On the Key Schedule Strength of PRESENT

Julio Cesar Hernandez-Castro[1], Pedro Peris-Lopez[2]
Jean-Philippe Aumasson [3]

[1] School of Computing, Portsmouth University, UK
[2] Information Security & Privacy Lab, TU-Delft, The Netherlands
[3] NagravisionSA, Cheseaux, Switzerland

Abstract. We present here the results of a playful research on how to measure the strength of a key schedule algorithm, with applications to PRESENT, including its two variants with 80 and 128 bit keys. We do not claim to have discovered any devastating weakness, but believe that some of the results presented, albeit controversial, could be of interest for other researchers investigating this cipher, notably for those working in impossible differentials and related key or slide attacks. Furthermore, in the case of PRESENT, key schedule features shown here may be exploited to attack some of the PRESENT-based hash functions. We carried out a probabilistic metaheuristic search for semi-equivalent keys, annihilators and entropy minima, and proposed a simple way of combining these results into a single value with a straightforward mathematical expression that could help in abstracting resistance to the set of presented analysis. Surprisingly, PRESENT−128 seems weaker than PRESENT−80 in the light of this new measure.

Keywords: Key Schedule, Semi-Equivalent Keys, Annihilators, Entropy Minimization, Simulated Annealing, PRESENT

1 Introduction

The PRESENT block cipher [7] is an ultra-lightweight substitution-permutation network aimed at extremely constrained environments such as RFID tags and sensor networks. Hardware efficiency was one of its most important design goals, and at 1570 GE it really is one of the very few realistic options in such constrained environments for providing an adequate security level. PRESENT works with two key lengths, 80 and 128 bits, and operates over 64-bit blocks with a very simple round scheme which is iterated 31 times. Each round is the composition of three operations: **addRoundKey**, **sboxLayer** and **pLayer**. The **addRoundKey** operation is simply an XOR between the 64-bit roundkey and the State; **sboxLayer** is a 64-bit nonlinear transformation which uses 16 parallel instances of a 4-to-4-bit S-box; **pLayer** is a bitwise permutation of the 64-bit internal state.

Previous works. Due to its very good performance and neat design, partly inspired by SERPENT [8], PRESENT has attracted a lot of attention from cryptanalysts: Wang presented a disputed [5] differential attack [9] against a reduced 16-round variant which requires 2^{64} texts and 2^{65} memory, Albrecht and Cid [11] presented another differential attack using algebraic techniques, with a very similar complexity against 16-rounds of the 80-bit variant, and 2^{113} operations against the 128-bit version with 19 rounds. A saturation attack was presented by Collard and Standaert [12], that is able to recover the key of a 24 round variant with 2^{57} texts and 2^{57} time. Another relevant attack was proposed by Ohkuma [13], where linear approximations and a class of weak keys were used against a 24-round variant requiring $2^{63.5}$ known texts.

Contribution. The contribution of this paper is twofold: we present a number of new results on the key schedule of PRESENT, and then combine these into a single value that abstracts to some point key schedule strength and aims to be useful for other block ciphers, and specifically for comparing between them. As a side result, we conclude that PRESENT-80 seems to have a stronger key schedule than PRESENT-128.

Organization. This paper is organized as follows. In Section 2, the key schedule of PRESENT is briefly introduced. The methodology and parameters used in our experimentation are explained in Section 3. In Section 4, we describe our results regarding semi-equivalent keys, annihilators and entropy minima. Later, we propose in Section 5 a measure to evaluate the strength of a given key schedule algorithm, and provide the corresponding values for PRESENT-80 and PRESENT-128. In Section 6, we extract some conclusions and describe what we consider will be interesting future research lines.

2 The Key Schedule of PRESENT

We focus in the following on the PRESENT key schedule, first on that of its 80-bit variant, later on that of PRESENT-128.

