Finding SHA-1 Characteristics: General Results and Applications

Christophe De Cannière[1,2] and Christian Rechberger[1]

[1] Graz University of Technology
Institute for Applied Information Processing and Communications
Inffeldgasse 16a, A–8010 Graz, Austria
{Christophe.DeCanniere,Christian.Rechberger}@iaik.tugraz.at
[2] Katholieke Universiteit Leuven, Dept. ESAT/SCD-COSIC,
Kasteelpark Arenberg 10, B–3001 Heverlee, Belgium

Abstract. The most efficient collision attacks on members of the SHA family presented so far all use complex characteristics which were manually constructed by Wang *et al.* In this report, we describe a method to search for characteristics in an automatic way. This is particularly useful for multi-block attacks, and as a proof of concept, we give a two-block collision for 64-step SHA-1 based on a new characteristic. The highest number of steps for which a SHA-1 collision was published so far was 58. We also give a unified view on the expected work factor of a collision search and the needed degrees of freedom for the search, which facilitates optimization.

1 Introduction

Shortcut attacks on the collision resistance of hash functions are usually differential in nature. In the differential cryptanalysis of block ciphers, characteristics with arbitrary starting and ending differences spanning less than the full number of rounds and having a sufficient high probability allow key recovery attacks faster than brute force. This contrasts the situation in the case of collision attacks on hash functions. Here characteristics of sufficiently high probability need to start and end with chaining input and output difference being zero, injected differences (via the message input) are expected to cancel out themselves.

Members of the MD4 hash function family like the widely used SHA-1 mix simple building blocks like modular addition, 3-input bit-wise Boolean functions and bit-wise XOR, combine them to steps and iterate these steps many times. High probability characteristics which are needed for fast collision search attacks exploit situations where differences with respect to one operation propagate with high probability through other building blocks as well. As an example, an XOR difference in the most significant bit of a word propagates with probability one through a modular addition. The best characteristics for SHA-1 are constructed such that these and similar effects are maximized. However they do not fulfill the requirement of zero differences at the chaining inputs/outputs which makes them not directly usable for fast collision search attacks. Earlier work on SHA-1 [2, 13] therefore consider characteristics which fulfill this requirement at the cost of a less optimal probabilities.

However, the fact that an attacker has complete control over the message input, and thus control over the propagation of all differences in the first steps, gives more freedom in the choice of good characteristics. Roughly speaking, the probability of complex characteristics spanning the first steps which *connect* to a desired high probability characteristic does not affect the performance of a collision search. Hence, finding these complex *connecting characteristics* helps to improve the performance of collision search attacks. In the case of SHA-1, finding such characteristics made differential collision search attacks on the full SHA-1 possible in the first place. To reflect the fact that the desired characteristics to connect to have usually probability one in a linearized model of the hash function, they are referred to as *L-characteristics*. The connecting characteristics do not have this property, hence the name *NL-characteristics*.

So far, little is known about the construction of these connecting NL-characteristics. Wang *et al.* describe in their seminal paper [20] an approach which is based on following and manipulating differences *manually* [23] in combination with a great deal of experience and intuition. Follow-up work on SHA-1 [16] as well as on MD4 [9], MD5 [3, 7, 8, 15] and SHA-0 [10] all build up on the characteristics given in the papers of Wang *et al.* [17, 20–22]. The only exception is recent work by Schläffer and Oswald [14] on the conceptually much simpler MD4, where an algorithm for finding new characteristics given the same message difference as originally used by Wang *et al.* is reported. No one succeeded so far in showing a similar ability in the case of SHA-1. By employing a new method and using SHA-1 as an example, we show in this article that finding useful NL-characteristics is also possible in more complex hash functions.

As shown in informal presentations by Wang [18, 19], the actual shape/design of these connecting NL-characteristics interacts with speed-up techniques at the final-search stage. These techniques are referred to as message modification techniques and little details about them in the context of SHA-1 are publicly known so far. To sum up, two important methods (finding connecting NL-characteristics and message modification) are not fully understood, but heavily affect the actual collision-search complexity. Therefore, it currently seems impossible to reason about the limits of these techniques, other than improving on the current results in an ad-hoc manner. Hence the need for automated search tools as the one presented in this paper.

