Practical attack on 8 rounds of the lightweight block cipher KLEIN

Jean-Philippe Aumasson[1], María Naya-Plasencia[2,*], and Markku-Juhani O. Saarinen[3]

[1] NAGRA, Switzerland
[2] University of Versailles, France
[3] Revere Security, USA

Abstract. KLEIN is a family of lightweight block ciphers presented at RFIDSec 2011 that combines a 4-bit Sbox with Rijndael's byte-oriented MixColumn. This approach allows compact implementations of KLEIN in both low-end software and hardware. This paper shows that interactions between those two components lead to the existence of differentials of unexpectedly high probability: using an iterative collection of differential characteristics and neutral bits in plaintexts, we find conforming pairs for four rounds with amortized cost below 2^{12} encryptions, whereas at least 2^{30} was expected by the preliminary analysis of KLEIN. We exploit this observation by constructing practical ($\approx 2^{35}$-encryption), experimentally verified, chosen-plaintext key-recovery attacks on up to 8 rounds of KLEIN-64—the instance of KLEIN with 64-bit keys and 12 rounds.

Keywords: block ciphers, cryptanalysis, lightweight cryptography

1 Introduction

Lightweight cryptography is concerned with the design, analysis, and implementation of cryptographic schemes—such as stream ciphers or authentication protocols—that minimize the consumption of resources, mainly ROM and RAM in software, and power, energy, and area in hardware. Research in lightweight cryptography is motivated by the growing number of low-resource computing devices such as RFID tags, network sensors, or low-end embedded software systems (for example, smartphones, digital cameras, portable GPS devices). The field has gained interest these last years with a multitude of new lightweight primitives, including the block ciphers PRESENT (CHES'08) [1], KATAN and KTANTAN (CHES'09) [2], PRINTcipher (CHES'10) [3], Hummingbird-2 (RFIDsec'11) [4], LED [5], Piccolo (CHES'11) [6] and the hash functions QUARK (CHES'10) [7], PHOTON (CRYPTO'11) [8], and SPONGENT (CHES'11) [9]. Most of those

*Supported by the National Competence Center in Research on Mobile Information and Communication Systems (NCCR-MICS), a center of the Swiss National Science Foundation under grant number 5005-67322 and by the French Agence Nationale de la Recherche through the SAPHIR2 project under Contract ANR-08-VERS-014

designs are hardware-oriented, and minimize area either using a combination of a small Sbox and a simplistic linear layer, or using a shift-register-based construction.

KLEIN is a new family of lightweight block ciphers, presented at RFIDSec 2011 by Gong, Nikova, and Law [10]. The instances KLEIN-64, KLEIN-80, and KLEIN-96 process 64-bit data blocks and respectively make 12, 16, and 20 rounds and accept keys of 64, 80, and 96 bits. Thanks to an involutive 4-bit Sbox and Rijndael's MixColumn, the KLEIN ciphers allow compact and low-memory implementations in low-end software and hardware. For example, on an Iris sensor node based on an 8-bit AVR microcontroller (ATmega128L), any of the KLEIN instances can be implemented with 97 bytes of RAM and approximately 4 KB or ROM. In 180 nm ASIC, approximately 2000 GE are needed for an implementation with 64-bit datapath.

The preliminary security analysis of KLEIN includes lower bounds on the number of active Sboxes in a differential characteristic. Namely, it is shown that any 4-round characteristic has at least 15 active Sboxes, which implies a probability below 2^{-90} for any characteristic of KLEIN-64. To the best of our knowledge, the best attack reported on KLEIN-64 is a key-recovery integral attack on five rounds with complexity 2^{48} [11, §4.2.1].

Contribution. We propose a refined differential analysis of KLEIN, showing that collections of iterative differential characteristics can be used to bypass the bound proven in the preliminary analysis. We exploit this observation by presenting practical chosen-plaintext key-recovery attacks on up to 8 rounds of KLEIN-64. Our results have been confirmed—and even refined—experimentally.

§2 starts with a brief description of KLEIN-64. Then §3 presents a high-probability differential, which is exploited in §4, first by constructing distinguishers, and then by building key-recovery attacks on top of the distinguisers. We conclude in §5.

2 Brief description of KLEIN

Our description differs in representation from that in [10], but is functionally equivalent. This is done to make the operation of some of our attacks more apparent.

