Colliding Keys for *SC2000-256*

Alex Biryukov[1] and Ivica Nikolić[2](✉)

[1] University of Luxembourg, Walferdange, Luxembourg
alex.biryukov@uni.lu
[2] Nanyang Technological University, Singapore, Singapore
inikolic@ntu.edu.sg

Abstract. In this work we present analysis for the block cipher *SC2000*, which is in the Japanese CRYPTREC portfolio for standardization. In spite of its very complex and non-linear key-schedule we have found a property of the full *SC2000-256* (with 256-bit keys) which allows the attacker to find many pairs of keys which generate identical sets of subkeys. Such colliding keys result in identical encryptions. We designed an algorithm that efficiently produces colliding key pairs in 2^{39} time, which takes a few hours on a PC. We show that there are around 2^{68} colliding pairs, and the whole set can be enumerated in 2^{58} time. This result shows that *SC2000-256* cannot model an ideal cipher. Furthermore we explain how practical collisions can be produced for both Davies-Meyer and Hiroses hash function constructions instantiated with *SC2000-256*.

Keywords: SC2000 · Block cipher · Key collisions · Equivalent keys · CRYPTREC · Hash function

1 Introduction

The block cipher *SC2000* [15] was designed by researchers from Fujitsu and the Science University of Tokyo, and submitted to the open call for 128-bit encryption standards organized by Cryptography Research and Evaluation Committees (CRYPTREC). Started in 2000, CRYPTREC is a program of the Japanese government set up to evaluate and recommend cryptographic algorithms for use in industry and institutions across Japan. An algorithm becomes a CRYPTREC recommended standard after two stages of evaluations. Unlike AES, eSTREAM and SHA-3 competitions, the evaluation stages of CRYPTREC do not have strictly defined time limits, but an algorithm progresses to the next stage (or becomes a standard), when its security level has been confirmed by a substantial amount of cryptanalysis. CRYPTREC takes into account all published cryptanalysis in academia and, as well, hires experts to evaluate the security of the algorithm. *SC2000* has passed the first two stages, and for a decade it was among the recommended standards.

Cryptanalysis of the full 6.5–7.5 round (depending on the key size) *SC2000* is still unknown, however, single-key attacks on round-reduced *SC2000* were presented in several papers: boomerang and rectangle attacks on 3.5 rounds

A. Joux and A. Youssef (Eds.): SAC 2014, LNCS 8781, pp. 77–91, 2014.
DOI: 10.1007/978-3-319-13051-4_5

by Dunkelman and Keller [7] and Biham et al. [2], high probability 3.5-round differential characteristics were used in 4.5-round attack by Raddum and Knudsen [13], iterative differential and linear characteristics resulting in attacks on 4.5 rounds by Yanami et al. [17], and a differential attack on 5 rounds by Lu [10].

In spite of considerable evaluation effort by world leading analysts, the cryptanalytic progress on the cipher was slow. A possible reason is given in one of the evaluation reports [16] – the authors state that "... the design is complicated and uses components which do not facilitate for easy analysis". Indeed, *SC2000* uses surprisingly large number of different operations: modular additions, subtractions and multiplications, bitwise additions (XOR), two S-boxes of different size (5 bits and 6 bits), diffusion layers based on multiplications by binary matrices, and rotations. Compared to the widely used design methods such as substitution-permutations (SP) networks (only S-boxes and diffusion layers), or ARX (additions, rotations and XOR), *SC2000* seems too complex, which in turn makes the analysis hard to perform. Moreover, in *SC2000* there are more operations in the key schedule than in the state – this may explain the absence of the key schedule attacks. This paper is the first analysis on the key schedule – we find a weakness in the complex key schedule that we exploit to find colliding keys, i.e. two different master keys that result in the same subkeys. Our result works on the full cipher and independently of the number of its rounds.

In [11] Matsui investigates the behavior of colliding key pairs for the stream cipher RC4. He shows that even in the case of a key size as small as 24 bytes, there are related keys that create the same initial state, hence they generate the same pseudo-random byte stream. In other words, the streams collide. Matsui's discovery is rather interesting and unexpected as the number of possible distinct initial states in RC4 is $256! \approx 2^{1684}$ while the number of states generated from 24-byte key is only 2^{192}. No key collisions should occur in any cipher (the key schedule should be injective), in particular in ciphers that have strictly expandable key schedule, where the accumulative size of the subkeys is larger than the size of the master key. The ratio of the expanded key size/master key size usually depends on the number of rounds and on the length of subkey input in each round. For example, in AES-256 this ratio is 7.5 as there are 15 128-bit subkeys produced from the 256-bit master key. Colliding keys are often called equivalent keys and the existence of such keys is known for a few ciphers. For instance, Robshaw [14] has shown that another CRYPTREC candidate, the block cipher CIPHERUNICORN-A, has equivalent keys. Kelsey et al. [9] found trivial equivalent keys for the Tiny Encryption Algorithm (TEA) block cipher. Furthermore, Aumasson et al. [1] have discovered that the ISDB Scrambling Algorithm, the cipher MULTI2, allows such keys as well.

For *SC2000-256*, despite the fact that the total size of the subkeys is 8 times larger than the size of the master key, we show that this cipher does not have an injective key schedule. There exists a set of 2^{68} pairs of colliding master keys – each pair is composed of two different master keys that after the key schedule lead to the same set of subkeys. Therefore encryptions of *any* plaintext under the first and under the second key produce the same ciphertext, hence the two

master keys are equivalent. We achieve the collisions in the subkeys by exploiting weaknesses in the two-stage key schedule: in the first stage we efficiently find a key pair that results in two intermediate keys with a special relation, which in turn is a sufficient condition for the second stage to produce the same subkeys.