Looking at the recent results of Wang *et al.* on SHA-1, we see that more degrees of freedom are needed for speedup-purposes. As mentioned in [18], message conditions and state variable conditions need to be fulfilled for that purpose. It is observed that "the available message space is tight", which refers to the remaining degrees of freedom.

The new view we propose unifies finding complex characteristics and speeding up the final search phase. By calculating the expected number of collisions, given the degrees of freedom, we tackle questions related to optimization. If the goal is to find *one* collision, why should the used method allow to find more than that? The new view gives an attacker the ability to exploit *all* available degrees of freedom.

The remainder of the paper is structured as follows. Subsequently we define some notation in Table 1. A short description of SHA-1 is given in Sect. 2. We tackle the core of the problem in Sect. 3, where we revisit the approach of finding collisions based on differential techniques. To do that, we generalize the concept of characteristics and introduce a new way to calculate the expected work to find a collision. Some examples are given there to illustrate the new concept. Based on that, in Sect. 4 we finally describe a way to automatically find the complex NL-characteristics needed. Also there we give examples which illustrate its behavior. As an application of the described technique, we give a two-block 64-step SHA-1 colliding message pair including all used characteristics in Sect. 5. Sect. 6 puts our contribution into the context of related and previous work. We conclude and survey future work in Sect. 7.

Table 1. Notation

notation	description
$X \oplus Y$	bit-wise XOR of X and Y
ΔX	difference with respect to XOR
$X + Y$	addition of X and Y modulo 2^{32}
δX	difference with respect to modular addition
X	arbitrary 32-bit word
x_i	value of the i-th bit
X^2	pair of words, shortcut for $(X,\, X^*)$
M_i	input message word i (32 bits)
W_i	expanded input message word t (32 bits)
$X \lll n$	bit-rotation of X by n positions to the left, $0 \leq n \leq 31$
$X \ggg n$	bit-rotation of X by n positions to the right, $0 \leq n \leq 31$
N	number of steps of the compression function

2 Short Introduction to SHA-1

SHA-1 [11], as most dedicated hash functions used today, is based on the design principles of MD4. First, the input message is padded and split into 512-bit message blocks. An 80-step compression function is then applied to each of these 512-bit message blocks. It has two types of inputs: a chaining input of 160 bits and a message input of 512 bits. Let $g(m, h)$ denote the compression function with message input m and chaining input h. The chaining input h_{n+1} for the next compression function is calculated by $h_n + g(m, h_n)$, called feed forward. The chaining variables for the first iteration are set to fixed values (referred to as IV). The result of the last call to the compression function is the hash of the message. The compression function basically consists of two parts: the message expansion and the state update transformation.

Message Expansion In SHA-1, the message expansion is defined as follows. The message is represented by 16 32-bit words, denoted by M_i, with $0 \leq i \leq 15$. In the message expansion, this input is expanded linearly into 80 32-bit words W_i. The expanded message words W_i are defined as follows:

$$W_i = \begin{cases} M_i, & \text{for } \ 0 \leq i \leq 15, \\ (W_{i-3} \oplus W_{i-8} \oplus W_{i-14} \oplus W_{i-16}) \lll 1 & \text{for } 16 \leq i \leq 79 \ . \end{cases} \quad (1)$$

State Update Transformation The state update transformation starts by copying the chaining input into the five 32-bit state variables $A, \ldots, E$, which are updated in 80 steps $(0, \ldots, 79)$ by using the word W_i and a round constant K_i in step i. A single step of the state update transformation is shown in Fig. 1. The function f in Fig. 1 depends on the step number: steps 0 to 19 (round 1)

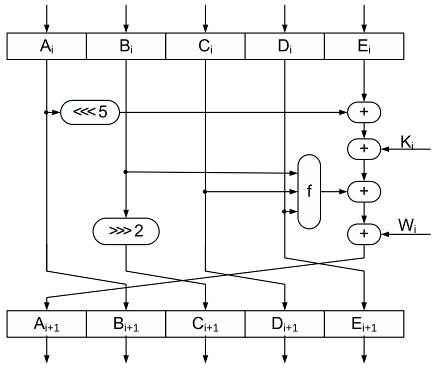


Fig. 1. One step of the state update transformation of SHA-1

use f_{IF} and steps 40 to 59 (round 3) use f_{MAJ}. The function f_{XOR} is applied in the remaining steps (round 2 and 4). The functions are defined as:

