Improved Differential Cryptanalysis of Round-Reduced Speck

Itai Dinur

Département d'Informatique, École Normale Supérieure, Paris, France

Abstract. Simon and Speck are families of lightweight block ciphers designed by the U.S. National Security Agency and published in 2013. Each of the families contains 10 variants, supporting a wide range of block and key sizes. Since the publication of Simon and Speck, several research papers analyzed their security using various cryptanalytic techniques. The best previously published attacks on all the 20 round-reduced ciphers are differential attacks, and are described in two papers (presented at FSE 2014) by Abed et al. and Biryukov et al.

In this paper, we focus on the software-optimized block cipher family Speck, and describe significantly improved attacks on all of its 10 variants. In particular, we increase the number of rounds which can be attacked by 1, 2, or 3, for 9 out of 10 round-reduced members of the family, while significantly improving the complexity of the previous best attack on the remaining round-reduced member. Our attacks use an untraditional key recovery technique for differential attacks, whose main ideas were published by Albrecht and Cid at FSE 2009 in the cryptanalysis of the block cipher PRESENT.

Despite our improved attacks, they do not seem to threaten the security of any member of Speck.

Keywords: Lightweight block cipher, Speck, cryptanalysis, differential attack, key recovery.

1 Introduction

In 2013 the U.S. National Security Agency published the Simon and Speck families of lightweight block ciphers [7]. Each block cipher family contains 10 variants and supports block sizes ranging from 32 to 128 and key sizes ranging from 64 to 256 bits. Both families of block ciphers have a simple and compact Feistel-like[1] design, but are optimized for different applications, where Simon is optimized for hardware and Speck is optimized for software implementations. Thus, Simon uses the basic hardware-friendly arithmetic operations of XOR, bitwise AND and bit rotation, whereas Speck is a pure ARX cipher (i.e., it uses modular addition, bit rotation and XOR operations).

[1] Simon is a Feistel structure, while Speck can be represented as a composition of two Feistel maps [7].

Since their publication, Simon and Speck received significant media attention, and were also subjects of extensive research in the cryptographic community, as several papers analyzed their security and performance [1, 4, 5, 9, 21]. In general, the best published attacks on all 20 round-reduced ciphers are differential attacks, described in the two papers [1, 9]. However, despite the extensive analysis, all 20 variants seem to have a sufficiently large security margin, and the current attacks do not threaten their security.

In this paper, we present improved attacks on all 10 members of the Speck family of block ciphers. In particular, we increase the number of rounds which can be attacked by 1, 2, or 3, for 9 out of 10 members of the family, while significantly improving the complexity of the previous attack on the remaining member. More specifically, we increase the number of rounds which can be attacked by 1 for 4 members, by 2 for 2 members, and by 3 for 3 members. In 3 of these cases, not only do we attack more rounds, but we also improve the complexity of the best previous attacks, which were applied to a weaker cipher. Moreover, in all of these cases, our attacks use less data than the previous attacks, and all of them require only a few megabytes of memory (typically improving the previous attack with respect to this parameter as well).

Surprisingly, our attacks do not exploit any newly found differential characteristic of Speck. In fact, our attacks completely reuse the characteristics presented in [1, 9], but are based on a significantly improved key recovery framework. As the basic idea behind this framework in very simple, at first, it seems quite strange that it was missed by the previous analysis. However, a closer look reveals that our key recovery technique is quite different from traditional techniques used in differential cryptanalysis. These key recovery techniques (called *counting* techniques) were published with the introduction of differential cryptanalysis [8]. Counting techniques remain, by far, the most common techniques to recover the key in differential attacks, and were thus naturally applied in the previous differential attacks on Speck [1, 9].

One of the main features of counting techniques in differential attacks is that the key material is typically recovered in chunks (i.e., in a divide-and-conquer manner) using statistical analysis. In order to recover a chunk of key material (e.g., some bits of the first and last round-keys), we analyze encrypted input pairs, each pair suggesting a value (or a few values) for this chunk. Right pairs (conforming to the characteristic) always suggest the correct value for the chunk, while wrong pairs suggest an arbitrary value. In order to be able to distinguish the correct suggestions from the incorrect ones, we require strong filtering which eliminates a large fraction of the wrong pairs (and the arbitrary key suggestions). Such a strong filtering requirement places a restriction on the number of rounds of the iterated block cipher which we can attack with a given differential characteristic. Namely, in order to attack an $(r + a)$-round cipher with an r-round characteristic, a needs to be sufficiently small such that the characteristic can be extended to deduce some linear constraints on the output of the cipher (e.g., some bits of the output difference are zero), allowing us to filter many of the wrong pairs.

In contrast to standard key recovery techniques (in particular, the ones used in previous attacks on Speck), in this paper we extend a differential characteristic by a (relatively) large number of rounds, and thus simple linear filtering can eliminate only a small fraction of the data. Consequently, we remain with too many suggestions for the key to mount an efficient attack using counting. On the other hand, this situation resembles some self-similarity attacks (such as reflection-based attacks [12–14]), in which the attacker does not have any characteristic that allows simple filtering. In such self-similarity attacks, the attacker encrypts multiple plaintexts and awaits a special event to occur (such as a reflection). The internal properties of the cipher assure that once this event occurs, the problem of attacking the full cipher is reduced to a simpler problem of attacking a sub-cipher with fewer rounds. The sub-cipher attack calculates suggestions for *the full secret key*, which the attacker tests using trial encryptions on the full cipher.[2] Since the attacker cannot detect the occurrence of the special event, the sub-cipher attack is executed for each plaintext (or plaintext structure). Thus, the complexity of the full attack is determined by the probability of the awaited event (which determines how many sub-cipher attacks we need to execute), and the average complexity of the sub-cipher attack.

In the scenario presented above for self-similarity attacks, the key is recovered in one chunk by a sub-cipher attack. However, there is nothing that prevents us from applying similar techniques in differential attacks. In fact, the last a rounds of the cipher in differential attacks can be viewed as a sub-cipher, and assuming the event that an encrypted pair conforms to the r-round characteristic (i.e., it is a right pair), we can mount a sub-cipher attack to obtain key suggestions for each encrypted pair, and enumerate each one of them, testing it using trial encryptions. As a right pair will always suggest the correct key value, the attack succeeds as soon as we finish executing the sub-cipher attack on this pair.

This generic key recovery framework for differential cryptanalysis was first proposed by Albrecht and Cid in [2], where it was applied to the block cipher PRESENT (and was further used in followup publications such as [3, 22]). Albrecht and Cid used algebraic techniques to enhance differential cryptanalysis, and specifically, devised Attack-C which formulates the sub-cipher as a system of non-linear equations, and solves it using algebraic tools (e.g., SAGE [20]). On the other hand, the sub-cipher attack can use various methods which do not necessarily exploit algebraic tools. Indeed, while we use the same framework as [2], our sub-cipher attack on Speck applies guess-and-determine techniques, and does not directly solve any system of non-linear equations. Furthermore, in [6] the generic framework was (implicitly) applied to the block cipher Zorro using a complex two-phase sub-cipher attack (in which the only equation systems directly solved are linear).

