Some Instant- and Practical-Time Related-Key Attacks on KTANTAN32/48/64

Martin Ågren

Dept. of Electrical and Information Technology, Lund University,
P.O. Box 118, 221 00 Lund, Sweden
martin.agren@eit.lth.se

Abstract. The hardware-attractive block cipher family KTANTAN was studied by Bogdanov and Rechberger who identified flaws in the key schedule and gave a meet-in-the-middle attack. We revisit their result before investigating how to exploit the weakest key bits. We then develop several related-key attacks, e.g., one on KTANTAN32 which finds 28 key bits in time equivalent to $2^{3.0}$ calls to the full KTANTAN32 encryption. The main result is a related-key attack requiring $2^{28.44}$ time (half a minute on a current CPU) to recover the full 80-bit key. For KTANTAN48, we find three key bits in the time of one encryption, and give several other attacks, including full key recovery. For KTANTAN64, the attacks are only slightly more expensive, requiring $2^{10.71}$ time to find 38 key bits, and $2^{32.28}$ for the entire key. For all attacks, the requirements on related-key material are modest as in the forward and backward directions, we only need to flip a single key bit. All attacks succeed with probability one. Our attacks directly contradict the designers' claims. We discuss why this is, and what can be learnt from this.

Keywords: cryptanalysis, related key, block cipher, key schedule, lightweight cipher, key-recovery

1 Introduction

KTANTAN is a hardware-oriented block cipher designed by De Cannière, Dunkelman and Knežević. It is part of the KATAN family [4] of six block ciphers. There are three variants KTANTANn where $n \in \{32, 48, 64\}$. All ciphers consist of 254 very simple, hardware-efficient rounds.

The only difference between KATAN and KTANTAN is the key schedule. The goal with KTANTAN is to allow an implementation to use a burnt-in key, which rules out loading the key into a register and applying some state updates to it in order to produce subkeys. Instead, subkeys are chosen as original key bits, selected according to a fixed schedule. This schedule is the same for all three variants.

Aiming for a lightweight cipher, the designers of KTANTAN did not provide the key schedule as a large table of how to select the key bits. Rather,

a small state machine generates numbers between 0 and 79. In this way, key bits can hopefully be picked in an irregular fashion. As shown by Bogdanov and Rechberger [3], the sequence in which the key bits are used has some unwanted properties.

We will revisit the result of Bogdanov and Rechberger. We adjust the presentation slightly, before using their observation to launch a related-key attack. Bogdanov and Rechberger noted this as a possible direction of research, but did not look into it further.

Related-key attacks have been known for almost twenty years [5, 1]. Like most other related-key attacks, the ones presented in this paper are quite academic in their nature. They are still a good measurement of the security of the cipher, which should appear as an ideal permutation, and several notable properties make the attacks in this paper very interesting:

1. They are minimal: they only require flipping one bit in the key and in several cases, it is enough for the attacker to use only one triplet: one plaintext and two ciphertexts.
2. They are extreme: we find a large number of key bits in time equivalent to just a few encryptions. For KTANTAN32, the entire key can be found in half a minute on a current CPU.
3. They never fail: All the properties exploited in this paper have probability one, meaning the correct (partial) key *always* shows the property we look for.
4. They directly contradict the designers' claims. We will discuss why this is, and what can be learnt from this.

The remainder of this paper is organized as follows: In Section 2 we describe the cipher KTANTAN, and Section 3 introduces (truncated) differentials. Section 4 discusses the result by Bogdanov and Rechberger [3]. Section 5 develops our attacks on KTANTAN32, while we summarize our results on KTANTAN48 and KTANTAN64 in Section 6. In Section 7 we compare our results to the designers' original claims on related-key security before concluding the paper in Section 8.

2 KTANTAN

The n-bit plaintext $P = p_{n-1} \dots p_0$ is loaded into the state of the cipher, which consists of two shift registers, L_1 and L_2, see Fig. 1. For KTANTAN32, these are of lengths $|L_1| = 13$ and $|L_2| = 19$. The other variants use longer registers.

The 254 rounds are denoted as round $0, 1, \dots, 253$. Each round uses two key bits, k_a^r and k_b^r, which are picked straight from the 80-bit master key. The key schedule is provided in Appendix A.

The contents of the registers are shifted, and the new bit in each register (L_1/L_2) is created from five or six bits from the other register (L_2/L_1), through

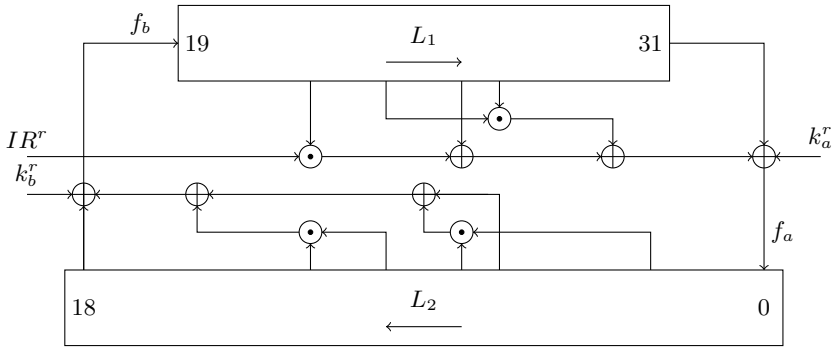

Fig. 1. An overview of KTANTAN32. In each clocking, one shift is made and two key bits, k_a^r and k_b^r, are added to the state. IR^r is a round constant which decides whether or not $L_1[3]$ is used in the state update or not. Indices denote how bits in the plaintext/ciphertext are identified. L_1 is shifted to the right and L_2 to the left.

Table 1. The parameters defining KTANTANn, where $n \in \{32, 48, 64\}$.

n	$\lvert L_1\rvert$	$\lvert L_2\rvert$	x_1	x_2	x_3	x_4	x_5	y_1	y_2	y_3	y_4	y_5	y_6
32	13	19	12	7	8	5	3	18	7	12	10	8	3
48	19	29	18	12	15	7	6	28	19	21	13	15	6
64	25	39	24	15	20	11	9	38	25	33	21	14	9

some simple functions of degree two. For all versions of KTANTAN, the update is specified by

$$f_a(L_1) = L_1[x_1] \oplus L_1[x_2] \oplus (L_1[x_3] \cdot L_1[x_4]) \oplus (L_1[x_5] \cdot IR^r) \oplus k_a^r$$
$$f_b(L_2) = L_2[y_1] \oplus L_2[y_2] \oplus (L_2[y_3] \cdot L_2[y_4]) \oplus (L_2[y_5] \cdot L_2[y_6]) \oplus k_b^r.$$

The indices are given by Table 1.