PRESENT-80 The key is stored in a register K and represented as $k_{79}k_{78} \ldots k_0$. At round i the 64-bit round key $K_i = \kappa_{63}\ \kappa_{62} \ldots \kappa_0$ consists of the 64 leftmost bits of the current contents of register K:

$$K_i = \kappa_{63}\kappa_{62} \ldots \kappa_0 = k_{79}k_{78} \ldots k_{16}$$

After extracting K_i, the key register $K = k_{79}k_{78} \ldots k_0$ is updated as described below:

Step 1 $[k_{79}k_{78} \ldots k_1k_0] = [k_{18}k_{17} \ldots k_{20}k_{19}]$
Step 2 $[k_{79}k_{78}k_{77}k_{76}] = \mathrm{S}[k_{79}k_{78}k_{77}k_{76}]$
Step 3 $[k_{19}k_{18}k_{17}k_{16}k_{15}] = [k_{19}k_{18}k_{17}k_{16}k_{15}] \oplus \mathit{round\_counter}$

PRESENT-128 The key is stored in a register K and represented as $k_{127}k_{126} \dots k_0$. At round i the 64-bit round key $K_i = \kappa_{63}\ \kappa_{62}\ \dots\ \kappa_0$ consists of the 64 leftmost bits of the current contents of register K:

$$K_i = \kappa_{63}\kappa_{62} \dots \kappa_0 = k_{127}k_{126} \dots k_{64}$$

After extracting K_i, the key register $K = k_{127}k_{126}\ \dots\ k_0$ is updated (we have **two** calls to **S** in this case) as described below:

Step 1 $[k_{127}k_{126} \dots k_1k_0] = [k_{66}k_{65} \dots k_{68}k_{67}]$
Step 2 $[k_{127}k_{126}k_{125}k_{124}] = \mathrm{S}[k_{127}k_{126}k_{125}k_{124}]$
Step 3 $[k_{123}k_{122}k_{121}k_{120}] = \mathrm{S}[k_{123}k_{122}k_{121}k_{120}]$
Step 4 $[k_{66}k_{65}k_{64}k_{63}k_{62}] = [k_{66}k_{65}k_{64}k_{63}k_{62}] \oplus \mathit{round\_counter}$

3 Methodology

We have used Simulated Annealing in our search for keys with useful properties, after having a limited success in getting similarly powerful results by analytical means. For example, we found that in the case of PRESENT-80, key bits k_{76}, k_{77}, k_{78} do not enter the key schedule Sbox until round 20, and during the 20 first rounds they only appear in 15 of the round keys. Thus the Hamming weight of the differences between round keys is 15 during the first 20 rounds. However, the results we got using Simulated Annealing were way more relevant.

The main advantage of Simulated Annealing is that it allows for a black-box look-up that performs generally much better than a random search, while having very little added computational costs. In every case, we proposed fitness functions that abstracted what kind of property we were looking for, and tune parameters for trying to get the best possible results.

We used our own implementation of a Simulated Annealing algorithm in Python, and followed a bisection method to tune parameters. We ran multiple experiments, each of them taking approximately three hours in our baseline computer. The best keys found were then double tested by a different member of our team, over a different PRESENT implementation.

3.1 Simulated Annealing

Simulated Annealing [1] is a combinatorial optimization technique based on the physical annealing of molten metals. It has been quite successfully used in different cryptanalytic attacks, such as those against the Permuted Perceptron Problem (PPP) [4, 3], or, more recently, Trivium [2].

We briefly describe the technique, closely following the presentation at [4] in the following:

A General Simulated Annealing Algorithm

INPUT: a temperature T, a cooling rate $\alpha \in (0, 1)$, N the number of moves at each temperature, $MaxFailedCycles$ the number of consecutive unsuccessful cycles before aborting, and IC_{Max} the maximum number of temperature cycles before aborting.