We stress that the *only* difference between our approach and the algebraic approach of Attack-C [2], is in the details of the sub-cipher attack. While this is a subtle difference, we believe that part of the reason that Attack-C was not

[2] Examples of sub-cipher attacks include the meet-in-the-middle and guess-and-determine attacks on round-reduced GOST, described in [12, 13].

considered in the previous attacks on Speck [1, 9], is that it promoted the use of black-box algebraic tools to perform the sub-cipher attack. As such black-box algorithms are often highly heuristic, and their running time is not very well understood, they have not become mainstream analysis tools. In this paper, we show that the sub-cipher attack can sometimes be performed by a simple algorithm with a better understood running time, and we hope that cryptanalysts will consider similar attacks in the future.

In order to generalize Attack-C of [2] to a broader key recovery framework for differential attacks, we call it an *enumeration framework*, as it enumerates suggestions for the full key proposed by a sub-cipher attack. This should be contrasted with *counting* techniques which extract partial key material from a few rounds of the cipher using statistical analysis (e.g., the 1, 2 and 3-round attacks of [8]).

In most cases, counting techniques for differential attacks seem to give the best results. This is perhaps due to the reason that when we extend the characteristic beyond the reach of these techniques, the sub-cipher attack becomes too expensive (as it needs to analyze dependent round-keys according to the key schedule), making the full differential attack inefficient.[3] However, as we show in this paper, in the case of Speck, the sub-cipher attack can be performed very efficiently, and results in significantly improved differential attacks (as in the case of Zorro [6]).

As previously mentioned, our sub-cipher attack on Speck is a guess-and-determine attack, and it is related to the similar attack of [12]. Furthermore, since Speck is an ARX cipher, we use techniques that were developed in the analysis of ARX cryptosystems and similar designs. In particular, our tools are related to several search algorithms for differential characteristics on these designs, such as [10, 15, 16, 18, 24].

The rest of this paper is organized as follows. We introduce our notation in Section 2, and provide a brief description of Speck in Section 3. The previous and our new results on the 10 Speck variants are summarized in Section 4. In Section 5, we describe the auxiliary algorithms used in our attacks (and in particular, overview the specific sub-cipher attack on Speck), while our full differential attacks in the enumeration framework are described in Section 6. Finally, we give the details of the sub-cipher attack in Section 7, and conclude the paper in Section 8.

2 Notations and Conventions

In this section, we describe the notations and conventions used in the rest of the paper.

Given a positive integer r, we denote by $x \ggg i$ the n-bit word obtained by rotating x by i bits to the right, and by $x \lll i$ the word obtained by rotating x by i bits to the left. Similarly, $x \gg i$ and $x \ll i$ denote a bitwise shift of x by i

[3] We note that analysis of dependent round-keys can sometimes be performed efficiently using algebraic tools, as claimed in [2].

bits to the right and left, respectively. We denote by $\neg x$ the bitwise negation of x.

Given two n-bit words x and y, we denote by $x \oplus y$ their n-bit XOR, by $x \boxplus y$ their n-bit addition over $GF(2^n)$, and by $x \boxminus y$ their difference over $GF(2^n)$. We further denote by $x \wedge y$ the bitwise AND of x and y.

Given an n-bit word x, we denote its i'th bit for $i \in \{0, 1, \ldots n-1\}$ by $x^{[i]}$. We note that operations on the bit indexes are performed modulo n, e.g. $x^{[n+5]} \equiv x^{[5]}$.

Conventions Throughout this paper, we use the standard conventions and calculate the time complexity of our attacks in terms of evaluations of the full cipher. The memory complexity of the attacks is calculated in terms of bytes.

3 Description of Speck

In this section, we give a short description of Speck. More details can be found in [7].

Speck is a family of block ciphers containing 10 variants. The variants are characterized by a block size of $2n$ bits (where n is the internal word size), and a key size of mn bits. The 10 variants are identified with a $2n/mn$ label, and defined with rotation constants α and β and a number of rounds T, as shown in Table 1.

The key schedule of Speck expands the initial m-word master key $(\ell_{m-2}, ..., \ell_0, k_0)$ into T round-key words $k_0, k_1, ..., k_{T-1}$ according to the following algorithm:

for $i = 0 \ldots T-2$ **do**
  $\ell_{i+m-1} \leftarrow (k_i \boxplus (\ell_i \ggg \alpha)) \oplus i$
  $k_{i+1} \leftarrow (k_i \lll \beta) \oplus \ell_{i+m-1}$
end for

The encryption function of Speck encrypts a plaintext of two n-bit words $P = (x_0, y_0)$, into a ciphertext $C = (x_T, y_T)$, using a sequence of T rounds according to the following algorithm (see Figure 1 for the round function):

for $i = 0 \ldots T-1$ **do**
  $x_{i+1} \leftarrow ((x_i \ggg \alpha) \boxplus y_i) \oplus k_i$
  $y_{i+1} \leftarrow (y_i \lll \beta) \oplus x_{i+1}$
end for

4 Summary of Previous and New Attacks on Speck

In this section, we summarize the previous and our new attacks on Speck, referring to Table 2. As the Speck family contains 10 variants, and each variant was analyzed by several papers, exhaustively listing all the dozens of previous

Variant $2n/mn$	Word Size n	Key Words M	Rounds T	α	β
32/64	16	4	22	7	2
48/72	24	3	22	8	3
48/96	24	4	23	8	3
64/96	32	3	26	8	3
64/128	32	4	27	8	3
96/96	48	2	28	8	3
96/144	48	3	29	8	3
128/128	64	2	32	8	3
128/192	64	3	33	8	3
128/256	64	4	34	8	3

Table 1. The Speck Family of Block Ciphers

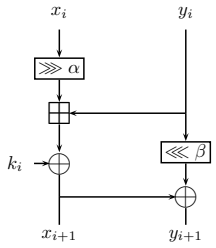


Fig. 1. The Round-Function of Speck

attacks is too tedious. Instead, for each Speck variant, we first choose the attacks which break the most number of rounds, and among these, we only refer to the attack with the best time complexity. As shown in Table 2, all the best previous attacks were described in the two papers [1, 9], and we additionally note that all of them are based on differential cryptanalysis and related techniques (such as rectangle attacks).

For each variant of Speck, Table 2 summarizes our attack which breaks the most number of rounds. We note that for each variant, we can also use our techniques to attack fewer rounds (using a shorter differential characteristic), but once again, we do not explicitly refer to these numerous attacks in this paper (with the exception of the 32/64 variant). As our attacks reuse the differential characteristics of [1, 9], we refer to these characteristics in the table, while describing them in more detail in Table 3. We note that since the internal differential transitions in each characteristic are not relevant for our attacks, Table 3 only gives the input and output differences for each characteristic, while their complete specification is described in [1, 9].

We now highlight some interesting features of the attacks summarized in Table 2. We first look at the 32/64 variant, on which the best previous attack could break 11 out of its 22 rounds, with data complexity of about a quarter of the entire code-book. Compared to this attack, our attack on 11 rounds uses less than a square root of the code-book (2^{14} plaintexts), requires less memory and has a slightly better time complexity. Furthermore, we can attack up to 13 rounds in time complexity which is significantly faster than exhaustive search, and up to 14 rounds with a marginal attack. For additional 2 variants (48/96 and 64/128), we can attack 3 more rounds than the best previous attack. For the 2 variants 48/72 and 48/96, we increase the number of rounds that can be attacked by 2. For 4 variants (96/96, 96/144, 128/128 and 128/256), we can attack 1 more round than the best previous attack. Note that for the 3 variants 96/96, 96/144 and 128/128, our attacks are also more efficient than the previous attacks in all complexity parameters (and particularly use much less memory). Finally, for the 128/192 variant, we attack the same number of rounds as the best previous attack, but improve it in all complexity parameters, and in particular use much less memory.

