A Practical Attack on KeeLoq*

Sebastiaan Indesteege[1,**], Nathan Keller[2,***], Orr Dunkelman[1], Eli Biham[3], and Bart Preneel[1]

[1] Department of Electrical Engineering ESAT/SCD-COSIC, Katholieke Universiteit Leuven. Kasteelpark Arenberg 10, B-3001 Heverlee, Belgium.
{sebastiaan.indesteege,orr.dunkelman,bart.preneel}@esat.kuleuven.be
[2] Einstein Institute of Mathematics, Hebrew University. Jerusalem 91904, Israel.
nkeller@math.huji.ac.il
[3] Computer Science Department, Technion. Haifa 32000, Israel.
biham@cs.technion.ac.il

Abstract. KeeLoq is a lightweight block cipher with a 32-bit block size and a 64-bit key. Despite its short key size, it is widely used in remote keyless entry systems and other wireless authentication applications. For example, authentication protocols based on KeeLoq are supposedly used by various car manufacturers in anti-theft mechanisms. This paper presents a practical key recovery attack against KeeLoq that requires 2^{16} known plaintexts and has a time complexity of $2^{44.5}$ KeeLoq encryptions. It is based on the slide attack and a novel approach to meet-in-the-middle attacks. The fully implemented attack requires 65 minutes to obtain the required data and 7.8 days of calculations on 64 CPU cores. A variant which requires 2^{16} chosen plaintexts needs only 3.4 days on 64 CPU cores. Using only 10 000 euro, an attacker can purchase a cluster of 50 dual core computers that will find the secret key in about two days. We investigated the way KeeLoq is intended to be used in practice and conclude that our attack can be used to subvert the security of real systems. An attacker can acquire chosen plaintexts in practice, and one of the two suggested key derivation schemes for KeeLoq allows to recover the master secret from a single key.

Key words: KeeLoq, cryptanalysis, block ciphers, slide attacks, meet-in-the-middle attacks.

1 Introduction

The KeeLoq technology [13] by Microchip Technology Inc. includes the KeeLoq block cipher and several authentication protocols built on top of it. The KeeLoq

* This work was supported in part by the Concerted Research Action (GOA) Ambiorics 2005/11 of the Flemish Government, by the IAP Programme P6/26 BCRYPT of the Belgian State (Belgian Science Policy), and in part by the European Commission through the IST Programme under Contract IST-2002-507932 ECRYPT.
** F.W.O. Research Assistant, Fund for Scientific Research — Flanders (Belgium).
*** This author is supported by the Adams Fellowship Program of the Israel Academy of Sciences and Humanities.

block cipher allows for very low cost and power efficient hardware implementations. This property has undoubtedly contributed to the popularity of the cipher in various wireless authentication applications. For example, multiple car manufacturers supposedly use, or have used KeeLoq to protect their cars against theft [5–7, 9, 17].[1]

Despite its design in the 80's, the first cryptanalysis of KeeLoq was only published by Bogdanov [5] in February 2007. This attack is based on the slide technique and a linear approximation of the non-linear Boolean function used in KeeLoq. The attack has a time complexity of 2^{52} KeeLoq encryptions and requires 16 GB of storage. It also requires the entire codebook, i.e., 2^{32} known plaintexts.

Courtois et al. apply algebraic techniques to cryptanalyse KeeLoq [7, 9]. Although a direct algebraic attack fails for the full cipher, they reported various successful slide-algebraic attacks. For example, they claim that an algebraic attack can recover the key when given a slid pair in 2.9 seconds on average. As there is no way to ensure or identify the existence of a slid pair in the data sample, the attack is simply repeated 2^{32} times, once for each pair generated from 2^{16} known plaintexts. They also described attacks requiring the entire codebook, which exploit certain assumptions with respect to fixed points of the internal state. The fastest of these requires 2^{27} KeeLoq encryptions and has an estimated success probability of 44% [9].

In [6], Bogdanov published an updated version of his attack. A refined complexity analysis yields a slightly smaller time complexity, i.e., $2^{50.6}$ KeeLoq encryptions while still requiring the entire codebook. This paper also includes an improvement using the work of Courtois et al. [7] on the cycle structure of the cipher. We note that the time complexity of the attack using the cycle structure given in [6] is based on an assumption from an earlier version of [7], that a random word can be read from 16 GB of memory with a latency of only 1 clock cycle. This is very unrealistic in a real machine, so the actual time complexity is probably much higher. In a later version of [7], this assumption on the memory latency was changed to be 16 clock cycles.

Our practical attack is based on the slide attack as well. However, unlike other attacks, we combine it with a novel meet-in-the-middle attack. The optimised version of the attack uses 2^{16} known plaintexts and has a time complexity of $2^{44.5}$ KeeLoq encryptions. We have implemented our attack and the total running time is roughly 500 days. As the attack is fully parallelizable, given x CPU cores, the total running time is only $500/x$ days. A variant which requires 2^{16} chosen plaintexts needs only $218/x$ days on x CPU cores. For example, for 10 000 euro, one can obtain 50 dual core computers, which will take about two days to find the key. Another, probably even cheaper, though illegal option would be to rent a botnet to carry out the computations.

KeeLoq is used in two protocols, the "Code Hopping" and the "Identify Friend or Foe (IFF)" protocol. In practice, the latter protocol, a simple challenge

[1] We verified these claims to the best of our ability, however, no car manufacturer seems eager to publically disclose which algorithms are used.

Table 1. An overview of the known attacks on KeeLoq.

Attack Type	Complexity			Reference
	Data	Time	Memory	
Time-Memory Trade-Off	2 CP	$2^{42.7}$	≈ 100 TB	[11]
Slide/Algebraic	2^{16} KP	$2^{65.4}$	?	[7, 9]
Slide/Algebraic	2^{16} KP	$2^{51.4}$	?	[7, 9]
Slide/Guess-and-Determine	2^{32} KP	2^{52}	16 GB	[5]
Slide/Guess-and-Determine	2^{32} KP	$2^{50.6}$	16 GB	[6]
Slide/Cycle Structure	2^{32} KP	$2^{39.4}$	16.5 GB	[7]
Slide/Cycle/Guess-and-Det.[a]	2^{32} KP	(2^{37})	16.5 GB	[6]
Slide/Fixed Points	2^{32} KP	2^{27}	> 16 GB	[9]
Slide/Meet-in-the-Middle	2^{16} KP	$2^{45.0}$	≈ 2 MB	Sect. 3.3
Slide/Meet-in-the-Middle	2^{16} KP	$2^{44.5}$	≈ 3 MB	Sect. 3.4
Slide/Meet-in-the-Middle	2^{16} CP	$2^{44.5}$	≈ 2 MB	Sect. 3.5
Time-Memory-Data Trade-Off	68 CP, 34 RK	$2^{39.3}$	≈ 10 TB	[2]
Related Key	66 CP, 34 $RK^{\ggg}$	negligible	negligible	Sect. A.1
Related Key	512 CP, 2 $RK^{\ggg}$	2^{32}	negligible	Sect. A.1
Related Key/Slide/MitM	2^{17} CP, 2 $RK^{\oplus}$	$2^{41.9}$	≈ 16 MB	Sect. A.2

Time complexities are expressed in full KeeLoq encryptions (528 rounds).
KP: known plaintexts; CP: chosen plaintexts
$RK^{\ggg}$: related keys (by rotation); $RK^{\oplus}$: related keys (flip LSB)

[a] The time complexity for this attack is based on very unrealistic memory latency assumptions and hence will be much higher in practice.

response protocol, is the most interesting target to acquire the data that is necessary to mount the attack. Because the challenges are not authenticated in any way, an attacker can obtain as many chosen plaintext/ciphertext pairs as needed from a transponder (e.g., a car key) implementing this protocol. Depending on the transponder, it takes 65 or 98 minutes to gather 2^{16} plaintext/ciphertext pairs.

