Cryptanalysis of KLEIN *

Virginie Lallemand and María Naya-Plasencia

Inria, France

Abstract. Due to the recent emergence of resource-constrained devices, cryptographers are facing the problem of designing dedicated lightweight ciphers. KLEIN is one of the resulting primitives, proposed at RFIDSec in 2011 by Gong et al. This family of software-oriented block ciphers has an innovative structure, as it combines 4-bit Sboxes with the AES MixColumn transformation, and has woken up the attention of cryptanalysts. Several security analyses have been published, in particular on the 64-bit key version. The best of these results could attack up to 10 rounds out of the total number of 12. In this paper we propose a new family of attacks that can cryptanalyze for the first time all the 12 rounds of the complete version of KLEIN-64. Our attacks use truncated differential paths and are based both on some of the notions developed in previous attacks and on our new ideas that allow to considerably improve the performance. To prove the validity of our attacks, we have implemented reduced-round versions of them. In particular we were able to reproduce a practical attack that recovers the whole key on 10 rounds, which also corresponds to the best practical attack against KLEIN-64.

Keywords: KLEIN, lightweight block cipher, truncated differential cryptanalysis, MixColumn, key-recovery.

1 Introduction

Design of lightweight and secure primitives has become one of the major interests of the cryptographic community in order to answer the requirements of a large number of applications, like RFID and wireless sensor networks. Through these last years an enormous amount of promising such primitives has been proposed, like PRESENT [7], LED [12], Spongent [6], ARMADILLO [5], PRINCE [8], PRINTcipher [14], KLEIN [11], LBlock [23] and Twine [22]. Correctly evaluating the security of these proposals has become a primordial task that merits all the attention from the community. This has been proved by the big number of security analyses of the previous primitives that has appeared (to cite a few: [16, 1, 9, 17, 20, 10, 19]).

On the other hand, most of now-a-days cryptanalysis results are analyses of round-reduced versions that do not apply to, nor break the full primitive studied. This trend could be explained by the recent improvements made by

*Partially supported by the French Agence Nationale de la Recherche through the BLOC project under Contract ANR-11-INS-011.

cryptographers in the last few years - partially resulting from competitions like AES, SHA-3 and eStream - that gave them solid bases and criteria to build secure designs. Nevertheless, it is still crucial to make security analyses in order to filter out the candidates that are not secure enough, and narrow down the list of available lightweight primitives to recommend the use of the secure ones.

KLEIN [11] is a lightweight block cipher proposed at RFIDSec2011 by Gong et al. Three versions were proposed for different key sizes: 64 bits, 80 bits and 96 bits, with 12, 16 and 20 rounds respectively. KLEIN is combining the AES `MixColumn` operations with 4-bit Sboxes. Since its publication, several cryptanalysts have been interested in its analysis and some results on round-reduced versions have been published [2–4, 13, 21, 24]. So far, the highest number of attacked rounds was 10 in the 64-bit version (out of 12), 11 out of 16 in the 80-bit version and 13 out of 20 in the 96-bit version. Recently biclique analyses ([3, 2]) appeared, but we can remark that this analyses require to perform an exhaustive search on the whole key and that the acceleration factors are very small.

We propose here a family of attacks that successfully exploits the slow diffusion between higher and lower nibbles in the cipher. They can be applied to the full 12-round KLEIN-64 with a time and data complexities of $2^{57.07}$ and $2^{54.5}$ respectively (other trade-offs are also possible). They apply to 13 and 14 rounds of KLEIN-80 and KLEIN-96 respectively, improving the previous attacks. We have also been able to implement some of our attacks, obtaining a practical key-recovery for 10 rounds of KLEIN-64. We recall that previous practical attacks reached at most 8 rounds. We have also been able to implement 8-round attacks with a considerably lower data complexity than previous ones and a faster execution.

The paper is organized as follows: the next two sections introduce KLEIN family of block ciphers (Section 2) and summarize the results of the security analyses that have been done so far and also recall some of their key ideas that are useful for our analysis (Section 3). Section 4 gives a generic description of our family of attacks and details the technical improvements we found to reduce the complexity. Section 5 is dedicated to the possible time-memory-data complexity trade-offs, and gives 4 examples of it while detailing the best time complexity attack we found on the 12-round version. Section 6 discusses and provides the results of the implementations we made to verify our attacks, and presents the 10-round practical attack on KLEIN-64. The paper ends with a conclusion.

2 Description of KLEIN

KLEIN is a family of lightweight block ciphers presented by Gong, Nikova and Law at RFIDSec2011. By design choice, it is implementation compact and has low-memory needs both in software and hardware, so it is a suitable family for resource-limited devices such as RFID tags and wireless sensors.

KLEIN encryption routine is a Substitution-Permutation Network that operates on 64-bit blocks. Three versions are proposed, denoted KLEIN-64, KLEIN-80 and KLEIN-96 with key-sizes of 64, 80 and 96 bits and 12, 16 and 20 rounds

respectively. Each round is composed of 4 layers: `AddRoundKey`, `SubNibbles`, `RotateNibbles` and `MixNibbles`.

More precisely, the state entering a round is first xored with the round-key through `AddRoundKey`. The result is then divided in 16 4-bit parts or *nibbles* that are all transformed by the same involutive 4×4 Sbox represented in Table 1. KLEIN designers chose this Sbox instead of a byte-wise one to minimize implementation costs and memory needs.

Table 1. KLEIN 4×4 Sbox

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

Then, `RotateNibbles` rotates the state two bytes to the left and finally `MixNibbles` applies Rijndael `MixColumn` transformation to each half of the state. Note that this last step is byte-wise while the others can be seen as nibble-wise. Contrary to `Rijndael`, the step `MixNibbles` is not omitted in the last round. A final whitening key is added at the end of the process so the encryption routine requires one more key than the number of rounds. The round-keys are computed from the MasterKey with the KeySchedule algorithm that follows a Feistel scheme. A complete description of it can be found in [11].

3 Previous Cryptanalysis

Since KLEIN proposal in 2011, several cryptanalysis have been published, mostly on the KLEIN-64 version: this block cipher has been attacked with differential cryptanalysis methods up to 8 rounds [4, 24] and an integral cryptanalysis was also proposed in [24] up to 7 rounds. All of these attacks reached the maximal number of rounds possible using their techniques. In [13] a study of the security of KLEIN is performed, and some ideas are proposed for trying to reach 9 rounds without success. The best previous attack on KLEIN-64 was made by Nikolić et al. in [21] and reached 10 rounds. This attack uses a new technique named the Parallel-Cut Meet-In-The-Middle (PC MITM) that can also apply to KLEIN-80 (up to 11 rounds) and to KLEIN-96 (up to 13 rounds). A summary of these results and of our best new ones is done in Table 2, where their complexities, number of rounds reached and version of the cipher are specified.