There is a round constant IR^r, 0 or 1, which decides whether a certain bit from L_1 is included in the feedback to L_2 or not, and is taken from a sequence with long period in order to rule out sliding attacks and similar.

For KTANTAN32, one state update is performed per round. In KTANTAN48 and KTANTAN64, there are two resp. three updates per round using the same key bits and round constant. This larger amount of state updates means the state mixing is faster, making our attacks slightly more expensive on the larger versions of KTANTAN. We use KTANTAN32 to describe our attacks but also give the characteristics for the attacks on KTANTAN48/64.

Note how the key bits are added linearly to the state. Only after three clockings will they start to propagate nonlinearly. This gives a very slow diffusion, which we will be able to use in our attacks.

We adopt and refine the notation from [3]: define $\phi_{r_1,r_2}(S, K)$ as the partial encryption that applies rounds $r_1, r_1 + 1, \ldots, r_2 - 1$ to the state S using key K. Similarly, $\phi^{-1}_{r_1,r_2}(S, K)$ applies the decryption rounds $r_2 - 1, \ldots, r_1 + 1, r_1$ to the

state S using key K. This allows us to decompose the full KTANTAN as e.g., $C = \phi_{127,254}(\phi_{0,127}(P, K), K)$.

The final encryption is denoted $C = c_{n-1} \dots c_0$.

2.1 On Bit Ordering and Test Vectors

We denote the key $K = k_{79} \dots k_0$ as in [3]. Test vectors for KTANTAN can be produced by the reference code. As an example, the all-ones key and the all-zeros plaintext produce the ciphertext `0x22ea3988`. Unfortunately, this does not highlight the bit order in the plaintext and, more importantly, the key. For completeness and using the reference code given by the designers, we thus provide the key `0xffffffffffffffffffff`, plaintext `0x00000001`, and ciphertext `0x8b4f0824` to indicate the bit orders involved.

3 (Truncated) Differentials

Differential cryptanalysis was publicly introduced by Biham and Shamir [2] in 1990. The idea is to study how a difference in the plaintext propagates through the state of the encryption. If a partial key is correctly guessed, this property should show up with some probability — ideally one but often very close to one half — while a bad guess should lead to a more random behaviour.

Knudsen [6] extended the technique to truncated differentials, where similar properties are studied only in some part of the state.

In [3], a differential is denoted by $(\Delta P, \Delta K) \rightarrow \Delta S$, where a difference in the plaintext and key gives a difference in the state some number of rounds into the encryption. We adopt and extend this notation. To denote truncated differentials, i.e., differentials where we only know the differences in certain bit positions, we will use a mask and a value denoted [mask : value]. As an example, [`00010a00:00010800`] denotes a known difference in bits 16, 11, and 9. In bits 16 and 11, there is a difference, while there is a bit-equality in bit 9. For the other bits, we do not know or care about the difference. In pseudo-C code, such a mask-value pair could be used to identify a match by

```
if ( ((s1^s2)&mask) == value ) { ... }.
```

In this paper, ΔK always involves only a single bit, so we will name this bit specifically, e.g., as in $(0, k_{32}) \rightarrow$ [`08075080 : 00000080`].

With each (truncated) differential, there is also a probability that it holds. In this paper, we only use differentials with probability one, which means there are only false positives, which can be ruled out by repeated filtering, and no false negatives. As a result, all attacks given in this paper have probability one of succeeding. When we give data complexities, these will be the expected number of samples needed to obtain a unique solution. Similarly, time complexities will account for the work needed to rule out false alarms. We assume that an alarm is raised with probability 2^{-b} for a differential that involves b bits.

Due to the unicity distance, we will always need some extra material in order to find a unique key. This is a fundamental property of KTANTAN as we can

Table 2. The nine most extreme key bits in both directions during encryption. Six bits do not appear before round 111, while six others are not used after round 131.

Key bit	Used first in round	Key bit	Used last in round
k_{13}	109	k_{38}	164
k_{27}	110	k_{46}	158
k_{59}	110	k_{15}	157
k_{39}	111	k_{20}	131
k_{66}	123	k_{74}	130
k_{75}	127	k_{41}	122
k_{44}	136	k_{3}	106
k_{61}	140	k_{47}	80
k_{32}	218	k_{63}	79

only access plaintexts and ciphertexts of 32 to 64 bits, but want to find a key consisting of 80 bits.

4 A Previous Result on KTANTAN

Bogdanov and Rechberger [3] note that some key bits are not used until very late in the cipher, while some others are never used after some surprisingly small number of rounds, see Table 2. Given a plaintext–ciphertext pair, this results in a guess-and-determine attack, where the "determine" part is a meet-in-the-middle: Guess 68 key bits. Of the twelve remaining key bits, six are not used in the first part of the cipher, meaning there are only $2^{12-6} = 2^6$ different states after calculating $\phi_{0,111}$ from the plaintext. Similarly, there are 2^6 possible states after calculating $\phi^{-1}_{132,254}$ from the ciphertext. By checking the 2^{12} combinations for matches, one can find the key. In KTANTAN32, one can use eight bits in the mid-cipher state to judge equality, so false positives should appear with rate 2^{-8}. Some additional plaintext–ciphertext pairs will help rule out the false positives, but they are needed anyway due to the unicity distance.

Bogdanov and Rechberger dub this a 3-subset meet-in-the-middle attack and give similar attacks for KTANTAN48 and KTANTAN64.

4.1 Reformulating the Attack

We note that the last step is not trivial, as the computations that need to be carried out in order to check for matches are similar to calculating the round functions themselves. Further, while the original authors choose to only use eight bits for matching, we have found that one can even use twelve bits, given by the mask `2a03cd44`. This slightly lowers the complexity of the attack as one can expect fewer false positives.

Summing up, we prefer to view the attack as follows:

1. Define $A_f = \{k_{63}, k_{47}, k_3, k_{41}, k_{74}, k_{20}\}$ and $A_b = \{k_{32}, k_{61}, k_{44}, k_{75}, k_{66}, k_{39}\}$.
2. Guess key bits $K \backslash (A_f \cup A_b)$.

Table 3. Probabilistic truncated differentials on the full KTANTAN32.