Table 1. The Sbox used by KLEIN. This Sbox is an involution.

x	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
$S[x]$	7	4	a	9	1	f	b	0	c	3	2	6	8	e	d	5

KLEIN is built from an involutive 4-bit Sbox (given in Table 1) and operations in GF(2^8). The field representation is defined by the irreducible polynomial

$x^8 + x^4 + x^3 + x + 1$ (as in Rijndael). During encryption, one can decompose the field operations to XOR and multiplication by 2. We write this multiplication operation algorithmically as a left shift with a conditional XOR:

$$L(x) = \begin{cases} (x \ll 1) & \text{if } x \wedge \texttt{80} = \texttt{00}, \\ (x \ll 1) \oplus \texttt{1b} & \text{if } x \wedge \texttt{80} = \texttt{80}. \end{cases}$$

A KLEIN round is composed of the following steps:

1. AddRoundKey, which XORs a round key to the 64-bit state.
2. SubNibbles, which applies the 4-bit Sbox to each nibble.
3. RotateNibbles, which left-rotates the state of 16 bits.
4. MixNibbles, which applies two MixColumn's in parallel.

The last round has an additional AddRoundKey operation after MixNibbles. Note that in the last round, MixNibbles is not omitted, unlike MixColumns in Rijndael.

Algorithm 1 KLEIN-64 encryption given 64-bit plaintext $P_0, \ldots, P_7$

1:	**for** $i = 0$ to 7 **do**	
2:	$V_i = P_i$	*Copy the plaintext to the state vector*
3:	**end for**	
4:	**for** $r = 0$ to 11 **do**	
5:	**for** $i = 0$ to 7 **do**	
6:	$V_i = V_i \oplus K_i^{(r)}$	AddRoundKey
7:	**end for**	
8:	**for** $i = 0$ to 7 **do**	
9:	$V_{i,0} = S[V_{i,0}]$	SubNibbles *(lower nibbles)*
10:	$V_{i,1} = S[V_{i,1}]$	SubNibbles *(higher nibbles)*
11:	**end for**	
12:	**for** $i = 0$ to 7 **do**	
13:	$T_{(i+6) \bmod 8} = V_i$	RotateNibbles
14:	**end for**	
15:	$V_0 = L(T_0 \oplus T_1) \oplus T_1 \oplus T_2 \oplus T_3$	MixNibbles *lower half*
16:	$V_1 = L(T_1 \oplus T_2) \oplus T_0 \oplus T_2 \oplus T_3$	
17:	$V_2 = L(T_2 \oplus T_3) \oplus T_0 \oplus T_1 \oplus T_3$	
18:	$V_3 = L(T_3 \oplus T_0) \oplus T_0 \oplus T_1 \oplus T_2$	
19:	$V_4 = L(T_4 \oplus T_5) \oplus T_5 \oplus T_6 \oplus T_7$	MixNibbles *higher half*
20:	$V_5 = L(T_5 \oplus T_6) \oplus T_4 \oplus T_6 \oplus T_7$	
21:	$V_6 = L(T_6 \oplus T_7) \oplus T_4 \oplus T_5 \oplus T_7$	
22:	$V_7 = L(T_7 \oplus T_4) \oplus T_4 \oplus T_5 \oplus T_6$	
23:	**end for**	
24:	**for** $i = 0$ to 7 **do**	
25:	$C_i = V_i \oplus K_i^{(n)}$	*Final half-round and copy to ciphertext*
26:	**end for**	

Algorithm 1 details the KLEIN-64 encryption. We index vectors as V_i where $V_{i,0}$ and $V_{i,1}$ are the low and high nibbles of byte i. Algorithm 2 describes the

KLEIN key setup for the three possible key lengths: 64, 80, and 96 bits. Observe that the key setup has the following properties

- The higher and lower nibbles are not mixed at all.
- The round counter has no effect on the higher nibbles with a 64-bit key and only during last round with a 80-bit key.
- If the higher nibbles are all 0 or 7, the higher nibbles will stay as 0 or 7 throughout the key setup.
- A higher-nibble fixed point for the 64-bit key setup is

$$7000007070700000 \mapsto 7000007070700000 \ .$$

Algorithm 2 KLEIN-64 key setup given 64-bit key K.

