Security Analysis of the Block Cipher SC2000

VERSION 1.1

FINAL REPORT

Alex Biryukov
University of Luxembourg, Luxembourg

Ivica Nikolić
Nanyang Technological University, Singapore

Contents

Chapter 1

Introduction

1.1 Description

SC2000 is 128-bit block cipher with a variable number of rounds that depends on the key size: 6.5 rounds for 128-bit key, and 7.5 rounds for 192-bit and 256-bit keys. The cipher is word-oriented – all of the transformations are word-wise, with the exception of the S-boxes which are applied to a subset of bits within a single word. Each round of *SC2000* is composed of two subkey addition layers I_{func}, one S-box layer B_{func}, and two one-round Feistel layers R_{func}. The layers can be described as follows:

- I_{func} - XOR of 4 subkey words to the state words
- B_{func} - application of S-boxes to the state words. First a matrix 4x32 is produced from the state words, and then an S-box is applied to each 4-bit column of the matrix
- R_{func} - One round Feistel is applied to the four words. The round function F is composed of S-box layer, and two linear layers

The key schedule of *SC2000* expands the initial master key into subkeys in three steps:

1. Extensions to 8 words. In the cases of 128-bit and 192-bit keys, the master key is extended to 8 words (256-bits) by copying words of the key
2. Intermediate key generation. The 8 words of the key are threated as 4 pairs of words. Each pair undergoes a series of transformation (involving S-boxes, linear transformations, XORs and modular additions) and for each pair 3 intermediate keys are produced.
3. Extended key generation. The 12 intermediate keys are extended to 56 (or 64) subkey words – each subkey word is generated from 4 intermediate words with application of word-wise operations: modular addition, XOR, and rotation.

SC2000 applies unconventionally high number of distinct transformations, including S-boxes of different sizes, bit-oriented matrix multiplication, logical operations AND,OR, XOR, NOT, word-oriented rotations, modular additions and subtractions in the key schedule. This mixture of transforms makes *SC2000* exceptionally hard to analyze.

1.2 Known Analysis

There are only a few published results on the analysis of *SC2000* . All the attacks are on round-reduced versions of the cipher: boomerang/rectangle attacks on 3.5 rounds [1], differential and linear attack on up to 4.5 rounds [5, 6] and a differential attack on 5 rounds [4].

1.3 Analysis of the Transformations

To estimate the resistance of *SC2000* against various attack, first we focus on each of the transformations used in the cipher. In particular, we analyze the S-boxes and the linear transformations.

1.3.1 S-box Analysis

The complexity of the differential attacks on a cipher is tightly related to the differential properties of the S-boxes used in the cipher. *SC2000* applies four different types of non-linear bijective S-boxes: one 6x6 S-box, one 5x5 S-box, and two 4x4 S-boxes. The maximal differential propagation probabilities are $2^{-4}, 2^{-4}, 2^{-2}$, respectively. Hence we can conclude that the S-boxes are optimal against differential attacks.

1.3.2 Analysis of the Linear Transformations

Two types of linear transformations are used in the round function: M_{func} and L_{func}. The branch number of the matrix multiplication M_{func}, i.e. the minimal number of active input and output bits, is 9. This is sub-optimal, as the maximal branch number for this type of transformations is 33, and in the linear case it is 12. Low branch number could lead to potential (impossible) differential attacks on higher number of rounds, due to the low diffusion of the bits. L_{func} can be seen as another matrix (64x64) multiplication with a branch number 2. However, we believe its main purpose is to introduce diffusion of bits between two words, rather than diffusion within a single word.

1.3.3 Analysis of the Word-oriented Transformations in the Key Schedule

Besides the previously mentioned transformations, the key schedule of *SC2000* applies additional operations such as modular additions, subtractions, and rotations. The rotation constant is always one, hence the avalanche effect of the combination rotation-addition is insufficient. This leads to the case where in the last transformation of the key schedule (the extended key generation) can be seen (almost) as a T-function[2], i.e. the more significant bits of the output do not depend on the less significant bits of the input. Additionally, the addition-rotation-XOR transforms are known to be susceptible to rotational attacks.

Chapter 2

An Attack on *SC2000-256*

2.1 A Dedicated Attack on *SC2000-256*

In this section we present an attack on *SC2000* with 256-bit keys. The attack exploits a weakness in the key-schedule which leads to key collisions for the cipher with practical complexity.