Our algorithm for finding a colliding key pair requires only 2^{39} operations, and we have tested our analysis in practice by implementing a search on a regular PC. The produced collisions (see Table 1) confirm the correctness of the analysis and the complexity of the algorithm. We show how an attacker can use the colliding key pairs in order to construct practical collisions in hash functions instantiated with *SC2000-256*. In both single-block-length Davies-Meyer hash, and in the double-block-length Hirose's hash [8] the level of collisions resistance drops from 64,128 to only 39 bits, if instantiated with *SC2000-256*. This suggests that *SC2000-256*, although possibly secure for encryption, has a serious key-schedule weakness and cannot model an ideal cipher.

2 Description of *SC2000-256*

SC2000 is 128-bit block cipher that supports 128, 192, and 256-bit keys. In this work we focus on 256-bit key cipher, further denoted as *SC2000-256*. This cipher has 7.5 rounds, but our analysis is independent of the number of rounds and the round function, as it is valid for any number and for any function. Therefore, in the sequel we describe only the key schedule.

Most of the operations in the key schedule are word-oriented. The only exception is S_{func}, which is a bijective non-linear operation that applies in parallel 5-bit and 6-bit S-boxes (see Fig. 1). The 32-bit input word is split into six chunks of sizes 6,5,5,5,5, and 6 bits, respectively, then 6-bit or 5-bit S-boxes (depending on the size of the chunk) are applied to the chunks, and finally the outputs of the S-boxes are concatenated to produce the final output of S_{func}.The remaining operations in the key schedule are all word-oriented, and include:

1. M_{func} : bijective linear transformation which is a multiplication by a 32×32 matrix. The input is seen as a vector of 32 elements, and it is multiplied by a binary matrix.
2. $+, \boxplus$: addition mod 2^{32}.
3. $-, \boxminus$: subtraction mod 2^{32}.
4. $\times, \boxtimes$: multiplication mod 2^{32}.

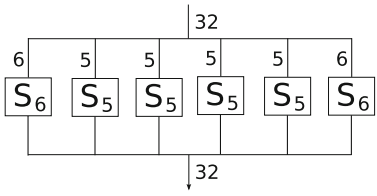


Fig. 1. The operation S_{func} used in the key schedule of *SC2000-256*.

5. $\oplus$: XOR (bitwise addition).
6. $\lll 1$: rotation by 1 bit to the left of 32-bit words.

The key schedule needs two steps (or phases) to produce the subkey words (used in the round functions) from the master key. At the beginning, it starts by dividing the 256-bit master key into eight 32-bit words $uk_j, j = 0, 1, \ldots, 7$, called master key words.

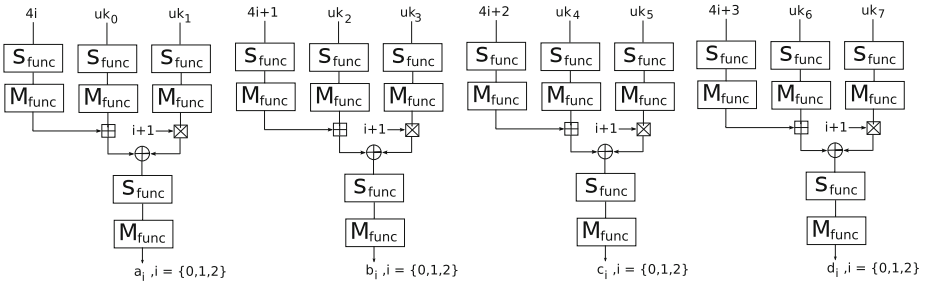
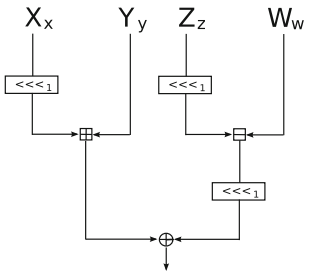


Fig. 2. The intermediate key generation used in *SC2000-256*.

The first phase, called intermediate key generation, takes the 8 words uk_j and outputs 12 intermediate key words $a_i, b_i, c_i, d_i, i = 0, 1, 2$ (see Fig. 2). It applies four similar transformations, called branches, to the four pairs of master key words: the first branch operates on uk_0, uk_1 and produces a_0, a_1, a_2, the second branch on uk_2, uk_3 and outputs b_0, b_1, b_2, etc. Hence each triplet of intermediate key words depends only on two master key words. In a pseudo code, this phase can be described as:

for $i = 0 \rightarrow 2$ **do**
$a_i \leftarrow M_f(S_f((M_f(S_f(uk_0)) + M_f(S_f(4i))) \oplus M_f(S_f(uk_1))))$
$b_i \leftarrow M_f(S_f((M_f(S_f(uk_2)) + M_f(S_f(4i+1))) \oplus M_f(S_f(uk_3))))$
$c_i \leftarrow M_f(S_f((M_f(S_f(uk_4)) + M_f(S_f(4i+2))) \oplus M_f(S_f(uk_5))))$
$d_i \leftarrow M_f(S_f((M_f(S_f(uk_6)) + M_f(S_f(4i+3))) \oplus M_f(S_f(uk_7))))$
end for