Most of the previous attacks took advantage of the almost full independence between higher and lower nibbles: either to build truncated differential paths ([4, 24]) or to split the cipher in two independent subciphers ([21]). More precisely [4, 21, 24] pointed out the fact that all encryption layers, with the exception of `MixNibbles`, are nibble-wise and do not mix higher nibbles with lower nibbles. The two analyses [4, 24] also gave an interesting property of the `MixColumn` structure, namely:

Proposition 1. *[4, 24] If the eight nibbles entering* `MixColumn` *are of the form* $0X0X0X0X$*, where the wild-card* X *represents any 4-bit value, then the output*

Version of KLEIN	Source	Rounds	Data	Time	Memory	Attacks
KLEIN-64	[24]	7	$2^{34.3}$	$2^{45.5}$	2^{32}	integral
	[24]	8	2^{32}	$2^{46.8}$	2^{16}	truncated
	[4]	8	2^{35}	2^{35}	-	differential
	[21]	10	1	2^{62}	2^{60}	PC MITM
	[3]	12	2^{39}	$2^{62.84}$	$2^{4.5}$	biclique
	Sec. 5.1	12	$2^{54.5}$	$2^{57.07}$	2^{16}	differential
KLEIN-80	[24]	8	$2^{34.3}$	$2^{77.5}$	2^{32}	integral
	[21]	11	2	2^{74}	2^{74}	PC MITM
	Sec. 5.6	13	2^{52}	2^{76}	2^{16}	differential
	[2]	16	2^{48}	2^{79}	2^{60}	biclique
KLEIN-96	[21]	13	2	2^{94}	2^{82}	PC MITM
	Sec. 5.6	14	$2^{58.4}$	$2^{89.2}$	2^{16}	differential
	[2]	20	2^{32}	$2^{95.18}$	2^{60}	biclique

Table 2. Previous results and some of our new results on KLEIN.

is of the same form if and only if the MSB of the 4 lower nibbles have all the same value. This case occurs with probability 2^{-3}.

Indeed, the higher nibble from the output of `MixColumn` belonging to the ith byte depends on: a) the 4 higher nibbles from the input and on b) the xor of the MSB of the input lower nibble from the ith byte and the MSB of the input lower nibble from the $i+1$th byte (mod 4). This information can be expressed as three quantities of one bit computed with the MSB of the input lower nibbles. This proposition allows to construct truncated differential paths with important probabilities leading to efficient distinguishers and key-recovery attacks up to 8 rounds, as the ones used in [4, 13, 24].

Also, as the pattern of the difference in the input of `MixColumn` is exactly the same as the one in the output, this truncated difference allows the attacker to build an iterative path. In the following, we denote by *iterative round* a round of KLEIN that goes from a difference with only lower nibbles active ($0X0X0X0X\ \ 0X0X0X0X$) to the same type of difference in the output. In [4] it was claimed that a difference with only lower nibbles active passes through an iterative round with probability $2^{-5.82}$, while in [24] it was computed as 2^{-6}. We will discuss the value of this probability in the next section.

An attentive study of the `KeySchedule` led to the following proposition:

Proposition 2. *[4, 13, 21, 24] In the* `KeySchedule` *algorithm, lower nibbles and higher nibbles are not mixed: the lower nibbles of any round-key can be computed directly from the lower nibbles of the master key. The same property holds for higher nibbles.*

Note that in KLEIN-64 case, since each round key is as long as the master key, the lower nibbles of any round-key can be computed directly from the lower nibbles of any other round-key.

This proposition means that the keySchedule can be seen as two independent and parallel subroutines involving each half of the key bits.

Propositions 1 and 2 are the two main ideas used in [4, 13, 21, 24], that we recycle and improve in our attacks.

4 Generic Description of Our Attacks

The best previous differential cryptanalyses of KLEIN basically exploited the iterative truncated differential path over R rounds for building attacks on $R+1$ rounds. No condition was imposed in the last round, making the whole output active.

Despite this, if we are given two ciphertexts (C, C') with a certain difference Δ_{out}, we can easily check if they verify the difference conditions at the output of round R (i.e., if they form a pair of values that might satisfy the differential path) without needing to guess any key-bits: we just have to apply to the output difference the linear transformation $\texttt{MixNibbles}^{-1}$, and check if the higher nibbles obtained are null, which will occur with a probability of 2^{-32}.

In previous cryptanalysis, since the number of rounds considered was relatively small, this big sieving of probability 2^{-32} appearing in the last round was sufficient to select only the pairs that conform the differential path. The set of possible key-bits involved in the differential path of the last round was then reduced to the ones that verify the conditions of round R, by only keeping the keys that generated higher nibble inputs with zero difference on each `MixColumn` from round R. Since this imposes $3 + 3 = 6$ bit-conditions as seen in Proposition 1, the number of possible key-bits involved is basically reduced by 2^{-6}.

In previous attacks several conforming pairs were produced (for example 6 in the 7-round attack) in order to iterate the filtering step of 2^{-6} and then recover the involved round-key bits. Contrary to those attacks which require to keep only the pairs that follow the differential path in the first step, we allow ourselves to conserve pairs that do not follow the differential path after the first filtering step. The pairs that verify the 32-bit conditions at round $R+1$ but do not satisfy the differential path are called *false alarms*. The idea is then to consider separately each *candidate* set made of one of these pairs and an associated possible part of the key (only the lower nibbles), i.e. a candidate triplet (C, C', k_{low}), and apply to it several filters to decide if the set is valid, that is if the pair is conforming the differential path while the considered part of the key is the correct one. Those several filters consist in using not only the conditions from the `MixNibbles` of round R but also the ones from the other rounds. Our sieves will then consist in checking the conformity of the candidates' differences going through all the successive `MixNibbles` operations, starting from the ciphertexts and inverting each round or starting from the plaintexts and considering the rounds in the forward direction. As we show in Section 4.3, reaching the next

sieve requires to make a guess of 6 information bits. Since the sieve is of 2^{-6} in the case of iterative rounds, the number of remaining candidates is unchanged after the iterative filters, i.e., on average, for each candidate triplet (C, C', k_{low}), only one candidate pair $(S, S')_r$ can be associated per iterative round. We will show how we can reduce the number of candidate triplets by first choosing a differential path with specific non-iterative first rounds, that provide a more powerfull filter, and second by comparing the informations obtained from the plaintexts/ciphertexts and from the filtering process.

The step for recovering the remaining key-bits is usually much cheaper.

In the following, we will explain in detail this attack in a generic way, including some technical improvements that have allowed us to reduce the complexities of the attack, and provide a result on the whole 12 rounds of KLEIN-64, the 64-bit key version, as well as much improved results on the other versions. We will start by studying in detail the properties and equations derived from `MixNibble`, showing how to correctly compute the probabilities of the path, the cost of inverting each round and of guessing the necessary key-bits. We will also provide the generic attack procedure with the corresponding complexities. Further, in Section 5, we present different possible trade-offs of this attack and the concrete results on the KLEIN ciphers, with one detailed example on the full 64-bit version that will also help the comprehension of the attacks.

4.1 `MixNibbles` properties and detailed equations

Let us denote the binary decomposition of the byte a by $(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7)$, where a_0 is the MSB and a_7 the LSB.

Proposition 3. *The values of the lower nibbles outputting* `MixColumn` *depend on the values of the lower nibbles at the input and on 3 quantities computed from the higher nibbles that we will call 3 information bits. More precisely, to compute the lower nibbles resulting of the operation* `MixColumn(a,b,c,d)`*, the 3 information bits are:*

$$\begin{cases} a_0 + b_0 \\ b_0 + c_0 \\ c_0 + d_0 \end{cases}$$

In particular, $a_0 + b_0$ is needed for computing the lower nibble at the first position, $b_0 + c_0$ the one in the second, $c_0 + d_0$ the one in the third and $d_0 + a_0$, which is the sum of the three of them, is needed for computing the lower nibble in the fourth position.