2.1.1 Description of the Framework

Further we define the attack framework and give complexities for the generic case.

Definition 1 *The triplet (P, K_1, K_2) is called a key collision for the cipher $E_K(P)$ if $K_1 \neq K_2$ and $E_{K_1}(P) = E_{K_2}(P)$.*

The basic idea of the key collisions is that it is hard to find two distinct keys and a plaintext that will produce the same ciphertext. There is no known algorithm that can produce such tripplet with a time complexity faster than $2^{n/2}$ for an ideal n-bit block cipher – this number is related to the complexity of producing collisions for an n-bit hash function.

Definition 2 *The tuple $(K_1, K_2, P_1, P_2, \ldots, P_m)$ is called m-multi key collision for the cipher $E_K(P)$ if $K_1 \neq K_2, P_i \neq P_j, i \neq j$ and $E_{K_1}(P_i) = E_{K_2}(P_i), i = 1, 2, \ldots, m$.*

Note that producing multi key collision for an ideal cipher requires more complexity than producing a collision as the later is a particular case of the former for $m = 1$. The complexity of producing m-multi key collisions for an ideal cipher can be estimated as the complexity of producing m multicollisions for a hash function, which is around $2^{\frac{m-1}{m}n}$ for an n-bit hash function.

Observation 1 *If for an n-bit block cipher $E_K(P)$, there exist two master keys K_1, K_2 such that the key schedule algorithm $KS(K)$ of the cipher produces the same set of subkeys, i.e. $\exists K_1, K_2, K_1 \neq K_2$ and $KS(K_1) = KS(K_2)$, then the* *complexity of producing 2^n-multi key collisions is not higher then the complexity of of finding K_1, K_2 that satisfy $KS(K_1) = KS(K_2)$.*

This comes from the fact that once the two distinct master keys produce the same set of subkeys, the encryption with K_1 for any plaintext P will produce the same ciphertext as the encryption with K_2. Hence the set of multi key collisions is maximal – its size equals to the size of the whole codebook.

2.1.2 The Attack on *SC2000-256*

For *SC2000-256* we are able to find two distinct master keys that produce the same subkey words and hence obtain the maximal 2^{128}-multi key collisions. To do so we have to analyze the two steps of the subkeys generation. The core idea is to produce carefully chosen pair of intermediate keys with a particular difference, such that when key pair is supplied to the extended key generation function, this procedure would cancel the difference. With superscripts 1 and 2 we denote the 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. The following observation will be used in the attack:

Observation 2 *Let each pair (X_1, X_2) of the intermediate key pairs (a_i^1, a_i^2), (b_i^1, b_i^2), (c_i^1, c_i^2), (d_i^1, d_i^2), $i = 0, 1, 2$ satisfies the conditions:*

$$X_2 = \overline{X_1} + 3 \tag{2.1}$$

$$X_1 \wedge \mathsf{0x8000000f} = \mathsf{0x80000003} \tag{2.2}$$

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

In short, the observation claims that if the most significant bits of each intermediate key word produced from the fist master key are set to 1, and the four last significant bits are set to 3, then to pass the extended key generation with probability 1, it is sufficient to find a second master key that will produce intermediate key words that are modular addition of the negation of the first intermediate key words and the value 3.

We will not prove formally this observation – it can be checked with a simple experiment on a PC. It is based on the properties of modular addition, rotation to the left by one bit and XOR. Indeed, the exact values of the key bits (1 for MSB and 3 for the four LSB), and the value 3 where obtained with a limited brute-force of the space of all values that pass the extended key generation with probability 1. Besides these, many more different values pass the generation with probability 1 – we have chosen our values randomly from the set of all values that satisfy similar observation.

Our next task is to produce a pair of intermediate key words that satisfies the conditions from the observation. For the sake of simplicity, we will take into account only the first condition, i.e. $X_2 = \overline{X_1} + 3$ and only later reintroduce the second condition. 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 $K_i = M_{func}(S_func(4 \cdot i)), i = 0, 1, 2$. Then, taking into account the intermediate key generation procedure, the condition 1 of the observation for the pairs $(a_i^1, a_i^2), i = 0, 1, 2$ is equivalent to solving the following system of equations:

$$(U_0^1 + K_1) \oplus U_1^1 = A^1 \tag{2.3}$$
$$(U_0^1 + K_2) \oplus 2 \cdot U_1^1 = B^1 \tag{2.4}$$
$$(U_0^1 + K_3) \oplus 3 \cdot U_1^1 = C^1 \tag{2.5}$$
$$(U_0^2 + K_1) \oplus U_1^1 = A^2 \tag{2.6}$$
$$(U_0^2 + K_2) \oplus 2 \cdot U_1^1 = B^2 \tag{2.7}$$
$$(U_0^2 + K_3) \oplus 3 \cdot U_1^1 = C^2 \tag{2.8}$$
$$A^2 = S_{func}^{-1}(M_{func}^{-1}(\overline{M_{func}(S_{func}(A^1))} + 3)) \tag{2.9}$$
$$B^2 = S_{func}^{-1}(M_{func}^{-1}(\overline{M_{func}(S_{func}(B^1))} + 3)) \tag{2.10}$$
$$C^2 = S_{func}^{-1}(M_{func}^{-1}(\overline{M_{func}(S_{func}(C^1))} + 3)) \tag{2.11}$$

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

$$(U_0^1 + K_1) \oplus U_1^1 = G((U_0^2 + K_1) \oplus U_1^1) \tag{2.12}$$
$$(U_0^1 + K_2) \oplus 2 \cdot U_1^1 = G((U_0^2 + K_2) \oplus 2 \cdot U_1^1) \tag{2.13}$$
$$(U_0^1 + K_3) \oplus 3 \cdot U_1^1 = G((U_0^2 + K_3) \oplus 3 \cdot U_1^1) \tag{2.14}$$

Hence, it is a system of three equations with four unknowns – theoretically, it has 2^{32} solutions for any values of K_1, K_2, K_3 and a bijective function G[1]. To find a solutions we will use to following algorithm:

1. Fix random A_1, B_1
2. Find U_0^1, U_1^1 that satisfy the first two equations, i.e. (2.3),(2.4)
3. Produce C_1 from U_0^1, U_1^1
4. Produce A_2, B_2 from A_1, B_1 with the function G
5. Find U_0^2, U_1^2 from A_2, B_2
6. Produce C_2 from U_0^2, U_1^2
7. Produce $\tilde{C_2}$ from C_2 with the function G

[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.

8. If C_2 is not equal to $\tilde{C_2}$ go to step 1

9. The quartet $(U_0^1, U_1^1, U_0^2, U_1^2)$ is the solution for the system

The values of C_2 and $\tilde{C_2}$ coincide with probability 2^{-32} hence the steps 1-7 have to be repeated at most 2^{32} times.

To use our algorithm we should be able to efficiently find solutions for the system of type:

$$(U_0 + K_1) \oplus U_1 = A \tag{2.15}$$

$$(U_0 + K_2) \oplus 2 \cdot U_1 = B \tag{2.16}$$

With basic algebraic transformations this system can be reduced to the form:

$$U_1 = ((K_1 - K_2) + (2 \cdot U_1 \oplus B)) \oplus A \tag{2.17}$$

$$U_0 = (A \oplus U_1) - K_1 \tag{2.18}$$

Hence we can solve the system if we can solve the equation (2.17).

Lemma 1 *There is an algorithm that with complexity linear in the size of the words can find the unique solution for the equation:*

$$X = ((2 \cdot X \oplus B) + C) \oplus A, \tag{2.19}$$

where A, B, C are some word constants.

Proof: We solve the equation bit by bit starting from the least significant and moving towards to most significant bit. We use subscripts to denote the bits within a word, e.g. X_5 is the sixth least significant bit of the word X. Note that the expression $2 \cdot X$ is a simple shift by one position to the left of X and therefore the s-th bit of $2 \cdot X$ is actually the $(s-1)$-th bit of X. Further, we present the algorithm inductively, i.e. we show how to find the LSB (i.e. 0-th bit) and, assuming we have solved for t-th, we show for $(t+1)$-th bit.

- *Bit 0.* For the LSB, the equation (2.19) takes form:

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

  hence the LSB of X_0 can be uniquely determined with a simple XOR of three bits. We also compute the carry (cr) from the addition $(2 \cdot X \oplus B) + C$, i.e. $cr_0 = B_0 \cdot C_0$.

- *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 constant number of operations.

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

The lemma gives us the complexity for the step 2 of the algorithm, i.e. we can solve to system for any A, B with a constant complexity (as $n = 32$).

