Better Steady than Speedy: Full break of SPEEDY-7-192

Christina Boura[1], Nicolas David[2], Rachelle Heim Boissier[1], and María Naya-Plasencia[2]

[1] Université Paris-Saclay, UVSQ, CNRS, Laboratoire de mathématiques de Versailles, 78000, Versailles, France
`firstname.lastname@uvsq.fr`
[2] Inria, France
`firstname.lastname@inria.fr`

Abstract. Differential attacks are among the most important families of cryptanalysis against symmetric primitives. Since their introduction in 1990, several improvements to the basic technique as well as many dedicated attacks against symmetric primitives have been proposed. Most of the proposed improvements concern the key-recovery part. However, when designing a new primitive, the security analysis regarding differential attacks is often limited to finding the best trails over a limited number of rounds with branch and bound techniques, and a poor heuristic is then applied to deduce the total number of rounds a differential attack could reach. In this work we analyze the security of the `SPEEDY` family of block ciphers against differential cryptanalysis and show how to optimize many of the steps of the key-recovery procedure for this type of attacks. For this, we implemented a search for finding optimal trails for this cipher and their associated multiple probabilities under some constraints and applied non-trivial techniques to obtain optimal data and key-sieving. This permitted us to fully break `SPEEDY-7-192`, the 7-round variant of `SPEEDY` supposed to provide 192-bit security. Our work demonstrates among others the need to better understand the subtleties of differential cryptanalysis in order to get meaningful estimates on the security offered by a cipher against these attacks.

Keywords: differential cryptanalysis · block ciphers · SPEEDY · security claim · key recovery

1 Introduction

Differential cryptanalysis is a very powerful technique to analyse block ciphers. It was introduced in 1990 by Biham and Shamir who used this method to break the Data Encryption Standard (DES). The idea of this technique applied to block ciphers is to exploit input differences that propagate through the cipher to output differences with a probability higher than what is expected for a random permutation.

Differential cryptanalysis is arguably the most well-known and studied technique in symmetric cryptography. Indeed, in the last 30 years, differential attacks have been applied to analyze a high number of primitives : [6–8, 4, 22, 9, 13, 19], to cite only a few. In parallel, several refinements and generalizations of the basic technique were introduced together with some new dedicated methods. One can for example mention the technique of truncated differentials [17], the use of structures to reduce data complexity (a technique already introduced in [8]), the technique of probabilistic neutral bits [15] or the conditional differential attacks [16]. However, applying differential cryptanalysis on a new cipher is in general a laborious, complex and potentially error-prone procedure. Indeed, combining together the different improvements and techniques for mounting interesting differential attacks is highly non-trivial. This is the reason why the designers of a new primitive provide most of the time only basic arguments on the security of their design against differential attacks. This is done for example by applying the branch-and-bound algorithm to determine the highest number of rounds covered by a single differential trail. Based on this, and without getting into too many details, designers then provide an estimate on the number of rounds that the key recovery steps could reach on top of the differential distinguisher. This estimation is used to state security claims, sometimes conservative, sometimes not, depending on the target application scenario. Examples of such kind of claims exist for almost all modern symmetric designs [10, 3, 12, 1, 2].

In this work we analyze `SPEEDY` against differential attacks. `SPEEDY` is a new ultra-low latency family of block ciphers [18], designed by Leander, Moos, Moradi and Rasoolzadeh. The authors provided in [18] a preliminary analysis that suggested that all versions of this cipher should be immune against this type of attack. However, we demonstrate here that `SPEEDY-7-192` can be fully broken with differential cryptanalysis. Our attack that uses improved techniques for the key-recovery part, demonstrates in practice that a more in-depth analysis of a primitive against differential cryptanalysis is necessary in order to provide precise estimates of its security margin.

Our contribution

We analyzed `SPEEDY`, a new ultra-low latency family of block ciphers [18] against differential attacks. More precisely, we managed to break the full version of `SPEEDY-7-192`, one of the three main variants of this family. This variant iterates over 7 rounds and its designers claimed 192-bit time and data security. Our attack has a time of $2^{187.75}$ and data complexity of $2^{187.27}$, and is thus more than 2^4 times faster than exhaustive search, contradicting therefore the security claim. We shared our results with the designers, that have agreed and acknowledged our attack. This attack is based on a 5.5-round distinguisher and is extended to 7 rounds, therefore it contradicts another claim of the designers : "the attacker cannot add more than one round to extend a distinguisher". Our attack is non-trivial and is based on improved techniques for the key-recovery part. We believe that most of these ideas could be generalized to be applied to differential attacks against other ciphers and we hope that this work can be seen as a step towards a general framework that could help in the future designers precisely estimate the security margin of their design against differential cryptanalysis.

Finally, we provide a brief summary of our differential attacks on the $\mathtt{r} = 5$ and $\mathtt{r} = 6$-round variants of SPEEDY-r-192 even if the attacks on the other variants do not contradict the designers' security claims, but they provide the best known attacks on these variants up to date. A summary of all our attacks together with other third-party cryptanalysis results on SPEEDY is given in Table 1.

Algorithm	# rounds attacked	Ref.	Data	Time (in C_E)	Memory	Security claim $(\mathcal{T}, \mathcal{D})$
SPEEDY-5-192	3	[21]	$2^{17.6}$	$2^{52.5}$	$2^{25.5}$	$(2^{128}, 2^{64})$
SPEEDY-5-192	5	this work	$2^{101.65}$	$2^{107.8}$	2^{42}	$(2^{128}, 2^{64})$
SPEEDY-6-192	5.5	this work	$2^{121.65}$	$2^{127.8}$	2^{42}	$(2^{128}, 2^{128})$
SPEEDY-6-192	6	this work	$2^{121.65}$	$2^{151.67}$	2^{42}	$(2^{128}, 2^{128})$
SPEEDY-7-192	7	this work	$2^{187.27}$	$2^{187.75}$	2^{42}	$(2^{192}, 2^{192})$

Table 1. Summary of SPEEDY cryptanalysis

The rest of the paper is organized as follows. In Section 2, we summarize the classical framework for differential attacks and deduce generic complexity formulas. In Section 3, we present the SPEEDY family of block ciphers and describe our methodology for finding good differential trails. Our attack on SPEEDY-7-192 is given in Section 4. Finally, our results on the other main variants of the SPEEDY family are briefly presented in Section 5. This section also discusses open problems and directions.

2 Differential cryptanalysis

Differential attacks are a very popular chosen-plaintext cryptanalysis technique against symmetric primitives [5]. The invention of this technique in 1990 was devastating for the ciphers of the time, as demonstrated by the breaks of both full FEAL and full DES [6, 8] among others. Similarly to the majority of attacks against block ciphers, differential attacks are built around a distinguisher. A differential distinguisher exploits as a distinguishing property the existence of a pair of differences $(a, b) \in \mathbb{F}_2^n$, where n is the block size, such that the input difference a propagates through some rounds of the cipher to the output difference b with a probability significantly higher than 2^{-n}. This distinguisher can then be extended a few rounds in both directions by adding some rounds that will serve as the key recovery part. In this part, an attacker will guess a reduced part of the key, and using this knowledge will be able to compute the first and/or the last state of the distinguisher in order to check if some plaintext or ciphertext pair follows the differential.

The goal of this section is to provide a global overview of a differential key recovery attack that extends a fixed differential in both directions together with generic formulas representing its time, data and memory complexity.

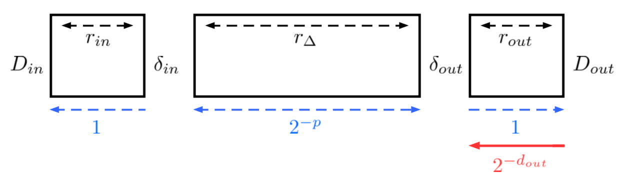


Fig. 1. Differential cryptanalysis context.

We start by considering a differential $\Delta = (\delta_{in}, \delta_{out})$ of probability $P = 2^{-p}$ covering r_Δ rounds. The difference δ_{in} (resp. δ_{out}) then maps to a truncated difference in D_{in}, r_{in} rounds before (resp. D_{out} and r_{out}) with probability 1. We denote by d_{in} (resp. d_{out}) the log_2 of the size of the input (resp. output) difference such that $|D_{in}| = 2^{d_{in}}$ (resp. $|D_{out}| = 2^{d_{out}}$). Note that the attack can be done in both directions (encryption or decryption) and the most interesting direction is determined by the concrete parameters. Without any further improvements, the data and time complexities should be the same in both directions, while the memory complexity if given by the size of one structure ($2^{d_{in}}$ or $2^{d_{out}}$). Similarly to our attack on `SPEEDY-7-192`, we will present a procedure by making calls to the decryption oracle (i.e. by generating ciphertexts). However, the general description of the attack remains unchanged no matter the direction. For this it suffices to replace in what follows "ciphertexts", D_{out} and d_{out} by "plaintexts", D_{in} and d_{in}.

Data complexity In order to have enough data to expect that one pair satisfies the differential, we will use *structures*, as it is often done in differential attacks. A structure is a set of ciphertexts that have a fixed value in the non-active bits, and that take all possible values in the remaining d_{out} bits. This approach permits us to build $(2^{2d_{out}-1})$ pairs inside a structure. The probability to start from a difference in D_{out} and to fall back to a difference δ_{out} is usually $2^{-d_{out}}$. This means that to have one pair that satisfies the differential trail, we need a total of $2^{p+d_{out}}$ pairs that we will obtain by using 2^s structures where s is such that $2^{s+2d_{out}-1} = 2^{p+d_{out}}$, that is $s = p - d_{out} + 1$. Therefore, we need to generate $2^{dout+s} = 2^{p+1}$ ciphertexts and thus the data complexity is $\mathcal{D} = 2^{p+1}$.

Pair sieving Since performing the key recovery phase with all the $2^{p+d_{out}}$ pairs is too costly in general, the attacker will very probably need to perform a sieving step which will permit her to discard pairs that cannot follow the differential trail. This can be done efficiently by just looking at the plaintext corresponding to each ciphertext inside a structure: the attacker will only keep those ciphertext pairs, for which the difference of the corresponding plaintext pairs belongs to D_{in}, i.e. only those pairs that have the same value on the $n - d_{in}$ non-active bits in the plaintext. This can be efficiently done by ordering the list of structures of size $2^{d_{out}}$ with respect to the values of the non-active bits in the plaintext, or even with a hash table, in order to avoid the logarithmic factor of sorting and sieving the table. The total number of pairs that will get through this sieve will be $2^{s+2d_{out}-1-n+d_{in}} = 2^{p-n+d_{in}+d_{out}}$. It is also possible to add an extra sieving step by looking at the concrete differences of active S-boxes. Indeed, by looking at the difference distribution table (DDT) of the primitive's S-box and by taking into account the activity pattern of each one of the active S-boxes, it is possible to further sieve the remaining pairs by removing all those that have an impossible difference on the concerned words. This approach was for example used in [11]. We denote by C_S the average cost of sieving a pair. This cost is in general quite small as it might simply correspond to a table lookup. However this is not always the case, as we will see in the attack of Section 4. Indeed, in our case, this cost will be a little higher than what would it have been with a straightforward approach, as we will consider simultaneously several configurations for the sieving filter.

Key recovery Although all of the pairs that were kept after the sieving step are candidates for having followed the differential, we now want to keep only those such that there exists an associated key that actually leads to the differential. By considering the first and last rounds, and performing partial key guesses that we will merge thanks to efficient list merging algorithms like the ones presented in [20], we can obtain quite low additional factors. In particular, we will denote by C_{KR}, the average cost to perform the key recovery steps per pair. The optimal way of doing this will depend on the round function structure of the analyzed cipher. However this is a step that can typically be done with a small factor. Its goal is to generate a final number of triplets formed by plaintext (or ciphertext) pairs and candidate associated keys that we expect smaller than the original number of pairs (and of the exhaustive search cost), and the cost of finding the secret key given these triplets is not expected to be the complexity bottleneck. In Section 4.4 we show some improved techniques to reduce this cost, and provide an example of such an accelerated key search in the context of `SPEEDY`.

Total time and memory complexity We denote by C_E the cost of one encryption. Taking into account the data generation, the data sieve and the key recovery steps described above, the *time complexity* $\mathcal{T}$[3] is given by

[3] If k is the size of the secret key, for the attack to be valid, the time complexity $\mathcal{T}$ should be smaller than $2^k C_E$.

$$\mathcal{T} = \left(2^{p+1} + 2^{p+1}\frac{C_S}{C_E} + 2^{s+2d_{in}-1-n+d_{out}}\frac{C_{KR}}{C_E}\right)C_E.$$

We present in the next two sections an application of the techniques introduced in Section 2 against the `SPEEDY` family of block ciphers. Section 3 is dedicated to the distinguisher part, while Section 4 describes the key recovery part for `SPEEDY-7-192`.

3 Finding Good Differentials on `SPEEDY`

We start by briefly presenting the specifications of the `SPEEDY` family of ciphers.

3.1 Specifications of the `SPEEDY` family of block ciphers

The `SPEEDY` family of ciphers is a family of lightweight block ciphers introduced by Leander, Moos, Moradi and Rasoolzadeh at CHES 2021 [18]. The main design goal of these primitives was to be fast in CMOS hardware by achieving extremely low latency. This goal was notably reached thanks to the design of a dedicated 6-bit bijective S-box.