The second phase, called extended key generation, takes the 12 intermediate key words and produces 64 subkey words $ek_i, i = 0, 1, \ldots, 63$, called extended key words. Each subkey word is obtained with a non-symmetric transformation (see Fig. 3) of four intermediate key words that come from different branches, hence every subkey word depends on all master key words. For each subkey word, to determine which four intermediate key words should be taken, and in what order, this phase requires two lookup tables. The first table *Order* specifies the order (recall that the transformation is non-symmetric, so the order matters) in which the words are put into the transformation. The second table *Index* determines which word within a branch should be taken. As a result, no two subkey words depend on the same intermediate key words put in the same order. Refer to Fig. 3



Order

t	X Y Z W
0	a b c d
1	b a d c
2	c d a b
3	d c b a
4	a c d b
5	b d c a
6	c a b d
7	d b a c
8	a d b c
9	b c a d
10	c b d a
11	d a c b

Index

s	x y z w
0	0 0 0 0
1	1 1 1 1
2	2 2 2 2
3	0 1 0 1
4	1 2 1 2
5	2 0 2 0
6	0 2 0 2
7	1 0 1 0
8	2 1 2 1

Fig. 3. The non-symmetric transformation used in the extended key generation (left), and the values of the lookup tables (right).

for a pictorial view of the non-symmetric transformation and for the values of the lookup tables. In a pseudo code, the second phase can be described as:

for $i = 0 \rightarrow 63$ **do**
  $s \leftarrow i \ (mod\ 9)$
  $t \leftarrow (i + \lfloor \frac{i}{36} \rfloor\ ) \ (mod\ 12)$
  $X \leftarrow Order[t][0]$
  $x \leftarrow Index[s][0]$
  $Y \leftarrow Order[t][1]$
  $y \leftarrow Index[s][1]$
  $Z \leftarrow Order[t][2]$
  $z \leftarrow Index[s][2]$
  $W \leftarrow Order[t][3]$
  $w \leftarrow Index[s][3]$
  $ek_i \leftarrow (X_h \lll 1 + Y_y) \oplus ((Z_z \lll 1 - W_w) \lll 1)$
end for

As the description of the key schedule suffices to understand our attack, for a full specification of the cipher we refer the interested reader to [15].

3 Key Collisions for *SC2000-256*

For *SC2000-256*, we show how to find two distinct master keys that produce the same subkey words and hence we obtain key collisions. The core idea of our analysis is a weakness in the second phase (extended key generation) – it can cancel a particular input difference (i.e. a particular difference in each pair of intermediate key words), resulting in subkey collisions. If we are able to deterministically find two master keys that after the first phase produce the particular difference, then the second phase will cancel the difference, and we will end up with collisions. Therefore, to present the analysis we focus on:

1. **(Second phase)** Specify the difference in intermediate key words, and prove that it leads to collisions after the second phase.
2. **(First phase)** Give an algorithm that finds two master keys that lead to the difference in the intermediate key words after the first phase.

This seemingly upside-down approach (at the beginning we analyze the second phase, and then the first), is taken to understand why the algorithm at step 2 has to target the specific difference (and not some other).

Notations. With superscripts 1 and 2 we denote various master, intermediate and extended keys for the first and respectively the second master key, e.g. a_0^2 is the first intermediate key produced from the second master key, ek_{10}^1 is the eleventh subkey word produced from the first master key. The subscript h stands for hexadecimal number, for instance $\texttt{80000000}_h$ is 2^{31}. With $\overline{X}$ we denote the bitwise negation of the word X, i.e. $\overline{X} = X \oplus (-1)$, while $\wedge$ stands for bitwise AND.

3.1 Specifying the Difference for the Second Phase

Let us start our analysis by focusing on the second phase. The following Lemma defines the required difference (and the additional conditions) in the intermediate key words, that plays the main role in the analysis:

Lemma 1. *Let each pair* (X_1, X_2) *in the set of pairs of intermediate key words* (a_i^1, a_i^2), (b_i^1, b_i^2), (c_i^1, c_i^2), (d_i^1, d_i^2), $i = 0, 1, 2$ *satisfy the following two conditions:*

$$\textbf{Condition 1} \quad X_2 = \overline{X_1} + 3,$$
$$\textbf{Condition 2} \quad X_1 \wedge \texttt{8000000f}_h = \texttt{80000003}_h.$$

Then the extended key generation will produce the same extended keys (subkey words), i.e. $ek_i^1 = ek_i^2, i = 0, \ldots, 63$.

The Lemma claims that if: (Condition 1) the pairs of intermediate words produced from the first and the second master key have the special relation, and (Condition 2) the intermediate key words produced from the first master key have particular values in five bits (the most significant, and the four least significant), then after the second phase they will lead to the same subkeys. The Condition 2 becomes clear in the proof, and when the five bits have this specific value, the probability that the subkey words collide is 1, otherwise it is less than 1.