5 Auxiliary Algorithms Used by Our Attacks

In this section, we describe the two auxiliary algorithms that are used by our attacks on Speck.

5.1 Key-Schedule Inversion

Given a sequence of m key words $k_{j-m}, \ldots, k_{j-1}$ for any $j \in \{m, m+1, \ldots, T\}$, we can efficiently invert the key schedule and calculate the master key: first, we determine k_{j-m-1} using the following key schedule equalities

$$\ell_{j+m-3} = k_{j-1} \oplus (k_{j-2} \lll \beta)$$
$$\ell_{j-2} = ((\ell_{j+m-3} \oplus (j-2)) \boxminus k_{j-2}) \lll \alpha$$
$$k_{j-m-1} = (k_{j-m} \oplus \ell_{j-2}) \ggg \beta.$$

Next, given $k_{j-m-1}, \ldots, k_{j-2}$, we iteratively continue the inversion of the key schedule and derive the master key.

5.2 Overview of the 2-Round Attack on Speck

In our basic attacks on Speck, we use an r-round differential characteristic with an initial difference, denoted by $(\Delta x_0, \Delta y_0)$, and a final difference, denoted by $(\Delta x_r, \Delta y_r)$. We devise an attack on $r+2$ rounds using a 2-round attack.

The enumeration framework poses the following problem: the $2n$-bit input difference $(\Delta x_r, \Delta y_r)$ to the last 2 rounds is fixed by the final difference of the differential characteristic, and the output difference $(\Delta x_{r+2}, \Delta y_{r+2})$ is known

Variant $2n/mn$	Rounds Attacked/ Total Rounds	Time	Data (CP)	Memory	Reference	Characteristic ID
32/64	11/22	$2^{46.7}$	$2^{30.1}$	$2^{37.1}$	[1]	-
32/64	**11/22**	2^{46}	2^{14}	2^{22}	This Paper	1
32/64	**12/22**	2^{51}	2^{19}	2^{22}	This Paper	2
32/64	**13/22**	2^{57}	2^{25}	2^{22}	This Paper	3
32/64	**14/22**	2^{63}	2^{31}	2^{22}	This Paper	4
48/72	12/22	2^{43}	2^{43}	NA	[9]	-
48/72	**14/22**	2^{65}	2^{41}	2^{22}	This Paper	5
48/96	12/23	2^{43}	2^{43}	NA	[9]	-
48/96	**15/23**	2^{89}	2^{41}	2^{22}	This Paper	5
64/96	16/26	2^{63}	2^{63}	NA	[9]	-
64/96	**18/26**	2^{93}	2^{61}	2^{22}	This Paper	6
64/128	16/27	2^{63}	2^{63}	NA	[9]	-
64/128	**19/27**	2^{125}	2^{61}	2^{22}	This Paper	6
96/96	15/28	$2^{89.1}$	2^{89}	2^{48}	[1]	-
96/96	**16/28**	2^{85}	2^{85}	2^{22}	This Paper	7
96/144	16/29	$2^{135.9}$	$2^{90.9}$	$2^{94.5}$	[1]	-
96/144	**17/29**	2^{133}	2^{85}	2^{22}	This Paper	7
128/128	16/32	2^{116}	2^{116}	2^{64}	[1]	-
128/128	**17/32**	2^{113}	2^{113}	2^{22}	This Paper	8
128/192	18/33	$2^{182.7}$	$2^{126.9}$	$2^{121.9}$	[1]	-
128/192	**18/33**	2^{177}	2^{113}	2^{22}	This Paper	8
128/256	18/34	$2^{182.7}$	$2^{126.9}$	$2^{121.9}$	[1]	-
128/256	**19/34**	2^{241}	2^{113}	2^{22}	This Paper	8

The "Characteristic ID" column refers to the IDs of the characteristics in Table 3, which are used in our attacks. The data is given in chosen plaintexts (CP).

Table 2. Previous Attacks and Our New Attacks on Speck

from the output. Furthermore, we are given actual output values (x_{r+2}, y_{r+2}) and $(x_{r+2} \oplus \Delta x_{r+2}, y_{r+2} \oplus \Delta y_{r+2})$. Our objective is to find all the possible independent round keys k_r and k_{r+1}, under which the difference of the 2-round partial decryptions of the pair (x_{r+2}, y_{r+2}) and $(x_{r+2} \oplus \Delta x_{r+2}, y_{r+2} \oplus \Delta y_{r+2})$ is equal to $(\Delta x_r, \Delta y_r)$. In general, we have $2n$ bits of variables and $2n$ bits of constraints (derived from the difference $(\Delta x_r, \Delta y_r)$). Thus, the problem can be formulated using an equation system, which has an average of one solution for an arbitrary pair of outputs (x_{r+2}, y_{r+2}) and $(x_{r+2} \oplus \Delta x_{r+2}, y_{r+2} \oplus \Delta y_{r+2})$. The

ID	Variant $2n$	Rounds	Probability	Reference	Input/Output Differences
1	32	7	2^{-13}	[1]	211 a04/850a 9520
2	32	8	2^{-18}	[1]	a60 a205/850a 9520
3	32	9	2^{-24}	[1]	a60 a205/802a d4a8
4	32	10	2^{-30}	[9]	8054 a900/40 542
5	48	11	2^{-40}	[9]	202040 82921/80a0 2085a4
6	64	15	2^{-60}	[9]	9 1000000/40024 4200d01
7	96	14	2^{-84}	[1]	2a20200800a2 322320680801/ 1008004c804 c0180228c61
8	128	15	2^{-112}	[1]	144304280c010420 6402400040024/ 180208402886884 80248012c96c80

The n-bit halves in each input/ouput difference are separated by a space.
Table 3. Differential Characteristics Used in Our Attacks

goal of the 2-round attack is to enumerate all the possible solutions for each given output pair as efficiently as possible.[4]

We note that it is not trivial that the equation system has an average of one solution, as the pairs of outputs are ciphertexts, whose corresponding plaintexts have the fixed initial difference $(\Delta x_0, \Delta y_0)$ to the characteristic. If such a plaintext pair diverges from the characteristic at its later rounds, then the difference after r rounds can potentially be close to $(\Delta x_r, \Delta y_r)$, which may result in non-random behavior. In fact, our experiments show that the average number of solutions is about 4 for characteristic 1 in Table 3, which has a relatively high probability of 2^{-13}. However, for the lower probability characteristics which we could test experimentally, the average number of solutions was only slightly higher than 1 (and lower than 2).

Our 2-round attack is given in Section 7. This attack exploits the (relative) simplicity of the Speck round function in order to recover the 2 final round keys of Speck with very low average time complexity. Indeed, our experiments show that for an output pair (x_{r+2}, y_{r+2}) and $(x_{r+2} \oplus \Delta x_{r+2}, y_{r+2} \oplus \Delta y_{r+2})$ (generated by plaintexts with the fixed initial difference $(\Delta x_0, \Delta y_0)$), the 2-round attack requires an average time which is smaller than 2 time units (i.e., 2 full encryptions of round-reduced Speck) for any characteristic that we use in the full differential attacks on Speck.