There are different `SPEEDY` variants that differ in block size, key size and number of rounds. More precisely, the block cipher `SPEEDY-r-`6ℓ has a block and key size of 6ℓ bits and is iterated over `r` rounds. The internal state is viewed as a $\ell \times 6$ rectangle-array of bits. Following the notation of [18], we will denote by $x_{[i,j]}$, $0 \le i < \ell$, $0 \le j < 6$, the bit located at row i and column j of the state x. Note that all indices start from zero and the zero-th bit or word is always considered to be the most significant one. Furthermore, if there is an addition or a subtraction in the indices of the state, this is done modulo ℓ for the first (row) index and in modulo 6 for the second (column) index.

The default block and key size for `SPEEDY` is 192 bits and this instance is denoted by `SPEEDY-r-192`. It is suggested to iterate this instance over 5,6 or 7 rounds. Next, we provide the specifications of the round function for `SPEEDY-r-192`. Note that for this variant, the state is seen as $(\ell \times 6)$-bit rectangle, with $\ell = 32$.

Round function of `SPEEDY-r-192` The internal state is first initialized with the 192-bit plaintext. Then, a round function $\mathcal{R}_r$ is applied to the state `r` times, where `r` is typically 5, 6 or 7. The round function is composed of four operations: First, `AddRoundKey` ($\mathtt{A}_{k_r}$) XORs the round subkey k_r to the state. Then, the `SubBox (SB)` operation applies a 6-bit S-box to each row of the state. Follows the `ShiftColumns (SC)` operation that rotates each column of the state by a different offset. These two operations (`SB` and `SC`) are repeated twice in an alternating manner. After this, the `MixColumns (MC)` operation multiplies each column of the state by a binary matrix. Finally, a constant c_r is XORed to the state by the `AddRoundConstant (AC)` operation. Note that, for the last round, the last `ShiftRows` as well as the `MixColumns` and the `AddRoundConstant` operations are omitted, while a post-whitening key is XORed to the state. The round function $\mathcal{R}_r$ for the rounds $0 \le r < \mathtt{r} - 1$ while also for the round $\mathtt{r} - 1$ are depicted in Figure 2.

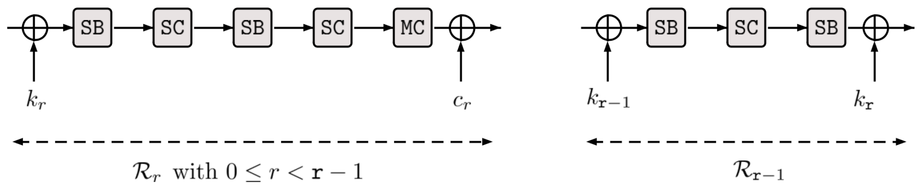


Fig. 2. The round function of `SPEEDY-r-192` for the first $\mathtt{r} - 1$ rounds (left) and the last round (right).

In the rest of our paper, we assume that the input (resp. output) to each of the described operations is a vector x (resp. y) $\in \mathbb{F}_2^{32\times 6}$.

- `AddRoundKey` ($\mathtt{A}_{k_r}$): The 192-bit round key k_r is XORed to the internal state. Hence,
$$y_{[i,j]} = x_{[i,j]} \oplus k_{r[i,j]}.$$
- `SubBox (SB)`: A 6-bit S-box S is applied to each row of the state. More precisely, for each row i, $0 \le i < 32$, `SB` operates as follows:
$$(y_{[i,0]}, y_{[i,1]}, y_{[i,2]}, y_{[i,3]}, y_{[i,4]}, y_{[i,5]}) = S(x_{[i,0]}, x_{[i,1]}, x_{[i,2]}, x_{[i,3]}, x_{[i,4]}, x_{[i,5]}).$$
The table representation of the S-box S is given in Table 2.

x	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S(x)	4	16	22	17	12	20	14	53	7	51	31	23	3	35	11	39
x	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
S(x)	36	52	6	21	8	28	10	61	37	33	13	5	1	41	9	45
x	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
S(x)	0	24	18	25	44	54	46	55	2	58	26	30	32	34	40	38
x	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
S(x)	48	60	19	29	56	62	59	63	50	42	27	15	49	43	57	47

Table 2. Table representation of the 6-bit S-box S

- `ShiftColumns (SC)`: This operation rotates the j-th column of the state, $0 \le j < 6$, upside by j bits:
$$y_{[i,j]} = y_{[i+j,j]}.$$

- `MixColumns (MC)`: The `MC` operation of `SPEEDY` applies column-wise and is based on a cyclic binary matrix $\alpha = (\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5, \alpha_6)$ whose values depend on the number of rows ℓ:

  $$y_{[i,j]} = x_{[i,j]} \oplus x_{[i+\alpha_1,j]} \oplus x_{[i+\alpha_2,j]} \oplus x_{[i+\alpha_3,j]} \oplus x_{[i+\alpha_4,j]} \oplus x_{[i+\alpha_5,j]} \oplus x_{[i+\alpha_6,j]}.$$

  Recall that the additions $i + \alpha_*$ are considered $\mod \ell$.
  For $\ell = 32$, $\alpha = (1, 5, 9, 15, 21, 26)$.
- `AddRoundConstant` ($\mathtt{A}_{c_r}$): The 192-bit round constant c_r is XORed to the internal state. Hence,

  $$y_{[i,j]} = x_{[i,j]} \oplus c_{r[i,j]}$$

  As this operation is not relevant to our analysis we omit the description of the constant values.

Key Schedule The 192-bit master key of `SPEEDY-r-192` is loaded to the state of the first round key k_0. To obtain the next round key, the key schedule consists in simply applying a bit-permutation `PB`. Hence,

$$k_{r+1} = \mathtt{PB}(k_r), \text{ with } k_{r+1[i',j']} = k_{r[i,j]},$$

such that

$$(i', j') := P(i, j) \text{ with } (6i' + j') \equiv (\beta \cdot (6i + j) + \gamma) \mod 6\ell,$$

where β and γ are parameters depending on the block length of the cipher and that satisfy the condition that $\gcd(\beta, 6\ell) = 1$. For `SPEEDY-r-192`, the parameters $\beta = 7$ and $\gamma = 1$ are suggested, leading to the permutation P described in Table 3.

Security Claims The authors made security claims for the three main versions of `SPEEDY-r-192`. For the 5-round version the authors expect no attack with complexity better than 2^{128} in time when data complexity is limited to 2^{64}. On the other hand, `SPEEDY-6-192` should achieve 128-bit security, while `SPEEDY-6-192` is expected to provide full 192-bit security.

3.2 Differential properties of `SPEEDY`

We describe in this section the differential properties of the non-linear and linear layer of `SPEEDY`.

Differential properties of the S-box The `SPEEDY` family of cipher employs a 6-bit S-box S whose differential uniformity is $\delta_S = 8$. This means that the highest probability of a differential transition through S is 2^{-3}. One particularity of this S-box that we exploit in our attacks is that almost all 1-bit to 1-bit differential transitions are possible. Moreover, these minimal weight transitions often have

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
P(i)	1	8	15	22	29	36	43	50	57	64	71	78	85	92	99	106
i	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
P(i)	113	120	127	134	141	148	155	162	169	176	183	190	5	12	19	26
i	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
P(i)	33	40	47	54	61	68	75	82	89	96	103	110	117	124	131	138
i	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
P(i)	145	152	159	166	173	180	187	2	9	16	23	30	37	44	51	58
i	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
P(i)	65	72	79	86	93	100	107	114	121	128	135	142	149	156	163	170
i	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
P(i)	177	184	191	6	13	20	27	34	41	48	55	62	69	76	83	90
i	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
P(i)	97	104	111	118	125	132	139	146	153	160	167	174	181	188	3	10
i	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
P(i)	17	24	31	38	45	52	59	66	73	80	87	94	101	108	115	122
i	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143
P(i)	129	136	143	150	157	164	171	178	185	0	7	14	21	28	35	42
i	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
P(i)	49	56	63	70	77	84	91	98	105	112	119	126	133	140	147	154
i	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
P(i)	161	168	175	182	189	4	11	18	25	32	39	46	53	60	67	74
i	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191
P(i)	81	88	95	102	109	116	123	130	137	144	151	158	165	172	179	186

Table 3. The bit-permutation P for SPEEDY-r-192 with $\beta = 7$ and $\gamma = 1$.

α/β	1	2	4	8	16	32
1	-	1	3	2	1	1
2	4	3	4	4	-	-
4	1	1	3	3	1	1
8	1	3	-	2	3	-
16	2	2	4	4	2	1
32	2	4	2	4	-	2

Table 4. Summary of all the 1-bit input differences α to 1-bit output differences β. The corresponding probability can be obtained by multiplying the coefficients of the table by 2^{-5}. The symbol - means that the corresponding transition is impossible.

a relatively high probability. Table 4 summarizes all these transitions, together with their corresponding probability.

The entire Difference Distribution Table (DDT) of S is provided as supplementary material. Another particularity is that 1-bit to 1-bit differential transitions can be chained within one round through the `SB` $\circ$ `SC` $\circ$ `SB` operation. All of them are possible and three of them achieve the maximum probability of 2^{-6}.

Differential properties of the `MixColumns` operation The branch number of the `MC` operation is 8, which is the maximum possible value for the vector α chosen. As the maximum differential probability over 1 round is 2^{-6}, this means that an upper bound on the probability of any differential transition over two rounds is $(2^{-6})^8 = 2^{-48}$. The inverse `MixColumns` operation is defined with the vector

$$\alpha^{-1} = (4, 5, 6, 7, 10, 12, 14, 15, 16, 18, 19, 20, 21, 22, 23, 24, 25, 28).$$

This means in particular, that a column with a single active bit, will lead after the inverse of the `MixColumns` operation to 19 active bits, while a column with two active bits will be transformed after the inverse `MixColumns` to a column with at least 12 active bits.

3.3 Searching for good differential trails

We describe in this section the methodology we followed to find the trails used in our attacks. Our idea was to precompute at first all good one-round trails and then chain them to create longer trails with high probability.

Searching for good one-round trails. Let M be the matrix used in the `MixColumns` operation. In order to find good one-round trails, we first computed and stored all ordered pairs of columns $(x, M(x)) \in \mathbb{F}_2^{32} \times \mathbb{F}_2^{32}$ such that both columns x and $M(x)$ have at most 7 active bits each. This led to a total of 5248 pairs $(x, M(x)) \in \mathbb{F}_2^{32} \times \mathbb{F}_2^{32}$. However, these 5248 pairs can be divided into 164 equivalence classes, each equivalence class corresponding to the 32 rotations of a different activity pattern inside a column. We then stored in a table $\mathtt{T}$ one representative per equivalence class and used these pairs to precompute and store all 1-round trails satisfying some particular criteria. To describe this phase we need to introduce the following notation. Let `st[0]` be the initial state for our computation. We denote by `st[1]` the resulting state after applying `MC` to `st[0]`, `st[2]` the state after applying `SB` to `st[1]`, `st[3]` the state after applying `SC` to `st[2]`, `st[4]` the state after applying `SB` to `st[3]`, `st[5]` the state after applying `SC` to `st[4]` and finally `st[6]` the state after applying `MC` to `st[5]`:

$$\mathtt{st[0]} \xrightarrow{\mathtt{MC}} \mathtt{st[1]} \xrightarrow{\mathtt{SB}} \mathtt{st[2]} \xrightarrow{\mathtt{SC}} \mathtt{st[3]} \xrightarrow{\mathtt{SB}} \mathtt{st[4]} \xrightarrow{\mathtt{SC}} \mathtt{st[5]} \xrightarrow{\mathtt{MC}} \mathtt{st[6]}.$$

We computed all such propagations (`st[0]`, `st[6]`) satisfying the following conditions:

- `st[0]` has a single active column $\mathtt{c}_0$ such that $(\mathtt{c}_0, M(\mathtt{c}_0)) \in \mathtt{T}$,
- `st[5]` has a single active column $\mathtt{c}_5$ such that $(\mathtt{c}_5, M(\mathtt{c}_5)) \in \mathtt{T}$,
- `st[2]` has at most two active bits per row,
- the probability of the trail (`st[0]`, `st[6]`) is strictly higher than 2^{-49}.

For all trails satisfying the above conditions, we stored in a table the states (`st[0]`, `st[5]`) together with the probability of the corresponding trail. We obtained a total number of 48615 one-round trails, which we stored. Note that each trail can be be shifted column-wise to form 32 other valid one-round trails. Thus, in total, there are 1555680 one-round trails which satisfy our criteria.

We now justify the criteria used for computing these 1-round trails. Our main constraint was computing time, as considering all 1-round trails is computationally infeasible. Furthermore, as we wanted to store the trails and reuse them, memory needed to be reasonable as well. Limiting the computation to states with a single active column before and after each `MixColumns` computation is a reasonable assumption, as states with more active columns would lead by the inverse `ShiftColumns` operation to many active rows. Furthermore, by doing initial experiments for computing long trails, we noticed that all good trails found never had more than 7 active bits in a column. This can be explained by the fact that more active rows naturally lead to lower probability transitions through the `SubBox` operation. We then limited the transition through the first `SubBox` operation to only transitions from rows with Hamming weight one to rows with Hamming weight at most two. While transitions activating in the output more bits per row can still lead to good trails respecting the other criteria, only a small proportion of these transitions does so, while the computational gain for not considering them is huge. Finally, we limited the probability of the trails to 2^{-49} in order not to have to store too many trails for the second phase. This particular bound came from our initial experiments, were we noticed that the probability of all 1-round trails that were part of the longer trails we found, had probability strictly higher than this bound.