1: $K^{(0)} = K$ *The first round key*
2: **for** $r = 1$ to 12 **do**
3: **for** $i = 0$ to 4 **do**
4: $K_i^{(r)} = K_{((i+1) \bmod 4)+4}^{(r-1)}$
5: $K_{i+m}^{(r)} = K_{((i+1) \bmod 4)+4}^{(r-1)} \oplus K_{(i+1) \bmod 4}^{(r-1)}$
6: **end for**
7: $K_2^{(r)} = K_2^{(r)} \oplus r$ *Round constant*
8: $K_{5,0}^{(r)} = S[K_{5,0}^{(r)}]$ *Nonlinear mixing*
9: $K_{5,1}^{(r)} = S[K_{5,1}^{(r)}]$
10: $K_{6,0}^{(r)} = S[K_{6,0}^{(r)}]$
11: $K_{6,1}^{(r)} = S[K_{6,1}^{(r)}]$
12: **end for**

3 A collection of differential characteristics

Our attack exploits a collection of iterative differential characteristics that have a same input difference and output differences in a specific 32-bit subspace (i.e. it is iterative). Below we first analyze the probability to follow one of those characteristics, which we successfully verified experimentally.

3.1 Observations

We first report four important observations that will allow us to identify high-probability differentials for KLEIN. We refer to MixColumn as the function executed twice within MixNibbles.

Observation 1. If the difference entering MixColumn is of the form 0000000X where X represents a non-zero difference in $\{1, \ldots, 7\}$—i.e. a nibble with null MSB—then the output difference is of the form 0Y0Y0Y0Y, where the wildcard Y represents a non-zero difference. That is, higher nibbles remain free of difference.

Observation 2. If the difference entering MixColumn is of the form 0X0X0X0X where the wildcard X represents a difference in $\{0, \ldots, 7\}$, then the output difference is of the form 0Y0Y0Y0Y, where Y represents a possibly null difference. Furthermore, the average number of non-zero Y's is 3.75, as one can experimentally verify. For example, the input difference 04020405 leads to the output difference 0f090100.

Observation 3. If the difference entering MixColumn is of the form 0X0X0X0X where the wildcard X represents a difference in $\{8, \ldots, f\}$, then the output difference is of the form 0Y0Y0Y0Y, where Y represents a (possibly zero) difference. Furthermore, the average number of non-zero Y's is 3.75. Note that, contrary to Observation 2, an X cannot be zero. For example, the input difference 0c0a080f leads to the output difference 010f0708.

Observation 4. Given a random difference, KLEIN's Sbox returns a difference in $\{1, \ldots, 7\}$ with probability $7/15 \approx 2^{-1.1}$, for a random input. If the difference is b or e, the probability is $3/4 \approx 2^{-0.42}$. These values can be verified either experimentally or using the difference distribution table in [11].

3.2 The collection of characteristics

Our attack exploits a truncated differential defined as a collection of (iterative) characteristics. That is, we not only set conditions on the intput/output differences, but also on the path followed to reach them.

Definition. To best exploit the first two observations, our collection of characteristics is such that higher nibbles remain inactive. A sufficient condition is that after SubNibbles the first four lower have differences either all in $\{0, \ldots, 7\}$, or all in $\{8, \ldots, f\}$. A similar condition is imposed on the last four lower nibbles. Fig. 1 gives a representation of the collection of characteristics. Note that the collection of characteristics is iterative, as it has the same conditions on the input as on the output at any round.

To maximize the probability at the first round, we choose an input difference b. At the first round, we thus have one active Sbox, and one active MixColumn. At the second round we always have four active Sboxes, and two active MixColumn's. Then we enter a state where all lower nibbles are active with high probability.

Probability analysis. We estimate the probability of our truncated differential as a collection of characteristics.

At the first round, it is sufficient that the input difference B gives a difference in $\{0, \ldots, 7\}$ after the Sbox. This occurs with probability 3/4, thus $p_1 \approx 2^{-0.42}$.

At the second round, there are four active nibbles entering SubNibbles. RotateNibbles propagates the four active nibbles to MixNibbles, wherein two lower nibbles are inactive in each half. Thus, the differences after SubNibbles in the

second round must be in $\{0,\ldots,7\}$. Since such a difference is reached with probability 7/15, we have $p_2 = (7/15)^4 \approx 2^{-4.40}$.