Proof. To prove the Lemma we focus on the extended key generation function

$$f(X, Y, Z, W) = (X \lll 1 + Y) \oplus [(Z \lll 1 - W) \lll 1].$$

We claim that if X, Y, Z, W are randomly chosen words with the MSB fixed to 1 and the four LSBs fixed to 3, then

$$f(X, Y, Z, W) = f(\overline{X} + 3, \overline{Y} + 3, \overline{Z} + 3, \overline{W} + 3).$$

Let us rewrite f as an XOR of two functions g, h, i.e.

$$\begin{aligned} f(X,Y,Z,W) &= g(X,Y) \oplus h(Z,W), \\ g(X,Y) &= X \lll 1 + Y, \\ h(Z,W) &= (Z \lll 1 - W) \lll 1. \end{aligned}$$

We will prove that

$$g(X,Y) \oplus g(\overline{X}+3, \overline{Y}+3) = \texttt{ffffffff}_h, \tag{1}$$

$$h(Z,Y) \oplus h(\overline{Z}+3, \overline{W}+3) = \texttt{ffffffff}_h, \tag{2}$$

and thus

$$\begin{aligned} f(X,Y,Z,W) \oplus f(\overline{X}+3, \overline{Y}+3, \overline{Z}+3, \overline{W}+3) = \\ g(X,Y) \oplus h(Z,W) \oplus g(\overline{X}+3, \overline{Y}+3) \oplus h(\overline{Z}+3, \overline{W}+3) = 0. \end{aligned}$$

We need to following supplementary facts:

Fact 1 *Let X, Y be 32-bit words. If $X \wedge \texttt{7fffffff}_h + Y \wedge \texttt{7fffffff}_h < 2^{31}$ then*

$$(X+Y) \lll 1 = X \lll 1 + Y \lll 1.$$

Proof. The fact can be seen as corollary of Theorem 4.11 from [5]. □

Fact 2 *For any values X, Y*

$$\overline{X+Y} = \overline{X} + \overline{Y} + 1.$$

Proof. Note that for any value V, $V \oplus \overline{V} = V + \overline{V} = -1$, and thus $\overline{V} = -1 - V$. Therefore:

$$\overline{X+Y} = -1 - (X+Y) = (-1-X) + (-1-Y) + 1 = \overline{X} + \overline{Y} + 1$$

□

Fact 3 *If $U \wedge m = 0$ then $\overline{U} \oplus (U+m) = \texttt{ffffffff}_h \oplus m$.*

Proof. When $U \wedge m = 0$, then $U + m = U \oplus m$. Therefore

$$\overline{U} \oplus (U+m) = \overline{U} \oplus U \oplus m = \texttt{ffffffff}_h \oplus m.$$

□

Now we are ready to present the proof of the Lemma. We will prove only the part for g – the part for h is similar and instead of modular addition we have to work with modular subtraction. Let us focus on (1). We get:

$$\begin{aligned} g(X,Y) \oplus g(\overline{X}+3, \overline{Y}+3) = & \quad (3) \\ = (X \lll 1 + Y) \oplus ((\overline{X}+3) \lll 1 + \overline{Y} + 3) = & \quad (4) \\ = (X \lll 1 + Y) \oplus ((\overline{X}) \lll 1 + 3 \lll 1 + \overline{Y} + 3) = & \quad (5) \\ = (X \lll 1 + Y) \oplus ((\overline{X \lll 1} + \overline{Y}) + 9) = & \quad (6) \\ = (X \lll 1 + Y) \oplus ((\overline{X \lll 1 + Y}) + 8) = & \quad (7) \\ = \overline{U} \oplus (U+8), & \quad (8) \end{aligned}$$

where $U = \overline{X \lll 1 + Y}$. The transition (4) to (5) is due to Fact 1 – the two least significant bits of $\overline{X}$ are 00 thus $\overline{X} \wedge \texttt{7fffffff}_h + 3 < 2^{31}$. Note, this is where we actually use the requirement of Condition 2: the two least significant bits of X must be '11'. The transition (6) to (7) is due to Fact 2. Finally, the four least significant bits of U are 0101 (again use of Condition 2!) and thus by Fact 3, $g(X, Y) \oplus g(\overline{X} + 3, \overline{Y} + 3) = \texttt{ffffffff}_h$. This concludes the proof. □

We have discovered the strange conditions of the Lemma (and then provided a formal proof), when we analyzed the behavior of the non-symmetric function f – it became clear that f cancels some modular differences. The similarity of the left and the right side (the function g and the function h without the final rotation) of f, and the fact that the rotations are only on 1 bit, suggested that there may exist a universal difference for the intermediate words, such that cancellation in f would occur when all four words have this difference. We started experimenting with various differences between X and $\overline{X}$, and various values for the two most significant bit (as in h we have twice rotation on 1 to the left, we took 2 bits), and several least significant bits. The experiments were implemented as an exhaustive computer search that tries all such differences and bit values, and for each combination checks the probability that f cancels the difference. The results of our experiment led to the actual Conditions 1,2.

3.2 Finding Pairs in the First Phase

Let us focus on the first phase and produce a pair of master key words that after this phase result in pairs of intermediate key words that comply with Conditions 1 and 2 of the Lemma. For the sake of simplicity, first we take into account only Condition 1, and later we consider Condition 2.