We claim by no means that the chosen criteria lead to all the one-round trails that could be part of optimal longer trails, however we believe that our strategy is a reasonable trade-off between optimality and efficiency.

Searching for longer trails. In a second step we used the precomputed 1-round trails to create longer ones. To do so, we started by chaining our precomputed one-round trails in order to obtain r-round trails.

To begin, we exhaustively ran through all the precomputed one-round trails and searched for the ones that can be chained. Recall here that the starting state and ending state of each round trail are the states just before the `MixColumns` application. The chaining condition is very simple and consists in simply verifying that the final state of a one-round trail is the same as a column-wise rotation of the starting state of the following one by an integer $0 \leq \iota < 32$. Note that when $\iota \neq 0$, the full one-round trail concerned is also rotated column-wise. Also note that doing so, we only obtain an element of an equivalence class modulo the column-wise rotation. In order to make our search efficient, we first sorted the states by Hamming weight and active column coordinate of their initial and final state. Following this procedure, we found 1437464 2-round trails, each of them giving by rotation another 32 valid 2-round trails. We followed a similar procedure to obtain 42991700 3-round trails which can also be rotated column-wise to obtain 32 times more valid 3-round trails. We were not able to use the same technique for r-round trails with $r \geq 4$, as the number of obtained 3-round trails was too high. Thus, we used the best 3-round trail (both in terms of probability and of its adaptability to the key recovery step) and chained it with potential one-round trails so as to obtain 4-round trails of interest.

From now on, for each r-round trail, we use the following notation. Let $\mathtt{st}^{\mathtt{start}}[\mathtt{k}]$ (resp. $\mathtt{st}^{\mathtt{end}}[\mathtt{k}]$) be the starting state (resp. the ending state) of each one-round trail composing the r-round trail, for $0 \leq k < r$. Denote also by $\mathtt{c}_{\mathtt{start}}$ the active column of $\mathtt{st}^{\mathtt{start}}[0]$ and by $\mathtt{c}_{\mathtt{end}}$ the active column of $\mathtt{st}^{\mathtt{end}}[\mathtt{r}-1]$. Let w_0 be the Hamming weight of $\mathtt{c}_{\mathtt{start}}$ (i.e. the number of active bits in $\mathtt{c}_{\mathtt{start}}$) and let w_1 be the Hamming weight of $M(\mathtt{c}_{\mathtt{end}})$, where M is the matrix used in `MixColumns`. Finally denote by P_k, the probability of the round k, for $0 \leq k < r$. The probability of the r-round trail, that we will call from now on *core trail*, is then given by $P_0 \times P_1 \times \cdots \times P_{r-1}$.

Extending the core trail. To build our attack, we need to choose an r-round trail that will be extended one round backwards and half a round forwards as shown in Figure 3. In this section, we describe the criteria that we used to select an r-round trail that is likely to result in a good $(r+1.5)$-round trail. The resulting $(r+1.5)$-round trail must have good probability, but also needs to be adapted to our key recovery step. In particular, as we will argue in detail in Section 4, it is important to have differentials that will allow for efficient sieving in the plaintext. In particular, it is desirable that the $(r+1.5)$-round trail we construct has a sufficient number of inactive rows on the plaintext.

First, as described above, we need to make sure that the r-round trail selected leads to a $(r+1.5)$-round trail with good probability. For a r-round trail, the probability of the resulting $(r+1.5)$-round trail can be upper-bounded by $2^{-(w_0+1)\times 3} \times P_0 \times P_1 \times \cdots \times P_{r-1} \times 2^{-w_1 \times 3}$. Indeed, if w_0 is the Hamming weight of $\mathtt{c}_{\mathtt{start}}$, then by computing backwards one round there will be at least

$w_0 + 1$ active S-boxes. As the highest probability transition through an S-box has probability 2^{-3}, the highest possible probability of this prepended round will be $2^{-(w_0+1)\times 3}$. In the same way, if w_1 is the Hamming weight of $M(\mathtt{c}_{\mathtt{end}})$, then there will be exactly w_1 active S-boxes through the first S-box layer of the next round. Thus, the probability of the appended half round will be at most $2^{-w_1\times 3}$. We generated all possible r-round core trails following the procedure described above and kept the ones providing high estimated probabilities.

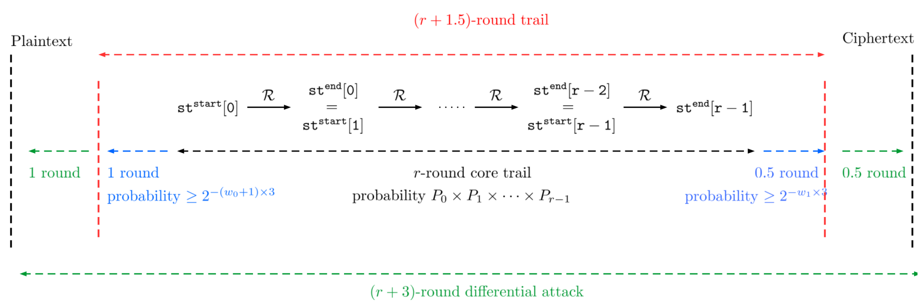
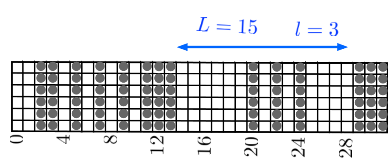


Fig. 3. Generating $(r + 1.5)$-round trails from core r-round trails and extending them to mount $(r + 3)$-round attacks on SPEEDY.

Second, we want the r-round trail selected to lead to a $(r + 1.5)$-round trail that has a significant number of inactive rows on the plaintext in order for the sieving step to be efficient. First, consider the initial state of the r-round trail. The rows that are active in this state are exactly the rows that will be active in the state that follows the first SC operation in round 0 of the $(r + 1.5)$-round trail. To achieve better sieving, we want the transition from this state through $\mathtt{SC}^{-1} \circ \mathtt{SB}^{-1}$ to lead to an initial state of the $(r + 1.5)$-round trail that has low Hamming weight. To achieve this, not only the number of active rows but also the way those are distributed inside this state play a role for the efficiency of the sieving procedure. Let L be the size of a block of consecutive rows, where all rows are non-active except for l out of them. An example of such a state is shown below with $L = 15$ and $l = 3$.



Large values of L combined with small values of l naturally lead to better complexities. Indeed, we can carefully control the l active rows with some probability at a given cost. By doing so, we can generate a number of inactive rows in the plaintext as high as $L - 5 - l$, thus leading to a sieving of $2^{-[(L-5-l)]\times 6}$.

Using the above criteria, we selected an r-round trail, which we then extended in two times, starting first by appending a round backwards. This led to an

$(r+1)$-round trail. Then, to further improve the probability of our trail $(r+1)$-round trail, we relied on the technique of multiple differentials.

3.4 Multiple differentials

The technique of multiple differentials consists in considering multiple $(r+1)$-round differential trails that all have the same input and output difference. To make the description of our technique simpler, we will describe how we built our multiple differentials in the case of our 7-round attack. In this case, $r = 4$. For our 7-round attack, the chosen 4-round core trail is the one displayed in red in Figure 4. This trail has probability $2^{-161.15}$. As shown in Figure 4, we extended it by one round backwards and obtained a 5-round trail of probability $p_{main} = 2^{-170.56}$. We call this trail the *main trail*. Note that it is possible to extend the 4-round core trail backwards with probability 2^{-6} for one round. However, this propagation, due to the diffusion properties of the inverse `MixColumns` transformation would lead to a column with 19 active bits (see Section 3.2). Such a scenario would have complicated the key-recovery phase and was not retained.

We limited our search to trails with probability smaller or equal to $p_{max} = p_{main} \times 2^{-25}$. Our new trails must thus verify that

- their input difference is such that the bits of coordinate

$$(\mathtt{i},\mathtt{j}) \in \{\mathtt{1},\mathtt{3},\mathtt{5},\mathtt{7},\mathtt{8},\mathtt{9},\mathtt{16},\mathtt{18},\mathtt{20},\mathtt{25},\mathtt{26},\mathtt{27},\mathtt{30},\mathtt{31}\} \times \{\mathtt{4}\}$$

  are active, whilst the other bits are inactive in the first state of Figure 4;
- their output difference is such the bits of coordinate

$$(\mathtt{i},\mathtt{j}) \in \{\mathtt{8},\mathtt{9},\mathtt{12},\mathtt{14},\mathtt{18},\mathtt{24},\mathtt{26}\} \times \{\mathtt{3}\}$$

  are active, whilst the other bits are inactive in the second state surrounded by red in Figure 4.

To build our new trails, we rely on an algorithm that operates round by round.

Initial round. We start by building a list of potential initial one-round trails. We denote the initial state by `st[0]`, the state after the application of `MC` by `st[1]`, and so on so forth as we did when constructing our one-round trails. We construct our initial one-round trails in a similar fashion to the way we constructed the one-round trails used to build our main trail. More precisely, we want our potential initial one-round trails to satisfy the following conditions:

- `st[0]` verifies the input condition;
- `st[5]` has a single active column $\mathtt{c}_5$ such that $(\mathtt{c}_5, M(\mathtt{c}_5)) \in \mathtt{T}$;
- `st[2]` has at most two active bits per row.

In order to make the search more efficient, we added constraints on these initial round trails' probability and Hamming weight, using the fact that $(\mathtt{st}[0], \mathtt{st}[6])$ must belong to a larger 5-round trail such that the probability of this larger trail is at most p_{max}. We will not describe these constraints in detail as they are very similar to previous techniques we used to build trails of reasonable probability. We obtained 10 potential initial round trails. Because of the second condition above, these new trails can be chained to our previously computed one-round trails. This property will be used to build our multiples.

Chaining the initial round. In order to find trails that satisfy our truncated differential constraints, we must now chain the potential initial round trails to the previously computed one-round trails. We do so in two steps in order for the chaining to be computationally feasible.

1. We chain the 2-round trails pre-computed to the potential initial one-round trails to form potential initial three-round trails. We get 11311 such 3-round trails.
2. We chain these potential initial 3-round trails to the previously computed 2-round trails to obtain 5-round trails.

We found 222 5-round trails that matched all our criteria. By adding their corresponding probabilities, we found a final probability of $2^{-169.87}$. As one can notice, using multiple differentials permits to improve the probability of the r-round differential, but this improvement is not as important as one would have expected by the number of found trails. This is due to the fact that all of the additional trails found had unfortunately quite bad probabilities compared to the main one.

5.5-round differential trail We describe now the 5.5-round differential trail we used to attack `SPEEDY-7-192` in the following section. This trail is depicted in Figure 4.

As stated before, the 5-round trail has probability $2^{-170.56}$, which is improved to $2^{-169.87}$ by using multiple trails. We then extended this differential 0.5 round forwards. For this step we followed a particular approach. To go through the last S-box layer of the distinguisher part (see the before last state of Figure 4) an attacker has several choices. One extreme would be to fix to some concrete output value the transitions through all active S-boxes. This comes at a cost of a certain probability, but if we choose the transitions carefully we can guarantee very few active rows on the ciphertext. The other extreme is to consider truncated output differences for all the active S-boxes of this state. Thus the transition through the `SubBox` layer happens with probability 1, but almost all rows will be active in the output leading to very large structures of ciphertexts. What we decided to do is a trade-off between these two scenarios. More precisely, we decided to fix the transition `0x4` $\rightarrow$ `0x8` for the actives S-boxes of rows 4, 12 and 30 and to allow more transitions for the S-boxes of rows 21 and 25. The choice of these two rows comes from the fact that after the `SC` operation, these two S-boxes activate some common rows. Our goal was to activate at most 7 rows after the `SC` operation (last state of Figure 4) and for this we computed the highest probability to have at most 4 rows active between rows 16 and 25 after `SC`. We exhausted all possible configurations and we found the best one to be the one having the rows $19, 21, 23$ and 25 active after `SC`. This corresponded to force the output difference of the S-box of row 21 to be of the form `(*,0,*,0,0,0)` and the output difference of the S-box of row 25 to be of the form `(*,0,*,0,*,0)`, where `*` means that the corresponding bit is potentially active. The probability then to start from any difference of the above form in rows 21 and 25 and to activate at most the rows $19, 21, 23$ and 25 after the `SC` is $2^{-3.41}$. This fact, together with the probability of $2^{-3.41}$ for the transition `0x4` $\rightarrow$ `0x8` for the other three active rows, gives a total probability of $2^{-13.64}$.

To summarize, as can be seen from Figure 4, our 5.5-round trail has then a total probability of

$$2^{-169.87} \times 2^{-13.64} = 2^{-183.51}.$$

4 Attack on `SPEEDY-7-192`

`SPEEDY-7-192` is the variant of the `SPEEDY` family suggested for applications where a security of 192 bits is needed. We show in this section, by using the techniques and ideas introduced earlier, how to recover the secret key of this version with less than 2^{192} encryptions. In addition, we will propose two ideas that will allow us to optimize the complexity of the attack: one, already used for instance in [11], is to not consider the rounds as blocks regarding their treatment with respect to the differential distinguisher or the truncated part, but include some row transitions in the differential and let the rest go as truncated in the same round which we will apply in the input and output of the attack; the other is to consider the detailed equations over two rounds with merging techniques that will allow us to optimize the complexity of the key guessing part.