Finally, as was previously noted by Bogdanov [6], we show that one of the two suggested key derivation algorithms is blatantly flawed, as it allows an attacker to reconstruct many secret keys once a single secret key has been exposed.

Given that KeeLoq is a cipher that is widely used in practice, side-channel analysis may also be a viable option for attacking chips that implement KeeLoq. However, we do not consider this type of attack in this paper. One could also attack the "Identify Friend or Foe (IFF)" protocol itself. For instance, as the responses are only 32 bits long, one could mount the birthday attack using 2^{16} known challenge/response pairs. This would not recover the secret key, thus posing less of a threat to the overall security of the system.

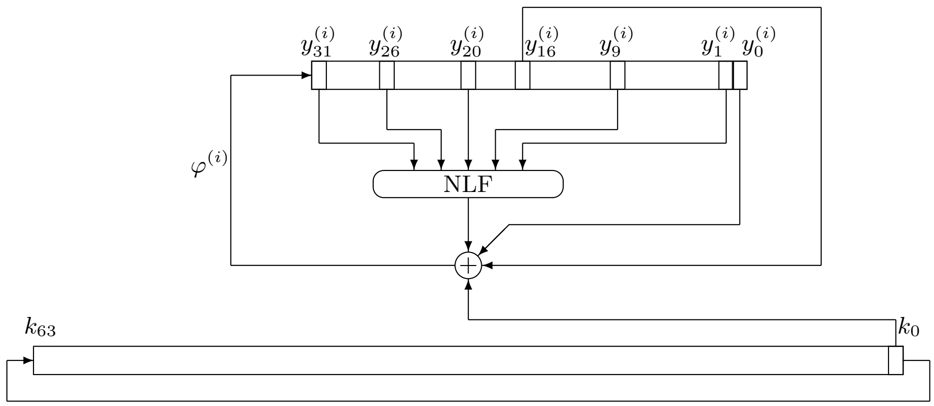

Fig. 1. The i-th KeeLoq encryption cycle.

Table 1 presents an overview of the known attacks on KeeLoq, including ours. In order to make comparisons possible, we have converted all time complexities to the number of KeeLoq encryptions needed for the attack.[2]

The structure of this paper is as follows. In Sect. 2, we describe the KeeLoq block cipher and how it is intended to be used in practice. Our attacks are described in Sect. 3. In Sect. 4 we discuss our experimental results and in Sect. 5 we show the relevance of our attacks in practice. Finally, in Sect. 6 we conclude. In Appendix A, we explore some related key attacks on KeeLoq that are more of theoretical interest.

2 Description and Usage of KeeLoq

2.1 The KeeLoq Block Cipher

The KeeLoq block cipher has a 32-bit block size and a 64-bit key. It consists of 528 identical rounds each using one bit of the key. A round is equivalent to an iteration of a non-linear feedback shift register (NLFSR), as shown in Fig. 1.

More specifically, let $Y^{(i)} = (y_{31}^{(i)}, \ldots, y_0^{(i)}) \in \{0,1\}^{32}$ be the input to round i $(0 \leq i < 528)$ and let $K = (k_{63}, \ldots, k_0) \in \{0,1\}^{64}$ be the key. The input to round 0 is the plaintext: $Y^{(0)} = P$. The ciphertext is the output after 528 rounds: $C = Y^{(528)}$. The round function can be described as follows (see Fig. 1):

$$\begin{aligned} \varphi^{(i)} &= \mathrm{NLF}\left(y_{31}^{(i)}, y_{26}^{(i)}, y_{20}^{(i)}, y_{9}^{(i)}, y_{1}^{(i)}\right) \oplus y_{16}^{(i)} \oplus y_{0}^{(i)} \oplus k_{i \bmod 64} \ , \\ Y^{(i+1)} &= (\varphi^{(i)}, y_{31}^{(i)}, \ldots, y_{1}^{(i)}) \ . \end{aligned} \tag{1}$$

[2] We list slightly better complexities for the attacks from [7, 9] because we used a more realistic conversion factor from CPU clocks to KeeLoq rounds (i.e., 12 rather than 4 CPU cycles per KeeLoq round).

The non-linear function NLF is a Boolean function of 5 variables with output vector $3A5C742E_x$ — i.e., NLF(i) is the i-th bit of this hexadecimal constant, where bit 0 is the least significant bit. We can also represent the non-linear function in its algebraic normal form (ANF):

$$\begin{aligned}\mathrm{NLF}(x_4, x_3, x_2, x_1, x_0) = {} & x_4x_3x_2 \oplus x_4x_3x_1 \oplus x_4x_2x_0 \oplus x_4x_1x_0 \oplus \\ & x_4x_2 \oplus x_4x_0 \oplus x_3x_2 \oplus x_3x_0 \oplus x_2x_1 \oplus x_1x_0 \oplus \\ & x_1 \oplus x_0 \ .\end{aligned} \tag{2}$$

Decryption uses the inverse round function, where i now ranges from 528 down to 1.

$$\begin{aligned}\theta^{(i)} &= \mathrm{NLF}\left(y_{30}^{(i)}, y_{25}^{(i)}, y_{19}^{(i)}, y_8^{(i)}, y_0^{(i)}\right) \oplus y_{15}^{(i)} \oplus y_{31}^{(i)} \oplus k_{i-1 \bmod 64} \ , \\ Y^{(i-1)} &= (y_{30}^{(i)}, \ldots, y_0^{(i)}, \theta^{(i)}) \ .\end{aligned} \tag{3}$$

There used to be some ambiguity about the correct position of the taps. Our description agrees with the "official" documentation [5, 6, 9, 15]. Additionally, we have used test vectors generated by an actual HSC410 chip [14], manufactured by Microchip Inc., to verify that our description and implementation of KeeLoq are indeed correct. Finally, we note that our attacks are unaffected by this difference.