When considering `MixColumn`$^{-1}$`(a,b,c,d)`*, a similar property holds for the computation of the output lower nibbles, which requires the following 3 informa-*

tion bits in addition to the input lower nibbles:

$$\begin{cases} a_1 + a_2 + b_2 + c_0 + c_1 + c_2 + d_0 + d_2 \\ a_1 + b_0 + b_1 + c_1 + d_0 + d_1 \\ a_0 + a_1 + a_2 + b_0 + b_2 + c_1 + c_2 + d_2 \end{cases}$$

Similarly and consequently with Proposition 1, the values of the higher nibbles outputting `MixColumn` *depend on the values of the higher nibbles at the input and on 3 quantities computed from the lower nibbles that we will call 3 information bits. More precisely, to compute the higher nibbles resulting of the operation* `MixColumn(a,b,c,d)`*, the 3 information bits are:*

$$\begin{cases} a_4 + b_4 \\ b_4 + c_4 \\ c_4 + d_4 \end{cases}$$

When considering $\texttt{MixColumn}^{-1}\texttt{(a,b,c,d)}$*, a similar property holds for the computation of the output higher nibbles, which requires the following 3 information bits in addition to the input higher nibbles:*

$$\begin{cases} a_5 + a_6 + b_6 + c_4 + c_5 + c_6 + d_4 + d_6 \\ a_5 + b_4 + b_5 + c_5 + d_4 + d_5 \\ a_4 + a_5 + a_6 + b_4 + b_6 + c_5 + c_6 + d_6 \end{cases} \quad (1)$$

The proof can be found in the full version of this article [15]. This proposition implies that an attacker that knows the values of the input lower (respectively higher) nibbles, has to guess 6 bits only to compute `MixNibbles` (or $\texttt{MixNibbles}^{-1}$) in its lower (respectively higher) nibbles. Since `MixColumn` is linear, Proposition 3 applies both to values and to differences. This proposition will be used in our attacks for two different things: computing the correct probabilities of each round, as well as showing how to invert the rounds.

4.2 Probability of one iterative round and other initial rounds

As previously said, the probability of one iterative round was correctly pointed out in [24] as 2^{-6}. In [4] the given probability was $2^{-5.82}$. In fact, if no condition is imposed on the input of one iterative round, the probability of verifying it is indeed 2^{-6}. We have been able to implement and verify this: we have found a probability extremely close to 2^{-6} per iterative round with no special condition on the input. As 2^{-6} can be seen as a close lower bound, we will consider this probability for the iterative rounds without conditions as it is the worst scenario for the attacker.

The difference in the probability computation can be introduced when, because of a specific form of the differential path and the branch number of `MixColumn` from previous rounds, some input nibbles are active or inactive for

sure. This will change the probability due to the Sbox: as already pointed out in [4], when a nibble is active with probability one, its probability of outputting a pair of values with a difference having its MSB to 0 is 7/15, but the probability of having its MSB to 1 is 8/15. We have correctly taken this into account and conducted new computations and experiments. All the configurations that we will use in our analysis are represented in Figure 2, so we use this figure as reference. We have obtained the following results for the rounds 1,2,3 and 4 represented in the figure:

- For round 1, as pointed out in [24], there exists only one possible difference with the 4 active lower nibbles that outputs after `MixColumn` a difference with only one lower nibble active and no difference in the higher nibbles: $(0d0b0e09)$. The probability of this round is then 2^{-16}, as we want to obtain a fixed difference after `SubNibbles`.
- For round 2 we know for sure that we have only one active `MixColumn`, and in its input, we have only one active nibble, that will be active with probability one. This means that the probability of verifying this round is $(\frac{7}{15}) = 2^{-1.1}$.
- For round 3 we have both `MixColumn` active, each with exactly two active nibbles in its input. The probability for this round is $(\frac{7}{15})^4 = 2^{-4.4}$, as the MSB of all the 4 active nibbles need to be 0.
- For round 4 the track of the influence of the branch number starts to be hard to follow. We have then performed experiments, and we could verify that the probability of this step is extremely close to 2^{-6}. Though a bit higher, for the sake of simplicity and for considering the worst case for the attacker, we consider it to be 2^{-6} (so the same case as iterative rounds).

4.3 Cost of inverting one round

By *inverting one iterative round* we consider the situation where a pair of values for the lower nibbles of the output is given and we want to obtain the possible pairs of values for the lower nibble inputs of that round. The lower nibbles of the corresponding round-key are supposed known.

To invert round r we are given a candidate (C, C', k_{low}), and its associated candidate pair at round r: $(S, S')_r$. At the end, we obtain $(S, S')_{r-1}$. As the lower nibbles of the key are known, we will omit `AddRoundKey` from the explanation, as we only work on the lower nibbles of the state and it can consequently be seen as transparent. The cost of inverting one round by using the naive way of guessing the 6 additional information bits (3 for each $\texttt{MixColumn}^{-1}$ as explained in Proposition 3) all at once, and next discarding the ones that do not verify the conditions of the previous round is of 2^6. We propose here a procedure for inverting one round with a cost of 2^4 instead of 2^6.

Improved cost of inverting one round. We first consider the case of inverting one iterative round. We will later see what happens when the round is not iterative. First, we make the 3-bit guess from the right part of Figure 1 to compute $\texttt{MixColumn}^{-1}$, $\texttt{RotateNibbles}^{-1}$ and $\texttt{SubNibbles}^{-1}$ from steps 1,2 and

3 on half of the known state of $(S, S')_r$. Then we have obtained the pair of values of the two most-left and of the two most-right lower nibbles from $(S, S')_{r-1}$, and with them we can compute the nibble differences a, b, c', d' at the output of `MixColumn` from step 4 in the figure. With the notations from Section 4.1, for each of the 2^3 values of the information bits that we have tried, we compute the following six bits in this exact order: $(a_5 + a_6 + b_6, a_5 + b_4 + b_5, a_4 + a_5 + a_6 + b_4 + b_6, c'_4 + c'_5 + c'_6 + d'_4 + d'_6, c'_5 + d'_4 + d'_5, c'_5 + c'_6 + d'_6)$. It is easy to verify that these quantities correspond each to one half of the last three equations from Proposition 3, equations that need to be verified in order to have the higher nibbles inactive in the input of `MixColumn`. The first three quantities correspond to the part of the equations associated to the left `MixColumn` from Figure 1, and the next three, to the right one.

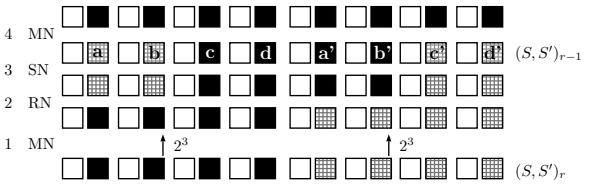

Fig. 1. Detail of inverting one round.