At the third round, there are on average 3.75 active nibbles coming from each MixColumn of the second round, and all four lower nibbles are active with probability 15/16. This is necessary to obtain four active nibbles with differences in $\{8,\ldots,f\}$ after SubNibbles, but not to obtain four active nibbles with differences in $\{0,\ldots,7\}$. The probability to obtain one the desired sets of differences in one half of the state is

$$\left(\frac{7}{15}\right)^{3.75} + \left(\frac{15}{16}\right) \times \left(\frac{8}{15}\right)^{4} .$$

Recall that a difference in $\{0,\ldots,7\}$ is reached with probability 7/15, and one in $\{8,\ldots,f\}$ with probability 8/15. As the halves of MixNibbles behave similarly,

$$p_3 = \left(\left(\frac{7}{15}\right)^{3.75} + \left(\frac{15}{16}\right)\left(\frac{8}{15}\right)^{4}\right)^{2} \approx 2^{-5.82} .$$

The differential is iterated through subsequent rounds, hence we have

$$p_4 = p_5 = p_6 = p_3 .$$

Note that, for the sake of simplicity, we do not consider the case when two inactive boxes occur in the same MixColumn, as this has a negligible impact on the probability obtained. Fig. 1 shows the cumulative probabilities for a sequence of 1 to 7 rounds of KLEIN.

3.3 Comparison with the lower bounds

The KLEIN paper [10] proves that any 4-round characteristic activates at least 15 Sboxes and has probability at most 2^{-30}. For comparison, our differential on 4 rounds has probability $2^{-16.45}$, yet it activates 21 Sboxes. Our result thus does not contradict the 2^{-30} bound, for it considers a collection of characteristics rather than a single one. Indeed, as argued in [10, §§3.1], "*the strength of a cipher against differential attacks is reflected by the maximum probability of a differential, i.e. a collection of characteristics*". However, the assumption that [10, §§3.1] *"one characteristic has a much larger probability than the other characteristics of the differential"* is proven wrong by our observations. That is, the maximal probability of a characteristic cannot be taken as an accurate estimate of the maximal probability of a differential, as the above assumption would imply.

4 Attacking KLEIN

This section shows how to exploit the differential described in §3 to attack reduced version of KLEIN-64. We start with the observation that *neutral bits* can be used to reduce the cost of finding values conforming to the differential.

Fig. 1. Representation of the collection of differential characteristics, where a square represents a nibble (white means inactive, black means possibly active). The two rightmost columns respectively show the round's probability and the cumulative probability of obtaining the differential.

1	SubNibbles RotateNibbles MixNibbles	□□ □□ □■ □□ □■ □□ □□ □□ □■ □■ □■ □■	□□ □□ □□ □□ □□ □□ □□ □□ □□ □□ □□ □□	$p_1 \approx 2^{-0.42}$	$2^{-0.42}$
2	SubNibbles RotateNibbles MixNibbles	□■ □■ □■ □■ □■ □■ □□ □□ □■ □■ □■ □■	□□ □□ □□ □□ □□ □□ □■ □■ □■ □■ □■ □■	$p_2 \approx 2^{-4.40}$	$2^{-4.82}$
3	SubNibbles RotateNibbles MixNibbles	□■ □■ □■ □■ □■ □■ □■ □■ □■ □■ □■ □■	□■ □■ □■ □■ □■ □■ □■ □■ □■ □■ □■ □■	$p_3 \approx 2^{-5.82}$	$2^{-10.64}$
4	SubNibbles RotateNibbles MixNibbles	□■ □■ □■ □■ □■ □■ □■ □■ □■ □■ □■ □■	□■ □■ □■ □■ □■ □■ □■ □■ □■ □■ □■ □■	$p_4 \approx 2^{-5.82}$	$2^{-16.45}$
5	SubNibbles RotateNibbles MixNibbles	□■ □■ □■ □■ □■ □■ □■ □■ □■ □■ □■ □■	□■ □■ □■ □■ □■ □■ □■ □■ □■ □■ □■ □■	$p_5 \approx 2^{-5.82}$	$2^{-22.27}$
6	SubNibbles RotateNibbles MixNibbles	□■ □■ □■ □■ □■ □■ □■ □■ □■ □■ □■ □■	□■ □■ □■ □■ □■ □■ □■ □■ □■ □■ □■ □■	$p_6 \approx 2^{-5.82}$	$2^{-28.08}$
7	SubNibbles RotateNibbles MixNibbles	□■ □■ □■ □■ □■ □■ □■ □■ □■ □■ □■ □■	□■ □■ □■ □■ □■ □■ □■ □■ □■ □■ □■ □■	$p_7 \approx 2^{-5.82}$	$2^{-33.90}$