2.2 Protocols built on KeeLoq

A device like the HCS410 by Microchip Technology Inc. [14] supports two authentication protocols based on KeeLoq: "KeeLoq Hopping Codes" and "KeeLoq Identify Friend or Foe (IFF)". The former uses a 16-bit secret counter, synchronised between both parties. In order to authenticate, the encoder (e.g., a car key) increments the counter and sends the encrypted counter value to the decoder (e.g., the car), which verifies if the received ciphertext is correct. In practice, this protocol would be initiated by a button press of the car owner.

The second protocol, "KeeLoq Identify Friend or Foe (IFF)" [14], is a simple challenge response protocol. The decoder (e.g., the car) sends a 32-bit challenge. The transponder (e.g., the car key) uses the challenge as a plaintext, encrypts it with the KeeLoq block cipher[3] under the shared secret key, and replies with the ciphertext. This protocol is executed without any user interaction whenever the transponder receives power and an activation signal via inductive coupling from a nearby decoder. Hence, no battery or button presses are required. It could for instance be used in vehicle immobilisers by placing the decoder near the ignition. Inserting the car key in the ignition would place the transponder within range of the decoder. The latter would then activate the transponder and execute the protocol, all completely transparent to the user. The car would then either disarm the immobiliser or activate the alarm, depending on whether the authentication was successful.

[3] This corresponds to what is called the "HOP algorithm" in [14]. The other option, the so-called "IFF algorithm", uses a reduced version of KeeLoq with 272 rounds instead of 528. Our attacks are also applicable to this variant, without any change.

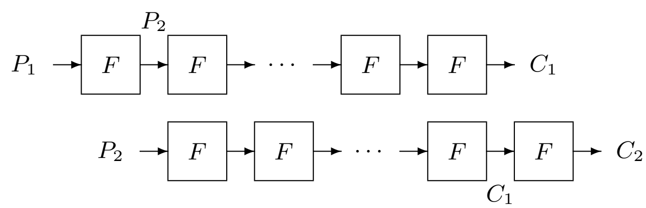

Fig. 2. A typical slide attack.

Of course both protocols can be used together in a single device, thereby saving costs. For example, the HCS410 chip [14] supports this combined mode of operation, possibly using the same secret key for both protocols, depending on the configuration options used.

3 Our Attacks on KeeLoq

This section describes our attacks on KeeLoq. We combine a slide attack with a novel meet-in-the-middle approach to recover the key from a slid pair. First we explain some preliminaries that are used in the attacks. Then we proceed to the description of the attack scenario using known plaintexts and a generalisation thereof. Finally, we show how chosen plaintexts can be used to improve the attack.

3.1 The Slide Property

Slide attacks were introduced by Biryukov and Wagner [3] in 1999. The typical candidate for a slide attack is a block cipher consisting of a potentially very large number of iterations of an identical key dependent permutation F. In other words, the subkeys are repeated and therefore the susceptible cipher can be written as

$$C = \underbrace{F\left(F\left(\ldots F(P)\right)\right)}_{r} = F^r(P) \ . \tag{4}$$

This permutation does not necessarily have to coincide with the rounds of the cipher, i.e., F might combine several rounds of the cipher.

A slide attack aims at exploiting such a self-similar structure to reduce the strength of the entire cipher to the strength of F. Thus, it is independent of the number of rounds of the cipher. To accomplish this, a so-called *slid pair* is needed. This is a pair of plaintexts that satisfies the slide property

$$P_2 = F(P_1) \ . \tag{5}$$

We depict such a slid pair in Fig. 2. For a slid pair, the corresponding ciphertexts also satisfy the slide property, i.e., $C_2 = F(C_1)$. By repeatedly encrypting this slid pair, we can generate as many slid pairs as needed [4, 10]. As each slid pair gives us a pair of corresponding inputs and outputs of the key dependent permutation F, it can be used to mount an attack against F.

KeeLoq has 528 identical rounds, each using one bit of the 64-bit key. After 64 rounds the key is repeated. So in the case of KeeLoq, we combine 64 rounds into F. However, because the number of rounds in the cipher is not an integer multiple of 64, a straightforward slid attack is not possible. A solution to this problem is to guess the 16 least significant bits of the key and use this to strip off the final 16 rounds. Then, a slide attack can be applied to the remaining 512 rounds [5, 7, 9].

In order to get a slid pair, 2^{16} known plaintexts are used. As the block size of KeeLoq is 32 bits, we expect that a random set of 2^{16} plaintexts contains a slid pair due to the birthday paradox.[4] Determining which pair is a slid pair is done by the attack itself. Simply put, the attack is attempted with every pair. If it succeeds, the pair is a slid pair, otherwise it is not.

3.2 Determining Key Bits

If two intermediate states of the KeeLoq cipher, separated by 32 rounds (or less) are known, all the key bits used in these rounds can easily be recovered. This was first described by Bogdanov [5], who refers to it as the "linear step" of his attack.

Let $Y^{(i)} = (y_{31}^{(i)}, \ldots, y_0^{(i)})$ and $Y^{(i+t)} = (y_{31}^{(i+t)}, \ldots, y_0^{(i+t)})$ be the two known states; $t \leq 32$. If we encrypt $Y^{(i)}$ by one round, the newly generated bit is

$$\varphi^{(i)} = \mathrm{NLF}\left(y_{31}^{(i)}, y_{26}^{(i)}, y_{20}^{(i)}, y_9^{(i)}, y_1^{(i)}\right) \oplus y_{16}^{(i)} \oplus y_0^{(i)} \oplus k_{i \bmod 64} \ . \tag{6}$$

Because of the non-linear feedback shift register structure of the round function and since $t \leq 32$, the bit $\varphi^{(i)}$ is equal to $y_{32-t}^{(i+t)}$, which is one of the bits of $Y^{(i+t)}$ and thus known. Hence

$$k_{i \bmod 64} = \mathrm{NLF}\left(y_{31}^{(i)}, y_{26}^{(i)}, y_{20}^{(i)}, y_9^{(i)}, y_1^{(i)}\right) \oplus y_{16}^{(i)} \oplus y_0^{(i)} \oplus y_{32-t}^{(i+t)} \ . \tag{7}$$

By repeating this t times, all t key bits can be recovered. The amount of computations that need to be carried out is equivalent to t rounds of KeeLoq. This simple step will prove to be very useful in our attack.

3.3 Basic Attack Scenario

We now describe the basic attack scenario, which uses 2^{16} known plaintexts. For clarity, the notation used is shown in Fig. 3 and a pseudocode overview is given in Fig. 4. We denote 16 rounds of KeeLoq by $g_{\hat{k}}$, where $\hat{k}$ denotes the 16

[4] The probability that a set of 2^{16} random plaintexts contains at least one slid pair is $1 - \left(1 - 2^{-32}\right)^{2^{32}} \approx 0.63$. Hence, the attack has a success probability of about 63%. With not much higher data complexity, higher success rates can be achieved.