Let $u_i^1, i = 0, \ldots, 7$ be the words of the first master key K_1, and $u_i^2, i = 0, \ldots, 7$ be the words of the second master key K_2. Let U_i be the corresponding words of the master keys after the application of S_{func} and M_{func}, i.e. $U_i^j = M_{func}(S_{func}(u_i^j))), i = 0, \ldots, 7, j = 1, 2$. Also, let I_i be the constants $I_{i+1} = M_{func}(S_{func}(4 \cdot i)), i = 0, 1, 2$. Then, taking into account the intermediate key generation procedure, Condition 1 for the pairs $(a_i^1, a_i^2), i = 0, 1, 2$ is equivalent to solving the following system of equations (refer to Fig. 4):

$$(U_0^1 + I_1) \oplus U_1^1 = A^1 \tag{9}$$
$$(U_0^1 + I_2) \oplus 2 \cdot U_1^1 = B^1 \tag{10}$$
$$(U_0^1 + I_3) \oplus 3 \cdot U_1^1 = C^1 \tag{11}$$
$$(U_0^2 + I_1) \oplus U_1^2 = A^2 \tag{12}$$
$$(U_0^2 + I_2) \oplus 2 \cdot U_1^2 = B^2 \tag{13}$$
$$(U_0^2 + I_3) \oplus 3 \cdot U_1^2 = C^2 \tag{14}$$
$$A^2 = S_{func}^{-1}(M_{func}^{-1}(\overline{M_{func}(S_{func}(A^1))} + 3)) \tag{15}$$
$$B^2 = S_{func}^{-1}(M_{func}^{-1}(\overline{M_{func}(S_{func}(B^1))} + 3)) \tag{16}$$
$$C^2 = S_{func}^{-1}(M_{func}^{-1}(\overline{M_{func}(S_{func}(C^1))} + 3)) \tag{17}$$

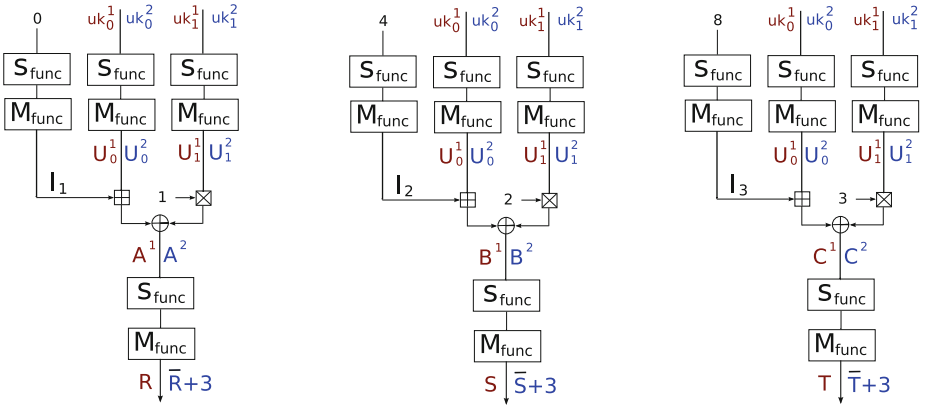


Fig. 4. The intermediate values used to describe the algorithm in the first branch (with outputs a_0, a_1, a_2). The values produced from the first master key are on the left and have a dark red color, while from the second are on the right and have blue color. R, S, T are 32-bit words that do not have specified values (Color figure online).

Let $G(x) = S^{-1}_{func}(M^{-1}_{func}(\overline{M_{func}(S_{func}(x))} + 3))$. Then the system can be rewritten as:

$$(U^1_0 + I_1) \oplus U^1_1 = G((U^2_0 + I_1) \oplus U^2_1)$$
$$(U^1_0 + I_2) \oplus 2 \cdot U^1_1 = G((U^2_0 + I_2) \oplus 2 \cdot U^2_1)$$
$$(U^1_0 + I_3) \oplus 3 \cdot U^1_1 = G((U^2_0 + I_3) \oplus 3 \cdot U^2_1)$$

The system has three equations and four unknowns $(U^1_0, U^1_1, U^2_0, U^2_1)$ – theoretically, for any values of I_1, I_2, I_3 and a bijective function G[1], it has 2^{32} solutions. To produce one solution, we find a good pair of master keys for the first two intermediate words (the first two equations of the system), and then we check if the pair is good as well for the third. The algorithm is as follows:

1. Fix random A_1, B_1.
2. Find U^1_0, U^1_1 that satisfy Eqs. (9) and (10).
3. Compute C^1 from U^1_0, U^1_1 with (11).
4. Produce A_2, B_2 from A_1, B_1 with the function G by (15),(16).
5. Find U^2_0, U^2_1 that satisfy Eqs. (12) and (13).
6. Compute C^2 from U^2_0, U^2_1 with (14).
7. Compute $\tilde{C}^2$ from C^1 with the function G by (17).
8. If C^2 is not equal to $\tilde{C}^2$ go to Step 1.
9. The quartet $(U^1_0, U^1_1, U^2_0, U^2_1)$ is the solution for the system.

[1] This is not always the case as the authors have tried to launch a much simpler attack with $A^1 = A^2, B^1 = B^2, C^1 = C^2$ and failed due to the fact that no solutions exist for such system.

The values of C^2 and $\tilde{C}^2$ coincide with probability 2^{-32}, hence to find a solution, we need to repeat around 2^{32} times the Steps 1–7. The complexity of each step is constant (just an application of a formula), with the exception of Steps 2 and 5 – here, we need to find the unknown (U_0, U_1) given the two equations:

$$(U_0 + I_1) \oplus U_1 = A$$
$$(U_0 + I_2) \oplus 2 \cdot U_1 = B$$

After basic algebraic transformations, they are reduced to the form:

$$U_1 = ((2 \cdot U_1 \oplus B) + (I_1 - I_2)) \oplus A \tag{18}$$
$$U_0 = (A \oplus U_1) - I_1 \tag{19}$$

Thus we want to efficiently solve Eq. (18)[2]. The complexity of finding the value of U_1 is given by the following Lemma. We note that the algorithm relies on solving a word equation and to a certain extend is similar to the algorithms from [3].