Our attack has a data complexity of $2^{187.27}$, a time complexity of $2^{187.82}$ and a memory complexity of 2^{42} and contradicts thus the designers' security claim for this variant, as has been acknowledged by them. More importantly, this cryptanalysis highlights that the security margin for this variant was over-estimated. Our attack uses the differential found with the ideas from Section 2 and the implemented method described in Section 3.3. As described before, the main differential trail depicted in Figure 4 covers 5.5 rounds and its probability, when taken together with its associated multiple trails described in Section 3.4, is $2^{183.51}$. The trail of Figure 4 can then be extended one round backwards and half a round forwards as shown in Figure 5, to finally cover 7 rounds. This fact contradicts a particular statement of the designers that wrote that a one-round security margin for the key-recovery part should be sufficient.

4.1 Trade-off between differential probability and efficient sieving

Our attack is performed in the decryption direction. The first step is to generate a number of relevant ciphertexts to implement the attack. If we impose no extra

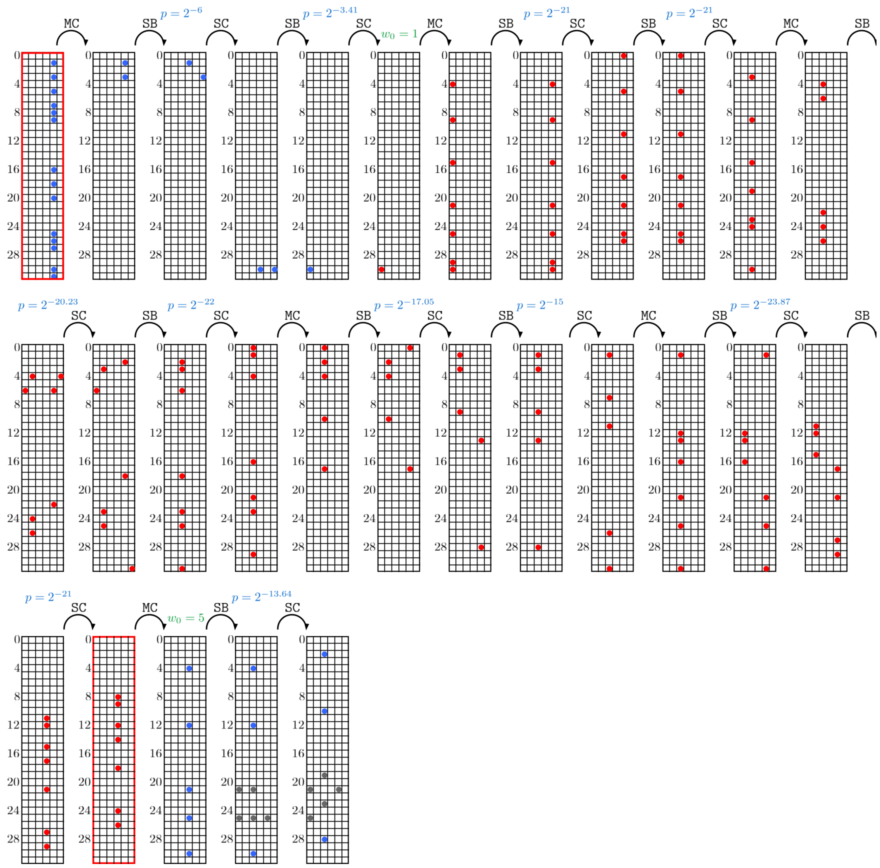


Fig. 4. 5.5-round differential trail used to attack `SPEEDY-7-192`. The red part corresponds to the 4-round core trail, while the blue part corresponds to the 1.5-round extension. Grey bits are bits with unknown difference. The two states surrounded in red are the starting and final states of the multiple differentials considered.

condition on the extension of the distinguisher to the plaintexts ($\delta_{in} \rightarrow D_{in}$ as denoted in Figure 1) then D_{in} will have all but one row active (see Figure 4), leading to a not very efficient sieving of the pairs, that would imply too many potential pairs to test. For this reason, we propose a first improvement. This improvement consists in restricting the permitted transitions through the second S-box layer of Round 0. More precisely, the condition is that the three active bits after the second S-box in rows 20, 22 and 24 only generate a maximum of three active rows (between rows 20 and 28) in the plaintext state. This happens with a certain probability P_{in}. As we show next, this probability is relatively high, and even if it decreases the overall differential probability a little bit, it allows to have 7 inactive rows (instead of 1 before) in the plaintext state.

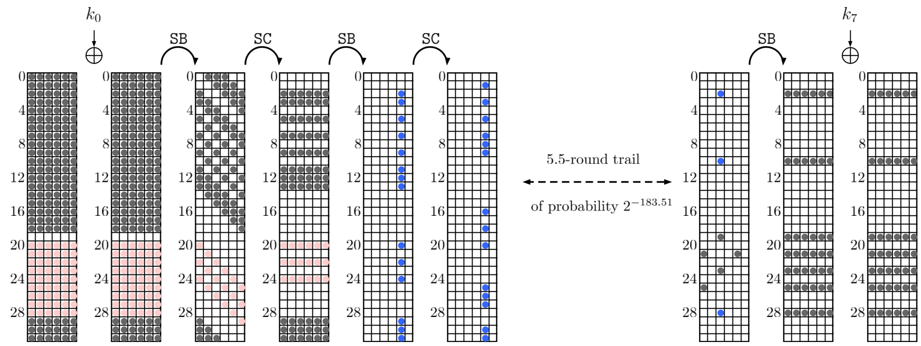


Fig. 5. Key recovery part of the 7-round attack against `SPEEDY-7-192`

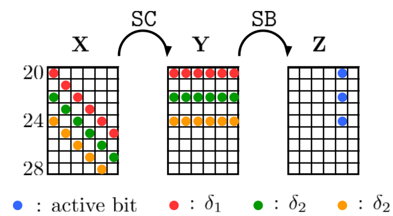


Fig. 6. Transition of rows 20, 22 and 24 through the inverse of the second `SB` of round 0.

We detail now how this probability can be computed. We start with the state $\mathbf{Z}$, corresponding to the state after the second S-box application of Round 0, where the rows 20, 22 and 24 all have an active difference of `000010`. Therefore, on the state $\mathbf{Y}$, we consider differences $\delta_1, \delta_2, \delta_3$ that propagate to `000010` through the S-box layer with probability $\mathbb{P}(\delta_1), \mathbb{P}(\delta_2), \mathbb{P}(\delta_3)$ respectively. Propagating backwards through `SC`, we obtain $\mathbf{X}_{\delta_1,\delta_2,\delta_3} = \texttt{SC}^{-1}(\mathbf{Y}_{\delta_1,\delta_2,\delta_3})$. According to our initial motivation, we will be interested in the states $\mathbf{X}_{\delta_1,\delta_2,\delta_3}$ that have at most three nonzero rows between rows 20 and 28. For this we can define the function $\mathbb{1}_3$ as follows:

$$\mathbb{1}_3(\mathbf{X}_{\delta_1,\delta_2,\delta_3}) = \begin{cases} 0 \text{ if } \mathbf{X}_{\delta_1,\delta_2,\delta_3} \text{has more than 3 nonzero rows} \\ 1 \text{ else.} \end{cases}$$

Therefore, the overall probability for the transition is given by the formula

$$P_{in} = \int_{\delta_1,\delta_2,\delta_3} \mathbb{1}_3(\mathbf{X}_{\delta_1,\delta_2,\delta_3})\mathbb{P}(\delta_1)\mathbb{P}(\delta_2)\mathbb{P}(\delta_3)$$

We computed this probability and obtained $P_{in} = 2^{-2.76}$. The 77 different possibilities for having at most 3 active rows between the rows 20 and 28 are provided as supplementary material. We take now this probability into account as part of the overall probability of the differential distinguisher, which becomes $2^{p^*} = 2^{-(183.51+2.76)} = 2^{-186.27}$.

4.2 Data generation

We build the data required for our attack in the decryption direction. Since there are 7 active rows on the ciphertexts, the size of each structure is $2^{7\times 6} = 2^{42}$. By following now the notations introduced in Section 2, we build 2^s structures of size 2^{42} each, such that 2^{s+42} equals $2^{186.27+1}$. This implies $2^s = 2^{145.27}$ structures and $2^{145.27+2\cdot 42-1} = 2^{228.27}$ potential pairs. The cost of this part is $2^{187.27}C_E$, where C_E is the cost of one encryption and can be estimated as $6 * (1 + 6 + 6 + 6) + 1 + 6 + 6 = 128$ bit-operations. Indeed, the cost of `MC` and of `SB` are considered as 6 bit-operations and the cost of `AK` is 1, and all the transformations are applied in parallel so we only consider one per round.

4.3 Sieving of the pairs

Using all the $2^{228.27}$ pairs for performing the attack would exceed the exhaustive search complexity. Therefore, we will start with a sieving step to eliminate pairs that cannot have followed the differential. This sieving will be applied with regards to the differences in the plaintext. Indeed, as can be seen from Figure 5, the good plaintext pairs have a zero-difference in the row 19 as well as 6 inactive rows between rows 20 and 28. The sieving will be performed on both the inactive and the active rows.

Inactive rows. Each inactive row represents a 6-bit filter. Since there are 7 inactive rows at the input, for each one of the 77 possible difference configurations in the plaintexts, the sieving obtained from these rows is 2^{-42}.

Active rows $[0-18]$ *and* $[29-31]$. We can proceed to a sieving on each of these 22 active rows by taking into account the first S-box layer of Round 0. To make this step clear, we start by explaining the sieve on row 0. Indeed, as can be seen from Figure 5, to follow the differential, a plaintext pair should generate after the application of the S-box a truncated difference of the form `(0,*,*,*,0,0)`. By looking at the DDT of `SPEEDY`'s S-box, we see that the input differences `0x12`, `0x1b` and `0x22` never propagate to an output difference of the form `(0,*,*,*,0,0)`. Thus, any pair with one of those three plaintext differences at row 0 can be sieved out. This gives us a filter of $log_2(61/64) = 0.07$, as seen in Table 5. The filters for the other active rows are computed similarly and are reported in Table 5.

Considering different patterns in the rows $[20-28]$. The active rows between rows 20 and 28 depend on the particular pattern considered. Recall that there are in total 77 possible patterns *pat*, and each one corresponds to a subset of exactly 3 active rows among rows 20 to 28 in the plaintext. We start from the difference `(0,0,0,0,1,0)` on the rows $20, 22$ and 24 after the second S-box of Round 0. Then we propagate this difference backwards through the two S-box layers of Round 0 and discard all the differences that don't follow the pattern considered. The number of possible differences on the plaintext allows us to filter $2^{-f_{pat}} = \frac{\#\text{Possible differences}}{2^{3\cdot 6}}$ (see Appendix B for the values of f_{pat}).

row	filter	row	filter	row	filter	row	filter
0	0.07	6	0.07	12	0.02	18	1.36
1	0.07	7	0	13	0	29	0.42
2	0	8	0.09	14	0	30	0.51
3	0.02	9	0.07	15	0.07	31	0.17
4	0.07	10	0.09	16	0.07		
5	0.14	11	0.07	17	0.48		
total filter 3.86							

Table 5. This table represents the filter in bits that we can perform by considering the set of input differences that can reach the set of output differences defined by the differential for the application of the S-box layer of Round 0.

Summarizing the sieving step. For a specific pattern *pat*, the sieving corresponding to the inactive rows is 2^{-42} while the one on the active rows is $2^{-3.86} \cdot 2^{-f_{pat}}$. Thus, the total number of potential pairs for the key recovery step is

$$\sum_{pat} 2^{228.27-45.86-f_{pat}} = 2^{181.92} \sum_{pat} 2^{-f_{pat}} = 2^{187.25}\,.$$

Let us point out here that this sieving step is the reason why we decided to perform the attack in the decryption direction, as using the 77 patterns in initial structures would have further increased the complexity.

4.4 Recovering the key

In this section, we describe our improved key recovery step. The key recovery algorithm is performed for each pair of data on the fly. As explained in the last section, the total number of pairs we will try in this step is $2^{-187.25}$. For each pair, we check whether there exists a secret key that allows the pair to follow the differential. If not, the pair is discarded. Otherwise, as we will show, we obtain a partial key on which all bits are determined but a small number n_l which is equal to 11 on average. For each of the remaining pairs and associated partial key, we then try exhaustively all possible 2^{n_l} keys. For each pair, the key recovery is divided into three stages which can be summarized as follows. First, we determine bits of the last subkey k_7 using the fact that if the pair follows the trail, then it must belong to δ_{out} before the last `SB` application. Since the key schedule of `SPEEDY` consists simply in a permutation of the key bits, this in turn constrains the bits of k_0. Second, we determine more bits of k_0 using the fact that the pair must belong to δ_{in}. Lastly, we determine a few extra key bits using the penultimate S-box.