[4] Recall that we have essentially no (linear) filtering conditions, and thus we must execute the sub-cipher attack for each encrypted input pair. Consequently, we are interested in the average time complexity of the algorithm.

6 Details of the Full Differential Attacks

In this section, we describe the details of our full differential attacks on Speck in the enumeration framework. In all the attacks, we assume that we have a differential characteristic that covers r rounds of the cipher with probability $p > 2 \cdot 2^{-2n}$. The attacks recover the mn-bit secret key of a variant with $r+m$ rounds using $2 \cdot p^{-1}$ chosen plaintexts, in expected time complexity of $2 \cdot p^{-1} \cdot 2^{(m-2)n}$. In other words, our attacks are faster than exhaustive search by a factor[5] of $p \cdot 2^{2n-1}$. For example, our attack on 11-round Speck 32/64 (with $m = 4$) uses a characteristic for $11 - 4 = 7$ rounds with $p = 2^{-13}$. Thus, its time complexity is $2 \cdot 2^{13} \cdot 2^{(4-2)16} = 2^{46}$, i.e. it is faster than exhaustive search for the 64-bit key by a factor of $p \cdot 2^{2n-1} = 2^{-13} \cdot 2^{31} = 2^{18}$.

The Full Differential Attack for $m = 2$ We first present the details of our attack for the Speck instances with $m = 2$ key words (and $m + 2$ rounds), and then extend the attack to the remaining instances, in which $m = 3$ or $m = 4$. We denote the initial difference of the characteristic (at the input of the cipher) by $(\Delta x_0, \Delta y_0)$, and its final difference (after r rounds) by $(\Delta x_r, \Delta y_r)$.

1. Request the encryptions of p^{-1} plaintext pairs P and $P' = P \oplus (\Delta x_0, \Delta y_0)$, and denote the ciphertexts by C and C', respectively. For each plaintext pair P and P':
   (a) Execute the 2-round attack of Section 7 using $(\Delta x_r, \Delta y_r)$, C and C'.
   (b) For each returned value of k_r and k_{r+1}, iteratively calculate $k_{r-1}, \ldots, k_0$ (as described in Section 5.1), and finally recover the master key. Test the master key using trial encryptions, and return it if the trial encryptions succeed.[6]

The attack requires $2 \cdot p^{-1}$ chosen plaintexts, and given that the r-round characteristic has probability p, we expect one plaintext pair P, P' to be a right pair (i.e., to follow the characteristic, and have a difference of $(\Delta x_r, \Delta y_r)$ after r rounds). Given the ciphertexts C, C', corresponding to the right pair, the 2-round attack will find the correct key, which will be returned by the full attack.

According to the analysis of Section 7, the 2-round attack has an average time complexity which is smaller than 2 time units, and thus the average processing time for each analyzed plaintext pair remains about 2. This implies that the total time complexity of the attack is about $2 \cdot p^{-1}$.

[5] Note that information theoretically, without considering the internal transitions of the differential characteristic, $p \cdot 2^{2n-1}$ is the best improvement factor that one can hope for, given $2 \cdot p^{-1}$ data.

[6] This step can be slightly optimized to replace many of the full trial encryptions by lighter Speck round evaluations, if we consider the internal transitions of the differential characteristic: while iteratively calculating $k_{r-1}, \ldots, k_0$, we partially decrypt C and C', and verify that they satisfy the differential characteristic for each round. If the verification fails for some round, we discard the key and continue.

The Full Differential Attack for $m = 3$ and $m = 4$ For $m = 3$ and $m = 4$, we attack variants with $r + 3$ and $r + 4$ rounds, respectively. The attacks on $m = 3$ and $m = 4$ are trivial extensions of the attack on the $m = 2$ variants, and work by guessing the last 1 and 2 round keys, respectively. Then, for each guess we apply a similar attack to the one applied for $m = 2$.

The data complexity of the attacks remain $2 \cdot p^{-1}$, while the time complexity increases to $2 \cdot p^{-1} \cdot 2^n$ and $2 \cdot p^{-1} \cdot 2^{2n}$ for $m = 3$ and $m = 4$, respectively.

7 The 2-Round Attack

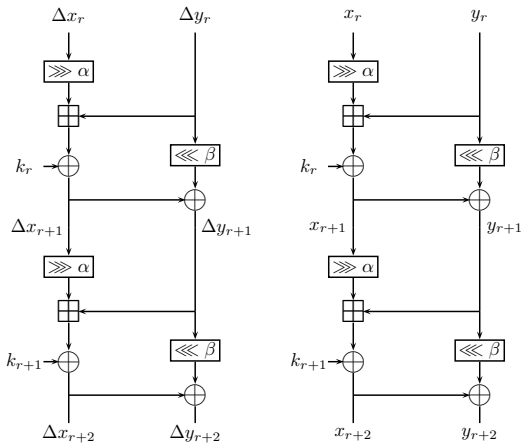
In this section, we present the details of our 2-round attack on Speck. As described in Section 5.2, we have an input difference $(\Delta x_r, \Delta y_r)$ to the two rounds (which is fixed by a differential characteristic), and we are given the actual output values (x_{r+2}, y_{r+2}) and $(x_{r+2} \oplus \Delta x_{r+2}, y_{r+2} \oplus \Delta y_{r+2})$. Our goal is to enumerate all the possible independent round keys k_r and k_{r+1}, under which the difference of the 2-round partial decryptions of the pair (x_{r+2}, y_{r+2}) and $(x_{r+2} \oplus \Delta x_{r+2}, y_{r+2} \oplus \Delta y_{r+2})$ is equal to $(\Delta x_r, \Delta y_r)$.

The notation we use in our analysis is given in Figure 2, where the XOR differential notation is given on the left, and the notation of the intermediate encryption values for (x_{r+2}, y_{r+2}) is given on the right. We further define $(x'_i, y'_i) = (x_i \oplus \Delta x_i, y_i \oplus \Delta y_i)$.

Note that $\Delta y_{r+1} = (\Delta x_{r+2} \oplus \Delta y_{r+2}) \ggg \beta$ and $\Delta x_{r+1} = \Delta y_{r+1} \oplus (\Delta y_r \lll \beta)$ are independent of the keys and can be calculated immediately. Thus, all the XOR differences in the scheme are completely determined. Similarly, the value $y_{r+1} = (x_{r+2} \oplus y_{r+2}) \ggg \beta$ can be calculated from the known (x_{r+2}, y_{r+2}), whereas (x_r, y_r) and x_{r+1} remain unknown. We further note that deriving the two round-keys is equivalent to deriving x_r and x_{r+1}, as their values allow us to calculate $k_{r+1} = (y_{r+1} \boxplus (x_{r+1} \ggg \alpha)) \oplus x_{r+2}$, and as $y_r = (x_{r+1} \oplus y_{r+1}) \ggg \beta$, then $k_r = (y_r \boxplus (x_r \ggg \alpha)) \oplus x_{r+1}$ can be calculated as well. Thus, in the following, we concentrate on deriving the intermediate values x_r and x_{r+1}.