Lemma 2. *There is an algorithm that, with complexity linear in the size of the words, finds the unique solution of the equation:*

$$X = ((2 \cdot X \oplus B) + C) \oplus A, \tag{20}$$

where A, B, C are some word constants.

Proof. Let us use subscripts to denote the bits of a word, e.g. X_5 is the sixth least significant bit of the word X. The multiplication $2 \cdot X$ is a shift to the left of X by one position, and therefore the $(s+1)$-th bit of $2 \cdot X$ is indeed the s-th bit of X, or in our notation $(2 \cdot X)_s = X_{s-1}$. We solve the above Eq. (20) bit by bit, starting from the least significant bit, and moving towards the most significant bit. In other words, we use a recursive algorithm: first show how to find the least significant (i.e. 0-th) bit, and assuming that we have found the t-th bit, describe how we can find the $(t+1)$-th bit. The equation involves modular addition, therefore for each bit we should keep track of the carry – with cr_i we denote the carry bit of $(2 \cdot X \oplus B) + C$ at i-th bit position.

- *Bit 0.* For the least significant bit, Eq. (20) takes the form:

  $$X_0 = B_0 \oplus C_0 \oplus A_0,$$

  hence the least significant bit of X_0 can be uniquely determined with a simple XOR of three bits, while $cr_0 = B_0 \cdot C_0$.
- *Bit $t+1$.* We assume we have the previous carry cr_t, and we have found the value for X_t. Then for the bit $t+1$, we have:

  $$X_{t+1} = X_t \oplus B_{t+1} \oplus C_{t+1} \oplus cr_t \oplus A_{t+1}$$

  and for the carry we get $cr_{t+1} = m(X_t \oplus B_{t+1}, C_{t+1}, cr_t)$, where $m(x, y, z) = xy \oplus xz \oplus yz$. Again, X_{t+1}, cr_{t+1} are determined uniquely with a constant number of operations.

[2] Once we have the value of U_1, we can easily find U_0 by (19).

As each step of the algorithm requires constant number of operations, and there are n steps in total (n is the word size), we can claim that the complexity of finding the unique solution is linear in the size of the words. □

The Lemma gives us the complexity for the Steps 2 and 5 of the algorithm, i.e. we can solve the system for any A, B with a constant complexity (since $n = 32$). As a result, the total complexity of the algorithm is 2^{32}.

Now we are ready to consider Condition 2. To satisfy this condition (as well as Condition 1), we have to slightly tweak our algorithm and make sure that we have the precise value in the 5 bits of each intermediate key word produced from the first master key. Further we present the full algorithm for computing the pair of master key words that produces the required intermediate key words in the first branch:

1. Fix random R, S such that the most significant bits of R, S are 1, and the values of the four least significant bits equal 3. Compute $A_1 = S^{-1}_{func}(M^{-1}_{func}(R))$, $B_1 = S^{-1}_{func}(M^{-1}_{func}(S))$.
2. Find U^1_0, U^1_1 that satisfy Eqs. (9) and (10).
3. Compute C^1 from U^1_0, U^1_1 with (11).
4. Produce A_2, B_2 from A_1, B_1 with the function G by (15),(16).
5. Find U^2_0, U^2_1 that satisfy Eqs. (12) and (13).
6. Compute C^2 from U^2_0, U^2_1 with (14).
7. Compute $\tilde{C^2}$ from C^1 with the function G by (17).
8. If C^2 is not equal to $\tilde{C^2}$, or in the word T such that $T = M_{func}(S_{func}(C^2))$, the most significant bit of T is not 1, or the value of the four least significant bits is not 3, go to Step 1.
9. The quartet $(U^1_0, U^1_1, U^2_0, U^2_1)$ is the solution for the system.

The new method of defining the values of A_1, B_1 introduced at Step 1, does not change the complexity of the algorithm (compared to the previous). On the other hand, the additional filter at Step 8 (condition on 5 bits) increases the frequency of repeating Steps 1–7 by a factor of 2^5. Hence, the total complexity of the algorithm is $2^{32} \cdot 2^5 = 2^{37}$.

The above complexity is required to find the values of $U^1_0, U^1_1, U^2_0, U^2_1$, which are the 4 words produced from a pair of two master key words in the first branch only. To find the precise value of the pair of two master key words, i.e. to find $(u^1_0, u^1_1), (u^1_1, u^2_1)$, we just need to invert S_{func}, M_{func} (see Fig. 4), hence $u^j_i = S^{-1}_{func}(M^{-1}_{func}(U^j_i)), i = 0, 1, j = 1, 2$. Computing the values of the master key words for all four branches can be done similarly, and with complexity $4 \cdot 2^{37} = 2^{39}$. Therefore, in 2^{39} we can find a pair of master key words, that after the first phase result in a pair of intermediate key words that comply with the conditions of Lemma 1, and thus after the second phase lead to colliding subkey words. Therefore, encryption of any plaintext under the first, and under the second key, produces the same ciphertext.

4 Results and Applications

We have implemented the above search for colliding keys on a PC and, in a matter of hours, we were able to find a pair of master keys (K_1, K_2) that produces the same subkeys. The words of the master keys are given in Table 1. These practical results confirm our analysis and the complexity of the algorithm.