We can store all of these 6-bit values in a list of size 2^3. Next, we perform a guess of the 3 bits needed for inverting the other half of $(S, S')_r$, and we can compute the nibble differences a', b', c, d from the figure. For each one of the 2^3 values that we try, we compute the following 6 bits, in this order: $(c_4 + c_5 + c_6 + d_4 + d_6, c_5 + d_4 + d_5, c_5 + c_6 + d_6, a'_5 + a'_6 + b'_6, a'_5 + b'_4 + b'_5, a'_4 + a'_5 + a'_6 + b'_4 + b'_6)$ and check if there is a match in the precomputed list, as we know that, for any $\texttt{MixColumn}^{-1}$ with no higher nibbles active in the input nor the output, the three expressions from equation (1) must be equal to 0. The associated values of the match will represent a value for a pair candidate at round $r - 1$, where $\Delta(S, S')_{r-1} = (a, b, c, d, a', b', c', d')$. A match occurs with probability of 2^{-6}, as we want to collide on 6 bits. As we have $2^3 \times 2^3$ pairs, on average we can expect only one value to stay, so only one value on average for $(S, S')_{r-1}$ per candidate triplet as previously announced, so inverting one iterative round does not increase the number of kept candidates. The cost of this step, given by applying the instant matching algorithm from [18] is about $2^3 + 2^3 = 2^4$.

If the round we want to invert is not iterative, there usually exist more difference conditions to be verified on the input, as in addition to the previous six bit conditions, some more differences might have been fixed, like for example non-active lower input Sboxes (in the non-iterative rounds we have considered in our study, this is always the case). The cost of inverting is exactly the same (2^4), but since there are more equations to verify, the probability that a candidate pass will be smaller than 2^{-6}: fewer candidates will be kept. The number of

candidates kept is even smaller when we invert the last round since we have to match both differences and values of the computed half-states.

4.4 Guessing the necessary key-bits

To perform the attack, we consider the pairs of ciphertexts (C, C') that verify the last round condition on 32 bits and we need to guess the values of the lower nibbles of the key (k_{low}) that verify as well the first round conditions. Then, we are able to compute the values and differences before the first `MixNibbles`, place where we will stop inverting and where we will do the match of values and differences recovered in the other direction. Proposition 2 shows us that a $|k|/2$ bit guess (where $|k|$ is the length of the key) of the lower nibbles of the key is sufficient to obtain all the lower nibbles of all the round-keys.

We use a divide-and-conquer approach to guess the bits from k_{low} and we are then able to build the candidates (C, C', k_{low}). This technique is similar to the one proposed for inverting one round. Instead of guessing all the lower nibbles of one key at the same time, we can perform this guess in an ingenious way that allows to reduce the computational cost.

We consider the first round of the path (as conditions before `MixNibbles` can be tested here without needing to guess any extra bits). From the plaintexts (from which we obtained the ciphertexts C, C') we first consider 4 lower nibbles that will be the inputs to the same `MixColumn`, and we guess the sixteen bits from k_{low} that are xored to this 4 nibbles in the first round. We next compute the inputs of the corresponding `MixColumn` and we only keep the keys that fulfill the needed conditions to verify round 1. These conditions will depend on the specific path considered. For example, in Figure 2, the conditions of the right `MixColumn` from the first round are verified with probability 2^{-16}, as only one difference can verify them (as explained previously), which means that among the 2^{16} keybits tried for this half, only one value will verify this part of the path. We repeat the procedure with the other 4 input lower nibbles and the other 16 key-bits. We combine both partial solutions and we obtain all the 2^{32+p_1} possible candidate triplets derived from (C, C'), where 2^{p_1} is the probability of verifying the first round of the path.

The cost in number of encryptions of recovering the candidate triplets for each pair of ciphertexts considered is then $2^{16} + 2^{16} + 2^{32+p_1} \times 2^{|k_{low}|-32}$, as in the first round only 32 bits from the lower nibbles intervene, and for realizing the whole attack we have to guess the remaining ones (for KLEIN-64 the last factor is 1 as $|k_{low}| = 32$).

4.5 Generic description of the attack procedure and complexity

First, we choose a truncated differential path over the desired number of rounds: several choices are possible, varying with the considered differences in the first three rounds of the path and leading to different time-memory-data trade-offs, as we will see in Section 5. Note that the size of the truncated difference entering the first round determines the size of the structures that we can build with the

input plaintexts: if the size is Δ_{in} bits, we can build about $2^{2\Delta_{in}-1}$ pairs with $2^{\Delta_{in}}$ inputs. In the following 2^p represents the probability of the whole path, R is the number of rounds of the differential path and $R+1$ is the number of rounds that we will attack. We denote by 2^{p_1}, 2^{p_2} and 2^{p_3} the probabilities of the first 3 rounds respectively, $p_1, p_2, p_3 \leq 0$. Since one iterative round has a probability of 2^{-6} of being verified and because of the forms of the considered paths, we have $p = p_1 + p_2 + p_3 - 6 \times (R-3)$.

1. Obtaining enough data: With the use of structures, we generate a certain number of ciphertexts such that we obtain enough pairs to be ensured to get one that verify our differential path: to obtain the required 2^{-p} pairs, we encrypt $\frac{2^{-p}}{2^{2\Delta_{in}-1}} 2^{\Delta_{in}}$ plaintexts so we perform $\frac{2^{-p}}{2^{2\Delta_{in}-1}} 2^{\Delta_{in}}$ encryptions.

2. Last-round filter: At this point, we can discard some pairs that for sure do not verify the differential path. As detailed in [4, 24], by inverting the output difference through the last `MixColumn` we can observe the value of the difference entering this transformation and then discard the ones that do not have the higher nibbles inactive. In practice, we construct a sorted list of all the higher nibbles values obtained by inverting `MixColumn` from the ciphertexts of a same structure (without considering the key addition) and look for collisions. Such a collision occurs with probability 2^{-32} so there are 2^{-p-32} remaining pairs of plaintexts.

3. Guess the involved key-bits: For each pair of plaintexts and their associated ciphertexts that collide at the previous step, we make two 16-bit guesses as explained in Section 4.4 and obtain possible values of the first 8 lower nibbles of the key. Since the conformity with round 1 is of probability 2^{p_1} it gives us $2^{-p-32+32+p_1}$ candidates formed by a pair of plaintexts and the 8 first lower nibbles of the master key. If the version attacked is KLEIN-64, the 8 lower nibbles correspond to the lower nibbles of the whole key but for KLEIN-80 and KLEIN-96, we have to make additional guesses to obtain all the possible lower nibble values. For KLEIN-64, KLEIN-80 and KLEIN-96 we obtain respectively $2^{-p-32+32+p_1}$, $2^{-p-32+32+p_1+8}$ and $2^{-p-32+32+p_1+16}$ possible candidates (C, C', k_{low}). This step requires $2 \times \frac{1}{12} \times 2^{16}$ encryptions, and allows us to compute the associated pair of half-states (associated to each candidate) at the input of the first `MixNibbles` that already satisfies the conditions from round 1. We will denote this pair of half-states by $(S, S')_1^*$.