7.1 A Basic 2-Round Attack

The problem of solving differential equations of addition (DEA) of the form $(x \oplus \delta_1) \boxplus (y \oplus \delta_2) = (x \boxplus y) \oplus \delta_3$ (where $\delta_1, \delta_2, \delta_3$ are given and x, y are unknown variables) is a basic problem in the analysis of ARX cryptosystems, and was extensively studied in several papers. In particular, [19] described an algorithm for solving such equations in time complexity which is linear in the total number of solutions. However, the previous algorithm is not directly applicable in our case, as we actually have two dependant equation systems (generated by two addition operations), and we want to efficiently solve them simultaneously. Moreover, the value of y in the DEA is fixed to y_{r+1} for one of the addition operations, and since the solutions vary according to this fixed value, the second equation system is of a different type than the one analyzed in [19]. Note that a standard DEA



Our notation of differences is given on the left, whereas our notation of values is given on the right.

Fig. 2. Two Rounds of Speck

has an average of 2^n solutions, whereas our equation system has an average of only 1 solution.

Given the complications above, it seems difficult to construct a generic algorithm that efficiently solves our type of equation systems for an arbitrarily large word size n. Thus, we concentrate on the word sizes $n \in \{16, 24, 32, 48, 64\}$ defined by the Speck family, and describe an algorithm whose complexity is estimated by experiments (rather than by a rigorous theoretical proof). As the full key recovery attack of Section 6 calls the 2-round attack with a fixed value of $(\Delta x_r, \Delta y_r)$ and many values of $(x_{r+2}, y_{r+2}), (x'_{r+2}, y'_{r+2})$, we are interested in the average complexity of the 2-round attack, where $(x_{r+2}, y_{r+2}), (x'_{r+2}, y'_{r+2})$ are chosen at random according to the procedure of the full key recovery attack.

In Appendix A, we devise a basic guess-and-determine algorithm which exploits the limited carry propagation of the addition operation in order to compute x_r and x_{r+1} bit by bit. It is related to several previous guess-and-determine algorithms such as the one of [12]. The analysis described in Appendix A shows that given some randomness assumptions on the problem, its average execution time is comparable to the execution time of a full Speck encryption.

The problem with this analysis is that the randomness assumptions do not hold in our case (as in many cases of DEA). In fact (as we show next), although we expect one solution on average, for almost any value of $(\Delta x_r, \Delta y_r)$, the distribution of solutions across the various instances is very far from uniform, and greatly depends on the values of the output pairs (x_{r+2}, y_{r+2}) and (x'_{r+2}, y'_{r+2}). More specifically, the solutions are distributed among a small fraction of the output pairs, whereas for the remaining output pairs, there are no solutions at

all. Such non-randomness properties have a negative effect of the performance of our basic guess-and-determine algorithm, as it can potentially make a large number of guesses for some bits of x_{r+1} and x_r (i.e., guess partial solutions), while discarding all (or a large fraction) of them at a later stage.[7] Nevertheless, the theoretical analysis based on randomness assumptions strongly indicates that an optimized variant of the attack can perform very efficiently.

7.2 Optimizing the Basic 2-Round Attack Using Filters

In order to optimize the basic algorithm, we notice that we can filter out very quickly a large fraction of the non-useful instances (with no solutions). The idea is to use efficient "look-ahead" (non-linear) filters that try to find a contradiction in the equation constraints before actually computing the solutions. The techniques we use to implement the filters are closely related to various search algorithms for differential characteristics for ARX-based and related cryptosystems (e.g., [10, 15, 16, 18, 24]).

These filtering techniques allow us to concentrate our efforts on a small fraction of "interesting" instances, and obtain an algorithm whose average time complexity is estimated (according to our simulations) to be smaller than 2 encryptions of Speck.[8]

One-Bit Filter This filter can be applied to any standard DEA $(x \oplus \delta_1) \boxplus (y \oplus \delta_2) = (x \boxplus y) \oplus \delta_3$. It was first described in [17], and it checks whether

$$eq(\delta_1 \ll 1, \delta_2 \ll 1, \delta_3 \ll 1) \wedge (\delta_1 \oplus \delta_2 \oplus \delta_3 \oplus (\delta_1 \ll 1)) = 0,$$

where $eq(a, b, c) = (\neg a \oplus b) \wedge (\neg a \oplus c)$ equals one at position i if and only if $a^{[i]} = b^{[i]} = c^{[i]}$. As shown in [17], a DEA for which the n-bit value of the filter is non-zero has no solutions and can be filtered out immediately.

This filter is called a 1-bit filter since for each bit position $i + 1$, it only depends the single bit position $i + 1$ of the input words (in addition to a 1-bit XOR difference $\delta_1 \oplus \delta_2 \oplus \delta_3$ at the previous position i). As applying the 1-bit filter involves only a few simple word operations, it requires much less time than a full Speck encryption, and given that it immediately filters out a large fraction of instances with no solutions, it can significantly reduce the running time of the algorithm.

In order to get an estimation of how the filter performs, we assume that all the values of $\delta_1, \delta_2, \delta_3$ are chosen at random. In this case, using the formula of [17],

[7] For example, assume that we want to solve the standard DEA over 16-bit words (given in hexadecimal) $(x \oplus 0000) \boxplus (y \oplus 0000) = (x \boxplus y) \oplus 8000$. If we solve the system from the LSB to the MSB, then we consider all 2^{30} partial solutions to the 15 LSBs of x and y, and then discard all of them at the MSB.

[8] We note that after applying the filters, one can try to apply standard counting techniques to recover some key bits in few first rounds of Speck. However, as we can solve the full equation system and test each suggested key efficiently, the counting techniques are not likely to significantly improve the complexity of the attacks.

an instance of a DEA will have a solution with probability of $q = 1/2 \cdot (7/8)^{n-1}$. Specifically, for $n = 16$, we have $q \approx 2^{-4}$, and for $n = 24, 32, 48, 64$ we have $q \approx 2^{-5.5}, 2^{-7}, 2^{-10}, 2^{-13}$, respectively.

In the case of Speck, we can immediately apply the filter once for round r and once for round $r+1$ (since all the XOR differences at the inputs and outputs of the addition operations are known). However, as there are clear dependencies between the various input and output XOR differences in 2 rounds of Speck (in fact, the input differences $\Delta x_r, \Delta y_r$ are fixed by the characteristic), the formula does not apply. Nevertheless, our experiments show that for the values of $\Delta x_r, \Delta y_r$ used in our attacks, the filters actually give slightly better results than expected from a random instance. This can be partially explained since $\Delta x_r, \Delta y_r$ have a relatively low hamming weight, as they are outputs of a high probability differential characteristic. As a result, the equality predicate in the filter of round r, $eq(\delta_1 \ll 1, \delta_2 \ll 1, \delta_3 \ll 1) = 0$, holds with a probability which is lower than the expected $1/4$.

As described above, our simulations show that we remain with less than a $2^{-8}, 2^{-11}, 2^{-14}, 2^{-20}, 2^{-26}$ fraction of the instances for $n = 16, 24, 32, 38, 64$, respectively, after applying the two 1-bit filters on rounds r and $r+1$. Although the fraction of remaining instances is small, our experiments indicate that when executing the basic algorithm on this small fraction, there are still instances on which we waste a lot of time computing partial solutions that are later discarded. For the smaller values of $n = 16$ and $n = 24$, the effect of these wasteful instances on the average complexity of the algorithm seems to be limited. However, for $n \in \{32, 48, 64\}$, their effect seems to be more significant, and is also more difficult to predict, as we can sample only a small fraction of the possible output pairs after $r+2$ rounds. Consequently, we use additional filters in order to further reduce the number of wasted partial solutions.