ALGORITHM

```
Step 1.: Let $T_0$ be the initial temperature. Increase it until the percentage of
    moves accepted within an inner loop of $N$ trials exceeds some threshold.
Step 2.: Set the iteration count $IC$ to zero, $finished = 0$, $ILaccepted = 0$
    (inner loops since last accepted move), and randomly generate a current
    solution $V_{curr}$.
Step 3.: While not($finished$) do
    Step 3.1.: Inner Loop: repeat $N$ times
        Step 3.1.1.: $V_{new}$=generateMoveFrom($V_{curr}$)
        Step 3.1.2.: Compute cost change $\Delta_{cost} = cost(V_{new}) - cost(V_{curr})$.
        Step 3.1.3.: If $\Delta_{cost} < 0$ accept the move, so $V_{curr} = V_{new}$.
        Step 3.1.4.: Otherwise, generate a random uniform value in $(0, 1) \to u$.
            If $e^{-\Delta_{cost}/T} > u$ accept the move, otherwise, reject it.
    Step 3.2.: If no move has been accepted after 3.1, then $ILaccepted =$
        $ILaccepted + 1$, else $ILaccepted = 0$.
    Step 3.3.: $T = \alpha * T$, $IC = IC + 1$.
    Step 3.4.: $finished = (ILaccepted > MaxFailedCycles)$ or $(IC > IC_{max})$
```

OUTPUT: The state V_{best} with the lowest cost obtained in the search.

We can informally say that Simulated Annealing is a type of Hill Climbing [2], that generally outperforms and has very little added cost over a random search, specially when compared with other much heavier heuristics such as Genetic Algorithms, Particle Swarm Optimization, etc. We have obtained the results presented in next Section with a standard Simulated Annealing algorithm, generally using an initial temperature of or close to $T_0 = 50$, a cooling rate $\alpha = 0.9999$, and a maximum iteration $IC_{max} = 200$.

4 Results

In this section, the strength of the PRESENT key schedule is examined in a combined search for semi-equivalent keys, annihilators, and entropy minimization.

K_1	K_2	KeySchedule Distance
0x8ba018d26545f5d3**4**dd1	0x8ba018d26545f5d3**2**dd1	35
0x60cf1262a6af**5d**01a7fb	0x60cf1262a6af**45**01a7fb	35
0x**8**3b4d3e2f49cbd4d5e2e	0x**a**3b4d3e2f49cbd4d5e2e	35
0x8ee**b**6a18106618d098da	0x8ee**2**6a18106618d098da	35
0xd0c**e**94581e6eda685d77	0xd0c**7**94581e6eda685d77	35
0x9f2**d**24499c081289fe11	0x9f2**8**24499c081289fe11	35
0x87668990280c70b**56**574	0x87668990280c70b**4e**574	34
0xf718fc4**e7**8a82353328a	0xf718fc4**f5**8a82353328a	34
0x6c96cfd0**1**ad1a5ca7900	0x6c96cfd0**7**ad1a5ca7900	34
0xaaf6b7f4d95**26**5eb3188	0xaaf6b7f4d95**02**5eb3188	**32**
0x5a46487f**2**82a052f1b0f	0x5a46487f**8**82a052f1b0f	**32**

Table 1. Semi-equivalent keys for PRESENT-80

4.1 The Search for Semi-Equivalent Keys

There are not equivalent keys for PRESENT-80 or PRESENT-128, because in both cases the PRESENT Key Schedule is an invertible mapping involving all input bits. So we started considering the search for semi-equivalent keys, that is, different keys that produce a very similar (w.r.t. the Hamming weight metric) key expansion.

The existence of semi-equivalent keys could potentially have more grievous consequences if, as it is the case of PRESENT [6], the block cipher is to be used as a hash function, because this could make finding collisions much easier.

The gap between the best pair of semi-equivalent keys we found, and the optimum value of zero (corresponding to a pair of equivalent keys) was uncomfortably close. In particular, we found key pairs (k_1, k_2) with a Hamming distance of 2 that produced very similar key expansions where all 32 round keys were at a Hamming distance of 2 bits or less, including 17 cases of distance 1 and 6 of distance zero. The total accumulated round key distance after 32 rounds of these two keys (shown in Table 1) was only 35.