Stage 1 - Last subkey addition (k_7). For each pair, we start by determining bits of k_7. As can be seen from Figure 5, the ciphertext pairs are active on the rows `[2,10,19,21,23,25,28]`. For the rows `[2,10,19,23,28]` (respectively row

`25`), we want the partial key to be such that these rows satisfy the differential `(0,0,1,0,0,0)` (`(1,0,0,0,0,0)` respectively) before the last `SB` application. For each pair, this determines $6 \times 6 = 36$ key bits on average. The case of row `21` is only slightly different. If active, there are 2^6 possibilities for the six key bits, but 4 different patterns are possible before `SB`. A correct pattern is thus reached with probability 2^{-4}. The row `21` can thus determine 6 additional key bits at the cost of about 2^2 guesses on average. This stage thus allows us to determine up to 42 key bits at the cost of a guess on 2^2 key bits.

On the potentially active ciphertext rows: For the sake of simplicity, we will consider in this analysis that the four rows in the output `[19,21,23,25]` are active, as it will simplify the key guessing procedure. In this case, we could just discard the pairs not verifying this, leaving us with $2^{24+24-1} - 2^{47-6} \approx 2^{46.91}$ pairs for the partial structure on 4 lines instead of 2^{47}, but with a higher probability of reaching a good difference before the penultimate `SB`. In practice, there is no need to discard this data and it can also be treated with similar methods, slightly more expensive than the one presented here as a few more key bits might have to be partially guessed. As this happens to a very small proportion of data, the difference in the cost will be negligible, and we therefore only explain the predominant case, with all the rows active. For rows `[2,10,28]`, we allow them to be non active, and this gives on average a probability of 2^{-x} of having a difference that can match the needed one (including the 0 difference), which generates on average around 2^x potential associated keys, so we can consider one on average.

row							row						
2	61	116	171	34	89	144	23	79	134	189	52	107	162
10	13	68	123	178	41	96	25	163	26	81	136	191	54
19	103	158	21	76	131	186	25	163	26	81	136	191	54
21	187	50	105	160	23	78							

Table 6. Last key guesses. Each row corresponds to one of the 7 active rows of the ciphertexts.

Stage 2 - First subkey addition (k_0) We now focus on the addition of the first subkey k_0. This key recovery stage is performed row by row, and the order in which each row is studied is important in order to keep our time complexity as low as possible. For each row, we will use available information from both `SubBox` layers of Round 0 to determine more triplets of possible pairs and an associated key. Table 7 helps us understand how to exploit the first S-box of Round 0 for rows `[0,...,18]` and `[29,30,31]`. Recall that these rows are active rows in the plaintext and that they allowed us to perform a specific sieving given in Section 4.3. For each row, Table 7 provides the following information:

- `Key determined` gives the number of key bits already determined during Stage 1 (*i.e.* with subkey k_7).
- `Key left` gives the number of key bits that remains to be determined for this row (note that the sum of `Key determined` and `Key left` is always 6).
- `Differential Filter` gives the value of the filter that was applied during the sieving step to each pair.
- `Fixed bits` gives the amount of inactive bits after the first `SubBox` layer.
- `First S-box Cost` gives the overall cost in bits for a given row to check the propagation through the first `SubBox`. Since one can precompute the valid pairs of values and associated partial keys for each row, this cost is equal to (`Key left` + `Differential filter - Fixed bits`) rather than `Key left`. For each row and for each key, the probability that they satisfy the differential is $2^{\texttt{Differentialfilter}-\texttt{Fixedbits}}$. In particular, for rows where the value of `First S-box Cost` is negative, then for each pair, there exists a key that satisfies the differential with probability < 1. Such rows thus allow us to discard more pairs.

Note that the rows 20 to 28 are considered differently. For the sake of simplicity, as this leads to many possible patterns, we won't consider them in our explanation of the generic key recovery, though they can be in each particular case. In addition, row 19, as it is inactive, does not provide any information about k_0 through the first `SB`.

To perform the key recovery, we will also look into the propagation through the second S-box. More precisely, we will use the conditions set on the rows `[2,3,5,7,9,11,12,13,29,31]` after the application of the second S-box to sieve the pairs. At the output of the second S-box, these rows must have the exact difference `(0,0,0,0,1,0)`. This provides us with a 2^{-6} filter, but it is not straightforward how to exploit it. Indeed, because of the `SC` step, each row at the output of the second S-box layer depends on 6 rows at the output of the first S-box layer. It thus seems that in order to get a 2^{-6} filter, one first has to guess 6 rows of k_7, which is very costly. However, we use several improved techniques in order to get filters without having to guess too many rows before the first S-box step. We describe these techniques through an example which can be found in the paragraph dedicated to the rows `[13,14,16]` below. We now provide the description of the key recovery. We describe in detail the first three step of Stage 2. Table 8 sums up the rest of Stage 2.

Rows `[17,31]`. We start by considering the rows 17 and 31. These rows allow us to perform a filter of $-0.83 - 0.52 = -1.35$ at the first S-box level.

Rows `[13,14,16]`. We next consider the rows 13, 14 and 16. To understand why these rows are the next ones we consider, one must take into account the second S-box transition. Indeed, consider the rows `[11,12,13]` after the second S-box transition. These three rows are active, and must thus have the exact difference `(0,0,0,0,1,0)`. Since these rows are positioned next to each other, one does not need to guess $3 \times 6 = 18$ rows at the input of the first S-box, but only 8, namely the rows `[11,12,...,18]`. Further, we show that to get a filter, one does not

row							Key determined	Key left	Differential Filter	Fixed bits	First S-box Cost
0	0	1	2	3	4	5	1	5	0.07	3	2.07
1	6	7	8	9	10	11	0	6	0.07	3	3.07
2	12	13	14	15	16	17	1	5	0	2	3
3	18	19	20	21	22	23	2	4	0.02	2	2.02
4	24	25	26	27	28	29	2	4	0.07	3	1.07
5	30	31	32	33	34	35	1	5	0.14	3	2.14
6	36	37	38	39	40	41	1	5	0.07	3	2.04
7	42	43	44	45	46	47	0	6	0	2	4
8	48	49	50	51	52	53	2	4	0.09	3	1.09
9	54	55	56	57	58	59	2	4	0.07	3	1.07
10	60	61	62	63	64	65	1	5	0.09	3	2.09
11	66	67	68	69	70	71	1	5	0.07	3	2.07
12	72	73	74	75	76	77	1	5	0.02	2	3.02
13	78	79	80	81	82	83	3	3	0	2	1
14	84	85	86	87	88	89	2	4	0	2	2
15	90	91	92	93	94	95	0	6	0.07	3	3.07
16	96	97	98	99	100	101	1	5	0.07	3	2.07
17	102	103	104	105	106	107	3	3	0.48	4	-0.52
18	108	109	110	111	112	113	1	5	1.36	5	1.36
19	114	115	116	117	118	119	1	5	**	6	**
20	120	121	122	123	124	125	1	5	*	5	*
21	126	127	128	129	130	131	1	5	*	5	*
22	132	133	134	135	136	137	2	4	*	4	*
23	138	139	140	141	142	143	0	6	*	4	*
24	144	145	146	147	148	149	1	5	*	3	*
25	150	151	152	153	154	155	0	6	*	3	*
26	156	157	158	159	160	161	2	4	*	4	*
27	162	163	164	165	166	167	3	3	*	4	*
28	168	169	170	171	172	173	1	5	*	5	*
29	174	175	176	177	178	179	1	5	1.42	4	2.42
30	180	181	182	183	184	185	0	6	0.51	4	2.51
31	186	187	188	189	190	191	4	2	0.17	3	-0.83

Table 7. This table represents the information used for efficiently solving the key-recovery part of the attack. Each line in the table is associated to the same row in the state. The column `Key determined` indicates how many bits are already known from Stage 1 (those bits are depicted in red), and `Key left` is the number of bits that remains to be known. `Fixed bits` represents the number of inactive bits after the first `SB` of Round 0 and that therefore can be used to perform a sieving on the candidate keys. The `cost` is the difference between the previous values, and the `second filter` denotes the active rows in the second `SB`, as they will provide an additional filtering to produce the fixed output difference.

need to guess all of the 8 rows which depend on `[11,12,13]` after the second S-box transition. We start by precomputing all the pair of values that are in the codomain of the function

$$a, b, c, d, e, f \mapsto \big(S^{-1}(a, b, c, d, e, f), S^{-1}(a, b, c, d, e \oplus 1, f)\big)$$

and store them in a table of size 2^6. We can thus build a precomputed table of size 2^{18} which contains all possible valid values of rows `[11,12,13]` at the entry the second S-box layer. We guess the rows `[13,14,16]` at the entry of the first S-box. Recall that the row `17` was already guessed in a first step. In total, 11 bits of the rows `[13,14,16,17]` later impact the rows `[11,12,13]` at the entry of the second S-box. There are thus 2^{22} possible pairs of values for these bits, whilst in total, the table contains 2^{18} possible pairs that verify the condition at the output of the S-box on the rows `[11,12,13]`. This thus results in a 2^{-4} filter. More precisely each pair matches a pre-computed valid pair in the table of size 2^{18} with probability 2^{-4}. In particular, whenever a pair is not discarded, the rows `[11,12,13]` before the second S-box are completely determined. This will allow us to filter more pairs as we guess more rows in the plaintext which impact the value of the rows `[11,12,13]` before the second S-box. The guess of rows $13, 14$ and 16 can be done following merging techniques developed in [14] resulting in a reduction of the guessing cost from $2^{3+2.07} = 2^{5.07}$ to $2^3 + 2^{2.07} + 2^{5.07-4} = 2^{3.83}$, or else can be more efficiently performed with small precomputations regarding this partial transitions with a cost for each step given by the number of remaining solutions, so $2^{5.07-4} = 2^{1.07}$ in this example.

Row `18`*.* The next step consists in guessing row 18. As we have described previously, for the pairs that have not been discarded yet, the three rows `[11,12,13]` before the second S-box are fixed. Two bits of row `18` later impact the value of these rows. Thus, we obtain an extra 2^{-2} filter. Therefore, as can be seen from Table 7 we obtain a partial guessing cost of $2^{1.36}$ and a partial data cost of $2^{-0.64}$.

The next steps of the key recovery can be derived similarly following Table 8. This table is to be read from top to bottom. Its columns provide the following informations:

– `Row guessed at the input`. This column displays the rows guessed. The first one are at the top and the last on is at the bottom.
– `Partial guessing cost`. From Table 7, we derive the cost of guessing each row and checking the first S-box transition.
– `Partial data cost`. For each line, this column displays the evolution of the data after the guess and filter step. This cost is the difference between the partial guessing cost and the filter.
– `Row determined at second S-box`. This column displays the 2nd-S-box rows that are fully determined after a given guess.

Line guessed at the input	Partial guessing cost	Partial data cost	Line determined at second S-box
17, 31	-1,35	-1.35	
13, 14, 16	3.61	1.07	11,12,13
18	1.36	-0.64	
11	2.07	0.07	9
9	1.07	-0.93	
12	3.02	-2.98	7
10	2.09	-1.91	
15	3.06	-2.94	
8	1.09	-0.91	5
7	4	0	
6	2.07	0.07	3
5	2.14	-1.86	2
4	2.07	-1.93	
3	2.02	-1.98	31
2	3	-1	30
1	3.07	-0.93	29
0	2.07	-3.93	
30	2.51	-1.49	
29	2.42	0.42	

Table 8. Key recovery near the second S-box.

At the end, the complexity of the key recovery up to know is given by the formula

$$\left[2^{187.25+2}2^{-1.35} + 2^{-1.35}(2^{3.61} + 2^{1.07}(2^{1.36} + 2^{-0.64}(2^{2.07} + 2^{0.07}(\cdots(2^{2.42} + 2^{0.42})))))\right] 2^{-7}C_E$$
$$= 2^{185.94}C_E$$

and the remaining data is $2^{166.1}$.

Stage 3 - Back to k_7 using the penultimate S-box. For the remaining key bits, we will study the penultimate S-box. A similar approach to the second S-box is applied here: instead of using the second S-box transition to perform a guess and filter approach, the penultimate S-box is used. We use Table 9 and Table 10 as a penultimate version of Table 7 and Table 8 respectively. We derive the complexity from the Table 10 and obtain a time complexity of $2^{164.52}C_E$ and we are left with $2^{161.1}$ data. At this point, the key bits of rows `[3,4,11,12,13]` remain to be guessed. Hence, we have determined the whole key except those 11 bits. We guess them and recover the 192-bit key, thus completing the key recovery. This last step complexity is $2^{161.1-7+11} = 2^{165.1}C_E$.

Complexity summary. The final time complexity of our attack is

$$\mathcal{T} = 2^{187.27}C_E + 2^{179.75}C_E + 2^{185.94}C_E + 2^{164.52}C_E + 2^{165.10}C_E = 2^{187.75}C_E$$

row		Key left	row		Key left
0	169 32 87 142 5 60	2	16	73 128 183 46 101 156	2
1	115 170 33 88 143 6	3	17	19 74 129 184 47 102	1
2	61 116 171 34 89 144	0	18	157 20 75 130 185 48	2
3	7 62 117 172 35 90	2	19	103 158 21 76 131 186	0
4	145 8 63 118 173 36	3	20	49 104 159 22 77 132	2
5	91 146 9 64 119 174	2	21	187 50 105 160 23 78	0
6	37 92 147 10 65 120	2	22	133 188 51 106 161 24	2
7	175 38 93 148 11 66	1	23	79 134 189 52 107 162	0
8	121 176 39 94 149 12	2	24	25 80 135 190 53 108	1
9	67 122 177 40 95 150	2	25	163 26 81 136 191 54	0
10	13 68 123 178 41 96	0	26	109 164 27 82 137 0	2
11	151 14 69 124 179 42	2	27	55 110 165 28 83 138	2
12	97 152 15 70 125 180	2	28	1 56 111 166 29 84	0
13	43 98 153 16 71 126	2	29	139 2 57 112 167 30	2
14	181 44 99 154 17 72	1	30	85 140 3 58 113 168	2
15	127 182 45 100 155 18	2	31	31 86 141 4 59 114	2

Table 9. Remaining key bits from the point of view of last round.