Multiple-Bit Filters These filters are generalizations of the 1-bit filters to larger blocks. They are built by breaking a system of n constraints into b-bit blocks of constraints with a relatively small number of parameters (e.g., a few bits of $\delta_1, \delta_2, \delta_3$ in a standard DEA), and analyzing each block independently. Obviously, if we encounter an equation system instance for which a block (with a certain value of the parameters) has no solutions for any possible values of its variables (e.g., a few bits of x and y in a standard DEA), then the full system has no solutions, and we can stop analyzing it. Given that the number of parameters that appear in a block is sufficiently small, we can exhaustively precompute and store for each of their possible values, a bit that specifies whether the block can potentially have a solution of not (by checking if the equations are satisfied for all possible values of its variables).

For a general system of equations, each block will contain many parameters and variables even for a small value of b, and thus the approach is useless. However, in our case, we can efficiently implement the b-bit filters, as it is easy to break simple equation systems based on ARX into blocks which are almost independent (the only dependency between the blocks are a few bits of carry).

The details of the b-bit filters that we use for our attacks are given in Appendix B. As described in the appendix , we select $b = 6$, and constructing the filters requires negligible precomputation time (compared to the time complexity of the full differential attacks), while their storage requires only a few megabytes of memory.

7.3 The Optimized 2-Round Attack

The details of our optimized 2-round attack are given in Appendix C. We implemented the optimized 2-round attack and estimated its average complexity by running the differential attack of Section 5.2 for all the analyzed 10 Speck variants. For Speck variants with $m > 2$, we ran 2 types of tests: one type in which we arbitrarily guessed the values of round keys k_{r+3}, k_{r+2} (or only k_{r+2} for $m = 3$), and one type in which we assumed that their correct value is known. In each test, we executed the optimized 2-round attack for a few randomly chosen keys with 2^{30} arbitrary input pairs that have the fixed input difference of the corresponding characteristic $(\Delta x_0, \Delta y_0)$. In all the tests, the average number of discarded partial solutions for an analyzed pair (which determines the average running time of the algorithm, as the expected number of solutions is close to 1) was smaller than 4, and we estimate that the average time complexity of the 2-round attack is smaller than 2 full encryptions of Speck. We note that we are somewhat less confident in the results for Speck instances with larger word sizes of $n \in \{32, 48, 64\}$, as we can only sample a small fraction of the possible input pairs. However, given the very good results obtained for $n \in \{16, 24\}$, it seems reasonable to believe that the quality of our approximations does not degrade significantly, and the performance of the algorithm is close to what is claimed.

8 Conclusions

In this paper, we presented significantly improved attacks on all 10 variants of the lightweight block cipher Speck, based on an enumeration framework for differential cryptanalysis. This framework tests suggestions for the key that are calculated by a sub-cipher attack, generalizing the algebraic-based framework of Albrecht and Cid [2]. The type of attacks presented in this paper can potentially break a cipher with many more rounds than the number covered by a differential characteristic, especially when the cipher uses a secret key which is larger than the block size (e.g., in our attack on 14-round Speck 32/64, we used a 10-round characteristic to attack 14 rounds of the cipher). Consequently, such sub-cipher attacks should be considered by designers when proposing new cryptosystems.

Since the framework is generic, finding any improved differential characteristic for a Speck variant would (almost) immediately give an improved attack on the full cipher (without the need to perform the low-level statistical analysis, typically required in key recovery attacks based on counting techniques). Furthermore, designing efficient sub-cipher attacks on more rounds of a Speck

variant, could also lead to improved attacks. However, such an attack would need to analyze dependencies in the round keys due to the key schedule.

Additional future work items include applying the enumeration framework to improve the best known attacks on more ciphers, and perhaps extending it to other types of attacks, which are different from attacks based on self-similarity and differential attacks.

References

1. Farzaneh Abed, Eik List, Jakob Wenzel, and Stefan Lucks. Differential Cryptanalysis of round-reduced Simon and Speck, 2014. Presented at FSE 2014. To Appear in Lecture Notes in Computer Science.
2. Martin R. Albrecht and Carlos Cid. Algebraic Techniques in Differential Cryptanalysis. In Orr Dunkelman, editor, *FSE*, volume 5665 of *Lecture Notes in Computer Science*, pages 193–208. Springer, 2009.
3. Martin R. Albrecht, Carlos Cid, Thomas Dullien, Jean-Charles Faugère, and Ludovic Perret. Algebraic Precomputations in Differential and Integral Cryptanalysis. In Xuejia Lai, Moti Yung, and Dongdai Lin, editors, *Inscrypt*, volume 6584 of *Lecture Notes in Computer Science*, pages 387–403. Springer, 2010.
4. Javad Alizadeh, Nasour Bagheri, Praveen Gauravaram, Abhishek Kumar, and Somitra Kumar Sanadhya. Linear Cryptanalysis of Round Reduced SIMON. Cryptology ePrint Archive, Report 2013/663, 2013. http://eprint.iacr.org/.
5. Hoda A. Alkhzaimi and Martin M. Lauridsen. Cryptanalysis of the SIMON Family of Block Ciphers. Cryptology ePrint Archive, Report 2013/543, 2013. http://eprint.iacr.org/.
6. Achiya Bar-On, Itai Dinur, Orr Dunkelman, Virginie Lallemand, and Boaz Tsaban. Improved Analysis of Zorro-Like Ciphers. *IACR Cryptology ePrint Archive*, 2014.
7. Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks, and Louis Wingers. The SIMON and SPECK Families of Lightweight Block Ciphers. Cryptology ePrint Archive, Report 2013/404, 2013. http://eprint.iacr.org/.
8. Eli Biham and Adi Shamir. Differential cryptanalysis of des-like cryptosystems. *J. Cryptology*, 4(1):3–72, 1991.
9. Alex Biryukov, Arnab Roy, and Vesselin Velichkov. Differential Analysis of Block Ciphers SIMON and SPECK, 2014. Presented at FSE 2014. To Appear in Lecture Notes in Computer Science.
10. Christophe De Cannière and Christian Rechberger. Finding SHA-1 Characteristics: General Results and Applications. In Xuejia Lai and Kefei Chen, editors, *ASIACRYPT*, volume 4284 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 2006.
11. Itai Dinur, Orr Dunkelman, and Adi Shamir. Improved Attacks on Full GOST. In Anne Canteaut, editor, *FSE*, volume 7549 of *Lecture Notes in Computer Science*, pages 9–28. Springer, 2012.
12. Takanori Isobe. A Single-Key Attack on the Full GOST Block Cipher. *J. Cryptology*, 26(1):172–189, 2013.
13. Orhun Kara. Reflection Cryptanalysis of Some Ciphers. In Dipanwita Roy Chowdhury, Vincent Rijmen, and Abhijit Das, editors, *INDOCRYPT*, volume 5365 of *Lecture Notes in Computer Science*, pages 294–307. Springer, 2008.
14. Gaëtan Leurent. Analysis of Differential Attacks in ARX Constructions. In Wang and Sako [23], pages 226–243.