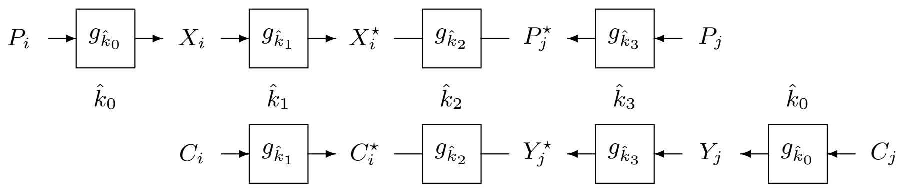

Fig. 3. The notation used in the attack.

for all $\hat{k}_0 \in \{0,1\}^{16}$ **do**
    for all plaintexts $P_i, 0 \leq i < 2^{16}$ **do**
        Partially encrypt P_i to X_i.
        Partially decrypt C_i to Y_i.
    for all $\underline{P_j^\star} \in \{0,1\}^{16}$ **do**
        for all plaintexts $P_j, 0 \leq j < 2^{16}$ **do**
            Determine the key bits $\hat{k}_3$.
            Partially decrypt Y_j to $Y_j^\star$.
            Save the tuple $\left\langle P_j^\star, Y_i^\star, \hat{k}_3 \right\rangle$ in a table.
        for all plaintexts $P_i, 0 \leq i < 2^{16}$ **do**
            Determine the key bits $\hat{k}_1$.
            Partially encrypt C_i to $C_i^\star$.
            for all collisions $\overline{C_i^\star} = \underline{Y_j^\star}$ in the table **do**
                Determine the key bits $\hat{k}_2$ from $X_i^\star$ and $P_j^\star$.
                Determine the key bits $\hat{k}_2'$ from $C_i^\star$ and $Y_j^\star$.
                if $\hat{k}_2 = \hat{k}_2'$ **then**
                    Encrypt 2 known plaintexts with the key $k = (\hat{k}_3, \hat{k}_2, \hat{k}_1, \hat{k}_0)$.
                    if the correct ciphertexts are found **then**
                        return success (the key is k)
return failure (i.e., there was no slid pair)

Fig. 4. The attack algorithm.

key bits used in these rounds. The 64-bit key k is split into four equal parts: $k = (\hat{k}_3, \hat{k}_2, \hat{k}_1, \hat{k}_0)$, where $\hat{k}_0$ contains the 16 least significant key bits.

As already mentioned in Sect. 3.1, the first step of the attack is to guess $\hat{k}_0$ — the 16 least significant bits of the key. This enables us to partially encrypt each of the 2^{16} plaintexts by 16 rounds (P_i to X_i) and partially decrypt each of the 2^{16} ciphertexts by 16 rounds (C_j to Y_j).

Encrypting X_i by 16 more rounds yields $X_i^\star$. Similarly, decrypting P_j by 16 rounds yields $P_j^\star$ (see Fig. 3). We denote the 16 most significant bits of $X_i^\star$ by $\overline{X_i^\star}$, and the 16 least significant bits of $P_j^\star$ by $\underline{P_j^\star}$. Note that, because $X_i^\star$ and $P_j^\star$ are separated by 16 rounds, it holds that $\overline{X_i^\star} = \underline{P_j^\star}$, provided that P_i and P_j form a slid pair. This is due to the structure of the cipher.

The next step in the attack is to apply a meet-in-the-middle approach. We guess the 16-bit value $\underline{P_j^\star}$. For each plaintext P_j we can then determine $\hat{k}_3$

using the algorithm described in Sect. 3.2. Indeed, as the other bits of $P_j^\star$ are determined by P_j, we know all of $P_j^\star$ when given the plaintext. There is always exactly one solution per plaintext. Using this part of the key, we can now partially decrypt Y_j to $Y_j^\star$. This result is saved in a hash table indexed by the 16-bit value $\underline{Y_j^\star}$. Each record in the hash table holds a tuple consisting of $P_j^\star$, $Y_j^\star$ and the 16 key bits $\hat{k}_3$.

Now we do something similar from the other side. For each plaintext we use the algorithm from Sect. 3.2 to determine $\hat{k}_1$. Again this can be done because we know all of $X_i^\star$, and there is exactly one solution per plaintext. Knowing $\hat{k}_1$, we partially encrypt C_i to $C_i^\star$.

Note that if P_i and P_j are indeed a slid pair their partial encryptions and decryptions (under the correct key) must "meet in the middle". More specifically, it must hold that $\overline{C_i^\star} = \underline{Y_j^\star}$. So, we look for a record in the hash table for which such a collision occurs. Because the hash table is indexed by $\underline{Y_j^\star}$ this can be done very efficiently. A slid pair produces a collision, provided the guesses for $\hat{k}_0$ and $\underline{P_j^\star}$ are correct. Therefore, we are guaranteed that all slid pairs are found at some point. Of course, a collision does not guarantee that the pair is actually a slid pair.

Finally, we check each candidate slid pair found. We determine the remaining key bits $\hat{k}_2$ from $X_i^\star$ and $P_j^\star$ and similarly $\hat{k}_2'$ from $C_i^\star$ and $Y_j^\star$. If $\hat{k}_2$ and $\hat{k}_2'$ are not equal, the candidate pair is not a slid pair. Note that we can determine the key bits one by one and stop as soon as there is a disagreement. This slightly reduces the complexity of the attack.

If $\hat{k}_2 = \hat{k}_2'$, we have found a pair of plaintexts and a key with the property that encrypting P_i by 64 rounds gives P_j and encrypting C_i by 64 rounds gives C_j. This is what is expected from a slid pair. It is however possible that the recovered key is not the correct key, so we can verify it by a trial encryption of one of the known plaintexts. Even if a wrong key is suggested during the attack, and discarded by the trial encryption, we are still guaranteed to find the correct key eventually, provided there is at least one slid pair among the given plaintexts.

Complexity Analysis. Using one round of KeeLoq as a unit, the time complexity of the attack can be expressed as

$$2^{16}\left(32 \cdot 2^{16} + 2^{16}\left(32 \cdot 2^{16} + 2^{16}\left(32 + N_{\text{coll}} \cdot V\right)\right)\right) \quad , \tag{8}$$

when N_{coll} denotes the expected number of collisions for a single guess of $\hat{k}_0$, $\underline{P_j^\star}$ and a given plaintext P_i, and V denotes the average cost of verifying one collision, i.e., checking if it leads to a candidate key and if this key is correct. This follows directly from the description of the attack. As the hash table has 2^{16} entries and a collision is equivalent to a 16-bit condition, $N_{\text{coll}} = 1$. In the verification step, we can determine one bit at a time and stop as soon as there is a disagreement, which happens with probability 1/2. Only when there is no disagreement after 16 key bits, we do two full trial encryptions to check the recovered key. Of course the second trial encryption is only useful if the first one gave the expected result. Hence, due to this early abort technique, the average cost of verifying one collision is

$$V = 2 \cdot \sum_{i=0}^{15} 2^{-i} + 2^{-16} \cdot \left(528 + 528 \cdot 2^{-32}\right) \approx 4 \ . \tag{9}$$

Thus the overall complexity of the attack is $2^{54.0}$ KeeLoq rounds, which amounts to $2^{45.0}$ full KeeLoq encryptions.