Now we can return to the condition 2 of the observation 2. We only have to slightly tweak our algorithm in order to satisfy this condition as well. Further we present the full algorithm for producing the intermediate key words:

1. Fix random A_1, B_1 such that the MSB of A_1, B_1 are 1, and the four least significant bits are 3
2. Find U_0^1, U_1^1 that satisfy the first two equations
3. Produce C_1 from U_0^1, U_1^1
4. Produce A_2, B_2 from A_1, B_1 using the function F
5. Find U_0^2, U_1^2 from A_2, B_2
6. Produce C_2 from U_0^2, U_1^2
7. Produce $\tilde{C_2}$ from C_2 using the function F
8. If C_2 is not equal to $\tilde{C_2}$, or the MSB of C_2 is 0, or the four LSB are not 3, go to step 1
9. The quartet $(U_0^1, U_1^1, U_0^2, U_1^2)$ is a solution for the system

Note, the additional conditions introduced in the step 1, do not change the complexity of the algorithm. On the other hand, the conditions in the step 8 increase the frequency of repeating steps 1-7 by a factor of 2^5 – this comes from the fact that there are conditions on 5 bits of C_2. Hence, the total complexity of the algorithm is $2^{32} \cdot 2^5 = 2^{37}$. Since we have to solve for four branches of the intermediate keys, i.e. a, b, c, d, the total complexity of finding intermediate key words is $4 \cdot 2^{37} = 2^{39}$. Once we have the pairs $(U_i^1, U_i^2), i = 0, \ldots, 7$, we can easily produce the pairs (u_i^1, u_i^2) by inverting the transformations at the beginning of the intermediate key generation, i.e. S_{func}, M_{func}, thus finding the the pair of master keys.

We have implemented our attack on a PC and found pair of master keys (K_1, K_2) that produce the same extended keys. The words of the master keys are given in Tbl. 2.1.

2.1.3 Other Attack Frameworks

The attack can trivially be extended to the case of finding multicollisions for the hash function produced with *SC2000-256* in Davies-Meyer mode, i.e. $H(M) = E_M(H_0) \oplus H_0$. As in the original attack the difference is only in the keys, in the hash function mode the difference will be only in the message words and

Table 2.1: 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`

thus multicollisions can be produced with complexity of 2^{39} calls to the hash function, beating even the generic bound 2^{64} for a single collision. Our attack can be seen as a related-key differential attack, where a non-zero input difference in the master key is canceled by the key schedule. Hence, if a high probability trail with a non-zero input difference and zero output difference can be built, this would lead to a related-key differential attack on *SC2000-256* . We have investigated this possibility, however we were unable to find such trail. The main obstacles are the two S-box layers in the intermediate key generation and inability to solve the main system probabilistically – each significantly reduces the probability of the differential trail resulting in trails below 2^{-256}.

2.1.4 Applications to *SC2000-128* and *SC2000-192*

For the cases of 128-bit and 192-bits, 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 depend on the keys of the first branch, and for 128-bit and 192-bit keys, the fourth branch depends on the second branch, the attack cannot be extended to *SC2000* with 128-bit and 192-bit keys.

Chapter 3

Analysis Against Various Attacks

Further we present the resistance of *SC2000* against different single-key and related-key attacks. In particular, we focus on:

- Classical Differential Cryptanalysis
- Impossible Differential Cryptanalysis
- Slide Attacks
- Rotational Attacks

3.1 Differential Cryptanalysis

Differential attacks are the most popular form of cryptanalysis for block ciphers. A widely accepted approach for designing a cipher resistant against differential attacks is the wide trail strategy used in AES. However, this design technique requires byte-oriented ciphers. Additionally, a few approaches were proposed for analysis of the resistance of a cipher against differential attacks, however they all assume the underlying cipher is byte-oriented. As *SC2000* fails to meet this requirement, the only known approach to analyze the cipher is ad-hoc, i.e. trying to build a differential trail (or possible a differential) using some heuristics and assumptions.

3.1.1 Single-key Differentials

In the single-key scenario we assume there is no difference in the key, and there some initial difference in the plaintext.

One-round Differential Trail

As R_{func} is a single-round Feistel, to produce the best trail for one round of *SC2000* we want a difference only in the second R_{func}. Hence we introduce difference at the same bit position in the words a, b, c, d of the plaintext. The S-box of B_{func} converts the difference into a single bit difference in a, the first R_{func} is passed for free, and only in the second R_{func}, there is only one active S-box. Hence, the probability of this trail is at least $2^{-3} \cdot 2^{-4} = 2^{-7}$.