Row	Guess	Data	Row	Guess	Data
28,27,26,25,24,23	5	-1	15,14	3	-3
22,21,20	4	-2	10,9,8,7,6,5	9	3
19,18	2	-4	2,1,0,31,30,29	11	5
17,16	3	-3			

Table 10. Key recovery near the penultimate S-box. This table displays the rows guessed at the output in the Row columns, the partial guessing cost in the Guess column and the partial data cost in the data column.

5 Discussion and conclusion

We presented in this work an attack on `SPEEDY-7-192` that fully breaks this variant of the `SPEEDY` family of ciphers. In parallel, we could also build attacks on other variants, even if our attacks on these smaller-round versions do not contradict the corresponding security claims. For completeness we provide a summary of these attacks, that are at the best of our knowledge the best known attacks on these versions.

`SPEEDY-5-192`. Following the trail depicted in Figure 7 and its associated multiple differential probability of $2^{98.04}$, computed as explained in Section 3.4, we can build a differential attack on 5 rounds, very similar to the 7-round one. We just need to take into account the new parameters. Regarding 5 rounds the new complexity is given by (with $C_E = 2^{6.47}$ here):

$$\mathcal{T} = (2^{101.65} + 2^{105.65}\frac{2^{8.58}}{C_E} + 2^{101.65}\frac{2^{7.62-1}}{C_E})C_E \approx 2^{107.8}C_E,$$

a data complexity of $2^{101.65}$ and a memory complexity of 2^{42}. The authors stated that this version should achieve 128-bit security when data complexity is limited to 2^{64}. Therefore, due to the data limitation, our attack does not contradict the security claim of the designers but still represents the best known attack against `SPEEDY-5-192`.

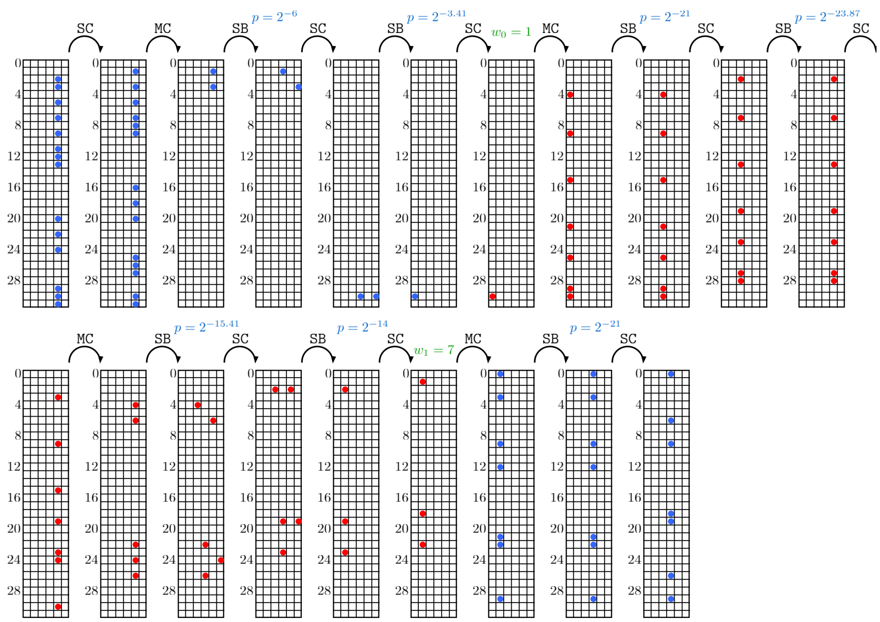


Fig. 7. 3.5-round differential trail used to attack `SPEEDY-5-192`. The red part corresponds to the 2-round core trail, while the blue part corresponds to the 1.5-round extension.

`SPEEDY-6-192`. For 5.5 and 6 rounds we can use the trail depicted in Figure 7 with multiple differential probability of $2^{118.55}$ and $2^{142.42}$ respectively. We take into account the new parameters and the complexities are (with $C_E^{5.5} = 2^{6.67}$ and $C_E^{6} = 2^{6.75}$):

$$\mathcal{T}_{5.5} = (2^{121.65} + 2^{125.65}\frac{2^{8.58}}{C_E^{5.5}} + 2^{121.65}\frac{2^{7.62-1}}{C_E^{5.5}})C_E^{5.5} \approx 2^{127.8}C_E,$$

and data complexity given by the first term and still a memory complexity of 2^{42}. The complexity of our attack is slightly below the claim for six rounds, which shows that SPEEDY-6-128 has less than half a round of security margin. We can do similar computations for 6 rounds to obtain $\mathcal{T}_6 = 2^{151.67} C_E^6$ with the same memory complexity and a data of $2^{140.20}$. To the best of our knowledge, our results are the best known attacks on SPEEDY and represent a significant gain over previous results.

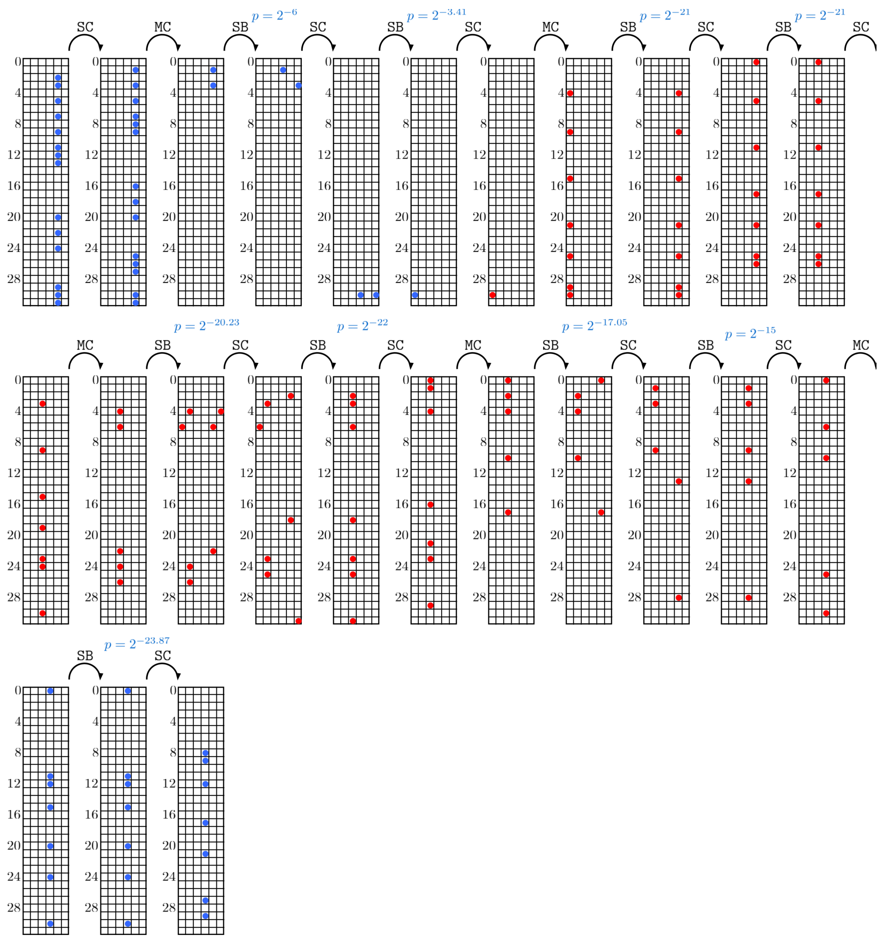


Fig. 8. 4.5-round differential trail used to attack SPEEDY-6-192. The red part corresponds to the 4-round core trail, while the blue part corresponds to the 1.5-round extension.

Open problems. We believe, as a future research, that it would be interesting to develop different algorithmic methods in order to search for higher-probability trails for SPEEDY. In parallel, different theoretical but also programming techniques should permit to improve our approach for finding multiple differentials.

Being able to find more trails of good probability would greatly increase the complexities of the attacks. In parallel, new tools that would permit to compute propagations through two rows at once with no constraint in the middle part would potentially permit to find better differentials. Finally, we believe that it would be interesting to develop an automatic tool for differential cryptanalysis that could give an approximate of the best attack complexity for certain types of ciphers.

References

1. Banik, S., Bao, Z., Isobe, T., Kubo, H., Liu, F., Minematsu, K., Sakamoto, K., Shibata, N., Shigeri, M.: WARP : Revisiting GFN for lightweight 128-bit block cipher. In: SAC 2020. LNCS, vol. 12804, pp. 535–564. Springer (2020)
2. Banik, S., Pandey, S.K., Peyrin, T., Sasaki, Y., Sim, S.M., Todo, Y.: GIFT: A small Present - towards reaching the limit of lightweight encryption. In: CHES 2017. LNCS, vol. 10529, pp. 321–345. Springer (2017)
3. Beierle, C., Jean, J., Kölbl, S., Leander, G., Moradi, A., Peyrin, T., Sasaki, Y., Sasdrich, P., Sim, S.M.: The SKINNY family of block ciphers and its low-latency variant MANTIS. In: CRYPTO 2016, Part II. LNCS, vol. 9815, pp. 123–153. Springer (2016)
4. Biham, E., Furman, V., Misztal, M., Rijmen, V.: Differential cryptanalysis of Q. In: FSE 2001. LNCS, vol. 2355, pp. 174–186. Springer (2001)
5. Biham, E., Shamir, A.: Differential cryptanalysis of DES-like cryptosystems. In: CRYPTO '90. LNCS, vol. 537, pp. 2–21. Springer (1990)
6. Biham, E., Shamir, A.: Differential cryptanalysis of Feal and N-Hash. In: EUROCRYPT '91. LNCS, vol. 547, pp. 1–16. Springer (1991)
7. Biham, E., Shamir, A.: Differential cryptanalysis of Snefru, Khafre, REDOC-II, LOKI and Lucifer. In: CRYPTO '91. LNCS, vol. 576, pp. 156–171. Springer (1991)
8. Biham, E., Shamir, A.: Differential cryptanalysis of the full 16-round DES. In: CRYPTO '92. LNCS, vol. 740, pp. 487–496. Springer (1992)
9. Blondeau, C., Gérard, B.: Multiple differential cryptanalysis: Theory and practice. In: FSE 2011. LNCS, vol. 6733, pp. 35–54. Springer (2011)
10. Bogdanov, A., Knudsen, L.R., Leander, G., Paar, C., Poschmann, A., Robshaw, M.J.B., Seurin, Y., Vikkelsoe, C.: PRESENT: an ultra-lightweight block cipher. In: CHES 2007. LNCS, vol. 4727, pp. 450–466. Springer (2007)
11. Broll, M., Canale, F., David, N., Flórez-Gutiérrez, A., Leander, G., Naya-Plasencia, M., Todo, Y.: New attacks from old distinguishers improved attacks on serpent. In: CT-RSA 2022. LNCS, vol. 13161, pp. 484–510. Springer (2022)
12. Canteaut, A., Duval, S., Leurent, G., Naya-Plasencia, M., Perrin, L., Pornin, T., Schrottenloher, A.: Saturnin: a suite of lightweight symmetric algorithms for post-quantum security. IACR Trans. Symmetric Cryptol. **2020**(S1), 160–207 (2020)
13. Canteaut, A., Fuhr, T., Gilbert, H., Naya-Plasencia, M., Reinhard, J.: Multiple differential cryptanalysis of round-reduced PRINCE. In: FSE 2014. LNCS, vol. 8540, pp. 591–610. Springer (2014)
14. Canteaut, A., Naya-Plasencia, M., Vayssière, B.: Sieve-in-the-middle: Improved MITM attacks. In: CRYPTO 2013, Part I. LNCS, vol. 8042, pp. 222–240. Springer (2013)

15. Choudhuri, A.R., Maitra, S.: Differential cryptanalysis of Salsa and ChaCha – An evaluation with a hybrid model. Cryptology ePrint Archive, Paper 2016/377 (2016), https://eprint.iacr.org/2016/377
16. Knellwolf, S., Meier, W., Naya-Plasencia, M.: Conditional differential cryptanalysis of nlfsr-based cryptosystems. In: ASIACRYPT 2010. LNCS, vol. 6477, pp. 130–145. Springer (2010)
17. Knudsen, L.R.: Truncated and higher order differentials. In: FSE 1994. LNCS, vol. 1008, pp. 196–211. Springer (1994)
18. Leander, G., Moos, T., Moradi, A., Rasoolzadeh, S.: The SPEEDY family of block ciphers engineering an ultra low-latency cipher from gate level for secure processor architectures. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2021**(4), 510–545 (2021)
19. Leurent, G.: Improved differential-linear cryptanalysis of 7-round Chaskey with partitioning. In: EUROCRYPT 2016, Part I. LNCS, vol. 9665, pp. 344–371. Springer (2016)
20. Naya-Plasencia, M.: How to improve rebound attacks. In: CRYPTO 2011. LNCS, vol. 6841, pp. 188–205. Springer (2011)
21. Rohit, R., Sarkar, S.: Cryptanalysis of reduced round SPEEDY. Cryptology ePrint Archive, Paper 2022/612 (2022), https://eprint.iacr.org/2022/612
22. Wang, M.: Differential cryptanalysis of reduced-round PRESENT. In: AFRICACRYPT 2008. LNCS, vol. 5023, pp. 40–49. Springer (2008)

A DDT of the SPEEDY S-box

Table 11 describes the Difference Distribution Table (DDT) of the S-box S. The rows correspond to input differences α and the columns to output differences β. An entry $\mathtt{DDT}[\alpha][\beta]$ provides the number of solutions to the equation:

$$\mathtt{DDT}[\alpha][\beta] = \#\{x \in \mathbb{F}_2^{64} : S(x) \oplus S(x \oplus \alpha) = \beta\}.$$

To ease readability, impossible transitions are represented with a '.'.