Differential	Probability
$(0, k_{32}) \rightarrow$ [00020000 : 00020000]	$.687 = .5 + .187$
$(0, k_{32}) \rightarrow$ [40000000 : 00000000]	$.640 = .5 + .140$
$(0, k_{32}) \rightarrow$ [40020000 : 00020000]	$.453 = .25 + .203$

3. Compute 2^6 partial encryptions $m_0, \ldots, m_{63}$ using $\phi_{0,127}$ for each choice of bit assignments for A_f.
4. Compute 2^6 partial decryptions $m'_0, \ldots, m'_{63}$ using $\phi^{-1}_{127,254}$ for each choice of bit assignments for A_b.
5. For the 2^{12} combinations, check twelve specific bits for equality:
   `if ( ((mi^m'j)&0x2a03cd44) == 0 ) { ... }`.
   Alarms will be raised with probability 2^{-12}, so we expect one alarm.
6. Use some additional plaintext–ciphertext pairs to rule out false alarms.

An implementation improvement is to only calculate those 12 bits that we actually need. We have then reached something similar to the original formulation of the attack, with the notable difference that we only perform the computations involved in matching ($\phi_{111,127}$, $\phi^{-1}_{127,132}$) once, during the 2^6-parts. (We can split at any round between and including 123 and 127, and still get twelve known (but different) bit positions to look at, but opted for 127 as it makes both halves equally expensive to calculate.)

5 Related-Key Attacks on KTANTAN32

We first study further how k_{32} enters the key schedule very late. We then formulate our attack idea and derive various attacks that find some parts of the key.

5.1 On the Bad Mixing of k_{32}

Key bit 32 is especially weak as it appears for the first time in round 218 of 254. We have thus studied this bit closer. It is worth noting that if the cipher had used 253 rounds rather than 254, there would have been one ciphertext bit that is linear in k_{32}. That is, there is a 253-round differential $(0, k_{32}) \rightarrow$ [00040000 : 00040000] with probability one. The single bit involved is state bit 18 in Figure 1, i.e., the leftmost bit in L_2. This bit is shifted out of the state in the very last round, so such a probability-one differential is not available on the full KTANTAN. However, there are some high-probability truncated differentials on the full KTANTAN as given in Table 3. We do not exploit these differentials in this paper, but note that they give a very non-random behaviour to the cipher.

5.2 The General Attack Idea

We will present several related-key attacks that recover some or all key bits. The general outline of our attacks can be formulated as follows: We group key bits

into disjoint subsets $A_0, \ldots, A_{l-1}$ of sizes $s_i = |A_i|$, $i = 0, \ldots, l-1$. These subsets do not necessarily need to collectively contain all 80 key bits. Define $s = \sum_i s_i$.

We attack these subsets one after another, i.e., when attempting to find the correct bit assignments for A_j, we assume that we already know the correct bit assignments for A_i, $i = 0, \ldots, j-1$. We then follow this simple outline:

1. Guess the bit assignments for A_j.
2. If the (truncated) differential matches, we have a candidate subkey.
3. If the (truncated) differential does not match, we discard the candidate subkey.

In the first step, we can make 2^{s_j} guesses for the subkey. Note that the last step can be performed without risk, since all our differentials have probability one. Due to this, we can immediately discard large numbers of guesses.

The second step of the attack can however give false positives. As already noted, we assume that an alarm is raised with probability 2^{-b} for a differential that involves b bits. To discard the false alarms, we can recheck the differential on more material.

After finding the key bits specified by $\cup_i A_i$, we can conclude by a brute force for the remaining $80-s$ key bits. The total complexity would be $2^{s_0}+\ldots+2^{s_{l-1}}+2^{80-s}$. However, the different operations in these terms have different costs. All time complexities in this paper will be normalized to KTANTAN calls, and also incorporate the expected increase of calculations due to false positives. We will denote this time measurement t and it will, depending on context, refer to the time required to recover either the full key or only some part of it.

5.3 A First Approach: Finding 28 Bits of the Key

Assume that we have a known plaintext P, and two ciphertexts C_0, C_1, where the difference is that k_{32} has been flipped in the unknown key between the calculations of the two ciphertexts. During the calculations of these two ciphertexts, the first 218 states followed the same development. Only after k_{32} entered could the calculations diverge to produce different ciphertexts.

Bogdanov and Rechberger give the probability-1 differential $(0, k_{32}) \rightarrow 0$ for 218 rounds. We note that this differential can be easily extended into 222 rounds, still with probability 1: $(0, k_{32}) \rightarrow$ 00000008. The flipped bit in ΔS is the linear appearance of k_{32}.

We will use "triplets" consisting of one plaintext and *two* ciphertexts to exploit these differentials. A first attempt to use such a plaintext–ciphertexts triplet in an attack could look like this: We note that there are 42 key bits used when decrypting into round 222, see Appendix B. We guess these bits and denote them by K. Denote by K' the same partial key but with k_{32} flipped. Calculate $S_0 = \phi^{-1}_{222,254}(C_0, K)$ and $S_1 = \phi^{-1}_{222,254}(C_1, K')$. For a correct guess, both ciphertexts will decrypt into the same state $S_0 = S_1$.

However, we will have problems with false positives. The first key bits to enter the decryptions, k_{71} and k_7, will enter into a lot of nonlinearity meaning

Table 4. Key bits recovered in Sections 5.3 and 5.5. In the second set, the 11 reappearing key bits have been underlined.

The 28 key bits guessed and found in Section 5.3, exploiting k_{32}. $\{k_0, k_1, k_2, k_4, k_5, k_7, k_8, k_{11}, k_{12}, k_{14}, k_{16}, k_{17}, k_{22}, k_{27}, k_{29},$ $k_{32}, k_{34}, k_{55}, k_{56}, k_{60}, k_{62}, k_{64}, k_{66}, k_{68}, k_{69}, k_{71}, k_{73}, k_{75}\}$
The 40 key bits guessed and found in Section 5.5, exploiting k_{63}. $\{\underline{k_7}, k_{10}, \underline{k_{11}}, \underline{k_{14}}, k_{15}, \underline{k_{17}}, k_{19}, k_{21}, \underline{k_{22}}, k_{25}, k_{26}, k_{28}, k_{30}, k_{31},$ $\underline{k_{34}}, k_{35}, k_{37}, k_{38}, k_{40}, k_{41}, k_{43}, k_{45}, k_{47}, k_{49}, k_{52}, k_{53}, k_{54},$ $\underline{k_{58}}, \underline{k_{60}}, \underline{k_{62}}, k_{63}, k_{67}, \underline{k_{68}}, \underline{k_{69}}, k_{70}, \underline{k_{71}}, k_{74}, k_{76}, k_{77}, k_{79}\}$

that a wrong guess here should give different partial encryptions S_0, S_1 with high probability. However, the very last bit we guess, k_{37}, will only enter linearly, and if the other 41 key bits are correct, we will have $S_0 = S_1$ no matter how we guess k_{37}.