We were able to obtain a lot more key pairs with these or even better characteristics, including multiple values of 34. The closest we got to a pair of equivalent keys was with key pairs

$$(0xaaf6b7f4d95265eb3188,\ 0xaaf6b7f4d95025eb3188)$$

and

$$(0x5a46487f282a052f1b0f,\ 0x5a46487f882a052f1b0f)$$

that produced very close round keys that differ only in 32 bits, that is, an average difference of a single bit for every round key.

Is this a statistically significant result? Yes, if we randomly flip one bit in the user key, this generates on average (statistics computed over $10,000$ random keys with random flips) 54.995 bits of accumulated changes in their corresponding round keys. This is relatively poor value, as in the case of a perfectly designed key expansion algorithm (i.e. a good hash function) the value should be on overage around $32 \cdot 32 = 1024$ bits. That result clearly shows that the PRESENT key schedule does not present a good degree of diffusion, and is far from having the avalanche effect.

We have found and shown in Table 1 even better results, with distances of 32, 34 and 35. The significance of our results is even clearer if we study the effect of flipping two random bits on a random key, which on average produces (statistics computed again over $10,000$ random keys with random flips but without repetition) 107.4244 bit changes. For three bit differences in the key, the average change in the sub-round keys is 157.2943, for four 204.744 bits, and 249.8925 for five bits. Values so small as 32, 34 and 35 should be considered insufficient for many applications, and too close to the optimum value of zero.

The results herein presented could be useful for other researchers of PRESENT-80 and, although by themselves do not constitute a major weakness, leave a discomforting low gap between our best findings and the global minimum that would lead to a pair of equivalent keys. Even if these pairs do not exist, our findings might easily lead to pseudo-collisions in one of the PRESENT-based hash functions. Furthermore, sparse differences in the key schedule have been successfully used before to cancel differences in the internal state [10].

Even worse results can be found for PRESENT−128, when for example the two keys

0x2a1145cfce0db6e38eaff175d39c90dc

and

0x2a1145cfcf0db6e38eaff175d39c90dc

with a difference of $0x10000000000000000000000$, generate round keys that only differ, as a whole, in 16 bits over 32 rounds. There are many other pairs with similar properties. The averages also reveal similar undesirable properties, with a random flip in a user key averaging only around 40.02 bit changes in the round keys, in contrast with the almost 55 bits modified by PRESENT−80. Worse results are also achieved by two random bit flips to PRESENT−128, with average values of 78.43 bits on the round keys, which is poor compared with the 107.42 of PRESENT−80. These results show that the 80-bit version has a stronger key schedule than the 128 bit variant – at least from the point of view of its diffusion characteristics.

4.2 Global Annihilators

We looked for keys that produced a set of round keys with a very low hamming weight. Such keys may be useful for impossible differential attacks, or for reducing the overall security of the cipher, because of course the key addition phase is way less strong under those low-hamming keys. The most interesting key we found in this regard is $0x862010e680100a028a10$, that generated an extremely low hamming weight of only 401 (for an average, for a random key, of around $32 \cdot 32 = 1024$ bits). In Table $2.A$ we show the round keys corresponding to this annihilator key.

Similarly, for PRESENT−128 we can obtain a value of 433.0 with key 0x484a0 4d32c22f3ae282001901034 81f3 (See Table 2.B).

Round	RoundKey	Hamming weight
0	0x862010e680100a02	15
1	0x14210c4021cd002	15
2	0x28002842188042	11
3	0x340000500050842	10
4	0x20100680000a002	8
5	0x4210040200d0002	9
6	0x280008420080402	8
7	0x400005000108402	7
8	0x100080000a0006	6
9	0x4210000200100005	7
10	0x8800084200004007	9
11	0x1000110001084005	8
12	0x8100020002200027	9
13	0x1100102000400042	7
14	0x82200204000f	9
15	0x1001000010440047	9
16	0x1000220020000200	5
17	0x5001020004400408	8
18	0x50080a0020400081	9
19	0x60102a0101400401	11
20	0xb0020c0205402022	13
21	0x90101600418040a2	13
22	0x80920202c0083b	14
23	0xa102801012404053	14
24	0xd020f42050020244	16
25	0x31015a041e840a0c	19
26	0xe80906202b4083dd	23
27	0x80283d0120c40565	19
28	0x820f700507a02416	21
29	0x70159041ee00a0fa	24
30	0x904e02b2083dcfL	23
31	0x1283e01209c0564e	22