As mentioned before, the data complexity of the attack is 2^{16} known plaintexts. The storage requirements are very modest. The attack stores the plaintext/ciphertext pairs, 2^{16} values for X_i and Y_i, and a hash table with 2^{16} records of 80 bits each. This amounts to a bit over 2 MB of RAM.

3.4 A Generalisation of the Attack

The attack presented in the previous section can be generalised by varying the number of rounds to partially encrypt/decrypt in each step of the attack. We denote by t_p the number of rounds to partially encrypt from the plaintext side (left on Fig. 3) and by t_c the number of rounds to partially decrypt from the ciphertext side (right on Fig. 3). More specifically, encrypting X_i by t_p rounds yields $X_i^\star$, encrypting C_i by t_p rounds yields $C_i^\star$. On the ciphertext side, $P_j^\star$ is obtained by decrypting P_j by t_c rounds and $Y_j^\star$ by decrypting Y_j by t_c rounds. Also, the partial keys $\hat{k}_0$ through $\hat{k}_3$ are adapted accordingly to contain the appropriate key bits.

Let t_o denote the number of bits that, provided P_i and P_j form a slid pair, overlap between $X_i^\star$ and $P_j^\star$. As $X_i^\star$ and $P_j^\star$ are separated by $48 - t_p - t_c$ rounds, it holds that $t_o = 32 - (48 - t_p - t_c) = t_p + t_c - 16$. The t_o least significant bits of $P_j^\star$ are denoted by $\underline{P_j^\star}$ and the t_o most significant bits of $X_i^\star$ are denoted by $\overline{X_i^\star}$.

Depending on the choices for the parameters t_p and t_c, the attack scenario has to be modified slightly. If $t_c < t_o$, not all plaintexts necessarily yield a solution for a given $\underline{P_j^\star}$ when determining $\hat{k}_3 = (k_{63}, \ldots, k_{64-t_c})$ because $t_o - t_c$ of the guessed bits overlap with plaintext bits. Similarly, if $t_c > t_o$, each plaintext is expected to offer multiple solutions because $t_c - t_o$ extra bits have to be guessed before all of $P_j^\star$ is known. From the other side, similar observations can be made.

In Sect. 3.3, the parameters were $t_p = t_c = 16$ which results in $t_o = 16$. It is clear that the choice of these parameters influences both the time and memory complexity of the attack.

Complexity Analysis. The generalisation leads to a slightly more complex formula for expressing the time complexity of the attack. Because of the duality between guessing extra bits and filtering because of overlapping bits, all cases can be expressed in a single formula, which is a generalisation of (8) (i.e., with $t_p = t_c = 16$, it reduces to (8)):

$$2^{16}\left(32 \cdot 2^{16} + 2^{t_o}\left(2t_c \cdot 2^{16+t_c-t_o} + 2^{16+t_p-t_o}\left(2t_p + N_{\text{coll}} \cdot V\right)\right)\right) \ . \qquad (10)$$

In the generalised case, finding a collision is equivalent to finding an entry in a table of $16 + t_p - t_o$ elements that satisfies a t_o bit condition, so $N_{\text{coll}} = 2^{16+t_c-t_o}/2^{t_o}$. Verifying a collision now requires an average effort of

$$V = 2 \cdot \sum_{i=0}^{47-t_p-t_c} 2^{-i} + 2^{t_p+t_c-48} \cdot \left(528 + 528 \cdot 2^{-32}\right) \qquad (11)$$

KeeLoq rounds. Simplification yields that the total complexity is equal to

$$32 \cdot 2^{32} + 2t_c \cdot 2^{32+t_c} + 2t_p \cdot 2^{32+t_p} + 4 \cdot 2^{80-t_p-t_c} + 528 \cdot 2^{32} \ . \qquad (12)$$

The optimum is found when $t_p = t_c = 15$ and thus $t_o = 14$, where the complexity reduces to $2^{53.524}$ KeeLoq rounds or $2^{44.5}$ full KeeLoq encryptions.

The memory requirements in the generalised case can also easily be evaluated. As before, 2^{16} plaintext/ciphertext pairs and 2^{16} values for X_i and Y_i are stored. The hash table now has $2^{16+t_p-t_o}$ entries of $64 + t_p$ bits each. For $t_p = t_c = 15$, the required memory is still less than 3 MB.

3.5 A Chosen Plaintext Attack

Using chosen plaintexts instead of known plaintexts, the attack can be improved. Consider the generalised attack from Sect. 3.4 in the case where $t_c < t_o$ (which is equivalent to $t_p > 16$). In this case, the $t_o - t_c$ least significant bits of the plaintext P_j are bits $(t_o, \ldots, t_c + 1)$ of $P_j^\star$. Hence, choosing the 2^{16} plaintexts in such a way that these $t_o - t_c$ least significant bits are equal to some constant, only 2^{t_c} guesses for $\underline{P_j^\star}$ have to be made at the beginning of the meet-in-the-middle step, instead of $\overline{2^{t_o}}$.

Complexity Analysis. As chosen plaintexts are only useful for the attack when $t_c < t_o$, we will only consider this case. The time complexity of the attack, in KeeLoq rounds, can be expressed as

$$2^{16}\left(32 \cdot 2^{16} + 2^{t_c}\left(2t_c \cdot 2^{16} + 2^{16+t_p-t_o}\left(2t_p + N_{\text{coll}} \cdot V\right)\right)\right) \ . \qquad (13)$$

The expected number of collisions is $N_{\text{coll}} = 2^{16}/2^{t_o}$. The verification cost, V, is given by (11). Simplification yields

$$32 \cdot 2^{32} + 2t_c \cdot 2^{32+t_c} + 2t_p \cdot 2^{48} + 4 \cdot 2^{80-t_p-t_c} + 528 \cdot 2^{32} \ . \qquad (14)$$

The optimum is found when $t_p = 20$, $t_c = 13$ and thus $t_o = 17$, where the attack has a time complexity of $2^{53.500}$ KeeLoq rounds or $2^{44.5}$ full KeeLoq encryptions.

It is clear that the (theoretical) advantage over the known plaintext attack from Sect. 3.4 is not significant. However, as is discussed in the next section, the chosen plaintext variant can provide a significant gain in our practical implementation, because the verification cost V turns out to be higher there.

The memory complexity is about 2 MB as in Sect. 3.3 because the size of the hash table is the same. The data complexity remains at 2^{16} plaintext/ciphertext pairs, but note that we now require chosen plaintexts instead of known plaintexts.