4.1 Finding and exploiting neutral bits

The term *neutral bit* was introduced by Biham and Chen [12] in the context of SHA0 cryptanalysis. In the input of some cryptographic function, a bit is said to be *neutral* with respect to a given differential (characteristic) when flipping this bit in an input conforming to the differential (characteristic) leads to a new input also conforming to that differential. Biham and Chen actually used sets of neutral bits (called k-neutral sets). A similar technique has been used in the context of block cipher cryptanalysis, for example in Biham et al.'s analysis of Skipjack [13].

In KLEIN, one can observe that the first two and last two input bytes in a plaintext block are neutral with respect to the first two rounds' collection of characteristics. Indeed,

- in the first round, those bytes first pass through the Sbox, then after RotateNibbles they form the 4-byte half of the state that is inactive in MixNibbles
- in the second round, our neutral bytes first pass through the Sbox—still independently of the other bytes and of the difference—then they are mixed with the other bytes within MixNibbles. However, since the conformance of the output of MixNibbles only depends on the active nibbles, our bytes remain neutral up to this stage.
- in the third round, values entering the Sbox depend on the first and last two input bytes; these are thus not neutral for the third round.

Therefore, given a pair of inputs satisfying the truncated differential, 2^{32} pairs conforming to the first two rounds can be obtained by varying the first two and last two input bytes. In other words, the conformance to the differential is independent from the values of those bytes.

It follows for example that after a 2^{28} effort to find a pair satisfying the 6-round differential, one can derive 2^{32} pairs for which the full differential is followed with probability $2^{-28.06+4.80} = 2^{-23.26}$. A new conforming pair can thus be found with effort $\approx 2^{23}$, instead of 2^{28} without exploiting neutral bits. With an extra effort of 2^{27} encryptions—which leaves the total complexity below 2^{29}—one expects to find 8 other conforming pairs.

4.2 Distinguisher for 7 rounds

Based on our 6-round differential characteristic, we construct a distinguisher for 7-round KLEIN-64. Our main observation is that for a pair conforming to the 6-round differential, the SubNibbles of round 7 has all higher nibbles inactive. Although MixNibbles may activate arbitrary nibbles at round 7, one can determine the differences before MixNibbles given only the output after 7 rounds, thanks to the linearity of MixNibbles. In other words, one can check that only lower nibbles were active after SubNibbles. A conforming pair is expected to be detected after 2^{28} observations, against 2^{32} ideally, which constitutes the distinguisher.

The distinguisher is actually more powerful: once a conforming pair is found in 2^{28}, one can produce approximately 8 other pairs with negligible extra cost, as explained in §§4.1.

4.3 Distinguisher for 8 rounds

The distinguisher for 7 rounds consisted in finding one (and possibly many) conforming pairs at a lower cost than for an ideal cipher. For 8 rounds, the distinguisher consists in finding several pairs (rather than one) with reduced complexity.

First, one collects approximately $2^{33.90}$ pairs, and records the ones that conform to the output difference as per the collection of characteristics in §3. One expects to record approximately 4 pairs satisfying the difference by chance, and one conforming to the collection of characteristics. Observe that the conforming pair can be identified using the neutral bits, as it is the only pair for which neutral bits will lead to an additional conforming pairs in approximately $2^{33.90-4.80} = 2^{29.10}$ trials.

It follows that by testing 2^{32} derived pairs for each of the (say) 5 pairs obtained initially, one new conforming pair is expected for the pairs obtained by chance, and about 8 new pairs for the one conforming to the characteristics. Therefore, with about $2^{33.90} + 4 \times 2^{32} \approx 2^{35}$, one expects to find twice more conforming pairs than ideally (16 vs. 8).

4.4 Key-recovery for 7 rounds

Our key-recovery attack for 7 rounds starts by using the distinguisher of §§4.2 to detect a pair satisfying the 6-round differential. Then, we exploit the invertibility of the final MixNibbles and RotateNibbles to determine the output differences of each nibble after the last SubNibbles (i.e. that of the seventh round. These differences should be null for all higher nibbles.

Then, the attack tries values of the lower nibbles (i.e., linear combinations of key bits) and pass them through the Sbox; the difference obtained is inverted through MixColumn; if the difference obtained has only lower nibbles active, then the guess is considered as possible.