Table 2.A

Round	RoundKey	Hamming weight
0	0x484a04d32c22f3ae	27
1	0x400400320206903e	15
2	0x402128134cb08bce	23
3	0x40001000c8081a40	10
4	0xc00084a04d32c22e	20
5	0x4000004003202068	9
6	0xc80002128134cb09	18
7	0x60000001000c8080	7
8	0xc22000084a04d32e	18
9	0x4080000004003200	6
10	0xc80880002128134e	16
11	0x40020000001000ca	7
12	0x82022000084a04e	12
13	0x4000080000004000	3
14	0xc420808800021282	12
15	0x2000000103	4
16	0xd81082022000084e	14
17	0x4c00000080000000	4
18	0xc860420808800025	13
19	0x8030000002000004	5
20	0xe821810820220005	14
21	0x8600c00000080005	8
22	0xf8a0860420808805	17
23	0x8618030000002005	10
24	0x88e2821810820226	17
25	0x4618600c00000086	12
26	0x4f238a086042080e	20
27	0x4518618030000004	12
28	0xee3c8e2821810827	25
29	0x1014618600c00007	14
30	0x18b8f238a0860427	24
31	0x1c40518618030007	17

Table 2.B

Table 2. Global Annihilators - PRESENT-80 (2.A) & PRESENT-128 (2.B)

4.3 Output Entropy Minimization

When minimizing the output entropy of the key schedule, we obtained the following interesting results for PRESENT−80 (see Table 4.A), which produced an entropy of 4.006811 bits per byte for key $0x62e00e7e01030028e80$. As shown in Table 4.B, the best result we found for PRESENT−128 had a much lower entropy of 3.744336 bits per byte, for key $0x55048c3882841800b8a669e49628e086$.

It is curious that all of the experiments ran with PRESENT−80 did point towards essentially the same key, with a very high correlation between the obtained optima, but this was clearly not the case for PRESENT−128. This may have different explanations, but the simplest one is that we are much closer to the global optima in the case of PRESENT−80 than in the case of PRESENT−128, or/and that there exists a reduced number of global optima for the 80-bit version and multiple ones for the 128-bit version. In any case, all scenarios clearly favor the design of the smaller version in terms of security, at least from this test point of view. The fact that the lowest output entropy found for the key schedule of the 128-bit variant is significantly smaller than the one found for the 80-bit version, after exactly the same computational effort in the search, further strengthens this point.

Round	RoundKeys
0	0x62e00e7e0103002
1	0x1d000c5c01cfc02
2	0xc0003a0018b802
3	0x3f0001800074002
4	0x2e0007e00030002
5	0xd00005c000fc002
6	0xc0001a0000b8002
7	0xf00018000340002
8	0xe0001e000300002
9	0x1c0003c0002
10	0x380002
11	0x2
12	0x6
13	0x4000000000000006
14	0x7000080000000007
15	0x10000e0001000007
16	0x2000020001c00028
17	0xc000040000400030
18	0x5000b80000800001
19	0x6000ca0017000019
20	0xb0000c00194002ea
21	0x9000760001800322
22	0xbb2000ec0003b
23	0xa00c8001764001d3
24	0xd000f40190002ec4
25	0x30075a001e80320c
26	0xe0bb0600eb4003dd
27	0xf0c83c1760c01d65
28	0x800f7e190782ec16
29	0x40759001ecf320fe
30	0x7bb0480eb2003df7
31	0x1c83ef760901d64f