4 Experimental Results

We have fully implemented and tested the attacks, using both simulated data and real data acquired from a HCS410 chip [14]. We made extensive use of bit slicing to do many encryptions in parallel throughout the implementation. However, because this parallelisation is not useful while verifying a collision, this verification step becomes more expensive in comparison. Hence, the optimal parameters for our implementation differ slightly from the theoretical ones. For the known plaintext attack from Sect. 3.4, the optimal parameters for our implementation were found to be $t_p = t_c = 16$. This means that, at least in our implementation, the best attack is the basic attack from Sect. 3.3. For the chosen plaintext attack, the optimal parameters are $t_p = 22$ and $t_c = 13$.

If we give the correct values for the 16 least significant key bits, the known plaintext attack completes in 10.97 minutes on average.[5] The chosen plaintext attack needs just 4.79 minutes to complete the same task.[6] This large difference can be explained by considering the impact of V, the cost of the verification step, on the time complexity of the attack. If V increases, and t_p and t_c are adapted as needed because their optimal values may change, the time complexity of the known plaintext attack increases much faster than the time complexity of the chosen plaintext attack does. Hence, even though their theoretical time complexities are the same, the chosen plaintext attack performs much better in our practical implementation because V is higher than the theoretical value.

We did not stop either of the attacks once a slid pair and the correct key were found, so we essentially tested the worst-case behaviour of the attack. This also explains the very small standard deviations of the measured running times. The machine used is an AMD Athlon 64 X2 4200+ with 1 GB of RAM (only one of the two CPU cores was used) running Linux 2.6.17. The attack was implemented in C and compiled with gcc version 4.1.2 (using the `-O3` optimiser flag). Critical parts of the code are written in assembly. Because the memory access pattern is random, but predictable to some extent, prefetching helped us to make maximum use of the cache memory.

The known plaintext attack performs over 288 times faster than the fastest attack with the same data complexity from [7, 9], although the actual increase

[5] We performed 500 experiments. The average running time was 658.15 s and the standard deviation was 1.69 s.

[6] We performed 500 experiments. The average running time was 287.17 s and the standard deviation was 0.55 s.

in speed is probably slightly smaller due to the difference in the machines used. Courtois et al. used (a single core of) a 1.66 GHz Intel Centrino Duo microprocessor [8]. The chosen plaintext attack performs more than 661 times faster, but this comparison is not very fair because chosen plaintexts are used. We note that the practicality of our results should also be compared with exhaustive key search due to the small key size. For the price of about 10 000 euro, one can obtain a COPACOBANA machine [12] with 120 FPGAs which is estimated to take about 1000 days to find a single 64-bit KeeLoq key.[7] Using our attack and 50 dual core computers (which can be obtained for roughly the same price), a KeeLoq key can be found in only two days.

5 Practical Applicability of the Attacks

5.1 Gathering Data

One might wonder if it is possible to gather 2^{16} known, or even chosen plaintexts from a practical KeeLoq authentication system. As mentioned in Sect. 2.2, a device like the HCS410 by Microchip Technology Inc. [14] supports two authentication protocols based on KeeLoq: "KeeLoq Hopping Codes" and "KeeLoq Identify Friend or Foe (IFF)". As the initial value of the counter used in "KeeLoq Hopping Codes" is not known, it is not easy to acquire known plaintexts from this protocol apart from trying all possible initial counter values. Also, since only 2^{16} plaintexts are ever used, knowing this sequence of 2^{16} ciphertexts suffices to break the system as this sequence is simply repeated.

The second protocol, "KeeLoq Identify Friend or Foe (IFF)" [14], is more appropriate for our attack. It is executed without any user interaction as soon as the transponder comes within the range of a decoder and is sent an activation signal. The challenges sent by the decoder are not authenticated in any way. Because of this, an adversary can build a rogue decoder which can be used to gather as many plaintext/ciphertext pairs as needed. The plaintexts can be fully chosen by the adversary, so acquiring chosen plaintexts is no more difficult than just known plaintexts. The only requirement is that the rogue decoder can be placed within the range of the victim's transponder for a certain amount of time. From the timings given in [14], we can conclude that one authentication completes within 60 ms or 90 ms, depending on the baud rate used. This translates into a required time of 65 or 98 minutes to gather the 2^{16} plaintext/ciphertext pairs. As these numbers are based on the maximum delay allowed by the specification [14], a real chip may respond faster, as our experiments confirm. No data is given with respect to the operational range in [14], because this depends on the circuit built around the HCS410 chip. However, one can expect the range to be short.

[7] The estimate was done by adapting the 17 days (worst case) required for finding a 56-bit DES key, taking into consideration the longer key size, the fact that more KeeLoq implementations fit on each FPGA, but in exchange take more clocks to test a key.

5.2 Key Derivation

The impact of the attack becomes even larger when considering the method used to establish the secret keys, as was previously noted by Bogdanov [6]. To simplify key management, the shared secret keys are derived from a 64-bit master secret (the manufacturer's code), a serial number and optionally a seed value [6, 15, 16]. The manufacturer's code is supposed to be constant for a large number of products (e.g., an entire series from a certain manufacturer) and the serial number of a transponder chip is public, i.e., it can easily be read out from the chip. The seed value is only used in the case of so-called "Secure Learning", and can also be obtained from a chip with relative ease [6, 15, 16]. The other option, "Normal Learning", does not use a seed value.

In both types of key derivation mechanisms, a 64-bit identifier is constructed, which contains the serial number, the (optional) seed and some fixed padding. Then, the secret key is derived from this identifier and the master secret using one of two possible methods. The first method simply uses XOR to combine the identifier and the master key. The consequence of this is that once a single key is known, together with the corresponding serial number and (optional) seed value, the master secret can be found very easily.

The second method is based on decryption with the KeeLoq block cipher. The identifier is split into two 32-bit halves which are decrypted using the KeeLoq block cipher, and concatenated again to form the 64-bit secret key. The master secret is used as the decryption key. Although much stronger than the first method, the master secret can still be found using a brute force search. Evidently, once the master secret is known, all keys that were derived from it are also compromised, and the security of the entire system falls to its knees. Thus, it is a much more interesting target than a single secret key. This may convince an adversary to legitimately obtain a car key, for the sole purpose of recovering the master key from its secret key.

6 Conclusion

In this paper we have presented a slide and meet-in-the middle attack on the KeeLoq block cipher which requires 2^{16} known plaintexts and has a time complexity of $2^{44.5}$ KeeLoq encryptions, and a variant using 2^{16} chosen plaintexts with the same theoretical time complexity.

We have fully implemented and tested both attacks. When given 16 key bits, the known plaintext attack completes successfully in 10.97 minutes. Due to implementation details, the chosen plaintext attack requires only 4.79 minutes when given 16 key bits. To the best of our knowledge, this is the fastest known attack on the KeeLoq block cipher.

Finally, we have shown that our attack can be used to attack real systems using KeeLoq due to the way it is intended to be used in practice. Moreover, one of the two suggested ways to derive individual Keeloq keys from a master secret is extremely weak, with potentially serious consequences for the overall security of systems built using the Keeloq algorithm.