Generalizing, we realize that the bits which enter the partial decryption "late" will not affect the comparison of S_0 and S_1 at all as they enter only linearly. We have found that there are only 28 key bits that affect the equality between S_0 and S_1. These bits are listed in Table 4.

We thus need to guess 28 bits and for each guess perform two partial decryptions of 32 out of 254 rounds. The total number of round function calls is expected to be $2^{28} \cdot 2 \cdot 32 = 2^{34}$, which corresponds to $2^{34}/254 \approx 2^{26.01}$ full KTANTAN evaluations. Thus the total time complexity of finding 28 bits is $t \approx 2^{26}$. All time complexities in the remainder of the paper will be calculated in this way.

By using brute-force for the remaining key bits, the entire key can be found in time $t \approx 2^{26} + 2^{62} \approx 2^{62}$.

5.4 Making it Faster

Rather than guessing 28 bits at once, we note that we can apply a divide-and-conquer approach to these bits, determining a few bits at a time. This will significantly improve the complexity of the attack. Due to the slow diffusion, we cannot find any truncated differential on 247 rounds or more for our purposes, but for 246 rounds, there is $(0, k_{32}) \rightarrow [\texttt{80050800} : \texttt{00000800}]$. This differential can be used to find three bits, $A_0 = \{k_{11}, k_{66}, k_{71}\}$, in time $t \approx 2^{-0.9}$. (That this work is performed in time less than one unit results from the fact that we only perform a small number of round calculations, compared to the full 254 rounds that are calculated in one time unit.)

We now know these three bits specified by A_0 and attempt to find more bits. There is no useful 245-round differential, but the 244-round truncated differential $(0, k_{32}) \rightarrow [\texttt{20054200} : \texttt{00000200}]$ can be used to obtain one additional key bit, specified by $A_1 = \{k_2\}$, with $t \approx 2^{-0.5}$.

Continuing with such small chunks, we can find the 28 bits with $t \approx 2^{-0.9} + 2^{-0.5} + \ldots \approx 2^{3.0}$. All differentials involved are listed in Table 5.

5.5 Using One Ciphertext and Two Plaintexts

k_{32} appeared very late in the encryption, and we exploited this above. Similarly, k_{63} is only used in the first 80 rounds, meaning that during *de*cryption it shows similar properties. With one ciphertext and two plaintexts, corresponding to a secret key with a flipped k_{63}, we can launch an attack similar to that above, with a truncated differential involving a single bit. With A_0 and using $\phi_{0,43}$, we guess and obtain 40 bits, listed in Table 4, using 40 data and $t \approx 2^{39.44}$. We can then exploit k_{63} for more subsets $A_1, \ldots, A_{15}$ and partial encryptions $\phi_{0,45}, \ldots, \phi_{0,71}$, finding in total 65 bits of the key still with $t \approx 2^{39.44}$. Concluding with a brute force for the remaining bits, we can find the entire key in $t \approx 2^{39.44} + 2^{15} \approx 2^{39.44}$. All subsets, truncated differentials, etc. can be found in Table 6.

5.6 Going in Both Directions for a Practical-Time Key-Recovery

We first go backwards in time $t \approx 2^{3.0}$ to find 28 bits as outlines above. We then go forwards using k_{63}. However, of the 40 bits we needed to guess above, we have learnt 11 while using k_{32}, so we only need to guess 29 bits. We have $t \approx 2^{28.44}$. Finally, we brute force the remaining $80 - 28 - 29 = 23$ bits. The total cost for finding the entire 80-bit key is $t \approx 2^{3.0} + 2^{28.44} + 2^{23} \approx 2^{28.47}$.

A similar attack has been implemented, and requires less than five minutes to recover the complete key using a single processor of a 2 Xeon E5520 (2.26 Ghz, quadcore). Utilizing all eight processors in parallel, the attack runs in 35 seconds. The implementation uses the more naive approaches for finding the first 28 bits, as this is easier to implement and leads to a total time complexity of about $t \approx 2^{28.71}$, which represents a negligible change from the attack described in this section.

We can use k_{63} for finding more key bits, and also exploit several different key bits. This attack does not require a concluding brute force, and recovers the entire key in $t \approx 2^{28.44}$. The truncated differentials involved can be found in Table 5.

In Table 5, note especially the differential on a single state bit involving 29 unknown key bits. This gives a large data requirement in order to rule out false positives, and gives a time complexity which dominates all other parts of the full key recovery attack. Any time improvements we make in other partial key recoveries will only be minor compared to this dominating term.

This leads to the interesting observation that if k_{32} had been *stronger*, i.e., appeared *earlier* in the key schedule, we might have been able to find more key bits at a higher cost ($> 2^3$) using it. This would then have lowered the data and time requirements for utilizing k_{63} which would have made the entire cipher *less* secure. Of course, had both key bits been stronger, the attack would again have become more expensive.

Table 5. The differentials used on KTANTAN32 in this paper. PCC means that the differential is of type $(\Delta P, \Delta K) \rightarrow \Delta S$, where S is the state some rounds into the encryption. Similarly, CPP means a differential $(\Delta C, \Delta K) \rightarrow \Delta S$, extending some rounds into the decryption. (The 'Rounds' column then denote the round into which we decrypt, not the number of decryption rounds.) The '#Key bits' column counts how many key bits need to be guessed. We also give the reduced number of guessed key bits in A_j when we have already acquired a part of the key, $\cup_{i<j} A_i$, by using the differentials found earlier in the table.

