciphers*, in: Proceeding of the Advanced Encryption Standard Conference—AES '04, (H. Dobbertin, V. Rijmen, A. Sowa, eds.), Lecture Notes in Comput. Sci., Vol. 3373, Springer-Verlag, Berlin, 2005, pp. 67–83.

[21] COURTOIS, N. T.: *The inverse S-box, non-linear polynomial relations and cryptanalysis of block ciphers*, in: Proceedings of The Advanced Encryption Standard Conference, Advanced Encryption Standard—AES '04, Lecture Notes in Comput. Sci., Vol. 3373, (H. Dobbertin, V. Rijmen, A. Sowa, eds.), Springer-Verlag, Berlin, 2005, pp. 170–188.

[22] COURTOIS, N. T.: *How fast can be algebraic attacks on block ciphers?*, in: Proceedings of Symmetric Cryptography, January 7–12, 2007, (E. Biham, H. Handschuh, S. Lucks, V. Rijmen, eds.), Dagstuhl Seminar Proceedings, Vol. 7021, Internationales Begegnungs—und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, 2007, pp. 7–12.

[23] COURTOIS, N. T.: *CTC2 and fast algebraic attacks on block ciphers revisited*, Cryptology ePrint Archive, Report 2007/152, 2007, http://eprint.iacr.org/2007/152.

[24] COURTOIS, N. T.—BARD, G. V.: *Algebraic cryptanalysis of the data encryption standard*, in: 11th IMA International Conference, Cryptography and Coding (S. Galbraith, ed.), Lecture Notes in Comput. Sci., Vol. 4887, Springer-Verlag, Berlin, 2007, 152–169, http://eprint.iacr.org/2006/402.

[25] BARD, G. V.—COURTOIS, N. T.—JEFFERSON, CH.: *Efficient methods for conversion and solution of sparse systems of low-degree multivariate polynomials over GF(2) via SAT-solvers*, Cryptology ePrint Archive, Report 2007/024, 2007, http://eprint.iacr.org/2007/024.

[26] FAUGÈRE, J.-CH.: *A new efficient algorithm for computing Gröbner bases (F_4)*, J. Pure Appl. Algebra **139** (1999), 61–88, www.elsevier.com/locate/jpaa.

[27] GRANBOULAN, L.—PORNIN, T.: *Perfect block ciphers with small blocks*, in: Proceedings of Fast Software Encryption—FSE '07 (A. Biryukov), Lecture Notes in Comput. Sci., Vol. 4593, Springer-Verlag, Berlin, 2007, pp. 452–465.

[28] GROSSMAN, E.K.—TUCKERMAN, B.: *Analysis of a Feistel-Like Cipher Weakened by Having no Rotating Key.* Technical Report: IBM Thomas J. Watson Research Center Report RC 6375, IBM, Poughkeepsie, New York, USA, 1977.

[29] KAHN, D.: *The Codebreakers.* The Comprehensive History of Secret Communication from Ancient Times to the Internet, Second Edition, Scribner (a division of Simon & Schuster), New York, USA. First published in 1967, new chapter added in 1996.

[30] MARRARO, L.—MASSACCI, F.: *Towards the formal verification of ciphers: logical cryptanalysis of DES*, in: Proceedings of the third LICS Workshop on Formal Methods and Security Protocols (E. Clarke, N. Heintze, eds.), Federated Logic Conferences——FLOC '99, 1999.

[31] MARNEWECK, K.: *An introduction to KeeLoq code hopping,* An Introduction to Keeloq Code Hopping, Technical Report: Microchip Technology Inc, TB003, Microchip Technology Inc, Chandler, Arizona, USA, 1996,
http://ww1.microchip.com/downloads/en/AppNotes/91002a.pdf.

[32] *Hopping code decoder using a PIC16C56, AN642*, Microchip.
http://www.keeloq.boom.ru/decryption.pdf.

[33] *Using KeeLoq to validate subsystem compatibility, AN827*, Microchip,
http://ww1.microchip.com/downloads/en/AppNotes/00827a.pdf.

[34] EÉN, N.—SÖRENSSON, N.: *MiniSat 2.0. An open-source SAT solver package*,
http://www.cs.chalmers.se/Cs/Research/FormalMethods/MiniSat.

[35] MIRONOV, I.—ZHANG, L.: *Applications of SAT solvers to cryptanalysis of hash functions*, in: Proceedings of The 9th International Conference on the Theory and Applications of Satisfiability Testing—SAT '06 (A. Biere, C. P. Gomes, eds.), Lecture Notes in Comput. Sci., Vol. 4121, Springer-Verlag, Berlin, 2006, pp. 102–115,
http://eprint.iacr.org/2006/254.

[36] *Random permutation statistics*, Wikipedia, 22 January, 2008, `http://en.wikipedia.org/wiki/Random permutation statistics`, the version was written by M. Riedel, `http://www.geocities.com/markoriedelde/papers/ /randperms.pdf`.

[37] Singular: A free computer algebra system for polynomial computations, `http://www.singular.uni-kl.de`.

Appendix. 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 A, we need to know how many fixed points, on average, 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 must be searched, in order 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.

TABLE 3. Fixed points of random permutations and their 8th powers.

Size of the domain	2^{12}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}
Experiments	1,000	10,000	10,000	10,000	10,000	100,000
Aborts ($n_1 < 2$)	780	7,781	7,628	7,731	7,727	76,824
Good examples ($n_1 \geq 2$)	220	2,219	2,372	2,269	2,273	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	2,482	2,483	4,918	9,752	19,829	39,707
Percentage (η)	60.60 %	60.62 %	60.11 %	59.59 %	60.51 %	60.59 %

In our experiment, we generated random permutations f of domain size 2^{12} through 2^{16}. We checked for fixed points by exhaustion. Indeed if that permutation had zero or one fixed points, then we would denote this as an abort. Then for those permutations that did not abort (i.e., those with 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 number of fixed points for f^8 is denoted n_8.

The nomenclature "average location" indicates how many domain points of f had to be tried before finding two points that are both fixed points of f and f^8 as well.

Received October 25, 2007

Nicolas T. Courtois
University College London
Computer Science MPEB
Gower Street
London WC1E 6BT
U.K.
E-mail: n.courtois@ucl.ac.uk

Gregory V. Bard
Department of Mathematics
Fordham University
John Mulcahey Hall
Bronx, NY, New York 10458
U.S.A
E-mail: bard@fordham.edu

Andrey Bogdanov
Horst Görtz Institute
for IT Security
Faculty of Electronics and
Information Technology
Ruhr University Bochum
Universität Straße 150
D-44780 Bochum
GERMANY
E-mail: abogdanov@crypto.rub.de