α/β	0	1	2		4				8								16																32																															
0	64	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
1	.	.	2	.	6	.	2	2	4	.	.	2	2	.	4	.	2	.	.	2	6	.	2	.	6	2	4	.	.	.	.	.	2	.	.	.	2	.	.	.	2	.	.	.	2	.	.	.	.	.	.	.	2	.	.	2	2	.	.	2	.	.	.	.
2	.	8	6	2	8	.	.	.	8	.	.	.	.	.	.	.	.	.	4	.	.	.	.	.	4	.	.	.	.	.	.	.	.	8	2	2	6	2	.	.	2	2	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
3	.	.	.	.	.	2	4	2	.	.	4	.	.	.	.	.	2	.	2	.	.	2	2	.	2	2	2	2	2	.	2	.	6	.	.	.	2	.	.	.	2	.	.	.	6	2	.	2	.	4	2	.	.	2	.	.	.	2	.	.	.	2	.	.
4	.	2	2	2	6	.	.	.	6	.	.	.	2	.	.	.	2	.	.	.	2	.	.	.	4	.	.	.	.	.	.	.	2	.	6	.	4	.	.	.	8	.	.	.	4	.	4	.	2	.	2	.	.	.	.	.	2	.	.	.	2	.	.	.
5	.	.	.	.	2	.	.	.	.	.	.	.	4	.	2	.	2	.	.	.	.	.	.	.	.	2	2	2	4	.	.	4	6	.	.	2	2	2	2	.	.	.	.	.	2	.	.	.	2	.	.	.	4	.	6	2	4	.	.	2	4	.	.	.
6	.	.	.	2	.	4	.	.	.	4	2	4	6	.	2	.	.	.	.	.	2	.	.	.	.	.	2	.	4	.	.	.	.	.	.	2	2	2	.	.	.	.	4	2	4	.	4	4	.	.	.	.	2	.	.	.	.	.	2	.	2	.	2	.
7	.	.	2	.	2	.	.	2	2	.	.	.	.	.	.	2	.	.	2	2	2	.	.	.	.	2	2	.	.	6	2	.	2	.	.	.	6	2	.	.	2	.	.	.	.	2	.	.	2	2	.	.	.	2	2	2	4	.	.	.	2	.	6	.
8	.	2	6	4	.	2	4	2	4	6	.	2	2	.	.	2	6	2	4	.	2	4	2	.	.	.	.	.	.	.	.	.	.	.	2	2	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	2	.	2	.	.	.	.	.	.	.	.
9	.	2	.	4	.	2	4	.	2	.	.	.	2	.	6	2	.	2	.	2	2	.	2	4	2	.	4	.	2	2	.	2	.	2	.	.	.	.	.	2	.	2	.	.	.	.	.	2	.	.	.	.	2	.	.	2	.	.	2	.	.	.	2	.
10	.	4	.	.	2	.	2	4	.	.	2	2	.	2	2	.	4	4	.	.	4	2	2	.	.	.	2	2	.	.	.	.	.	2	2	4	.	.	.	.	.	2	.	2	.	.	.	.	.	4	.	4	2	.	.	2	.	.	.	.	.	.	.	.
11	.	.	2	.	.	.	.	2	.	.	2	.	2	.	2	2	2	.	.	2	.	2	2	4	.	.	.	2	.	.	4	2	.	2	.	2	.	4	.	4	2	.	.	2	.	2	.	2	2	.	.	.	.	.	2	.	.	4	.	.	2	.	.	2
12	.	2	.	.	.	.	.	4	.	.	2	2	2	.	.	2	.	2	.	.	2	.	.	2	.	.	.	.	.	4	.	.	4	.	2	.	.	2	.	2	.	2	2	.	.	2	2	.	2	.	2	2	2	.	2	.	4	.	4	.	.	2	.	2
13	.	.	.	.	.	.	.	.	.	.	.	.	2	4	.	2	.	.	2	4	.	.	2	.	.	2	2	2	2	.	.	.	.	.	2	.	4	.	.	2	.	2	2	2	.	2	.	.	4	4	.	.	6	2	.	.	2	2	.	.	2	.	.	2
14	.	.	.	2	.	2	2	2	.	6	.	.	.	2	.	2	.	.	.	4	.	4	.	.	.	4	.	.	2	.	.	.	.	.	2	2	.	.	.	.	4	.	.	.	2	2	.	2	.	2	4	.	.	.	4	2	.	.	.	2	2	.	2	.
15	.	.	.	.	.	2	.	.	.	2	.	2	.	4	.	.	.	4	2	.	4	.	.	.	2	.	.	4	.	.	.	2	.	.	.	2	.	2	2	.	.	.	.	2	4	.	.	2	4	.	4	.	.	4	.	2	2	2	.	.	.	4	.	.
16	.	4	4	.	8	.	.	.	8	4	4	.	.	.	.	.	4	6	6	.	2	2	.	.	.	.	.	.	.	.	.	.	2	.	2	.	4	.	.	.	.	.	.	.	.	.	.	.	4	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
17	.	.	.	2	.	2	.	2	2	.	2	2	2	.	2	2	2	2	2	2	.	2	2	2	.	.	4	.	2	.	.	2	.	.	2	.	.	.	4	.	4	.	2	.	.	.	2	.	2	.	.	2	2	.	.	.	.	.	.	.	2	.	.	2
18	.	.	2	.	.	4	4	.	.	4	6	.	.	4	4	.	.	.	.	2	.	.	2	2	.	6	.	.	.	.	.	.	.	.	2	.	.	4	.	.	2	4	.	.	.	.	.	.	.	.	2	.	2	2	4	.	.	.	2	.	.	.	.	.
19	.	.	2	2	.	.	.	.	.	.	.	2	.	4	.	4	2	2	2	2	.	.	2	2	.	.	.	2	.	2	2	.	.	2	4	.	.	.	2	.	.	2	2	.	2	.	4	.	2	2	2	.	.	2	.	.	.	.	.	.	.	2	4	.
20	.	.	2	.	.	.	4	.	4	.	2	.	4	.	.	.	.	2	2	.	.	.	2	.	.	.	2	.	4	.	.	.	4	.	2	2	.	.	6	.	2	4	2	.	2	.	.	.	.	2	.	4	.	.	.	.	2	.	2	.	.	2	.	.
21	.	.	.	.	.	.	4	.	.	.	2	2	.	.	2	.	4	.	.	.	.	.	.	.	4	.	.	4	.	.	2	.	2	.	4	.	2	.	.	4	.	2	2	2	.	2	2	.	2	.	4	4	.	2	.	.	2	.	.	.	.	2	2	.
22	.	4	.	.	.	.	2	.	.	.	2	2	2	2	6	.	.	.	.	.	.	2	.	.	.	.	2	.	.	.	8	.	.	.	.	.	.	.	2	4	.	.	4	4	.	4	2	.	.	.	.	.	.	2	.	.	.	.	.	6	.	.	2	2
23	.	.	4	2	.	.	2	2	.	.	2	.	.	.	.	.	.	2	4	.	2	.	2	.	.	4	2	.	.	.	.	.	.	2	.	2	.	2	.	.	.	.	4	.	2	2	2	4	.	4	.	.	.	.	2	.	.	4	4	.	.	.	.	2
24	.	2	2	2	.	.	2	4	.	4	.	2	.	2	.	.	.	.	.	2	2	.	2	2	6	4	2	2	2	4	.	.	.	2	.	2	.	.	.	.	.	.	.	.	.	.	.	.	.	2	6	.	.	.	.	.	.	.	.	.	.	2	.	2
25	.	2	2	.	.	.	.	6	.	.	4	.	.	2	2	.	.	2	.	2	.	2	.	4	.	.	2	.	2	4	2	2	.	.	.	2	.	4	.	.	.	.	4	2	.	2	.	.	.	.	.	2	.	2	2	.	.	.	.	.	2	.	2	.
26	.	2	.	2	.	2	.	.	.	4	.	.	.	.	.	2	2	2	.	.	.	4	.	2	2	2	.	4	4	2	2	2	2	.	2	2	.	.	2	2	.	.	2	.	.	.	.	.	2	.	.	4	.	.	.	.	.	2	.	4	.	.	.	.
27	.	2	.	4	.	2	2	.	.	.	.	.	.	.	.	4	.	.	2	.	.	.	2	2	.	2	.	.	.	4	2	4	.	2	.	.	.	.	2	2	.	4	.	4	.	.	2	2	.	.	.	2	4	2	.	2	2	.	.	.	.	.	2	.
28	.	.	.	2	2	2	.	.	.	.	.	.	.	2	.	2	2	.	.	.	.	2	.	2	.	.	.	.	2	.	4	2	.	4	.	.	.	.	.	2	.	2	2	2	.	.	.	2	2	2	2	2	.	4	2	.	2	2	2	4	2	.	.	.
29	.	.	2	.	2	.	2	2	.	.	.	2	.	.	.	.	.	2	2	.	.	.	.	.	2	.	.	2	2	2	2	.	2	4	2	.	.	4	.	.	.	2	.	2	2	4	.	.	.	2	.	.	.	4	.	2	2	.	.	4	2	2	.	.
30	.	2	.	.	.	4	.	.	.	.	.	2	.	4	2	.	.	2	.	2	2	.	.	4	2	.	2	.	2	2	.	.	2	.	2	2	.	.	.	.	.	.	.	2	.	.	.	2	.	.	2	2	.	.	2	4	2	6	.	.	.	2	.	2
31	.	2	.	4	2	2	.	.	.	2	.	.	.	.	.	.	4	.	.	.	2	.	4	.	.	4	.	.	.	.	.	2	.	2	.	2	2	2	2	.	2	2	.	.	.	2	.	4	.	2	.	.	.	.	.	.	.	4	2	.	.	4	.	4
32	.	4	8	.	4	4	.	.	8	4	2	2	.	.	.	.	.	.	.	.	2	2	2	2	.	.	.	.	.	.	.	.	4	.	4	4	.	.	.	.	.	.	.	.	.	.	.	.	6	2	.	.	.	.	.	.	.	.	.	.	.	.	.	.
33	.	2	2	4	2	.	4	.	.	.	.	.	.	2	.	6	2	.	.	2	2	.	.	.	4	.	.	4	2	.	2	.	.	2	.	.	2	.	2	.	.	.	2	.	.	2	.	.	.	2	.	.	.	2	2	.	2	2	2	.	2	.	.	.
34	.	.	.	4	.	4	4	.	.	4	.	.	.	.	.	.	.	.	.	.	.	.	4	.	.	.	.	.	.	4	.	.	.	.	4	4	.	.	.	.	.	4	.	4	.	4	2	2	.	.	2	2	.	.	2	2	4	.	.	.	.	.	2	2
35	.	.	2	.	.	2	.	2	.	.	.	2	.	.	2	.	2	2	.	.	.	2	.	.	.	4	.	.	2	2	.	.	.	.	2	.	.	6	.	6	.	4	2	.	.	.	2	.	.	.	.	.	2	.	.	4	2	2	6	2	.	.	.	.
36	.	2	.	2	.	.	.	.	.	.	4	.	4	.	.	.	.	4	2	.	2	.	.	.	.	4	.	.	4	.	.	.	4	.	2	.	.	.	4	2	2	.	4	.	2	.	.	.	.	2	.	2	2	.	.	2	4	2	.	.	.	2	.	.
37	.	.	.	.	.	.	2	.	.	.	.	.	2	.	2	.	2	.	.	2	4	2	.	4	.	.	2	2	.	.	.	.	.	4	2	.	.	2	.	2	.	.	.	2	2	.	4	.	.	.	2	.	2	.	2	.	.	6	.	.	4	2	.	4
38	.	4	.	.	.	.	.	2	.	2	.	2	.	2	4	.	.	.	.	.	.	2	.	.	.	2	2	2	2	2	2	2	.	.	.	.	.	.	2	4	.	.	2	2	.	2	2	2	.	.	.	.	.	2	.	.	.	.	4	2	2	2	2	2
39	.	.	.	.	.	.	2	.	.	.	2	.	.	2	2	4	.	.	.	.	.	4	2	.	2	.	.	4	.	2	.	2	2	2	.	2	.	2	2	.	.	.	.	.	.	2	.	.	.	2	2	2	2	2	2	2	2	2	.	.	.	4	2	.
40	.	.	.	.	.	.	2	2	.	.	.	.	2	2	2	2	2	.	4	2	.	4	2	2	2	.	2	.	.	4	4	.	.	.	.	.	.	2	2	.	.	.	2	2	2	.	.	2	.	.	2	2	.	.	2	2	.	4	.	.	.	.	.	.
41	.	.	.	.	2	2	4	.	2	4	.	2	.	.	4	2	2	4	2	.	.	.	4	.	.	2	.	.	.	2	.	2	.	.	.	2	.	2	.	.	2	.	.	.	.	2	2	.	.	.	.	2	2	2	2	.	.	.	.	.	2	.	2	2
42	.	.	.	.	.	.	.	.	.	.	.	.	.	.	2	2	.	2	.	2	4	2	4	2	.	.	.	.	.	.	2	2	.	.	.	.	2	.	.	2	.	.	2	2	.	2	2	.	.	4	2	2	2	.	2	.	.	2	2	4	2	2	.	4
43	.	.	.	.	.	.	.	2	.	.	2	2	2	2	.	.	.	.	2	2	.	4	.	2	.	.	2	.	.	.	2	.	.	4	.	.	.	2	.	4	.	.	2	2	6	.	.	2	.	2	2	.	2	2	.	2	2	.	.	.	2	2	2	.
44	.	.	.	2	.	2	.	.	.	.	.	.	.	.	2	.	.	2	.	.	2	4	.	.	.	2	2	.	.	2	.	4	2	.	.	.	2	.	.	.	.	.	.	4	.	.	2	.	2	4	4	.	.	.	4	2	.	.	6	2	.	.	4	2
45	.	.	.	.	.	4	.	.	.	.	.	.	.	.	.	2	2	.	2	.	2	.	2	.	.	4	.	4	.	.	.	2	.	.	2	2	.	.	4	.	2	2	.	.	2	.	2	2	4	2	2	.	.	2	2	.	.	4	.	.	.	2	4	.
46	.	.	.	.	2	.	.	.	.	.	2	2	2	.	.	2	.	4	2	4	.	.	2	2	.	.	.	2	.	2	4	.	.	2	.	.	.	.	.	.	.	2	.	.	2	.	.	.	.	.	.	4	2	2	4	2	2	2	2	2	.	.	2	2
47	.	.	.	.	.	.	.	4	.	.	.	2	.	2	2	2	.	.	2	2	.	.	.	.	.	.	6	.	2	4	.	.	2	.	2	.	2	.	.	2	.	2	.	.	.	2	.	.	2	4	.	2	4	2	.	2	2	.	2	2	.	2	.	.
48	.	.	.	.	2	2	.	.	4	.	4	4	4	.	.	.	.	.	.	.	2	.	.	2	2	2	.	4	.	.	.	.	.	4	.	.	6	.	.	2	.	.	4	.	4	.	.	.	.	.	4	.	4	4	.	.	.	.	.	.	.	.	.	.
49	.	2	2	.	.	2	.	2	.	2	.	2	2	.	2	.	2	.	2	4	.	.	.	.	.	.	.	.	.	.	2	6	2	.	.	4	2	.	.	.	2	.	.	.	2	2	.	2	2	2	2	.	2	.	.	.	2	.	.	.	2	2	2	.
50	.	.	.	.	.	.	2	.	.	.	.	4	4	4	.	6	.	.	.	.	.	.	.	2	.	.	.	.	2	.	.	.	.	.	.	.	.	.	6	.	.	4	.	4	.	6	.	.	.	.	.	.	.	.	4	2	.	.	4	.	2	2	.	6
51	.	2	.	2	.	.	.	2	2	.	.	.	.	4	.	.	.	2	.	.	.	2	2	.	.	2	2	.	.	.	2	.	2	2	.	.	.	2	2	2	2	.	2	2	2	.	.	2	2	.	2	2	.	2	.	2	.	.	.	4	2	2	.	2
52	.	.	2	4	2	2	.	.	4	2	.	.	.	.	.	.	2	.	.	4	.	.	.	.	4	.	.	.	.	2	.	.	.	.	2	2	2	2	2	.	8	.	.	.	.	2	4	.	.	2	.	2	.	.	2	.	2	.	.	.	4	.	.	.
53	.	2	2	.	2	.	.	2	.	.	.	.	.	.	.	.	2	2	2	4	.	.	.	2	2	.	.	.	2	.	.	.	.	.	.	.	2	.	2	.	2	4	2	4	2	2	2	2	4	2	.	.	.	.	.	2	.	.	2	4	2	.	.	.
54	.	.	.	2	.	2	.	2	.	4	4	2	.	4	.	.	.	.	.	.	.	.	.	2	2	.	2	2	2	.	.	2	.	.	.	2	.	2	.	2	.	.	2	.	2	2	4	4	.	.	.	.	.	.	.	2	.	2	.	.	2	.	4	2
55	.	.	.	2	.	2	.	2	.	2	2	.	2	.	2	.	.	4	2	2	.	2	.	.	.	.	2	.	.	.	.	2	.	2	.	.	.	2	2	.	4	2	.	2	.	.	.	4	2	.	2	4	2	.	.	.	2	2	.	2	.	2	.	.
56	.	.	.	.	.	.	.	.	.	.	2	2	2	2	2	2	.	.	.	.	.	.	.	.	2	.	4	2	4	2	2	4	.	.	2	.	.	2	2	2	2	.	.	.	.	2	.	.	2	.	.	.	.	.	2	4	.	2	2	2	.	2	2	2
57	.	.	2	4	2	2	.	.	.	.	2	.	2	.	.	2	.	.	.	.	2	2	.	2	2	2	.	2	2	.	2	.	.	4	2	.	.	2	.	2	.	.	2	.	2	.	2	.	.	.	.	.	2	.	2	2	.	.	.	2	2	4	2	.
58	.	.	.	.	2	.	.	.	.	.	.	.	.	2	.	.	.	.	.	.	2	.	.	.	.	2	.	2	2	2	4	6	2	.	.	.	2	2	.	.	.	.	4	2	.	.	2	6	.	.	.	2	.	2	.	2	4	.	2	4	.	.	2	2
59	.	2	.	.	2	2	.	2	.	.	.	4	.	.	.	.	.	.	.	.	.	2	.	2	2	.	2	.	.	2	.	2	2	.	.	6	.	2	.	2	2	.	.	.	.	2	4	.	.	.	.	2	2	2	2	.	.	.	2	.	8	.	.	2
60	.	2	.	.	2	.	.	.	.	2	2	.	.	2	.	.	2	.	.	.	.	2	2	2	2	.	.	2	.	.	.	2	.	.	.	4	2	.	2	.	2	.	.	.	.	.	2	2	.	2	2	4	2	2	.	.	.	.	4	2	.	.	6	2
61	.	2	.	.	.	.	2	.	.	.	.	2	.	2	.	.	2	.	.	4	4	.	2	.	.	2	.	.	.	.	2	.	4	2	.	.	.	.	4	2	2	2	.	2	2	.	2	2	.	.	2	.	2	.	.	.	.	4	2	.	.	2	.	4
62	.	.	2	.	.	.	.	2	2	2	.	2	4	.	.	.	2	.	4	2	.	.	2	2	.	.	.	4	.	.	.	2	.	2	.	.	.	.	2	4	.	2	.	.	.	.	.	.	2	.	.	2	.	2	.	2	.	.	2	2	4	2	2	2
63	.	.	2	2	.	.	2	2	2	.	.	.	.	.	.	4	.	2	2	.	.	.	2	2	2	.	.	.	2	.	2	.	2	2	.	.	4	.	.	.	.	4	.	2	.	2	.	2	4	.	.	.	.	2	.	2	2	.	.	4	.	2	.	2