Type	Rounds	#Key bits	A_j	Differential
PCC	246	3	$\{k_{11}, k_{66}, k_{71}\}$	$(0, k_{32}) \rightarrow$ [80050800 : 00000800]
PCC	244	4/1	$\{k_2\}$	$(0, k_{32}) \rightarrow$ [20054200 : 00000200]
PCC	243	7/3	$\{k_5, k_7, k_{73}\}$	$(0, k_{32}) \rightarrow$ [1006a100 : 00000100]
PCC	242	8/1	$\{k_4\}$	$(0, k_{32}) \rightarrow$ [08075080 : 00000080]
PCC	241	11/3	$\{k_{32}, k_{68}, k_{75}\}$	$(0, k_{32}) \rightarrow$ [8407a840 : 00000040]
PCC	239	14/3	$\{k_1, k_{34}, k_{69}\}$	$(0, k_{32}) \rightarrow$ [a107ea10 : 80000010]
PCC	238	15/1	$\{k_0\}$	$(0, k_{32}) \rightarrow$ [d087f508 : 40000008]
PCC	237	17/2	$\{k_8, k_{16}\}$	$(0, k_{32}) \rightarrow$ [e847fa84 : 20040004]
PCC	236	19/2	$\{k_{12}, k_{17}\}$	$(0, k_{32}) \rightarrow$ [f427fd42 : 10020002]
PCC	234	20/1	$\{k_{64}\}$	$(0, k_{32}) \rightarrow$ [bd0fff50 : 04008000]
PCC	233	21/1	$\{k_{27}\}$	$(0, k_{32}) \rightarrow$ [de87ffa8 : 02004000]
PCC	232	22/1	$\{k_{29}\}$	$(0, k_{32}) \rightarrow$ [ef47ffd4 : 01002000]
PCC	231	24/2	$\{k_{14}, k_{62}\}$	$(0, k_{32}) \rightarrow$ [f7a7ffea : 00801000]
PCC	230	25/1	$\{k_{60}\}$	$(0, k_{32}) \rightarrow$ [fbd7fff5 : 00400800]
PCC	229	27/2	$\{k_{22}, k_{56}\}$	$(0, k_{32}) \rightarrow$ [fdeffffa : 00200400]
PCC	222	28/1	$\{k_{55}\}$	$(0, k_{32}) \rightarrow$ [ffffffff : 00000008]
CPP	43	40/29	A_{16} (see below)	$(0, k_{63}) \rightarrow$ [00000001 : 00000001]
CPP	45	45/4	$\{k_3, k_9, k_{18}, k_{33}\}$	$(0, k_{63}) \rightarrow$ [00000005 : 00000004]
CPP	46	49/2	$\{k_{20}, k_{24}\}$	$(0, k_{63}) \rightarrow$ [0000000b : 00000008]
CPP	51	52/1	$\{k_6\}$	$(0, k_{63}) \rightarrow$ [0000017f : 00000108]
CPP	55	54/1	$\{k_{51}\}$	$(0, k_{63}) \rightarrow$ [000017ff : 00001080]
CPP	57	57/1	$\{k_{72}\}$	$(0, k_{63}) \rightarrow$ [00085fff : 00084200]
CPP	58	58/1	$\{k_{46}\}$	$(0, k_{63}) \rightarrow$ [0010bfff : 00108400]
CPP	60	59/1	$\{k_{23}\}$	$(0, k_{63}) \rightarrow$ [0042ffff : 00421000]
CPP	61	60/1	$\{k_{48}\}$	$(0, k_{63}) \rightarrow$ [008dffff : 00842000]
CPP	67	62/1	$\{k_{65}\}$	$(0, k_{63}) \rightarrow$ [237fffff : 21080000]
CPP	68	64/1	$\{k_{50}\}$	$(0, k_{63}) \rightarrow$ [46ffffff : 42100000]
CPP	71	65/1	$\{k_{36}\}$	$(0, k_{63}) \rightarrow$ [37ffffff : 10800000]
CPP	83	68/1	$\{k_{78}\}$	$(0, k_3) \rightarrow$ [00000155 : 00000040]
CPP	98	70/1	$\{k_{42}\}$	$(0, k_{41}) \rightarrow$ [000017ff : 00001080]
CPP	102	71/1	$\{k_{57}\}$	$(0, k_{41}) \rightarrow$ [00217fff : 00210800]
CPP	115	72/1	$\{k_{59}\}$	$(0, k_{74}) \rightarrow$ [046955ff : 04214008]
CPP	116	73/1	$\{k_{13}\}$	$(0, k_{74}) \rightarrow$ [08daabff : 08428010]
CPP	118	75/1	$\{k_{39}\}$	$(0, k_{74}) \rightarrow$ [237aafff : 210a0040]
PCC	172	70/2	$\{k_{44}, k_{61}\}$	$(0, k_{61}) \rightarrow$ [00050000 : 00040000]
$A_{16} = \{k_{10}, k_{15}, k_{19}, k_{21}, k_{25}, k_{26}, k_{28}, k_{30}, k_{31}, k_{35}, k_{37}, k_{38}, k_{40}, k_{41}, k_{43}, k_{45}, k_{47}, k_{49}, k_{52}, k_{53}, k_{54}, k_{58}, k_{63}, k_{67}, k_{70}, k_{74}, k_{76}, k_{77}, k_{79}\}$				

Table 6. The attack parameters for finding 65 key bits with $t \approx 2^{39.44}$, exploiting k_{63}.

Type	Rounds	#Key bits	A_j	Differential
CPP	43	40	A_0 (see below)	$(0, k_{63}) \rightarrow$ [00000001 : 00000001]
CPP	45	45/5	$\{k_3, k_5, k_9, k_{18}, k_{33}\}$	$(0, k_{63}) \rightarrow$ [00000005 : 00000004]
CPP	46	49/4	$\{k_2, k_{20}, k_{24}, k_{73}\}$	$(0, k_{63}) \rightarrow$ [0000000b : 00000008]
CPP	47	51/2	$\{k_1, k_{56}\}$	$(0, k_{63}) \rightarrow$ [00000017 : 00000010]
CPP	51	52/1	$\{k_6\}$	$(0, k_{63}) \rightarrow$ [0000017f : 00000108]
CPP	53	53/1	$\{k_8\}$	$(0, k_{63}) \rightarrow$ [000005ff : 00000420]
CPP	55	54/1	$\{k_{51}\}$	$(0, k_{63}) \rightarrow$ [000017ff : 00001080]
CPP	56	55/1	$\{k_{55}\}$	$(0, k_{63}) \rightarrow$ [00002fff : 00002100]
CPP	57	57/2	$\{k_{12}, k_{72}\}$	$(0, k_{63}) \rightarrow$ [00085fff : 00084200]
CPP	58	58/1	$\{k_{46}\}$	$(0, k_{63}) \rightarrow$ [0010bfff : 00108400]
CPP	60	59/1	$\{k_{23}\}$	$(0, k_{63}) \rightarrow$ [0042ffff : 00421000]
CPP	61	60/1	$\{k_{48}\}$	$(0, k_{63}) \rightarrow$ [008dffff : 00842000]
CPP	65	61/1	$\{k_{16}\}$	$(0, k_{63}) \rightarrow$ [08dfffff : 08420000]
CPP	67	62/1	$\{k_{65}\}$	$(0, k_{63}) \rightarrow$ [237fffff : 21080000]
CPP	68	64/2	$\{k_4, k_{50}\}$	$(0, k_{63}) \rightarrow$ [46ffffff : 42100000]
CPP	71	65/1	$\{k_{36}\}$	$(0, k_{63}) \rightarrow$ [37ffffff : 10800000]
$A_0 = \{k_7, k_{10}, k_{11}, k_{14}, k_{15}, k_{17}, k_{19}, k_{21}, k_{22}, k_{25}, k_{26}, k_{28}, k_{30}, k_{31}, k_{34}, k_{35}, k_{37}, k_{38}, k_{40}, k_{41}, k_{43}, k_{45}, k_{47}, k_{49}, k_{52}, k_{53}, k_{54}, k_{58}, k_{60}, k_{62}, k_{63}, k_{67}, k_{68}, k_{69}, k_{70}, k_{71}, k_{74}, k_{76}, k_{77}, k_{79}\}$				

5.7 Minimizing the Data Complexities

When using truncated differentials involving only a few bits, the probabilities of getting false positives are high, which leads to large data requirements. For the forward direction, we can use the 62-round differential $(0, k_{63}) \rightarrow$ [011bffff : 01084000]. It requires guessing 41 bits and the false-alarm probability is 2^{-21}. The total time complexity for obtaining the full key then becomes $t \approx 2^{39.97}$. The data requirement is one and two triplets, respectively, in the backward and forward directions.