Two and more round Differential Trails

Note that as there are two S-box layers per round, and the S-boxes are bijective, in any two-round differential trail, there have to be at least 4 active S-boxes. It is trivial however to see that such two-round trail does not exist. Moreover, due to the multiple linear diffusion layers, finding an optimal trail for two-rounds is infeasible. Hence, we cannot prove that our trails are optimal, i.e. trails with better probability might exist. One strategy to build a two-round trail is to start with a difference at the beginning of the second round, and expand the difference forward through the second round, and backwards through the first round, thus obtaining the initial starting and ending difference of the trail. We would want to avoid as many active S-boxes as possible, in order to get a trail with a higher probability. Let this difference be one bit difference in a. Then, in the forward direction, there would be only two active S-boxes, hence the probability in round 2 is 2^{-7}. In the backward direction, due to the branch number of M_{func}, is follows that at the input of M_{func}, there have to be at least 8 active bits, which means there where at least two active S-boxes in the F, thus the probability of F is at most 2^{-8}. If the 8 bits were from only two distinct S-boxes, it means at the input where might have been only 2 active bits. Thus in B_{func} of round 1, there were two active S-boxes, hence the probability of this part is at most 2^{-6}. In total, we get that this type of two-round trail has a probability of at most $2^{-7-8-6} = 2^{-23}$. It is unclear if a trail with such probability exist at all (there might be inconsistencies with the linear layers in the first round). For trails on more than two rounds, the above analysis does not produce any meaningful results as there are two many possible ways the branching could occur.

3.1.2 Related-key Differentials

To analyze the resistance of *SC2000* against related-key attacks we focus on the highly non-linear key schedule and try to build differential trails with maximal probability only for the key schedule. For the differential probability of the active S-boxes we take 2^{-4}, and assume the probability of modular additions to be as the one presented in [3], i.e. if two words X, Y, have a difference $\Delta X, \Delta Y$, then the best probability for the difference in Z = X+Y is $2^{-hamming(\Delta X \oplus \Delta Y)}$.

Intermediate Key Generation

First let us focus on the intermediate key exchange procedure within a single branch (out of the four branches) and assume there is a difference only in one of the two input master words. By running a brute force on the space of all single-word input differences in the first master key of the branch, we have found the best probability of a differential trail to be 2^{-25} for a single intermediate key. By launching an additional brute force, we have found the accumulated probability of all three intermediate keys for a branch – the best probability is 2^{-38}. The significant improvement over the theoretical $2^{-25 \cdot 3} = 2^{-75}$, is due to the fact that S-boxes in first S_{func} are counted only once. Similarly, when the difference is in the second key of the branch, then the best probability is 2^{-24} for constants 1 and 2, and 2^{-25} for constant 3, while the accumulated probability for all of the three intermediate keys in the branch 2^{-34}. As all of the transformations are bijective, it means that each of the single word case difference results in a situation where all three output words have difference, e.g. if there is difference only in $uk[0]$ (or only in $uk[1]$) then there would be a difference in $a[0], a[1], a[2]$. To summarize, we get:

Observation 3 *If in the intermediate key generation, there is a difference only in the first word of the branch, then the probability of the differential trail in this branch is at most* 2^{-38}*, and all three intermediated subkey words would have a difference.*

Observation 4 *If in the intermediate key generation, there is a difference only in the second word of the branch, then the probability of the differential trail in this branch is at most* 2^{-34}*, and all three intermediated subkey words would have a difference.*

Therefore, if we count only the intermediate key generation, and there is difference in a single word even in all of the four branches the probability of the best trail is $2^{-34 \cdot 4} = 2^{-132} > 2^{-256}$. Hence theoretically related-key differential attacks are possible and thus we have to investigate more complex cases (and take into account the probability of the extended key generation as well).