Since the inverse MixNibbles produces lower only active nibbles given lower only active nibble with probability 2^{-3}, we can reduce the search space from 2^{16} to $2^{16}/2^3 = 2^{13}$ for each of the two MixColumn instances. Overall, this reduces the cost of key-recovery to 2^{58} trials. The attack always succeeds, as all candidate keys are tried within the 2^{58} trials.

The attack can be improved by using several conforming pairs. Using neutral bits, one can generate 8 more pairs in 2^{27}. It is expected that 6 are sufficient to identify the correct combination of key bits, by taking the intersection of the 2^{13}-element sets determined for each conforming pair. One can thus recover the 32 bits corresponding to the XOR between the lower nibbles of the ciphertexts, and those after the last SubNibbles. Since these bits are a linear combination of the key bits, it is equivalent to recovering 32 key bits (due to the linear independency of the 32 equations). The 32 key bits left can then be bruteforced in 2^{32}.

The attack thus recovers the complete 64-bit key with fewer than 2^{33} encryptions.

4.5 Key-recovery for 8 rounds

One can extend the strategy of the 7-round attack to 8 rounds, using the trick mentioned in §§4.3 to detect the pair conforming to the collection of characteristics when "false alarms" (i.e. values conforming to the input/output differential but not necessarily to the collection of characteristics). Within fewer than 2^{34} encryptions, one thus identifies a conforming pair with high probability.

Using neutral bits, one expects to produce approximately 8 other conforming pairs after 2^{32} trials. This is more than enough to identify with certainty 32 bits of the last subkey, as done in the attack on 7 rounds. Overall, the 64 bits of the last subkey (and thus of the original key) can be found with complexity below 2^{35} encryptions.

4.6 Experimental verification

We experimentally verified the correctness of the probablities reported in §3 as well as the correctness of the distinguishers and key recovery attacks claimed. Namely, we implemented the chosen-plaintext attack that aims to recover the lower nibbles of the last subkey. As exact complexities cannot be fully confirmed experimentally, we just checked that the order of magnitude was consistent with the expected complexities.

Since no reference code of KLEIN is published by its designers, we wrote our own reference C implementation of KLEIN-64 and made sure it matched the test vectors provided in [11]. A reference implementation of KLEIN is very easily written: it took us less than one hour to implement KLEIN-64, by reusing available code of MixColumn to implement MixNibbles, and implementing SubNibbles with a look-up table.

Based on our reference KLEIN-64 code, we implemented the method that recovers the combinations of subkey bits that are XORed with the higher nibbles of the state after the last SubNibbles: a first conforming pair is found by bruteforce, then neutral bits are used to find five more conforming pairs. Below we copy examples of outputs of our program for the attack on 6 and 7 rounds, reporting the number of trials done for finding each of the pairs, the value found—to make sure that all are distinct—as well as the total time of the attack:

```
$ ./attack 6
test vector ok
soundness ok
Pair found in 2^22.85: c093c2304ac8b7ca
Pair found in 2^18.41: ccc0c2304ac855a6
Pair found in 2^15.81: b4efc2304ac8fa2d
Pair found in 2^17.42: bbddc2304ac81c53
Pair found in 2^14.72: 9b53c2304ac8bdd2
Pair found in 2^19.26: 40c9c2304ac86349
Subkey lower nibbles recovered:
745a
     a1ba
Actual subkey lower nibbles:
```

```
745a a1ba
10 seconds elapsed


$ ./attack 7
test vector ok
soundness ok
Pair found in 2^27.29: 0e45d5ed12117e30
Pair found in 2^23.78: dd50d5ed12114908
Pair found in 2^23.74: d78dd5ed12112a02
Pair found in 2^24.45: fdb7d5ed1211f745
Pair found in 2^22.69: a4e3d5ed121123bc
Pair found in 2^15.82: 6286d5ed12116f2c
Subkey lower nibbles recovered:
1bda
     5d7d
Actual subkey lower nibbles:
1bda 5d7d
296 seconds elapsed
```

In the list of pairs found, the first one is found by bruteforce and the subsequent ones are derived using neutral bits, thereby reducing the cost by a factor $2^{4.82}$ on average. We used an Athlon64 X2 Dual Core 4400+. Although our code is slightly optimized (e.g. with SubNibbles implemented as 8-bit look-ups) and `gcc`-compiled with speed-optimization flags (`-O3 -m64 -march=athlon64 -fomit-frame-pointer -funroll-loops`), the attacks can probably be sped up further.