Acknowledgements. We would like to thank Wim Aerts and Elke De Mulder for their help with the experiments. Also, we would like to thank the reviewers for their helpful comments.

References

1. Eli Biham, *New Types of Cryptanalytic Attacks Using Related Keys*, Journal of Cryptology, Vol. 7, No. 4, pp. 229–246, Springer-Verlag, 1994.
2. Alex Biryukov, Sourav Mukhopadhyay and Palash Sarkar, *Improved Time-Memory Tradeoffs with Multiple Data*, Proceedings of Selected Areas in Cryptography 2005, Lecture Notes in Computer Science 3897, pp. 245–260, Springer-Verlag, 2006.
3. Alex Biryukov and David Wagner, *Slide Attacks*, Proceedings of Fast Software Encryption '99, Lecture Notes in Computer Science 1636, pp. 245–259, Springer-Verlag, 1999.
4. Alex Biryukov and David Wagner, *Advanced Slide Attacks*, Advances in Cryptology, Proceedings of EUROCRYPT 2000, Lecture Notes in Computer Science 1807, pp. 586–606, Springer-Verlag, 2000.
5. Andrey Bogdanov, *Cryptanalysis of the KeeLoq block cipher*, Cryptology ePrint Archive, Report 2007/055, 16 February 2007, `http://eprint.iacr.org/2007/055/`.
6. Andrey Bogdanov, *Attacks on the KeeLoq Block Cipher and Authentication Systems*, 3rd Conference on RFID Security 2007 (RFIDSec 2007), available online at `http://rfidsec07.etsit.uma.es/slides/papers/paper-22.pdf`.
7. Nicolas T. Courtois and Gregory V. Bard, *Algebraic and Slide Attacks on KeeLoq*, Cryptology ePrint Archive, Report 2007/062, 8 May 2007, `http://eprint.iacr.org/2007/062/`.
8. Nicolas T. Courtois, personal communication, 31 May 2007.
9. Nicolas T. Courtois, Gregory V. Bard and David Wagner, *Algebraic and Slide Attacks on KeeLoq*, Proceedings of Fast Software Encryption 2008, Lecture Notes in Computer Science, Springer-Verlag, to appear.
10. Soichi Furuya, *Slide Attacks with a Known-Plaintext Cryptanalysis*, Proceedings of Information and Communication Security 2001, Lecture Notes in Computer Science 2288, pp. 214–225, Springer-Verlag, 2002.
11. Martin E. Hellman, *A Cryptanalytic Time-Memory Trade-Off*, IEEE Transactions on Information Theory, vol. 26, pp. 401–406, 1980.
12. Sandeep Kumar, Christof Paar, Jan Pelzl, Gerd Pfeiffer, Manfred Schimmler, *Breaking Ciphers with COPACOBANA — A Cost-Optimized Parallel Code Breaker*, Proceedings of Cryptographic Hardware and Embedded Systems 2006, Lecture Notes in Computer Science 4249, pp. 101–118, Springer, 2006.
13. Microchip Technology Inc. *KeeLoq® Authentication Products*, `http://www.microchip.com/keeloq/`
14. Microchip Technology Inc., *HCS410 KeeLoq® Code Hopping Encoder and Transponder Data Sheet*, `http://ww1.microchip.com/downloads/en/DeviceDoc/40158e.pdf`
15. Microchip Technology Inc., *AN642: Code Hopping Decoder using a PIC16C56*, `http://www.keeloq.boom.ru/decryption.pdf`
16. Microchip Technology Inc., *TB001: Secure Learning RKE Systems using KeeLoq Encoders*, `http://ww1.microchip.com/downloads/en/AppNotes/91000a.pdf`
17. Wikipedia, *KeeLoq* , `http://en.wikipedia.org/wiki/KeeLoq` , August 2007.

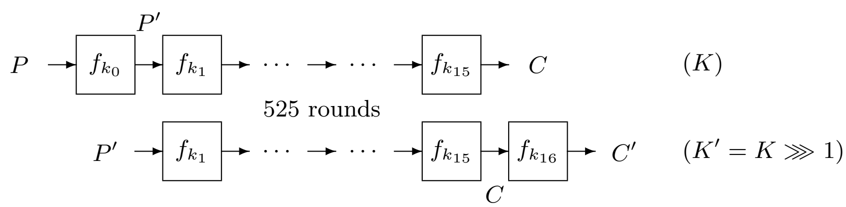

Fig. 5. A related-key attack using keys related by rotation.

A Related-Key Attacks on KeeLoq

Related-key attacks [1] exploit the relations between the encryption processes under different but related keys.

In this appendix we present two related-key attacks on KeeLoq. The first attack is a very efficient attack using pairs of keys related by rotation. The second attack is an improvement of the attack presented in Sect. 3.3 using pairs of keys related by flipping the least significant bit of the key.

A.1 A Related-Key Attack Using Keys Related by Rotation

The first attack exploits the extremely simple way in which the key is mixed into the state during encryption.

Denote a full encryption of a plaintext P by KeeLoq with the key K by $E_K(P)$, and encryption through a single round with the subkey bit k by $f_k(P)$. Consider a pair (K, K') of related-keys, such that $K' = (K \ggg 1)$. If for a pair (P, P') of plaintexts we have $P' = f_{k_0}(P)$, where k_0 is the LSB of K, then $E_{K'}(P') = f_{k_{16}}(E_K(P))$. Indeed, in this case the encryption of P' under the key K' is equal to the encryption of P under K shifted by one round (see Fig. 5). This property, which is clearly easy to check, can be used to retrieve two bits of the secret key K.

Consider a plaintext P. We note that there are only two possible values of $f_{k_0}(P)$, i.e., $1||(P \gg 1)$ and $0||(P \gg 1)$. Hence, we ask for the encryption of P under the key K and for the encryption of the two plaintexts $P'_0 = 0||(P \gg 1)$ and $P'_1 = 1||(P \gg 1)$ under the related-key K', and check whether the ciphertexts satisfy the relation $E_{K'}(P') = f_{k_{16}}(E_K(P))$. This check is immediate, since $E_K(P)$ and $f_{k_{16}}(E_K(P))$ have 31 bits in common. Exactly one of the candidates (P'_0 or P'_1) is expected to satisfy the relation. This pair satisfies also the relation $P' = f_{k_0}(P)$.

At this stage, since P' and P are known, we can infer the value of k_0 immediately from the update rule of KeeLoq, using the relation $P' = f_{k_0}(P)$. Similarly, we can retrieve the value of k_{16} from the relation $E_{K'}(P') = f_{k_{16}}(E_K(P))$. Hence, using only three chosen plaintexts encrypted under two related-keys, we can retrieve two key bits with a negligible time complexity.

In order to retrieve additional key bits, we repeat the procedure described above with the pair of related-keys $(K', K'' = (K' \ggg 1))$ and one of the plaintexts P'_0 or P'_1 examined in the first stage. As a result, we require the encryption of two additional chosen plaintexts (under the key K''), and get two additional key bits: k'_0 and k'_{16}, which are equal to k_1 and k_{17}.