When there is difference in both of the input words of the branch, the situation becomes more complicated and the brute force space is too large for practical implementations as there are 2^{64} possible output differences after the first S-box layer. First, let us focus on the number of active words per branch – for the sake of simplicity we analyze on the first branch. Let U_0, U_1 be the values $U_i^j = M_{func}(S_{func}(uk^j[i]), i = 1, 2, j = 1, 2,$ where $(uk^0[0], uk^1[0]), (uk^0[1], uk^1[1])$ are the pairs of related key words with a non-zero difference. Also let $K_i = M_{func}(S_{func}(4 * i))$. If we require zero differences in all three words of the output of the intermediate key generation, then this can be translated into the following system of equations:

$$(U_0^0 + K_0) \oplus U_1^0 = (U_0^1 + K_0) \oplus U_1^1 \tag{3.1}$$

$$(U_0^0 + K_1) \oplus 2U_1^0 = (U_0^1 + K_1) \oplus 2U_1^1 \tag{3.2}$$

$$(U_0^0 + K_2) \oplus 3U_1^0 = (U_0^1 + K_2) \oplus 3U_1^1 \tag{3.3}$$

Now, let us take only the first two equations of this system, i.e. (3.1), (3.2).

Observation 5 *The system (3.1), (3.2) has a unique trivial solution* $U_0^0 = U_0^1, U_1^0 = U_1^1$.

Proof: Let us first find the value of the least significant bits (LSB). From (3.2), it follows that for the LSB $U_0^0 = U_0^1$, as multiplication by 2 is simple shift to the left, hence the LSB of U_1^0, U_1^1 play no role in this equation. Therefore, from (3.1), for the LSB is follows that $U_1^0 = U_1^1$, and thus the LSB of the related master words must coincide.

If we repeat the following process iteratively, we will obtain that for each bit, it has to hold a similar fact, hence the related master key words coincide, and therefore the input difference is zero. This contradicts the initial condition that the input difference was non-zero. □

Thus we get that the difference in the output words $a[0], a[1]$ cannot be simultaneously zero. A similar observation holds if we try to solve the system composed of the equations (3.2), (3.3), i.e.:

Observation 6 *The system (3.2), (3.3) has a unique trivial solution* $U_0^0 = U_0^1, U_1^0 = U_1^1$.

The last possible system, composed of (3.1),(3.3) can have a non-trivial solution however, i.e.:

Observation 7 *The system (3.1), (3.3) always has a non-trivial solution.*

Based on the above observations we can conclude that:

Observation 8 *If in the intermediate key generation, there is a difference in both of the incoming words of the branch, then either all three intermediated subkey words have a difference result in a trail with a maximal probability of* 2^{-20}*, or only one word does not have a difference (trail with probability of at most* 2^{-16} *or only the first and the third word does not have a difference - trail with probability of* 2^{-12}.

Extended Key Generation

In the extended key generation the XOR difference can go through three different operations: XOR, modular addition and rotations. Only the modular addition are passed probabilistically. As we are looking for the upper bound, we assume that if two words contain difference, this difference passes the modular addition with probability 1, i.e. the case when $\Delta X = \Delta Y$ from [3]. However, once the equality is fixed, then if the words are matched again (through other words) with rotation 1, then we take the probability 2^{-1}. For example, if in the generation of the extended key words, we have the sum $a[0] \lll 1 + b[2]$, and both $a[0]$ and $b[2]$ contain difference, then we can assume they have the same difference (with the first difference rotated on one bit to the right), thus the modular addition is passed with probability 1. However, if in some other extended key generation words we also assume that both $a[0] \lll 1 + c[2]$ and $d[1] \lll 1 + b[2]$ are passed with probability 1, then the probability of the difference in the addition $c[2] \lll 1 + d[1]$ cannot be 1 anymore (unless the difference was $111\ldots1$, but then the probability of the intermediate key generation would have been much lower).

If only one of the terms in the addition has a difference, then the probability of the modular addition is at most 2^{-1}.

The Complete Key Schedule

To find the upper bound on the probability we have implemented a brute force on all the possible combinations of active words at the input of the key schedule. In the search we use the results of observations 3,4,8. Depending on the number of active input words in a branch, we produce all the possible output (at the intermediate key generation) combinations of active words (with the probabilities mentioned in the observations). Then, we add the probability of the extended key generation to produce the final probability, i.e. upper bound on the probability of the best trail in the key schedule. The search resulted in a best trail with probability 2^{-29}. It is achieved when the third branch contains two input active words, that after the intermediate key generation produce difference in all of the three words of the third branch.