Table 1. Example of colliding pair of master keys for *SC2000-256*

K_1	0x59d0d459	0x4473d8dd	0xcc7d3064	0xd3bbda93
	0x8ff60b58	0xe9dc073d	0x8776c115	0x743c9cfe
K_2	0x10672240	0xb94214ff	0x2bc72c50	0x539cdd3e
	0xf9e9f251	0x921811fa	0x35bf5b7f	0x82ab8bdd
ek^1, ek^2	0xff582ab3	0x4d261f23	0xcb9f9ad3	0x7c81f9c2
	0x0997d523	0xc42fc563	0x2172df72	0x95d8dcb3
	0x18121223	0x9d034e02	0x1baa1423	0xe9190113
	0x4d148522	0xd9247b13	0xb49e6723	0xa393b3e3
	0x3953dbc3	0xb2f85ee2	0x0c17c0a2	0x29d7a162
	0x45ba8593	0x14eb6423	0xe4780213	0xdf8f8b23
	0xd7b48013	0xb5a368a3	0xc47fffc3	0xdee3ff23
	0x4f279343	0xb4a34873	0xe2881a63	0x0c1b8372
	0xae1a47e3	0x3285cd02	0x96418533	0x8a904d03
	0xf1633b43	0x0664d382	0x35fb0a83	0xe246b6c2
	0x8fc44d93	0x2fe1e763	0xd2823073	0x530dffc2
	0xe7dd8fe3	0xe4503972	0xad5f9022	0xdebed232
	0x10a9a642	0x9db60612	0x3ea3de03	0x5ed728a2
	0x3941d142	0xd961e823	0x43df53b2	0x7d7f7a82
	0x766512c3	0x6d9e3863	0xaacccc73	0xf74a2b92
	0x9ca25a32	0xd6a613e2	0x94819ca3	0xc98a4542

Our next task is to find the number of colliding key pairs. A careful look at the proposed algorithm reveals the number. At Step 1, we can choose $2^{32-5} = 2^{27}$ possible values for A_1, and the same number for B_1. The equations at Steps 2 and 5 can be solved always, thus there are $2^{2 \cdot 27} = 2^{54}$ different values for the tuple $(U_0^1, U_1^1, C^1, A^2, B^2, U_0^2, U_1^2, C^2, \tilde{C}^2)$ obtained at Steps 2–7. The condition at Step 8 filters $2^{32+5} = 2^{37}$ tuples, therefore we end up with $2^{54-37} = 2^{17}$ possible different values for $(U_0^1, U_1^1, U_0^2, U_1^2)$, and thus there are 2^{17} values for $(uk_0^1, uk_1^1, uk_0^2, uk_1^2)$. This is for the first branch only – if we take into account the four branches, in total there are $2^{4 \cdot 17} = 2^{68}$ colliding key pairs in *SC2000-256*. It is interesting to note that the collisions are found independently for each branch. Thus, to find all 2^{68} colliding key pairs we need only around $4 \cdot 2^{54} = 2^{56}$ operations.

Application to hash functions. The key collision attack on *SC2000-256* leads to practical collisions for single-block-length hash function instantiated with *SC2000-256*. Assume that the compression function $C(M, H)$ is based on the Davies-Meyer construction[3], i.e. $C(M, H) = E_M(H) \oplus H$. For the cipher $E_M(H)$, we find two colliding keys $M_1, M_2, M_1 \neq M_2$, such that for any plaintext H, it holds $E_{M_1}(H) = E_{M_2}(H)$. Therefore, in 2^{39} time we can find collisions for the compression function as

$$C(M_1, H) \oplus C(M_2, H) = E_{M_1}(H) \oplus H \oplus E_{M_2}(H) \oplus H = 0.$$

The collisions do not depend on the chaining value H, hence the above result holds for the hash function as well.

Double-block-length hash constructions based on *SC2000-256*, are not collision resistant as well. Let us take Hirose's construction [8], where the 256-bit compression function $C(g, h, M)$ (here g, h are two 128-bit chaining values, M is 128-bit message block) is based upon a cipher $E_K(P)$ with 256-bit key K and 128-bit state, and it is defined as

$$C(g, h, M) = E_{h||M}(g) \oplus g || E_{h||M}(g \oplus c) \oplus g \oplus c,$$

where $||$ is concatenation of two 128-bit words, and c is some non-zero constant. Hirose proved the collision resistance level of this construction to be around 2^{128}, when the underlying cipher is ideal. However, if we use *SC2000-256*, and two colliding keys $h_1||M_1, h_2||M_2$, then for any 128-bit chaining value g we obtain

$$\begin{aligned}
&C(g, h_1, M_1) \oplus C(g, h_2, M_2) = \\
&= [E_{h_1||M_1}(g) \oplus g || E_{h_1||M_1}(g \oplus c) \oplus g \oplus c] \oplus [E_{h_2||M_2}(g) \oplus g || E_{h_2||M_2}(g \oplus c) \oplus g \oplus c] = \\
&= E_{h_1||M_1}(g) \oplus g \oplus E_{h_2||M_2}(g) \oplus g || E_{h_1||M_1}(g \oplus c) \oplus g \oplus c \oplus E_{h_2||M_2}(g \oplus c) \oplus g \oplus c = \\
&= 0^{128} || 0^{128}.
\end{aligned}$$

Therefore, instead of 128-bit collision level achieved when using an ideal cipher, we obtain only 39 bits in the case of *SC2000-256*.