4. Inverting the rounds: At this point we start inverting the rounds from the candidates that we have obtained, generating possible pairs $(S, S')_r$ for r from R to 1. That step requires 2^4 round encryptions per inversion and per triplet, as detailed in Section 4.3. During the iterative rounds, the number of possible triplets stays the same, contrary to what happens during the non-iterative rounds where the number of candidates is reduced (see Section 4.3). The attack is performed one triplet at a time. Once we have computed

$(S, S')_1$, we have to guess the 6 bits needed to invert the first `MixNibbles`, and next we have to match values and active differences with the already computed values $(S, S')_1^*$.
In the differential paths that we will consider, like the one represented in Figure 2, if we denote by 2^q the filtering probability obtained when inverting rounds 2,3 and 4 (in some cases round 4 or even 3 adds just a filter of 2^{-6}, but to be general we include it in the special case), the total number of remaining candidates in the end is $2^{-p-32+32+p_1+6\times4+q}$, $2^{-p-32+32+p_1+8+6\times4+q}$ and $2^{-p-32+32+p_1+16+6\times4+q}$ respectively for KLEIN-64, KLEIN-80 and KLEIN-96. If the number of remaining candidates is smaller than $2^{|k_{low}|}$, as there is one possible value for k_{low} per candidate, the cost of recovering the key is smaller than the one of exhaustive search. In practice, after inverting all the rounds, the number of remaining candidates is currently very small.
The cost of this step is given by the initial candidate triplets $2^{-p-32+32+p_1}$ multiplied by 2^4 (cost of inverting), multiplied by the number of inverted rounds and by the relative cost to one encryption of each inverted round. In the next section we will see an illustrative and detailed example of this computation.

5. Recovering the whole key: Finally, we have to recover the higher nibbles of the key, which can be done by an exhaustive search or better (as it is explained in Section 4.6), and we have recovered the whole key.

4.6 Higher Nibbles recovery

To recover the complete key, we can perform a more efficient attack than the exhaustive search for the remaining key-bits, by deducing information from the 6-bit guesses associated to the candidates that remain after the sieving process. One more time, each candidate that passes the sieving process will be studied separately; we make the hypothesis that the candidate is valid so the 6-bit guesses give us the values that must take certain combinations of intermediate states.

For instance, at the first round, we know the value that must be taken by the sum of the MSB of the higher nibbles of the state entering the two first `MixColumn`. For each possible value of the higher nibbles of the key, we will compute that state from the plaintext and check this 6-bit condition. The keys that pass the test will undergo the second sieve resulting from the second round information bits and so on.

To limit the number of encryptions required to complete this step, we can one more time use the independences between the 2 half-states during the `MixNibbles` operation. We first make a guess on the 16 higher nibbles of the master key that are added to the 32 middle bits of the state at the first round and that impact the value of the 32 bits at the output of the left `MixColumn`. We then check the 3 corresponding information bits and realise the same operations for the 16 key bits required to compute the right `MixColumn`. Since 2^{16} half-state encryptions are required for each half, the first round filter requires $2^{16} \times \frac{1}{2} \times 2$ round encryptions.

Next, as previously said, each round gives a 6-bit filter for the higher nibbles of the key. For example, for KLEIN-64 the total complexity of the higher nibbles recovery is of $(2^{16} + 2^{26} + 2^{20} + 2^{14} + 2^8 + 2^2) \times \frac{1}{12} = 2^{22.4}$ complete encryptions. This procedure needs to be repeated for each candidate recovered during the k_{low} search.

5 Different Time-Memory-Data Trade-Offs

Various differential truncated paths are possible, each one leading to different time-memory-data trade-offs. We have studied many different possibilities and we present here the 4 cases that have provided better results. The only difference between them is the shape of the wanted differences in the first three rounds of the path. We will explain one of them applied to the full 12 rounds of KLEIN-64 in a detailed way as an illustration of the attacks, and next present the other cases considered, also with results given on the 64-bit key version. In the next section we will provide the results obtained with some considered cases in all the versions of the cipher, for several number of rounds up to the highest number that could be reached (which is 12, 13 and 14 for the 64-, 80-, and 96-bit versions respectively).

We recall here that the probability values that we will use are well detailed and explained in Section 4.2.

5.1 Case I

In this case, we consider a truncated differential path that corresponds to the one in [24]. This path is depicted in Figure 2, including the details of our attack. This is the application that will provide us the best time complexity attack on the full 12 rounds of KLEIN-64.

1. Obtaining enough data: Using Figure 2 it is easy to verify that the differential path has a probability of $p = 2^{-16-1.1-4.4-6\times 8} = 2^{-69.5}$ and that $\Delta_{in} = 16$. Here, the probabilities correspond to the ones discussed in Section 4.2. We need to generate pairs such that we can expect with a good probability that one among them will verify the whole path. Since one structure allows us to build 2^{16} plaintexts, which lead to $\frac{2^{16}\times(2^{16}-1)}{2} \approx 2^{31}$ pairs, we have to build $2^{69.5-31} = 2^{38.5}$ structures to have among all the pairs one verifying the whole path. This step then requires $2^{38.5} \times 2^{16} = 2^{54.5}$ full encryptions, which correspond to the data complexity of the attack. As it is smaller than 2^{64}, the whole codebook, we can obtain such an amount of plaintexts. Since the true conforming pair is necessarily composed of 2 plaintexts from the same structure, one structure can be treated after the other so the required memory is of 2^{16} plaintexts.
2. Last-round filter: we compare the values obtained when inverting the last `MixNibbles` on each ciphertext and keep the pairs that have a difference with inactive higher nibbles at this point. There remain $2^{69.5-32} = 2^{37.5}$ candidate pairs of plaintext. The cost of this step is negligible.