Table 4.A

Round	RoundKeys
0	0x55048c3882841800
1	0x5d14cd3c92c51c10
2	0x55541230e20a1060
3	0x55745334f24b1470
4	0x55555048c3882840
5	0x5555d14cd3c92c50
6	0x55555541230e20a0
7	0x55555745334f24b0
8	0x55555555048c3880
9	0x5555555d14cd3c90
10	0x55555555541230e0
11	0x55555555745334f0
12	0x55555555555048c0
13	0x5555555555d14cd0
14	0x5555555555554120
15	0x5555555555574530
16	0x5555555555555500
17	0x5555555555555d10
18	0x5555555555555550
19	0x5555555555555570
20	0x5555555555555550
21	0xce55555555555550
22	0x5555555555555550
23	0x5839555555555550
24	0x5555555555555553
25	0xae60e55555555553
26	0xa555555555555553
27	0xdfb9839555555553
28	0xd695555555555552
29	0x987ee60e55555552
30	0x9b5a555555555552
31	0xf61fb9839555552

Table 4.B

Table 3. Entropy minimization - PRESENT−80 (A) & PRESENT−128 (B)

5 Measuring the Strength of a Key Schedule

In the following, we propose to combine the presented results in a single measure that could give a general idea of the strength of a given key schedule algorithm, and ease analysis and comparison between different block cipher proposals. We emphasize here that this measure is completely independent of the structured used in the cipher (e.g. Feistel or SP-network) and only dependent on the key schedule function. Although we acknowledge that security is a quite complex concept that can not be fully abstracted in a single value, we however believe that this abstraction could be useful in a number of testing and designing scenarios.

The simplest and most natural way of combining these results in a single value is simply by multiplying them, like in the following formula:

$$S_{KeySchedule} = \frac{-1}{\ln(S_{Annihilators} \cdot S_{EquivalentKeys} \cdot S_{OutputEntropy})} \tag{1}$$

The above expression particularized with the results obtained over the PRESENT variants, produces the following values:

$$S_{PRESENT_{80}} = \frac{-1}{\ln(\frac{401}{1024} \cdot \frac{32}{1024} \cdot \frac{4.006811}{8})} = 0,19628$$

$$S_{PRESENT_{128}} = \frac{-1}{\ln(\frac{433}{1024} \cdot \frac{16}{1024} \cdot \frac{3.744336}{8})} = 0,17304$$

And we take $S_{KeySchedule} = 0$ if any of the $S_j = 0$.

Although this way of combining different strength results into a single value could be improved, it at least provides a simple, quick and easy way of comparing between different algorithms or variants. This is exactly our case, where contrary to our first thoughts it seems clear that the keyschedule of PRESENT−80 is, at least from the point of view of test S, more secure than that of its cousin PRESENT−128.

The overall result is consistent with the general impression extracted from the different tests, which is that the keyschedule of PRESENT−128 is less robust than that of PRESENT−80 despite the two calls to the S-box done in PRESENT−128. It seems these two lookups are not enough, and more should be allowed to offer a similarly strong key schedule.

6 Conclusions

We present a first attempt to measure the strength of a key schedule algorithm in a single and straightforward fashion, allowing meaningful comparisons between different algorithms and different variants of the same algorithm. This could potentially be useful for designing purposes, when a quick way of comparing different decisions could be handy, and for claiming and testing security properties of new block cipher proposals. Improvements to our proposed measure are possible, and we will work on then in future works, but in its current state we have proved it is good enough to analyze in some depth the key schedules of both PRESENT−80 and PRESENT−128. Our results point out to a quite good overall design that only seems slightly worrisome regarding semi-equivalent keys, specially taking into account the proposal of PRESENT as a basis for various lightweight hash functions.

We also found the slightly surprising and maybe controversial result that, in the light of these tests, the PRESENT−80 key schedule seems to be more robust than that of PRESENT−128. We think that this first attempt to measure cryptographic strength combining the results of multiple heuristic lookups opens an avenue for new kinds of more complex analysis that could help in better understanding some of the recent lightweight cryptographic primitives.