15. Gaëtan Leurent. Construction of Differential Characteristics in ARX Designs Application to Skein. In Ran Canetti and Juan A. Garay, editors, *CRYPTO (1)*, volume 8042 of *Lecture Notes in Computer Science*, pages 241–258. Springer, 2013.
16. Helger Lipmaa and Shiho Moriai. Efficient Algorithms for Computing Differential Properties of Addition. In Mitsuru Matsui, editor, *FSE*, volume 2355 of *Lecture Notes in Computer Science*, pages 336–350. Springer, 2001.
17. Florian Mendel, Tomislav Nad, and Martin Schläffer. Finding SHA-2 Characteristics: Searching through a Minefield of Contradictions. In Dong Hoon Lee and Xiaoyun Wang, editors, *ASIACRYPT*, volume 7073 of *Lecture Notes in Computer Science*, pages 288–307. Springer, 2011.
18. Souradyuti Paul and Bart Preneel. Solving Systems of Differential Equations of Addition. In Colin Boyd and Juan Manuel González Nieto, editors, *ACISP*, volume 3574 of *Lecture Notes in Computer Science*, pages 75–88. Springer, 2005.
19. W. A. Stein et al. *Sage Mathematics Software*. The Sage Development Team. `http://www.sagemath.org`.
20. Harshal Tupsamudre, Shikha Bisht, and Debdeep Mukhopadhyay. Differential Fault Analysis on the families of SIMON and SPECK ciphers. Cryptology ePrint Archive, Report 2014/267, 2014. http://eprint.iacr.org/.
21. Meiqin Wang, Yue Sun, Nicky Mouha, and Bart Preneel. Algebraic Techniques in Differential Cryptanalysis Revisited. In Udaya Parampalli and Philip Hawkes, editors, *ACISP*, volume 6812 of *Lecture Notes in Computer Science*, pages 120–141. Springer, 2011.
22. Xiaoyun Wang and Kazue Sako, editors. *Advances in Cryptology - ASIACRYPT 2012 - 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, December 2-6, 2012. Proceedings*, volume 7658 of *Lecture Notes in Computer Science*. Springer, 2012.
23. Xiaoyun Wang and Hongbo Yu. How to Break MD5 and Other Hash Functions. In Ronald Cramer, editor, *EUROCRYPT*, volume 3494 of *Lecture Notes in Computer Science*, pages 19–35. Springer, 2005.

A Details of the Basic 2-Round Attack

In this section, we give the details of our basic 2-round attack, which computes x_r and x_{r+1} bit by bit given $(x_{r+2}, y_{r+2}), (x'_{r+2}, y'_{r+2})$, assuming the knowledge of $(\Delta x_r, \Delta y_r)$.

In general, we have (for any round $\hat{r}$)

$$((x_{\hat{r}} \ggg \alpha) \boxplus y_{\hat{r}}) \oplus ((x'_{\hat{r}} \ggg \alpha) \boxplus y'_{\hat{r}}) = \Delta x_{\hat{r}+1},$$

and we denote by $z_{\hat{r}}$ the n-bit carry word generated by the addition operation $(x_{\hat{r}} \ggg \alpha) \boxplus y_{\hat{r}}$, and by $z'_{\hat{r}}$ the carry word generated by $(x'_{\hat{r}} \ggg \alpha) \boxplus y'_{\hat{r}}$.

The basic procedure $1RProcedure(\hat{r}, i)$ analyzes round $\hat{r}$ of Speck at bit index $i \in \{0, 1, 2, \ldots, n-1\}$, and is given below. The procedure assumes that we know the XOR input/output differences of round $\hat{r}$ (we actually need only a few bits of these differences), and requires the additional 1-bit value $y_{\hat{r}}^{[i]}$, and the values of the 2 carry bits $z_{\hat{r}}^{[i]}, z_{\hat{r}}^{'[i]}$. The procedure guesses the value of the bit $x_{\hat{r}}^{[i+\alpha]}$, and computes the next 2 carry bits $z_{\hat{r}}^{[i+1]}, z_{\hat{r}}^{'[i+1]}$.

1. For of the 2 possible values of $x_{\hat{r}}^{[i+\alpha]}$:
   (a) Compute $x_{\hat{r}}^{[i+\alpha]} \boxplus y_{\hat{r}}^{[i]} \boxplus z_{\hat{r}}^{[i]}$ and determine the next carry bit $z_{\hat{r}}^{[i+1]}$.
   (b) Compute $x_{\hat{r}}^{'[i+\alpha]} = x_{\hat{r}}^{[i+\alpha]} \oplus \Delta x_{\hat{r}}^{[i+\alpha]}$.
   (c) Compute $x_{\hat{r}}^{'[i+\alpha]} \boxplus y_{\hat{r}}^{'[i]} \boxplus z_{\hat{r}}^{'[i]}$ and determine the next carry bit $z_{\hat{r}}^{'[i+1]}$.
   (d) Check whether $z_{\hat{r}}^{[i+1]} \oplus z_{\hat{r}}^{'[i+1]} \oplus \Delta x_{\hat{r}}^{[i+1+\alpha]} \oplus \Delta y_{\hat{r}}^{[i+1]} = \Delta x_{\hat{r}+1}^{[i+1]}$, and if equality does not hold, discard the guess. Otherwise, output the current value of $x_{\hat{r}}^{[i+\alpha]}$ and the computed carry bits $z_{\hat{r}}^{[i+1]}, z_{\hat{r}}^{'[i+1]}$.

In order to analyze 2-rounds, we assume the we know the values of the XOR input/output differences of rounds $\hat{r}, \hat{r}-1$, and require the 2 additional 1-bit values $y_{\hat{r}}^{[i]}, y_{\hat{r}}^{[i+\alpha]}$, and the values of the 4 carry bits $z_{\hat{r}}^{[i]}, z_{\hat{r}}^{'[i]}, z_{\hat{r}-1}^{[i+\alpha-\beta]}, z_{\hat{r}-1}^{'[i+\alpha-\beta]}$. The procedure guesses the value of the bits $x_{\hat{r}}^{[i+\alpha]}, x_{\hat{r}-1}^{[i+2\alpha-\beta]}$, and computes the next 4 carry bits $z_{\hat{r}}^{[i+1]}, z_{\hat{r}}^{'[i+1]}, z_{\hat{r}-1}^{[i+1+\alpha-\beta]}, z_{\hat{r}-1}^{'[i+1+\alpha-\beta]}$. The $2RProcedure(\hat{r}, i)$ algorithm is given below.

1. Run $1RProcedure(\hat{r}, i)$ and for each returned solution:
   (a) Compute $y_{\hat{r}-1}^{[i+\alpha-\beta]} = x_{\hat{r}}^{[i+\alpha]} \oplus y_{\hat{r}}^{[i+\alpha]}$.
   (b) Run $1RProcedure(\hat{r}-1, i+\alpha-\beta)$

The guess-and-determine algorithm calls $2RProcedure(\hat{r}, i)$ with $\hat{r} = r+1$ for various indexes i in order to recover the full values of x_{r+1} and x_r. Note that all the input/output XOR differences to the last 2 rounds of Speck are known, and the full $y_{\hat{r}} = y_{r+1}$ is known as well, and thus we are only missing the value of the carry bits.