$$f_{IF}(B, C, D) = B \wedge C \oplus \overline{B} \wedge D \quad (2)$$

$$f_{MAJ}(B, C, D) = B \wedge C \oplus B \wedge D \oplus C \wedge D \quad (3)$$

$$f_{XOR}(B, C, D) = B \oplus C \oplus D \ . \quad (4)$$

Note that $B_i = A_{i-1}$, $C_i = A_{i-2} \ggg 2$, $D_i = A_{i-3} \ggg 2$, $E_i = A_{i-4} \ggg 2$. This also implies that the chaining inputs fill all A_j for $-4 \leq j \leq 0$. Thus it suffices to consider the state variable A, which we will for the remainder of this paper.

3 Collision Attacks Revisited

The objective of this paper is to develop a method to find SHA-1 characteristics which are suitable for collision attacks. However, in order to solve this problem, we first have to determine exactly what 'suitable' means in this context. In this section, we will therefore consider collision attacks and characteristics in a general setting, and analyze how the choice of the characteristic affects the work factor of the attack.

3.1 How Dedicated Collision Attacks Work

If we are given an n-bit hash function whose output values are uniformly distributed and use it to hash an arbitrary pair of messages, then we expect the hash values to collide with a probability of 2^{-n}. Hence, without knowing anything about the internals of the hash function, we should be able to find a collision after trying out 2^n pairs. Since any set of 2^n pairs will do, this approach can be turned into a birthday attack requiring only $2^{n/2}$ hash evaluations.

Instead of testing arbitrary pairs, dedicated collision attacks try to use the internal structure of the hash function to locate a special subset of message pairs which (1) are considerably more likely to collide than random pairs, and (2) can efficiently be enumerated. A particularly effective way to construct such subsets is to restrict the search space to message pairs with a fixed difference. The goal is to pick these differences in such a way that they are likely to propagate through the hash function following a predefined differential characteristic which eventually ends in a zero difference (a collision).

As was observed in [4], the probability for this to happen can be increased by restricting the subset even further and imposing conditions not only on the differences but also on the *values* of specific (expanded) message bits. Moreover, since the internal variables of a hash function only depend on the message (and not on a secret key as for example in block ciphers), we can also restrict the set of message pairs by imposing conditions on the state variables. Depending on their position, however, these conditions might have a considerable impact on the efficiency to enumerate the messages fulfilling them. This important point is analyzed in detail in Sect. 3.3.

3.2 Generalized Characteristics

In order to reflect the fact that both the differences and the actual values of bits play a role in their attack, Wang *et al.* already extended the notion of differential characteristics by adding a sign to each non-zero bit difference (1 or -1). In this paper we generalize this concept even further by allowing characteristics to impose arbitrary conditions on the values of pairs of bits.

The conditions imposed by such a generalized characteristic on a particular pair of words X^2 will be denoted by ∇X. It will turn out to be convenient to represent ∇X as a set, containing the values for which the conditions are satisfied, for example

$$\nabla X = \{X^2 \mid x_7 \cdot x_7^* = 0,\, x_i = x_i^* \text{ for } 2 \leq i < 6,\, x_1 \neq x_1^*, \text{ and } x_0 = x_0^* = 0\}\,.$$

In order to write this in a more compact way, we will use the notation listed in Table 2. Using this convention, we can rewrite the example above as

$$\nabla X = [\texttt{7?----x0}]\,.$$

Table 2. Possible conditions on a pair of bits

(x_i, x_i^*)	$(0,0)$	$(1,0)$	$(0,1)$	$(1,1)$
?	✓	✓	✓	✓
-	✓	-	-	✓
x	-	✓	✓	-
0	✓	-	-	-
u	-	✓	-	-
n	-	-	✓	-
1	-	-	-	✓
#	-	-	-	-
3	✓	✓	-	-
5	✓	-	✓	-
7	✓	✓	✓	-
A	-	✓	-	✓
B	✓	✓	-	✓
C	-	-	✓	✓
D	✓	-	✓	✓
E	-	✓	✓	✓

A generalized characteristic for SHA-1 is then simply a pair of sequences $\nabla A_{-4}, \ldots, \nabla A_N$ and $\nabla W_0, \ldots, \nabla W_{N-1}$.