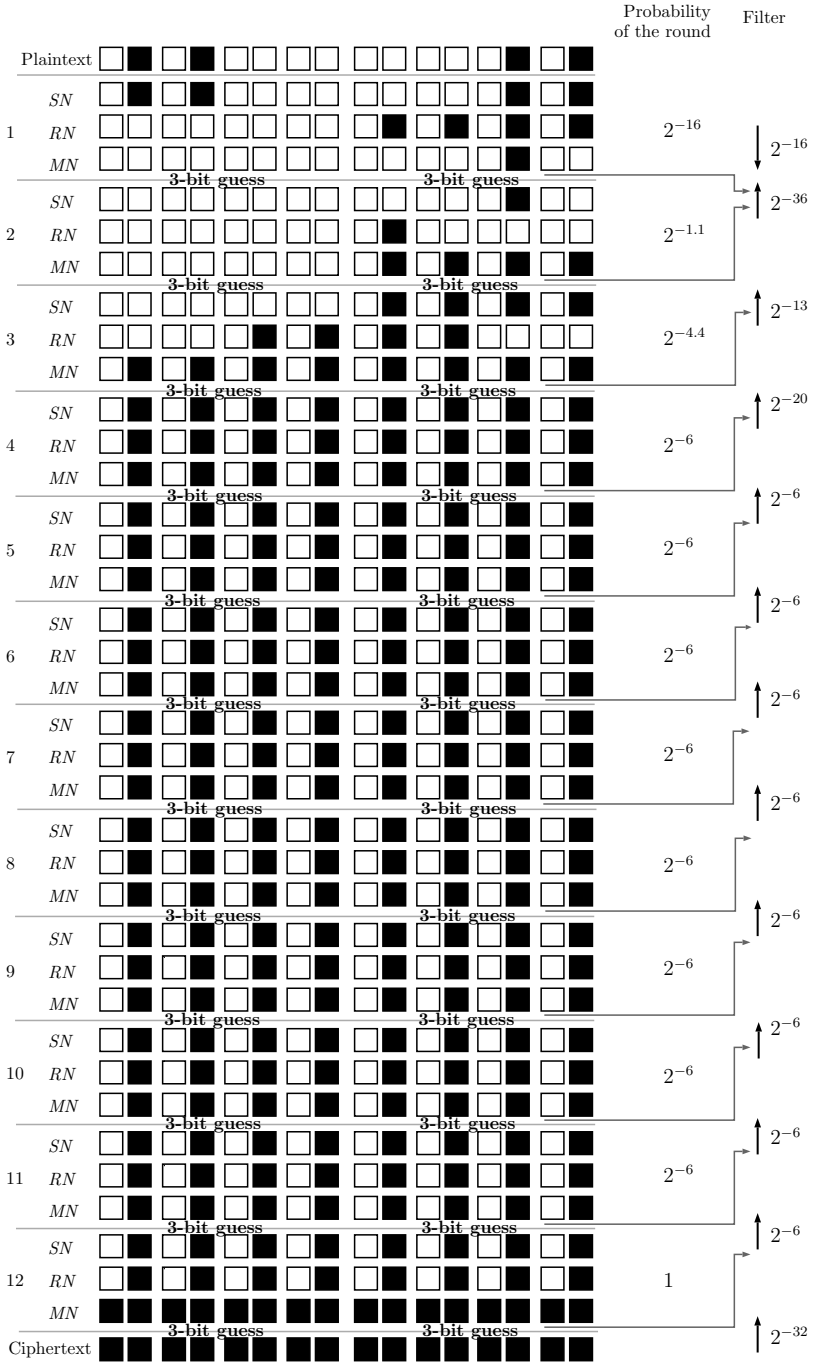

Fig. 2. Attack on 12 rounds of KLEIN-64 using case I

3. Guess the involved key-bits: At this point, we perform the optimized guess of the lower nibbles of the key, k_{low}, as seen in Section 4.4, so we obtain $2^{32-16} \times 2^{37.5} = 2^{53.5}$ candidates (C, C', k_{low}) with $(2^{16} \times \frac{1}{2} \times \frac{1}{12}) \times 2$ encryptions. So far, we have 2^{16} possible keys k_{low} for each candidate pair (C, C'). For each candidate set, we know the values and differences at the input of `MixNibbles` of the first round, $(S, S')_1^*$, and these values already verify the conditions imposed through the first `MixNibbles`.
4. Inverting the rounds: Each time that we obtain one of the $2^{53.5}$ candidates, we start inverting rounds and generating the candidate pairs from $(S, S')_{11}$ to $(S, S')_4$ (so we invert 8 rounds). We can see in Figure 2 that for all of these rounds, the amount of bits guessed compensates the probability of verifying the path, and we expect to obtain one pair candidate per round and per candidate set. As we can see in the figure, the remaining probabilities for inverting rounds 4, 3, 2 and the `MixNibbles` transform from round 1 (so to arrive to the previously computed values $(S, S')_1^*$) is $2^q = 2^{-20-13-36} = 2^{-69}$. Indeed, the filter probability at the end of round 3 is of $2^{-(4\times4)-4} = 2^{-20}$, since we need to have 4 lower nibbles inactive and two MSB of the 4 active lower nibbles to 0 and the input of `MixNibbles` as depicted in Figure 2. The probability of the filter at the end of round two is of $2^{-13} = 2^{-(3\times4)-1}$ because of the three inactive lower nibbles at the input of the active `MixColumn` and the MSB of the active one, and the filter probability at the end of the first round is of $2^{-36} = 2^{-32-4}$ as we have to collide with the previously forward computed 32 bits of values and 4 bits of differences. If we apply the formula from Section 4.5, we can compute the number of remaining candidate triplets as $2^{69.5-32+32-16+6\times4-69} = 2^{8.5}$, meaning also that we recover only $2^{8.5}$ possibilities for the 32 bits in k_{low}. The term $2^{6\times4}$ comes from the fact that we have to guess the 6 bits for inverting 4 rounds, namely rounds 4, 3, 2 and also 1, as we want to match the values in $(S, S')_1^*$. The cost of this step is $2^{69.5-32+32-16} \times 2 \times \frac{1}{2} \times \frac{8}{12} \times (2^3 + 2^3) \simeq 2^{56.9}$ encryptions for the inversion of the first 8 rounds. And for inverting the remaining 4 rounds, we have a complexity of $2^{53.5} \times \frac{1}{12} \times (2^3 + 2^3) + 2^{53.5+6-20} \times \frac{1}{12} \times (2^3 + 2^3) + 2^{39.5+6-13} \times \frac{1}{12} \times (2^3 + 2^3) + 2^{32.5+6} \times \frac{1}{12} \times (2^3 + 2^3) \simeq 2^{53.9}$. This part of the attack will be the bottleneck of the total time complexity: $2^{56.9} + 2^{53.9} = 2^{57.07}$.
5. Recovering the whole key: Finally, we recover the higher nibbles with the process explained in Section 4.6. The cost of this step is $2^{8.5} \times 2^{22.4}$, so the bottleneck in terms of time complexity for recovering the whole key is the one of the previous step.

This version requires a total of $2^{57.07}$ encryptions, $2^{54.5}$ data and 2^{16} memory.

5.2 Case II

We use a truncated differential path associated to the path given in [4] (the value of the input difference is not fixed). As there is only one active nibble at the beginning of the path, the structures will be the smallest ones that we

will use (size of $\Delta_{in} = 4$), and the memory will be very small, while the data complexity will be bigger than in other cases.

For the case of 12 rounds of KLEIN-64, we have a probability for the path of $2^{-1.1-4.4-6\times 9} = 2^{-59.5}$. The amount of data needed is $2^{59.5-7+4} = 2^{56.5}$, and the memory needed is 2^4. The bottleneck in the time complexity is given by $2^{59.5-32+32-1.1} \times 2 \times \frac{1}{2} \times \frac{9}{12} \times (2^3 + 2^3) \simeq 2^{61.98}$. The number of remaining candidates is $2^{58.39+6-16-4+6-12-1+6-4-32} = 2^{7.39}$. The time for recovering the remaining bits of the key is $2^{7.39} \times 2^{22.4} = 2^{29.79}$.

5.3 Case III

This attack uses a path with an iterative round at every round, and Δ_{in} is consequently 32. This attack has low data complexity, but the highest memory, and not very good time complexity in the case of an attack on 12-round KLEIN-64.

For the scenario on 12 rounds of KLEIN-64, the time complexity is very close to the one of the exhaustive search. We have a probability for the path of $2^{-6\times 11} = 2^{-66}$. The amount of data needed is $2^{3+32} = 2^{35}$, and the memory needed is 2^{32}. The cost of the time bottleneck is given by $2^{66-32+32-6} \times 2 \times \frac{1}{2} \times \frac{11}{12} \times (2^3 + 2^3) = 2^{63.87}$. The number of remaining candidates is $2^{60+6-64} = 2^2$. The time for recovering the remaining bits of the key is $2^2 \times 2^{22.4} = 2^{24.4}$.

5.4 Case IV

In this case, we use a truncated path that starts with a difference of the form $(0X0X000000000X0X)$, when represented as 16 nibbles. this one has the same time complexity than the second one, better data and worse memory.