As the carries computed by index j are input to procedure $j+1$, we perform calls to $2RProcedure(r+1, i)$ with sequential labels $i, i+1, \ldots, n-1, 0, \ldots, i-1$ (from LSB to MSB) in order to recover x_{r+1} and x_r. As a result, we only have to guess the carries required by the initial procedure. Since there is no carry into the LSB (i.e., $z_{r+1}^{[0]} = z_{r+1}^{'[0]} = 0$), then we start with procedure $i = 0$ to minimize the number of carry guesses. Furthermore, the value of $z_r \oplus z'_r$ can be computed from $(\Delta x_r \ggg a) \oplus \Delta y_r \oplus \Delta x_{r+1}$, and thus we actually need to guess only one carry bit before executing the first procedure. Finally, after the execution of the last procedure (with index $n-1$), we can derive the actual value of this guessed carry bit and obtain an additional filtering condition.

As the time complexity of the guess-and-determine algorithm is proportional to the number of guesses it makes, we need to carefully analyze the ratio between the number of guessed bits, and the number of filtering conditions used to filter the guesses. The algorithm $2RProcedure(\hat{r}, i)$ guesses the values of the two bits $x_{\hat{r}}^{[i+\alpha]}, x_{\hat{r}-1}^{[i+2\alpha-\beta]}$ and uses two filtering conditions (one in each call to $1RProcedure(\hat{r}, i)$ at Step 1.(d)). Thus, assuming that the analyzed instance behaves randomly, we expect the number of guesses at each stage of the execution of the algorithm to remain constant. Such randomness assumptions lead to the conclusion that the average execution time of the algorithm is comparable to the execution time of a full Speck encryption (perhaps even smaller, as we analyze only 2 rounds). However, as noted in Section 7.1, these randomness assumptions do not hold in our case.

B Details of the Implementation of Multi-Bit Filters

In this section, we describe how to construct the b-bit filters for the differential equations obtained by the system $(x \boxplus a) \oplus ((x \oplus \delta_1) \boxplus a') = \delta_2$, where x is a variable and $a, a', \delta_1, \delta_2$ are known constants (parameters). This is a general description of the equation system obtained for round $r+1$ of Speck (where $x = (x_{r+1} \ggg \alpha), a = y_{r+1}, a' = y'_{r+1}, \delta_1 = (\Delta x_{r+1} \ggg \alpha), \delta_2 = \Delta x_{r+2})$.

We denote by z the carry generated by $x \boxplus a$, and by z' the carry generated by $(x \oplus \delta_1) \boxplus a'$. We examine the system at indexes i to $i+b-1$, with the additional carry into bit $i+b$. This system has $b+2$ variables: $z^{[i]}, z'^{[i]}, x^{[i,\ldots,i+b-1]}$ (for $i = 0$ there are only b variables, as the carry bits are 0) and b constraints which depend on the $4b+1$ parameters $a^{[i,\ldots,i+b-1]}, a'^{[i,\ldots,i+b-1]}, \delta_1^{[i,\ldots,i+b-1]}, \delta_2^{[i,\ldots,i+b-1]}, (\delta_1 \oplus \delta_2 \oplus a \oplus a')^{[i+b]}$. However, if we assume that we already executed the 1-bit filter above on the system, then we know that the XOR of the carries and differences at bit i is 0. This allows us to remove one parameter, and remain with $4b$ parameters.[9] The general algorithm for calculating the b-bit filter is given below:

1. Allocate a bit array A of size 2^{4b}.
2. For each value of the $4b$ parameters $j = 0, 1, \ldots, 2^{4b} - 1$:
   (a) For each value of the $b+2$ variables:
      i. Check whether all the b constraints are satisfied, and if they are, set $A[j] = 1$ and increment j by going to Step 2.
   (b) Since there is not solution with the current parameters, set $A[j] = 0$.

Calculating the b-bit filter requires 2^{4b} bits of memory and its time complexity is 2^{5b+2} in the worst case.

For the specific case of Speck, we have an additional dependency between the parameters, as $\Delta x_{r+1} = y_{r+1} \oplus y'_{r+1} \oplus (\Delta y_r \lll \beta)$, and Δy_r is a fixed value. Thus, we can reduce the number parameters to $3d$, but as a result, the filter depends on Δy_r, and we need to calculate it separately for each bit index $i \in \{0, 1, \ldots, n-b\}$ for which we want to apply the filter.

In our attacks, we use $b = 6$, and thus one 6-bit filter requires 2^{18} bits, or 2^{15} bytes. If we want to apply a filter for each index i for Speck with the largest word size of $n = 64$, then the memory complexity of the filters is less than $64 \cdot 2^{15} = 2^{21}$ bytes, and thus the filters require no more than a few megabytes of storage. In order to get an estimation of how the filters perform, we implemented their general version (with $4b$ parameters), and calculated that for a random value of the $4b$ parameters, the probability to pass both a 1-bit filter and a 6-bit filter starting from index $i > 0$ is about 0.285 (whereas the probability that a 6-bit word passes just a 1-bit filter is $(7/8)^6 \approx 0.45$). However, for larger n-bit words, we will apply the filters to consecutive indexes, resulting in strong

[9] We can additionally reduce the number of parameters by assuming that the 1-bit filter passed on the b-bit words, but this is more complicated and we do not elaborate on this possible optimization in this paper.

correlations between their applications, and making it more difficult to estimate the probability of passing the filters.

We note that it is possible to construct b-bit filters that analyze blocks of the full equation system for 2-round Speck, rather than just for one round. However, these filters require a larger amount of memory (of about 2^{6b}) and can only be efficiently used for small values of b. Of course, one can try to combine several filters of different types and sizes, and it is interesting to study how to construct efficient algorithms to solve a given system of equations using various combinations of filters. However, in this paper, we are mainly interested in the equation systems generated by 2-round Speck. For this specific case, we describe in Appendix C an algorithm which gives very good results in practice.

C Details of the Optimized 2-Round Attack

The details of the optimized 2-round attack are give below. Recall that the algorithm is given the value of $(x_{r+2}, y_{r+2}), (x'_{r+2}, y'_{r+2})$, while $(\Delta x_r, \Delta y_r)$ is a fixed by a differential characteristic. The algorithm recovers all possible values of x_r, x_{r+1}, assuming that we calculated during preprocessing, 6-bit filters for rounds r and $r+1$ of Speck, for all indexes $i \in \{0, 1, \ldots, n-6\}$. Additionally, the attack uses the $1RProcedure(\hat{r}, i)$ algorithm of Appendix A.

1. Apply the 1-round filter of Section 7.2 to rounds $r+1$ and r, and if a filter fails, return $NULL$.
2. For $i \in \{0, 1, \ldots, n-6\}$, apply the corresponding 6-bit filter at index i to round $r+1$, and if a filter fails, return $NULL$.
3. For $i \in \{0, 1, \ldots, n-1\}$:
   (a) Run $1RProcedure(r+1, i)$, iteratively recovering x_{r+1}. For each partial solution, compute $y_r^{[i+\alpha-\beta]}$ (as in Step 1.(a) of $2RProcedure(r+1, i)$).
   (b) If $i \geq 6$, apply the 6-bit filter to round r at index $i-6+\alpha-\beta$, and if the filter fails, discard the partial solution.
4. For each solution for x_{r+1}:
   (a) For $i \in \{0, 1, \ldots, n-1\}$:
      i. Run $1RProcedure(r, i)$, iteratively recovering x_r.
   (b) Output all the solutions for x_r, and the current value of x_{r+1}.