Application to SC2000-128 and SC2000-192. For the cases of 128-bit and 192-bit key *SC2000*, the last four, respectively two, words entering the intermediate key generation are copies of the original master key words. Hence in these cases two, respectively one, branches of the intermediate key generation has to be satisfied probabilistically. As there are 96 conditions per branch, and the remaining freedom per branch is 2^{32}, and the branches are cross dependent, i.e. for 128-bit key, the third branch depends on the keys of the first branch, and for 128-bit and 192-bit keys, the fourth branch depends on the second branch, the analysis for *SC2000-256* cannot be extended to *SC2000* with 128-bit and 192-bit keys.

[3] Among the analyzed by Preneel-Govaerts-Vandewalle [12] secure single- block-length modes, only Davies-Meyer mode, i.e. $C(M, H) = E_M(H) \oplus H$ allows 256-bit messages inputs and 128-bit chaining values.

5 Conclusion

We have shown that the key schedule of *SC2000-256* is not injective, and the cipher has 2^{68} pairs of colliding keys. These pairs are due to the two weaknesses in the key schedule: the non-symmetric function used in the second phase easily cancels a particular difference, and the four branches in the first phase are independent and can be attacked separately. Based on the combination of the two weaknesses, we have derived an algorithm that in 2^{39} operations finds one pair of colliding keys, and in 2^{56} finds all of them. As a proof of concept we have produced colliding keys in a matter of a few hours on a PC. Thus the collision resistance level of hash functions based on *SC2000-256* is very low.

SC2000-256 suffers from a practically exploitable security weakness and cannot model an ideal cipher. In spite of the *SC2000-256* cipher being in the CRYPTREC portfolio for more than 10 years and in spite of considerable previous evaluation, this paper is the first to discover this design flaw. This is probably due to complexity of the design and is an example in favor of clean and easy to analyze design strategies.

Acknowledgement. The authors would like to thank Jérémy Jean for proposing improvements to the results and the anonymous reviewers of SAC 2014 for their helpful comments. Ivica Nikolić is supported by the Singapore National Research Foundation Fellowship 2012 NRF-NRFF2012-06.

References

1. Aumasson, J.-P., Nakahara Jr., J., Sepehrdad, P.: Cryptanalysis of the ISDB scrambling algorithm (MULTI2). In: Dunkelman [6], pp. 296–307
2. Biham, E., Dunkelman, O., Keller, N.: New results on boomerang and rectangle attacks. In: Daemen and Rijmen [4], pp. 1–16
3. Biryukov, A., Gauravaram, P., Guo, J., Khovratovich, D., Ling, S., Matusiewicz, K., Nikolić, I., Pieprzyk, J., Wang, H.: Cryptanalysis of the LAKE hash family. In: Dunkelman [6], pp. 156–179
4. Daemen, J., Rijmen, V. (eds.): FSE 2002. LNCS, vol. 2365. Springer, Heidelberg (2002)
5. Daum, M.: Cryptanalysis of hash functions of the MD4-Family. Ph.D. Thesis, Ruhr-Universität Bochum, May 2005
6. Dunkelman, O. (ed.): FSE 2009. LNCS, vol. 5665. Springer, Heidelberg (2009)
7. Dunkelman, O., Keller, N.: Boomerang and rectangle attacks on SC2000. In: NESSIE 2nd Workshop (London) (2001)
8. Hirose, S.: Some plausible constructions of double-block-length hash functions. In: Robshaw, M. (ed.) FSE 2006. LNCS, vol. 4047, pp. 210–225. Springer, Heidelberg (2006)
9. Kelsey, J., Schneier, B., Wagner, D.: Key-schedule cryptanalysis of IDEA, G-DES, GOST, SAFER, and Triple-DES. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 237–251. Springer, Heidelberg (1996)
10. Lu, J.: Differential attack on five rounds of the SC2000 block cipher. In: Bao, F., Yung, M., Lin, D., Jing, J. (eds.) Inscrypt 2009. LNCS, vol. 6151, pp. 50–59. Springer, Heidelberg (2010)

11. Matsui, M.: Key collisions of the RC4 stream cipher. In: Dunkelman [6], pp. 38–50
12. Preneel, B., Govaerts, R., Vandewalle, J.: Hash functions based on block ciphers: A synthetic approach. In: Stinson, D.R. (ed.) CRYPTO 1993. LNCS, vol. 773, pp. 368–378. Springer, Heidelberg (1994)
13. Raddum, H., Knudsen, L.R.: A differential attack on reduced-round SC2000. In: Vaudenay, S., Youssef, A.M. (eds.) SAC 2001. LNCS, vol. 2259, pp. 190–198. Springer, Heidelberg (2001)
14. Robshaw, M.: A cryptographic review of Cipherunicorn-A. Technical Report, CRYPTRECT Technical report (2001)
15. Shimoyama, T., Yanami, H., Yokoyama, K., Takenaka, M., Itoh, K., Yajima, J., Torii, N., Tanaka, H.: The block cipher SC2000. In: Matsui, M. (ed.) FSE 2001. LNCS, vol. 2355, pp. 312–327. Springer, Heidelberg (2002)
16. Technical Report 1087. Analysis of SC2000 (2000). http://www.ipa.go.jp/security/enc/CRYPTREC/fy15/doc/1087_sc2000.pdf
17. Yanami, H., Shimoyama, T., Dunkelman, O.: Differential and linear cryptanalysis of a reduced-round SC2000. In: Daemen and Rijmen [4], pp. 34–48