3.3 Work Factor and Probabilities

Let us now assume that we are given a complete characteristic for SHA-1, specified by $\nabla A_{-4}, \ldots, \nabla A_N$ and $\nabla W_0, \ldots, \nabla W_{N-1}$. Our goal is to estimate how much effort it would take to find a pair of messages which follows this characteristic, assuming a simple depth-first search algorithm which tries to determine the pairs of message words M_i^2 one by one starting from M_0^2.

In order to estimate the work factor of this algorithm, we will compute the expected number of visited nodes in the search tree. But first we introduce some definitions.

Definition 1. *The* message freedom $F_W(i)$ *of a characteristic at step i is the number of ways to choose W_i^2 without violating any (linear) condition imposed on the expanded message, given fixed values W_j^2 for $0 \le j < i$.*

We note that since the expanded message in SHA-1 is completely determined by the first 16 words, we always have $F_W(i) = 1$ for $i \ge 16$.

Definition 2. *The* uncontrolled probability $P_u(i)$ *of a characteristic at step i is the probability that the output A_{i+1}^2 of step i follows the characteristic, given that all input pairs do as well, i.e.,*

$$P_u(i) = P\left(A_{i+1}^2 \in \nabla A_{i+1} \mid A_{i-j}^2 \in \nabla A_{i-j} \text{ for } 0 \le j < 5, \text{ and } W_i^2 \in \nabla W_i\right)\,.$$

Definition 3. *The* controlled probability $P_c(i)$ *of a characteristic at step i is the probability that there exists at least one pair of message words W_i^2 following the characteristic, such that the output A_{i+1}^2 of step i follows the characteristic, given that all other input pairs do as well, i.e.,*

$$P_c(i) = P\left(\exists W_i^2 \in \nabla W_i : A_{i+1}^2 \in \nabla A_{i+1} \mid A_{i-j}^2 \in \nabla A_{i-j} \text{ for } 0 \le j < 5\right) .$$

With the definitions above, we can now easily express the number of nodes $N_s(i)$ visited at each step of the compression function during the collision search. Taking into account that the average number of children of a node at step i is $F_W(i) \cdot P_u(i)$, that only a fraction $P_c(i)$ of the nodes at step i have any children at all, and that the search stops as soon as step N is reached, we can derive the following recursive relation:

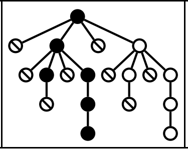
$$N_s(i) = \begin{cases} 1 & \text{if } i = N\,, \\ \max\left\{N_s(i+1) \cdot F_W(i)^{-1} \cdot P_u^{-1}(i),\, P_c^{-1}(i)\right\} & \text{if } i < N\,. \end{cases}$$

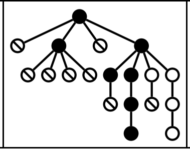
The total work factor is then given by

$$N_w = \sum_{i=1}^{N} N_s(i)\,.$$

In order to understand what the different quantities defined above represent, it might be helpful to walk through a small example. Table 3 shows two hypothetical search trees with corresponding values of F_W, P_u, and P_c. The nodes which are visited by the search algorithm, and hence contribute to the complexity of the collision search, are filled. Note that the values of $P_c(i)$ do not always influence the complexity of the attack. The trees in Table 3, however, are examples where they do.

Table 3. How P_c affects the search tree

i	tree[a]	F_W	$P_u(i)$	$P_c(i)$	$N_s(i)$
0:		4	1/2	1	1
1:		4	1/2	1	1
2:		1	1/2	1/2	2
3:		1	1	1	1
4:					1

i	tree	F_W	$P_u(i)$	$P_c(i)$	$N_s(i)$
0:		4	1/2	1	1
1:		4	1/2	1/2	2
2:		1	1/2	1/2	2
3:		1	1	1	1
4:					1

[a] Both ○ and ● represent values of W_{i-1}^2 which lead to a consistent A_i^2; the nodes visited by the search algorithm are filled. Inconsistent values are denoted by ⊘.