On the other hand, it will be overambitious and highly arguable to claim that this single value can be an undisputed measure of key schedule strength. Firstly, overall strength is quite tricky to define. We only aim here to have contributed a first step in this direction. Furthermore, we also acknowledge that looking at the key schedule algorithm in complete isolation as we do here could, in some cases, make little sense because a weaker key schedule does not always lead to a weaker cipher if this is properly accounted for in the design of the round function.

Apart from investigating possible improvements to the key schedule strength measure just proposed, we plan to compare that of the two key schedule algorithms explored here against that of the AES, and other lightweight algorithms like KATAN and KTANTAN [14]. We will also research for more complex ways of measuring the difficulty of the problems associated with finding keys with security-relevant properties, particularly with measures based in problem landscape complexity as in [2].

References

1. S. Kirkpatrick, Gelatt, Vecchi. Optimization by Simulated Annealing. Science, 220 (4598): pp. 671–680, 1983
2. Julia Borghoff, Lars R. Knudsen, Krystian Matusiewicz. Analysis of Trivium by a Simulated Annealing Variant. Proceedings of Ecrypt II Workshop on Tools for Cryptanalysis 2010
3. Lars R. Knudsen and Willi Meier. Cryptanalysis of an Identification Scheme Based on the Permuted Perceptron Problem. In Advances in Cryptology, Eurocrypt '99, pp. 363–374, LNCS 1592, 1999
4. John A. Clark and Jeremy L. Jacob. Fault Injection and a Timing Channel on an Analysis Technique. In L.R. Knudsen (Ed.) Eurocrypt 2002, LNCS 2332, pp. 181–196, 2002
5. Manok Kuman, Pratibha Yadav and Meena Kumari. Flaws in Differential Cryptanalysis of Reduced Round PRESENT. `http://eprint.iacr.org/2010/407`
6. Bogdanov A, Leander G, Paar C, et al. Hash Functions and RFID Tags : Mind the Gap. 2008: pp. 283–299
7. Bogdanov A, Knudsen LR, Leander G, et al. PRESENT: An Ultra-Lightweight Block Cipher. Lecture Notes in Computer Science. 2007, v. 4727, pp. 450–466. Available at: `http://dx.doi.org/10.1007/978-3-540-74735-2_31`
8. R. Anderson, E. Biham, and L. Knudsen, Serpent: A proposal for the Advanced Encryption Standard, First Advanced Encryption Standard (AES) Conference, 1998
9. M. Wang, Differential cryptanalysis of reduced-round PRESENT, AFRICACRYPT 2008, First International Conference on Cryptology in Africa, Casablanca, Morocco, June 11-14, 2008. Lecture Notes in Computer Science, vol. 5023, 2008, pp. 40-49
10. Onur Ozen, Kerem Varici, Cihangir Tezcan and Celebi Kocair. Lightweight Block Ciphers Revisited: Cryptanalysis of Reduced Round PRESENT and HIGHT. In Boyd, Gonzalez-Benito (Eds.) Proceedings of the 2009 Australasian (ACISP) Conference, pp. 90–107, July 1-3, 2009. Lecture Notes in Computer Science LNCS v. 5594
11. M. Albrecht and C. Cid, Algebraic techniques in differential cryptanalysis, Fast Software Encryption International Workshop, FSE 2009, Leuven, Belgium, February 22-25, 2009, Lecture Notes in Computer Science, vol. 5665, 2009, pp. 193-208
12. B. Collard and F. Standaert, A statistical saturation attack against the block cipher PRESENT, CT-RSA, Lecture Notes in Computer Science, vol. 5473, 2009, pp. 195-210

13. Kenji Ohkuma, Weak Keys of Reduced-Round PRESENT for Linear Cryptanalysis, Selected Areas in Cryptography, Lecture Notes in Computer Science, Volume 5867, pp. 249–265, 2009.
14. Christophe De Canniere, Orr Dunkelman and Miroslav Knezevic. KATAN & KTANTAN A Family of Small and Efficient Hardware-Oriented Block Ciphers. Proceedings of CHES 2009, LNCS 5747, Springer, 2009