Table 11. DDT of the SPEEDY S-box

B Top Pattern

We detail here the 77 patterns used for the sieving of the pairs on the plaintext together with the filter f_{pat} associated. Each vector describes a set of 3 active rows among rows 20 to 28. A 1 describes an active row. The leftmost coordinate corresponds to row 20, while the rightmost one to row 28.

1.	[1,0,1,0,1,0,0,0,0]	2.12
2.	[1,0,1,0,0,1,0,0,0]	2.15
3.	[1,0,1,0,0,0,1,0,0]	2.63
4.	[1,0,1,0,0,0,0,1,0]	2.47
5.	[1,0,1,0,0,0,0,0,1]	3.92
6.	[1,0,0,1,1,0,0,0,0]	1.91
7.	[1,0,0,1,0,1,0,0,0]	1.93
8.	[1,0,0,1,0,0,1,0,0]	2.11
9.	[1,0,0,1,0,0,0,1,0]	2.16
10.	[1,0,0,1,0,0,0,0,1]	3.70
11.	[1,0,0,0,1,1,0,0,0]	1.55
12.	[1,0,0,0,1,0,1,0,0]	1.41
13.	[1,0,0,0,1,0,0,1,0]	1.87
14.	[1,0,0,0,1,0,0,0,1]	2.37
15.	[1,0,0,0,0,1,1,0,0]	1.76
16.	[1,0,0,0,0,1,0,1,0]	1.21
17.	[1,0,0,0,0,1,0,0,1]	2.94
18.	[1,0,0,0,0,0,1,1,0]	2.46
19.	[1,0,0,0,0,0,1,0,1]	2.72
20.	[1,0,0,0,0,0,0,1,1]	3.39
21.	[0,1,1,0,1,0,0,0,0]	1.48
22.	[0,1,1,0,0,1,0,0,0]	1.48
23.	[0,1,1,0,0,0,1,0,0]	2.03
24.	[0,1,1,0,0,0,0,1,0]	1.87
25.	[0,1,1,0,0,0,0,0,1]	3.30
26.	[0,1,0,1,1,0,0,0,0]	1.46
27.	[0,1,0,1,0,1,0,0,0]	1.46
28.	[0,1,0,1,0,0,1,0,0]	1.65
29.	[0,1,0,1,0,0,0,1,0]	1.70
30.	[0,1,0,1,0,0,0,0,1]	3.07
31.	[0,1,0,0,1,1,0,0,0]	1.29
32.	[0,1,0,0,1,0,1,0,0]	1.34
33.	[0,1,0,0,1,0,0,1,0]	1.68
34.	[0,1,0,0,1,0,0,0,1]	2.28
35.	[0,1,0,0,0,1,1,0,0]	1.56
36.	[0,1,0,0,0,1,0,1,0]	1.10
37.	[0,1,0,0,0,1,0,0,1]	2.80
38.	[0,1,0,0,0,0,1,1,0]	2.28
39.	[0,1,0,0,0,0,1,0,1]	2.68
40.	[0,1,0,0,0,0,0,1,1]	3.22
41.	[0,0,1,0,0,1,1,0,0]	0.81
42.	[0,0,1,0,0,1,0,1,0]	0.63
43.	[0,0,1,0,0,1,0,0,1]	2.12
44.	[0,0,1,0,1,1,0,0,0]	0.23
45.	[0,0,1,0,1,0,0,0,1]	0.99
46.	[0,0,1,0,0,0,1,0,1]	1.87
47.	[0,0,1,0,0,0,0,1,1]	1.24
48.	[0,0,1,1,1,0,0,0,0]	0.85
49.	[0,0,1,1,0,1,0,0,0]	0.65
50.	[0,0,1,1,0,0,1,0,0]	0.98
51.	[0,0,1,1,0,0,0,1,0]	0.84
52.	[0,0,1,1,0,0,0,0,1]	2.27
53.	[0,0,1,0,1,0,1,0,0]	0.57
54.	[0,0,1,0,1,0,0,1,0]	0.40
55.	[0,0,1,0,0,0,1,1,0]	1.34
56.	[0,1,1,1,0,0,0,0,0]	2.10
57.	[0,0,0,1,0,1,1,0,0]	0.83
58.	[0,0,0,1,0,1,0,1,0]	0.60
59.	[0,0,0,1,0,1,0,0,1]	1.67
60.	[0,0,0,1,1,1,0,0,0]	0.21
61.	[0,0,0,1,1,0,0,0,1]	0.94
62.	[0,0,0,1,0,0,1,0,1]	1.93
63.	[0,0,0,1,0,0,0,1,1]	1.29
64.	[0,0,0,1,1,0,1,0,0]	0.51
65.	[0,0,0,1,1,0,0,1,0]	0.60
66.	[0,0,0,1,0,0,1,1,0]	0.65
67.	[1,1,0,1,0,0,0,0,0]	2.98
68.	[0,0,0,0,1,1,1,0,0]	0.61
69.	[0,0,0,0,1,1,0,1,0]	0.46
70.	[0,0,0,0,1,1,0,0,1]	1.53
71.	[0,0,0,0,1,0,1,0,1]	1.03
72.	[0,0,0,0,1,0,0,1,1]	1.03
73.	[0,0,0,0,1,0,1,1,0]	1.05
74.	[0,0,0,0,0,1,1,1,0]	1.05
75.	[0,0,0,0,0,1,1,0,1]	1.65
76.	[0,0,0,0,0,1,0,1,1]	1.08
77.	[1,1,0,0,0,1,0,0,0]	1.68