6 Attacking KTANTAN48 and KTANTAN64

We summarize our results on KTANTAN32 in Table 7. Similar attacks can be realized on the two other members of the KTANTAN family, i.e., KTANTAN48 and KTANTAN64. The corresponding complexities are found in Table 8 and Table 9, respectively, and the differentials in Appendices C and D.

Complexities have been optimized in both dimensions: using a small amount of related-key data, and using low time complexities.

We give full key-recovery attacks, but also some partial-key recoveries with extremely low time complexities, similar to the $2^{3.0}$ attack on KTANTAN32 for 28 bits. We also give costs on finding the smallest possible set of key bits.

Generally, the first step is done in the backwards direction, exploiting k_{32}. Following this, we switch to the forward direction and k_{63}. For more advanced

Table 7. Characteristics for some attacks on KTANTAN32. We typically first go backwards, exploiting k_{32}, then forwards using k_{63}, then perhaps forwards exploiting several other key bits before reverting to backwards, using k_{61}. Slashes indicate shift of direction, commas separate needed triplets for different flipped key bits. Differentials and other details are found in Tables 5 and 6.

KTANTAN32		80 bits	80 bits	28 bits	3 bits
Low time	Time	$2^{28.44}$	$2^{28.47}$	$2^{3.02}$	$2^{-0.90}$
	Data	$1/29, 1, 1, 1/1$	$1/29$	1	1
Low data	Time	$2^{39.44}$	$2^{39.97}$	as above	as above
	Data	$-/1$	$1/2$	as above	as above

Table 8. Characteristics for some attacks on KTANTAN48.

KTANTAN48		80 bits	36 bits	3 bits
Low time	Time	$2^{31.77}$	$2^{4.73}$	$2^{0.01}$
	Data	3/32	3	3
Low data	Time	$2^{37.34}$	$2^{31.66}$	as above
	Data	1/1	1	as above

attacks, we can use more key bits in the forward direction: k_3, k_{41}, k_{74}. We may then end using more backward calculations on k_{61}. Attacks that require less data are completed through a brute force.

Note that the benefit of using more data quickly becomes very marginal. Thus, the implementation overhead may consume any theoretic advantage of the extremely data-consuming attacks.

6.1 Possible Improvements

We have used a greedy approach for finding the differentials used in this paper. As an example, on $\phi_{0,248}$, there is the truncated differential $(0, k_{32}) \rightarrow [\texttt{00021000} : \texttt{00001000}]$, but due to the slow diffusion we cannot find any key bits using it with probability one. This forces us to use the differential $(0, k_{32}) \rightarrow [\texttt{80050800} : \texttt{00000800}]$ on $\phi_{0,247}$, where three key bits affect the differential so all three bits need to be guessed. We could truncate this truncated differential further to only involve a single bit, possibly allowing us to only guess a single key bit. In this

Table 9. Characteristics for some attacks on KTANTAN64.

KTANTAN64		80 bits	38 bits	13 bits
Low time	Time	$2^{32.28}$	$2^{10.75}$	$2^{10.71}$
	Data	13/17	13	13
Low data	Time	$2^{36.54}$	$2^{30.53}$	as above
	Data	1/1	1	as above

way, we could perhaps partition the 28 bits that can be recovered using k_{32} into 28 subsets $A_0, \ldots, A_{27}$, and reach a very small time complexity for the attack. We have not investigated this optimization as the time complexities are already impressive enough.

Note that for the key recovery attack on KTANTAN32 the time complexity is dominated by exploiting k_{63} to find the 29-bit subkey defined by A_{16} (see Table 5). For this, we already use a one-bit truncated differential so this cannot be improved by the technique outlined above.

7 Comparison to Specification Claims

In the specification of KTANTAN, the authors state the design goal that "no related-key key-recovery or slide attack with time complexity smaller than 2^{80} exists on the entire cipher" [4]. They also claim to have searched for related-key differentials on KTANTAN. However, it appears the approach has been randomized over the huge space of differences in plaintext and key. With hindsight, the authors should have made sure to try differentials where we flip only some small number of plaintext or key bits. This strategy would have been a good choice due to the bitwise and irregular nature of the key schedule coupled with the slow diffusion of the state. If all key bits had been investigated individually, it would have become apparent e.g., that k_{32} could not affect encryptions before round 218, that one state bit in KTANTAN32 only contained this key bit linearly until the very last round, and that there are some high-probability truncated differentials on the full KTANTAN32.

Note that the first reference implementation of KTANTAN provided by the designers used an incorrect key schedule. The pre-proceedings version of [3] only improved the exhaustive search slightly, while with the correct key schedule, i.e., the one described in the design document, the attack eventually published in [3] gave a more significant speedup. As the incorrect key schedule was in a sense better than the intended one, the original search for related-key differentials might have indicated a better behaviour of the cipher than one carried out with the correct key schedule. Still, even on the incorrect key schedule, using low-weight differentials would have alerted the designers to the unwanted behaviour of some key bits.

8 Conclusion

We have presented several weaknesses related to the key schedule of KTANTAN. We first noted how the exceptionally weak key bit k_{32} allowed for a nonrandomness result on KTANTAN32.

As the main result, we then derived several related-key attacks allowing for (partial-)key recovery: With a single triplet, 3 bits can be found in time $2^{-0.90}$ and 28 bits can be obtained in time $2^{3.0}$. Using one triplet in the backward and 29 in the forward direction, the full 80-bit key is recovered in time $2^{28.47}$. Requiring only three triplets, the full key is instead recovered in time $2^{39.97}$. Our

implementation of one of the attacks verifies the general attack idea and the specific results.