For the case of 12 rounds of KLEIN-64, we have a probability for the path of $2^{-3-4-6\times 9} = 2^{-61}$. The amount of data needed is $2^{16+30} = 2^{46}$, and the memory needed is 2^{16}. The bottleneck in the time complexity is given by $2^{61-32+32-3} \times 2 \times \frac{1}{2} \times \frac{9}{12} \times (2^3 + 2^3) = 2^{61.57}$. The number of remaining candidates is $2^{61-3+6\times 3-20-48} = 2^8$. The time for recovering the remaining bits of the key is $2^8 \times 2^{22.4} = 2^{30.4}$.

5.5 Results on KLEIN-64

The complexity of the 4 different trade-offs presented here are summarized in Table 3 and depicted on Figure 3. Table 4 provides the results obtained on KLEIN-64 using case I for different numbers of rounds.

5.6 Results on KLEIN-80 and KLEIN-96

For KLEIN-80 and 96 we provide in Table 5 the obtained results for several numbers of rounds using different cases. Case I does not reach a lot of rounds, as the data complexity exceeds 2^{64}, which is the maximal amount of data available, after 13 rounds.

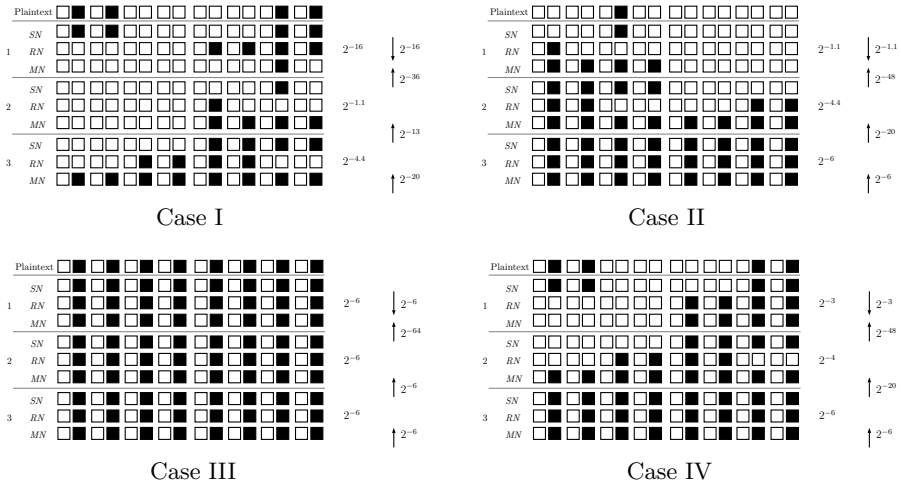

Fig. 3. Beginnings of the truncated differential paths of our 4 trade-offs

Case	p1	p2	p3	Data	Time	Memory
I	-16	-1.1	-4.4	$2^{54.5}$	2^{57}	2^{16}
II	-1.1	-4.4	-6	$2^{56.5}$	2^{62}	2^{4}
III	-6	-6	-6	2^{35}	$2^{63.9}$	2^{32}
IV	-3	-4	-6	2^{46}	$2^{61.6}$	2^{16}

Table 3. Summary of the probabilities and complexities of our 4 trade-offs

Rounds	Data	Time	Memory
8	$2^{30.5}$	$2^{31.7}$	2^{16}
9	$2^{36.5}$	2^{38}	2^{16}
10	$2^{42.5}$	$2^{44.4}$	2^{16}
11	$2^{48.5}$	$2^{50.6}$	2^{16}
12	$2^{54.5}$	$2^{57.07}$	2^{16}

Table 4. Best time complexities for recovering the whole key for several round-reduced variants of KLEIN-64

6 Implementation and verification

We have experimentally verified the efficiency of the proposed attacks by implementing some variants in C language. We wrote our own implementation of KLEIN-64 with look-up tables for `MixColumn` and its inverse and we verified it with the test vectors given in KLEIN specifications [11].

Version	Case	Rounds	Data	Time	Memory
80	I	12	$2^{54.5}$	2^{65}	2^{16}
80	I	13	$2^{60.5}$	$2^{71.1}$	2^{16}
80	II	13	$2^{62.5}$	2^{76}	2^{4}
80	III	13	2^{41}	2^{78}	2^{32}
80	IV	13	2^{52}	2^{76}	2^{16}
96	III	14	2^{47}	2^{92}	2^{32}
96	IV	14	$2^{58.4}$	$2^{89.2}$	2^{16}

Table 5. Best complexities for recovering the whole key for several round-reduced variants of KLEIN-80 and -96

We then implemented the attack exactly as described in Section 5: the complete key is recovered in 2 steps with first the search for the lower nibbles with a truncated differential and then the search for the higher nibbles with an improved exhaustive search.

In particular, we have been able to implement the first successful practical attack on KLEIN-64 up to 10 rounds. For this, we have considered case I, the one having the smallest time complexity and average data and memory needs. We used several speed-optimization flags and a computer with an Intel(R) Xeon(R) CPU W3670 at 3.20GHz (12MB cache), and with 8GB of RAM. Our program shows that the proposed attack works and recovers the correct key.

Below we report the outputs of our program for an attack on 10 rounds. The field `structure` refers to the randomly chosen values at the beginning, i.e. the 12 nibbles common to all the plaintexts so that the differences between 2 plaintexts are only in the first and last 2 lower nibbles. *Plaintext 1* and *Plaintext 2* form the conforming pair found that enabled us to determine the lower nibbles of the key. Once the lower nibbles are found, the higher ones are recovered in a few seconds.

```
NB rounds: 10
MasterKeyToFind:         66 a2 fa 17  23 19 39 bd

structure:               c0 70 03 39  de 72 30 e0
Plaintext 1:             c3 79 03 39  de 72 34 e4
Plaintext 2:             c2 77 03 39  de 72 30 e1
lower nibbles found:     06 02 0a 07  03 09 09 0d
Complete Key:            66 a2 fa 17  23 19 39 bd
number of pltxt:         624712959124  ie: 2^39.184403
number of false alarms:  4764629       ie: 2^22.183932
number of structures:    9532364
time elapsed:            1254407.310000 sec
```

We checked manually that the 2 returned plaintexts conform the differential path and compared our theoretical results with the practical ones. First,

we notice that this result required by chance little less data than theoretically predicted ($2^{42.5}$) and second we notice that the ratio of false alarms meets the theory: since we encrypted $2^{39.184}$ plaintexts, we were able to create $2^{54.184}$ pairs so we expected a total of $2^{54.184-32} = 2^{22.184}$ pairs to pass the first test which is really close to the observed number of false alarms. This experiment takes near to 15 days.

The experiments on 9-round versions took us an average of 2 days to recover the complete key. Some of them are detailed in the full version of this paper [15]. Since that experiments were quicker, we were able to do several ones to compute average values. As for the previous result, some experiments needed less data than expected theoretically (by chance) but in average on 4 tests $2^{36,483}$ plaintexts were encrypted (which is really close to the $2^{36.5}$ expected), $2^{19,483}$ false alarms appeared and 47 hours were required.

7 Conclusion

In this paper we propose the first attack on the full version of KLEIN-64. It improves the previous results from 10 to 12 rounds. For the 80-bit and 96-bit key versions we have provided several attacks on 13 and 14 rounds respectively. We have implemented round-reduced versions of our attacks, and have been able to verify the theoretical complexities that we have predicted and to validate our assumptions. In particular, we have successfully implemented an attack on 10 rounds of KLEIN-64. This is the practical attack realized on the highest number of rounds, as previous results could not reach more than 8 rounds.