The 8-round attack took much more time to verify than the attack on 7 rounds, due to the possible finding of false alarms. For example, one failed experiment returned after a few hours

```
Pair found in 2^32.27: 0beeadb61e7d4787
Pair found in 2^29.80: 0beeadb61e7d4787
Pair found in 2^31.61: 0beeadb61e7d4787
```

Such results suggest that the first pair found was not conforming to the collection of characteristics, as the tentative use of neutral bits failed to find distinct conforming pairs. Nevertheless, we were able to find conforming pairs for the 8-round attack. We were particularly lucky with the following experiment:

```
$ ./attack 8
test vector ok
soundness ok
Pair found in 2^28.21: fb5248c1a424ca3e
Pair found in 2^26.43: 00b848c1a424882f
Pair found in 2^28.54: 180b48c1a4245a09
Pair found in 2^26.78: 1ee948c1a4246b1d
Pair found in 2^25.81: 226848c1a424362e
Pair found in 2^27.56: 2e3548c1a424f161
```

```
Subkey lower nibbles recovered:
d42c
     d515
Actual subkey lower nibbles:
d42c d515
1344 seconds elapsed
```

The experiments reported above demonstrate that the attacks described in §4 do work and succeed in recovering the key with a complexity that seems to be in line with our analytical estimates.

5 Conclusion

We presented practical, experimentally verified attacks on the lightweight cipher KLEIN-64 reduced to up to 8 rounds, out of 12 in total. Our attack is made possible by a high-probability differential described as a large collection of differential characteristics. Our results suggest that combining a 4-bit Sbox (as used in Serpent) with the byte-oriented MixColumn linear layer (as used in Rijndael/AES) is not an optimal strategy, as far as security is concerned. This work is the first third-party analysis of KLEIN published, to our best knowledge. Future works may seek to extend our attacks to more rounds of KLEIN.

Acknowledgments

We would like to thank the reviewers of INDOCRYPT 2011 for their insightful comments.

References

1. Bogdanov, A., Knudsen, L.R., Leander, G., Paar, C., Poschmann, A., Robshaw, M.J.B., Seurin, Y., Vikkelsoe, C.: PRESENT: An ultra-lightweight block cipher. In: CHES. (2007)
2. Cannière, C.D., Dunkelman, O., Knezevic, M.: KATAN and KTANTAN - a family of small and efficient hardware-oriented block ciphers. In: CHES. (2009)
3. Knudsen, L.R., Leander, G., Poschmann, A., Robshaw, M.J.B.: PRINTcipher: A block cipher for IC-printing. In: CHES. (2010)
4. Engels, D., Saarinen, M.J.O., Smith, E.M.: The Hummingbird-2 lightweight authenticated encryption algorithm. In: RFIDsec. (2011)
5. Guo, J., Peyrin, T., Poschmann, A., Robshaw, M.: The LED block cipher. In: CHES. (2011)
6. Shibutani, K., Isobe, T., Hiwatari, H., Mitsuda, A., Akishita, T., Shirai, T.: Piccolo: An ultra-lightweight blockcipher. In: CHES. (2011)
7. Aumasson, J.P., Henzen, L., Meier, W., Naya-Plasencia, M.: Quark: A lightweight hash. In: CHES. (2010)
8. Guo, J., Peyrin, T., Poschmann, A.: The PHOTON family of lightweight hash functions. In: CRYPTO. (2011)

9. Bogdanov, A., Knezevic, M., Leander, G., Toz, D., Varici, K., Verbauwhede, I.: SPONGENT: A lightweight hash function. In: CHES. (2011)
10. Gong, Z., Nikova, S., Law, Y.W.: KLEIN: a new family of lightweight block ciphers. In: RFIDSec. (2011)
11. Gong, Z., Nikova, S., Law, Y.W.: KLEIN: a new family of lightweight block ciphers. http://doc.utwente.nl/73129/ (2011)
12. Biham, E., Chen, R.: Near-collisions of SHA-0. In: CRYPTO. (2004)
13. Biham, E., Biryukov, A., Dunkelman, O., Richardson, E., Shamir, A.: Initial observations on Skipjack: Cryptanalysis of Skipjack-3XOR. In: SAC. (1998)