Finally, note that none of these attacks are directly applicable to KATAN. The slow diffusion, which allowed for e.g., the $2^{3.0}$-attack on 28 bits, is present also in KATAN, but one needs a weak key bit in order to exploit this.

For the design of future primitives with a bitwise key schedule such as the one in KTANTAN, we encourage designers to carefully study how individual key bits are used, either by specifically ensuring that they are used both early and late in the key schedule, or by investigating all differentials of modest weight.

Acknowledgment

This work was supported by the Swedish Foundation for Strategic Research (SSF) through its Strategic Center for High Speed Wireless Communication at Lund. The author wishes to thank Andrey Bogdanov and Christian Rechberger for their valuable comments, and the anonymous reviewers for their insightful remarks.

References

1. E. Biham. New types of cryptanalytic attacks using related keys. *Journal of Cryptology*, 7(4):229–246, 1994.
2. E. Biham and A. Shamir. *Differential Cryptanalysis of the Data Encryption Standard*. Springer-Verlag, 1993.
3. A. Bogdanov and C. Rechberger. A 3-subset meet-in-the-middle attack: cryptanalysis of the lightweight block cipher KTANTAN. In *Selected Areas in Cryptography—SAC 2010*, volume 6544 of *Lecture Notes in Computer Science*, pages 229–240. Springer-Verlag, 2010.
4. C. De Cannière, O. Dunkelman, and M. Knežević. KATAN and KTANTAN — a family of small and efficient hardware-oriented block ciphers. In *Cryptographic Hardware and Embedded Systems—CHES 2009*, volume 5747, pages 272–288. Springer-Verlag, 2009.
5. L. R. Knudsen. Cryptanalysis of LOKI 91. In J. Seberry and Zheng Y, editors, *Advances in Cryptology—AUSCRYPT'92*, volume 718 of *Lecture Notes in Computer Science*, pages 196–208. Springer-Verlag, 1993.
6. L. R. Knudsen. Truncated and higher order differentials. In B. Preneel, editor, *Fast Software Encryption'94*, volume 1008 of *Lecture Notes in Computer Science*, pages 196–211. Springer-Verlag, 1995.

A The Key Schedule of KTANTAN

r	k_a^r	k_b^r	r	k_a^r	k_b^r	r	k_a^r	k_b^r	r	k_a^r	k_b^r	r	k_a^r	k_b^r	r	k_a^r	k_b^r	r	k_a^r	k_b^r	r	k_a^r	k_b^r
0	63	31	1	31	63	2	31	63	3	15	47	4	14	14	5	60	76	6	40	40	7	49	17
8	35	67	9	54	22	10	45	77	11	58	26	12	37	69	13	74	10	14	69	69	15	74	10
16	53	21	17	43	43	18	71	7	19	63	79	20	30	62	21	45	45	22	11	11	23	54	70
24	28	60	25	41	41	26	3	19	27	38	70	28	60	28	29	25	73	30	34	34	31	5	21
32	26	74	33	20	52	34	9	41	35	2	18	36	20	68	37	24	56	38	1	33	39	2	2
40	52	68	41	24	56	42	17	49	43	3	35	44	6	6	45	76	76	46	72	8	47	49	17
48	19	51	49	23	55	50	15	63	51	14	46	52	12	28	53	24	72	54	16	48	55	1	49
56	2	34	57	4	20	58	40	72	59	48	16	60	17	65	61	18	50	62	5	53	63	10	58
64	4	36	65	8	8	66	64	64	67	64	0	68	65	1	69	51	19	70	23	55	71	47	47
72	15	15	73	78	78	74	76	12	75	73	9	76	67	3	77	55	23	78	47	47	79	63	31
80	47	79	81	62	30	82	29	77	83	26	58	84	5	37	85	10	26	86	36	68	87	56	24
88	33	65	89	50	18	90	21	69	91	42	42	92	5	5	93	58	74	94	20	52	95	25	57
96	3	51	97	6	38	98	12	12	99	56	72	100	16	48	101	33	33	102	3	3	103	70	70
104	60	28	105	41	41	106	67	3	107	71	71	108	78	14	109	77	13	110	59	27	111	39	39
112	79	15	113	79	79	114	62	30	115	45	45	116	59	27	117	23	71	118	46	46	119	13	29
120	42	74	121	52	20	122	41	73	123	66	2	124	53	69	125	42	42	126	53	21	127	27	75
128	38	38	129	13	13	130	74	74	131	52	20	132	25	57	133	35	35	134	7	7	135	62	78
136	44	44	137	73	9	138	51	67	139	22	54	140	29	61	141	11	43	142	6	22	143	44	76
144	72	8	145	65	65	146	50	18	147	37	37	148	75	11	149	55	71	150	46	46	151	77	13
152	75	75	153	70	6	154	61	29	155	27	59	156	39	39	157	15	31	158	46	78	159	76	12
160	57	73	161	34	34	162	69	5	163	59	75	164	38	38	165	61	29	166	43	75	167	70	6
168	77	77	169	58	26	170	21	53	171	43	43	172	7	23	173	30	78	174	44	44	175	9	25
176	18	66	177	36	36	178	9	9	179	50	66	180	36	36	181	57	25	182	19	67	183	22	54
184	13	45	185	10	10	186	68	68	187	56	24	188	17	49	189	19	51	190	7	39	191	14	30
192	28	76	193	40	40	194	1	1	195	66	66	196	68	4	197	57	25	198	35	35	199	55	23
200	31	79	201	30	62	202	13	61	203	10	42	204	4	4	205	72	72	206	48	16	207	33	33
208	51	19	209	39	71	210	78	14	211	61	77	212	26	58	213	21	53	214	11	59	215	6	54
216	12	44	217	8	24	218	32	64	219	64	0	220	49	65	221	18	50	222	37	37	223	11	27
224	22	70	225	28	60	226	9	57	227	2	50	228	4	52	229	8	40	230	0	0	231	48	64
232	32	32	233	65	1	234	67	67	235	54	22	236	29	61	237	27	59	238	7	55	239	14	62
240	12	60	241	8	56	242	0	32	243	0	16	244	16	64	245	32	32	246	1	17	247	34	66
248	68	4	249	73	73	250	66	2	251	69	5	252	75	11	253	71	7						

B Key Bits Used in Round 222 and Forward

0	1	2	4	5	7	8	9	11	12	14	16	17	22	27	28	29	32	34	37	40
48	50	52	54	55	56	57	59	60	61	62	64	65	66	67	68	69	70	71	73	75

C Differentials for KTANTAN48

The differentials used on KTANTAN48 are given in Table 10.

Table 10. Similar to Table 5, this table gives the truncated differentials used on KTANTAN48.