The main weakness of the cipher might be the fact that the `MixColumn` transformation does not correctly mix higher and lower nibbles, as it is the only transform that does so. Maybe considering other matrices instead could lead to a more solid construction. Also, the fact that the KeySchedule does not mix higher and lower nibbles helps the cryptanalyst to perform a reduced partial key search, so a stronger KeySchedule could help to prevent the attacks.

We believe that the family of attacks presented, though clearly dedicated to the cryptanalysis of KLEIN, might apply to other ciphers with big independences between two parts of the state.

References

1. Mohamed Ahmed Abdelraheem, Céline Blondeau, María Naya-Plasencia, Marion Videau, and Erik Zenner. Cryptanalysis of ARMADILLO2. In *ASIACRYPT*, volume 7073 of *Lecture Notes in Computer Science*, pages 308–326. Springer, 2011.
2. Farzaneh Abed, Christian Forler, Eik List, Stefan Lucks, and Jakob Wenzel. Biclique Cryptanalysis Of PRESENT, LED, And KLEIN. Cryptology ePrint Archive, Report 2012/591, 2012. http://eprint.iacr.org/.
3. Zahra Ahmadian, Mahmoud Salmasizadeh, and Mohammad Reza Aref. Biclique Cryptanalysis of the Full-Round KLEIN Block Cipher. Cryptology ePrint Archive, Report 2013/097, 2013. http://eprint.iacr.org/.

4. Jean-Philippe Aumasson, María Naya-Plasencia, and Markku-Juhani O. Saarinen. Practical attack on 8 rounds of the lightweight block cipher KLEIN. In *INDOCRYPT*, volume 7107 of *Lecture Notes in Computer Science*, pages 134–145. Springer, 2011.
5. Stéphane Badel, Nilay Dagtekin, Jorge Nakahara, Khaled Ouafi, Nicolas Reffé, Pouyan Sepehrdad, Petr Susil, and Serge Vaudenay. ARMADILLO: A multi-purpose cryptographic primitive dedicated to hardware. In *CHES*, volume 6225 of *Lecture Notes in Computer Science*, pages 398–412. Springer, 2010.
6. Andrey Bogdanov, Miroslav Knezevic, Gregor Leander, Deniz Toz, Kerem Varici, and Ingrid Verbauwhede. SPONGENT: A lightweight hash function. In *CHES*, volume 6917 of *Lecture Notes in Computer Science*, pages 312–325. Springer, 2011.
7. Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin, and C. Vikkelsoe. PRESENT: An ultra-lightweight block cipher. In *CHES*, volume 4727 of *Lecture Notes in Computer Science*. Springer, 2007.
8. Julia Borghoff, Anne Canteaut, Tim Güneysu, Elif Bilge Kavun, Miroslav Knezevic, Lars R. Knudsen, Gregor Leander, Ventzislav Nikov, Christof Paar, Christian Rechberger, Peter Rombouts, Søren S. Thomsen, and Tolga Yalçin. PRINCE - A Low-Latency Block Cipher for Pervasive Computing Applications - Extended Abstract. In Xiaoyun Wang and Kazue Sako, editors, *ASIACRYPT*, volume 7658 of *Lecture Notes in Computer Science*, pages 208–225. Springer, 2012.
9. Julia Borghoff, Lars R. Knudsen, Gregor Leander, and Søren S. Thomsen. Cryptanalysis of PRESENT-like ciphers with secret s-boxes. In *FSE*, volume 6733 of *Lecture Notes in Computer Science*, pages 270–289. Springer, 2011.
10. Baudoin Collard and François-Xavier Standaert. A Statistical Saturation Attack against the Block Cipher PRESENT. In *Topics in Cryptology - CT-RSA 2009*, Lecture Notes in Computer Science 5473, pages 195–210. Springer Verlag, 2009.
11. Zheng Gong, Svetla Nikova, and Yee Wei Law. KLEIN: A new family of lightweight block ciphers. In *RFIDSec*, volume 7055 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2011.
12. Jian Guo, Thomas Peyrin, Axel Poschmann, and Matthew J. B. Robshaw. The LED block cipher. In *CHES*, volume 6917 of *Lecture Notes in Computer Science*, pages 326–341. Springer, 2011.
13. Vikash Kumar Jha. Cryptanalysis of lightweight block ciphers. Aalto University Master's Thesis, 2011. https://into.aalto.fi/download/attachments/9382995/VikashKumarJha_thesis.pdf.
14. Lars R. Knudsen, Gregor Leander, Axel Poschmann, and Matthew J. B. Robshaw. PRINTcipher: A block cipher for IC-Printing. In *CHES*, volume 6225 of *Lecture Notes in Computer Science*, pages 16–32. Springer, 2010.
15. Virginie Lallemand and María Naya-Plasencia. Cryptanalysis of KLEIN (full version). Cryptology ePrint Archive, Report 2014/090, 2014. http://eprint.iacr.org/.
16. Gregor Leander, Mohamed Ahmed Abdelraheem, Hoda AlKhzaimi, and Erik Zenner. A cryptanalysis of PRINTcipher: The invariant subspace attack. In *CRYPTO*, volume 6841 of *Lecture Notes in Computer Science*, pages 206–221. Springer, 2011.
17. Florian Mendel, Vincent Rijmen, Deniz Toz, and Kerem Varici. Differential analysis of the LED block cipher. In *ASIACRYPT*, volume 7658 of *Lecture Notes in Computer Science*. Springer, 2012.
18. María Naya-Plasencia. How to improve rebound attacks. In *CRYPTO*, volume 6841 of *Lecture Notes in Computer Science*, pages 188–205. Springer, 2011.

19. María Naya-Plasencia and Thomas Peyrin. Practical cryptanalysis of ARMADILLO2. In *FSE*, volume 7549 of *Lecture Notes in Computer Science*, pages 146–162. Springer, 2012.
20. Ivica Nikolic, Lei Wang, and Shuang Wu. Cryptanalysis of round-reduced LED. In *FSE 2013*, Lecture Notes in Computer Science. Springer, 2013. To appear.
21. Ivica Nikolic, Lei Wang, and Shuang Wu. The parallel-cut meet-in-the-middle attack. Cryptology ePrint Archive, Report 2013/530, 2013. http://eprint.iacr.org/.
22. Tomoyasu Suzaki, Kazuhiko Minematsu, Sumio Morioka, and Eita Kobayashi. TWINE : A lightweight block cipher for multiple platforms. In Lars R. Knudsen and Huapeng Wu, editors, *Selected Areas in Cryptography*, volume 7707 of *Lecture Notes in Computer Science*, pages 339–354. Springer, 2012.
23. Wenling Wu and Lei Zhang. LBlock: A Lightweight Block Cipher. In *ACNS 2011*, volume 6715 of *Lecture Notes in Computer Science*, pages 327–344. Springer, 2011.
24. Xiaoli Yu, Wenling Wu, Yanjun Li, and Lei Zhang. Cryptanalysis of reduced-round KLEIN block cipher. In *Inscrypt*, volume 7537 of *Lecture Notes in Computer Science*. Springer, 2011.