Related-key Differential Trails

Obviously the upper bound on the probability of the best trail in the key schedule , i.e. 2^{-29}, is insufficient to give any meaningful results as the claim, that the probability of the best related-key differential trails cannot be higher than 2^{-29}, gives no security margin for *SC2000* in the related-key scenario. The biggest obstacle again is the complexity of both the key schedule and the state transformation, in particular the linear layers M_{func} and L_{func}. We cannot exclude the possibility of launching related-key attacks as it seems that high probability differential trails might exist in the key schedule. On the other hand it is unclear how to combine these trails with the trails in the state since the word-oriented state transformations do not allow feasible exhaustive search of all possible combinations of trails in the key schedule and in the state.

3.2 Impossible Differential Attacks

Impossible differentials can be very powerful form of attacks but mainly on byte-oriented ciphers. In the case of word-oriented ciphers, the diffusion between the words is much faster, and usually full active state is reached in a few rounds making the cipher resistant against impossible attacks even with a small number of rounds. This is the case for *SC2000* as well. First we examine related-key impossible differential attacks. Note, one can build probability 1 differentials for the keys schedule only with respect to active (non-active) words, i.e. truncated differentials, as the transformations in the key schedule are non-linear. When there is difference either in a single word within the branch, or in both words of the branch, all three intermediate keys in the branch will have difference. Hence, even if there is an input difference in only one of the branches, due to the extended key generation procedure, there would be a difference in all of the subkey words. Therefore, related-key impossible differentials would not give any advantage to the attacker over the single-key impossible differentials. Thus it is sufficient to examine only the single-key differentials and ignore the subkey additions in each round.

Each round begins with a B_{func} which besides non-linearity introduces an additional diffusion between the bits (at the same position) of all four words in the state. Therefore, if there is even only a single bit difference in the plaintext, after the application of B_{func}, each of the four words will have at least one bit undetermined. The F_{func} assures that this single bit difference in one word would make the difference in the whole (or at least the big part) word undetermined. Since F_{func} is applied to all four of the state (in two applications of the R_{func}), all (or the big part) of the bits in the four words will be undetermined. Therefore, probability 1 differential can be built not more than on one round, and thus we expect that no impossible differential are achievable on more than two rounds of *SC2000* . With various techniques a round at the beginning and at the end might be skipped – however, the high security margin guarantees that no impossible differential attack can be mounted on the full-round cipher.

3.3 Slide Attacks

Slide attacks are applicable to ciphers that have similar rounds. This is not the case for *SC2000* mainly because of the key schedule – there is no simple relation between two consecutive subkeys. The extended key generation procedure produces extended keys (round subkeys) that depend on the intermediate keys in a random manner. This results in a situation where two consecutive subkeys have no simple relation. Hence, we can conclude that *SC2000* is resistant against single-key and related-key slide attacks.

3.4 Rotational Attacks

The addition-rotation-XOR structure of the extended key generation procedure is highly susceptible to rotational attacks. However, the vast use of S-boxes (in particular S-boxes of different size) in the key schedule as well as in the state, make *SC2000* resistant against rotational attacks.

Bibliography

[1] O. Dunkelman, N. Keller, et al. Boomerang and rectangle attacks on sc2000. 2001.

[2] A. Klimov and A. Shamir. Cryptographic applications of t-functions. In M. Matsui and R. J. Zuccherato, editors, *Selected Areas in Cryptography*, volume 3006 of *Lecture Notes in Computer Science*, pages 248–261. Springer, 2003.

[3] H. Lipmaa and S. Moriai. Efficient algorithms for computing differential properties of addition. In M. Matsui, editor, *FSE*, volume 2355 of *Lecture Notes in Computer Science*, pages 336–350. Springer, 2001.

[4] J. Lu. Differential attack on five rounds of the sc2000 block cipher. In F. Bao, M. Yung, D. Lin, and J. Jing, editors, *Inscrypt*, volume 6151 of *Lecture Notes in Computer Science*, pages 50–59. Springer, 2009.

[5] H. Raddum and L. R. Knudsen. A differential attack on reduced-round sc2000. In S. Vaudenay and A. M. Youssef, editors, *Selected Areas in Cryptography*, volume 2259 of *Lecture Notes in Computer Science*, pages 190–198. Springer, 2001.

[6] H. Yanami, T. Shimoyama, and O. Dunkelman. Differential and linear cryptanalysis of a reduced-round sc2000. In J. Daemen and V. Rijmen, editors, *FSE*, volume 2365 of *Lecture Notes in Computer Science*, pages 34–48. Springer, 2002.