We can repeat this procedure 16 times to get bits $k_0, \ldots, k_{31}$ of the secret key. Then, the procedure can be repeated with the 16 related keys of the form $(K \ggg 32), (K \ggg 33), \ldots, (K \ggg 47)$ to retrieve the remaining 32 key bits. The attack then requires 66 plaintexts encrypted under 34 related keys (two plaintexts under each of 32 keys, and a single plaintext under the two remaining keys), and a negligible time complexity.

An option to reduce the required amount of plaintexts and related keys in exchange for a higher time complexity, is to switch to an exhaustive key search after a suitable number of key bits has been determined. For example, if 32 key bits remain to be found, a brute force search can be conducted in several hours on a PC, or even much less on FPGAs.

Another variant of the attack, requiring less related-keys, is the following. Denote the encryption of a plaintext P through r rounds of KeeLoq with the key $k = (k_0, \ldots, k_{r-1})$ by $f_k^r(P)$. Consider a pair of related-keys of the form $(K, K' = K \ggg r)$. If a pair of plaintexts (P, P') satisfies $P' = f_k^r(P)$, then the corresponding ciphertexts satisfy $E_{K'}(P') = f_{k'}^r(E_K(P))$, where $k' = (k_{16}, \ldots, k_{16+r-1})$. Since $E_K(P)$ and $f_{k'}^r(E_K(P))$ have $32 - r$ bits in common, this property is easy to check.

However, when $r > 1$, the task of detecting P' such that $P' = f_k^r(P)$ is not so easy. Actually, there are 2^r candidates for P', and hence during the attack we have to check 2^r candidate pairs. On the other hand, we can reduce the data complexity of this stage of the attack to $2^{1+r/2}$ by using structures: The first structure S_1 consists of $2^{r/2}$ plaintexts, such that the $32 - r$ least significant bits are equal to some constant C in all the plaintexts of the structure, and the other bits are arbitrary. The second structure S_2 also consists of $2^{r/2}$ plaintexts, such that the $32 - r$ most significant bits are equal to the same constant C in all the plaintexts of the structure, and the other bits are arbitrary. By birthday paradox arguments on the 2^r possible pairs (P, P') such that $P \in S_1$ and $P' \in S_2$ we expect one pair for which $P' = f_k^r(P)$, and this pair can be used for the attack.

In the attack, we go over the 2^r possible pairs and check whether the colliding bits of the relation $E_{K'}(P') = f_{k'}^r(E_K(P))$ are satisfied. If $r \leq 16$, this check discards immediately most of the wrong pairs. After finding the right pair, $2r$ bits of the key can be found using the algorithm presented in Sect. 3.2.

By choosing different values of r, we can get several variants of the attack:

1. Using $r = 16$, we can recover 32 key bits, and then the rest of the key can be recovered using exhaustive key search. The data complexity of the attack is 512 chosen plaintexts encrypted under two related-keys (256 plaintexts under each key), and the time complexity is 2^{32} KeeLoq encryptions.
2. Using $r = 8$ twice (for the pairs $(K, K \ggg 8)$, and $(K \ggg 8, K \ggg 16)$) we retrieve 32 key bits, and exhaustively search the remaining bits. The data complexity of the attack is 64 chosen plaintexts encrypted under three related-keys (16 plaintexts under two keys, and 32 plaintexts under the third key), and the time complexity is 2^{32} KeeLoq encryptions.

3. Using $r = 8$ four times (for the pairs $(K, K \ggg 8)$, $(K \ggg 8, K \ggg 16)$, $(K \ggg 32, K \ggg 40)$, and $(K \ggg 40, K \ggg 48)$) we can retrieve the full key. The data complexity of the attack is 128 chosen plaintexts encrypted under six related-keys (16 plaintexts under four keys, and 32 plaintexts under two keys), and the time complexity is negligible.

Other variants are also possible, and provide a trade-off between the number of chosen plaintexts and the number of related-keys.

A.2 Improved Slide/Meet-in-the-Middle Attack Using Related-Keys

Using a related-key approach, we can improve the attack presented in Sect. 3.3. Denote the encryption of a plaintext P through 64 rounds of KeeLoq under the key K by $g_K(P)$. Denote by e_0 the least significant bit of a word. We observe that if two related-keys (K, K') satisfy $K' = K \oplus e_0$, i.e., they differ in the least significant bit, and two plaintexts (P, P') satisfy $P' = P \oplus e_0$, then we have $g_K(P) = g_{K'}(P')$. Indeed, in the first round of encryption the key difference and the data difference cancel each other. As a result, after the first round the intermediate values in both encryptions are equal, and the key difference is not mixed into the data until the 65-th round. Thus, the intermediate values after 64 rounds are equal in both encryptions.

Now, recall that in Sect. 3.1, the pair (P_i, P_j) is called a slid pair if it satisfies $P_j = g_K(P_i)$. The attack searches among 2^{32} candidates for a slid pair, and then the key can be easily retrieved. Note that by the observation above, if (P_i, P_j) is a slid pair with respect to K, then the pair $(P_i \oplus e_0, P_j)$ is a slid pair with respect to $K' = K \oplus e_0$, and thus $E_{K'}(P_j) = g_{(K' \ggg 16)}(E_{K'}(P_i \oplus e_0))$. This additional slid pair can be used to improve the check of candidate slid pairs, and thus to reduce the time complexity of the attack.

More in detail, (10) can be rewritten as

$$2^{16}\left(48 \cdot 2^{16} + 2^{t_o}\left(3t_c \cdot 2^{16+t_c-t_o} + 2^{16+t_p-t_o}\left(3t_p + N_{\text{coll}} \cdot V\right)\right)\right) \ . \tag{15}$$

The expected number of collisions becomes $N_{\text{coll}} = 2^{16+t_c-t_o}/2^{2t_o}$. Verifying a collision now costs on average V KeeLoq rounds, where

$$V = \sum_{i=0}^{47-t_p-t_c}\left(2 \cdot 2^{-2i} + 2^{-2i-1}\right) + 2^{2t_p+2t_c-96} \cdot \left(528 + 528 \cdot 2^{-32}\right) \ . \tag{16}$$

Simplification yields:

$$48 \cdot 2^{32} + 3t_c \cdot 2^{32+t_c} + 3t_p \cdot 2^{32+t_p} + 3.33 \cdot 2^{96-2t_p-2t_c} + 528 \cdot 2^{32} \ . \tag{17}$$

The optimum is situated at $t_p = t_c = 12$ where the time complexity of the attack is $2^{50.9}$ KeeLoq rounds, or $2^{41.9}$ full KeeLoq encryptions.

Summarising the attack, the data complexity is 2^{17} chosen plaintexts encrypted under two related-keys (2^{16} plaintexts under each key), and the time complexity is $2^{41.9}$ KeeLoq encryptions. The memory complexity is about 16 MB.