Type	Rounds	#Key bits	A_j	Differential
PCC	246	3/3	$\{k_7, k_{11}, k_{73}\}$	$(0, k_{32}) \rightarrow$ [000000010000 : 000000000000]
PCC	242	7/4	$\{k_2, k_4 k_{32}, k_{71}\}$	$(0, k_{32}) \rightarrow$ [000000010100 : 000000000000]
PCC	241	11/4	$\{k_5, k_{64}, k_{66}, k_{75}\}$	$(0, k_{32}) \rightarrow$ [00000001c040 : 000000000000]
PCC	240	18/7	A_3 (see below)	$(0, k_{32}) \rightarrow$ [000c00007010 : 000000000000]
PCC	239	19/1	$\{k_{17}\}$	$(0, k_{32}) \rightarrow$ [700011c04000 : 000000000000]
PCC	238	20/1	$\{k_{56}\}$	$(0, k_{32}) \rightarrow$ [1c001c701000 : 000000000000]
PCC	237	23/3	$\{k_{12}, k_{14}, k_{60}\}$	$(0, k_{32}) \rightarrow$ [0c7001f1c400 : 000400000000]
PCC	236	24/1	$\{k_{62}\}$	$(0, k_{32}) \rightarrow$ [071c01fc7100 : 000100000000]
PCC	235	25/1	$\{k_{55}\}$	$(0, k_{32}) \rightarrow$ [1c701ff1c040 : 000040000000]
PCC	234	26/1	$\{k_{27}\}$	$(0, k_{32}) \rightarrow$ [871c1ffc7010 : 000010000000]
PCC	233	30/4	$\{k_{29}, k_{54}, k_{61}, k_{67}\}$	$(0, k_{32}) \rightarrow$ [e1c71fff1c04 : 000004010000]
PCC	232	32/2	$\{k_{22}, k_{65}\}$	$(0, k_{32}) \rightarrow$ [f871dfffc701 : 00000100c000]
PCC	230	33/1	$\{k_{48}\}$	$(0, k_{32}) \rightarrow$ [cf871ffffc70 : 000000100c00]
PCC	229	34/1	$\{k_{59}\}$	$(0, k_{32}) \rightarrow$ [f3e1dfffff1c : 000000040300]
PCC	225	36/2	$\{k_{40}, k_{52}\}$	$(0, k_{32}) \rightarrow$ [fff3ffffffff : 000000003000]
CPP	54	53/32	A_{15} (see below)	$(0, k_{63}) \rightarrow$ [000000000002 : 000000000000]
CPP	55	54/1	$\{k_6\}$	$(0, k_{63}) \rightarrow$ [000000000009 : 000000000000]
CPP	57	57/3	$\{k_{23}, k_{46}, k_{51}\}$	$(0, k_{63}) \rightarrow$ [00000000009f : 00000000000c]

$A_3 = \{k_0, k_1, k_8, k_{16}, k_{34}, k_{68}, k_{69}\}$

$A_{15} = \{k_3, k_9, k_{10}, k_{15}, k_{18}, k_{19}, k_{20}, k_{21}, k_{24}, k_{25}, k_{26}, k_{28}, k_{30}, k_{31}, k_{33}, k_{35}, k_{37}, k_{38}, k_{41}, k_{43}, k_{45}, k_{47}, k_{49}, k_{53}, k_{58}, k_{63}, k_{70}, k_{72}, k_{74}, k_{76}, k_{77}, k_{79}\}$

D Differentials for KTANTAN64

The differentials used on KTANTAN64 are given in Table 11.

Table 11. Similar to Table 5, this table gives the truncated differentials used on KTANTAN64.

Type	Rounds	#Key bits	A_j	Differential
PCC	241	13/13	A_0 (see below)	$(0, k_{32}) \rightarrow$ [0000000000000400 : 0000000000000000]
PCC	237	21/8	A_1 (see below)	$(0, k_{32}) \rightarrow$ [0000000704000000 : 0000000000000000]
PCC	236	27/6	A_2 (see below)	$(0, k_{32}) \rightarrow$ [00c000007e800000 : 0000000000000000]
PCC	235	29/2	$\{k_{29}, k_{61}\}$	$(0, k_{32}) \rightarrow$ [f800007fc0100000 : 0000000e00000000]
PCC	234	30/1	$\{k_{22}\}$	$(0, k_{32}) \rightarrow$ [3f00007ff8020000 : 00000001c0000000]
PCC	233	32/2	$\{k_{54}, k_{67}\}$	$(0, k_{32}) \rightarrow$ [c7e0007fff004000 : 0000000038000000]
PCC	232	33/1	$\{k_{65}\}$	$(0, k_{32}) \rightarrow$ [78fc007fffe00800 : 0000000007000000]
PCC	228	34/1	$\{k_{48}\}$	$(0, k_{32}) \rightarrow$ [f8c78fffffffe00 : 0000070038000000]
PCC	226	36/2	$\{k_{40}, k_{50}\}$	$(0, k_{32}) \rightarrow$ [ffe31e7fffffffff8 : 0000000000e00000]
PCC	225	38/2	$\{k_9, k_{52}\}$	$(0, k_{32}) \rightarrow$ [fffc63fffffffff : 00000000001c0000]
CPP	58	55/33	A_{10} (see below)	$(0, k_{63}) \rightarrow$ [0000000000000003 : 0000000000000001]
CPP	59	59/2	$\{k_{46}, k_{51}\}$	$(0, k_{63}) \rightarrow$ [000000000000001f : 000000000000000e]
CPP	69	65/1	$\{k_{36}\}$	$(0, k_{63}) \rightarrow$ [00000407ffffffff : 0000040380000000]

$$A_0 = \{k_2, k_4, k_5, k_7, k_{11}, k_{17}, k_{32}, k_{64}, k_{66}, k_{69}, k_{71}, k_{73}, k_{75}\}$$
$$A_1 = \{k_1, k_{16}, k_{34}, k_{55}, k_{56}, k_{60}, k_{62}, k_{68}\}$$
$$A_2 = \{k_0, k_8, k_{12}, k_{14}, k_{27}, k_{59}\}$$
$$A_{10} = \{k_3, k_6, k_{10}, k_{15}, k_{18}, k_{19}, k_{20}, k_{21}, k_{23}, k_{24}, k_{25}, k_{26}, k_{28}, k_{30}, k_{31}, k_{33}, k_{35}, k_{37}, k_{38}, k_{41}, k_{43}, k_{45}, k_{47}, k_{49}, k_{53}, k_{58}, k_{63}, k_{70}, k_{72}, k_{74}, k_{76}, k_{77}, k_{79}\}$$