Let us now illustrate the previous concepts with two examples on 64-step SHA-1. In the first example, shown in Table 4, we consider a generalized characteristic which does not impose any conditions, except for a fixed IV value at the input of the compression function and a collision at the output. The values of $N_s(i)$ in the table tell us that the search algorithm is expected to traverse nearly the complete compression function 2^{160} times before finding a colliding pair (note that from here on all values listed in tables will be base 2 logarithms).

In the example of Table 5, we force the state variables and the expanded message words to follow a given differential characteristic starting from the output of the 16th step (i.e., $A_{16}, \ldots, E_{16}$). How such diffential characteriscs can be found will be explained in Sect. 4. The most significant effect is that the five consecutive uncontrolled probabilities of 2^{-32} in the previous example move up to steps 11–15, where their effect on the number of nodes is completely neutralized by the degrees of freedom in the expanded message, resulting in a considerable reduction of the total work factor.

The examples above clearly show that small probabilities have a much larger impact on the work factor when they occur after step 16 (where $F_W(i) = 1$). Therefore, when constructing characteristics, we will in the first place try to optimize the probabilities in the second part of the compression function (steps 16 to $N-1$), even if this comes at the cost of a significant decrease of probabilities in the first part.

4 Constructing Characteristics

Having the necessary tools to estimate the work factor corresponding to any given generalized characteristic, we now turn to the problem of finding characteristics which minimize this work factor.

The search method presented in this section constructs characteristics by iteratively adding more conditions as long as it improves the work factor. During this process, two important tasks need to be performed: (1) determining when and where to add which condition, and (2) letting conditions propagate and avoiding inconsistent conditions. We first discuss the second problem.

4.1 Consistency and Propagation of Conditions

When analyzing the interaction of bit conditions imposed at the inputs and the outputs of a single step of the state update transformation, three situations can occur: (1) the conditions are inconsistent, (2) the conditions are consistent, and (3) the conditions are consistent, provided that a number of additional bit conditions are fulfilled as well (the conditions are said to propagate). This third case is illustrated in Table 6, where the conditions imposed on the expanded message words in the previous example propagate to the state variables. It should be noted that such consistency checks can be implemented in a very efficient way, thanks to the fact that bits at different bit positions only interact through the carries of the integer additions.

4.2 Determining Which Conditions to Add

In Sect. 3.3 we noted that conditions in a characteristic affect the work factor in very different ways depending on the step where they are enforced. This is also

Table 4. Example 1, no conditions

i	∇A_i	∇W_i	F_W	$P_u(i)$	$P_c(i)$	$N_s(i)$
-4:	00001111010010111000011111000011					
-3:	01000000110010010101000111011000					
-2:	01100010111010110111001111111010					
-1:	11101111110011011010101110001001					
0:	01100111010001010010001100000001	????????????????????????????????	64	0.00	0.00	0.00
1:	????????????????????????????????	????????????????????????????????	64	0.00	0.00	0.00
	...	...				
12:	????????????????????????????????	????????????????????????????????	64	0.00	0.00	0.00
13:	????????????????????????????????	????????????????????????????????	64	0.00	0.00	0.00
14:	????????????????????????????????	????????????????????????????????	64	0.00	0.00	32.00
15:	????????????????????????????????	????????????????????????????????	64	0.00	0.00	96.00
16:	????????????????????????????????	????????????????????????????????	0	0.00	0.00	160.00
17:	????????????????????????????????	????????????????????????????????	0	0.00	0.00	160.00
	...	...				
59:	????????????????????????????????	????????????????????????????????	0	-32.00	0.00	160.00
60:	--------------------------------	????????????????????????????????	0	-32.00	0.00	128.00
61:	--------------------------------	????????????????????????????????	0	-32.00	0.00	96.00
62:	--------------------------------	????????????????????????????????	0	-32.00	0.00	64.00
63:	--------------------------------	????????????????????????????????	0	-32.00	0.00	32.00
64:	--------------------------------					

Table 5. Example 2, less message freedom, better work factor by specifying a suitable message difference

i	∇A_i	∇W_i	F_W	$P_u(i)$	$P_c(i)$	$N_s(i)$
	...					
0:	01100111010001010010001100000001	-xx-----------------------------	32	0.00	0.00	0.00
1:	????????????????????????????????	xxx-----------------------x-x-x-	32	0.00	0.00	0.00
	...	...				
7:	????????????????????????????????	-xx-----------------------xx--x-	32	0.00	0.00	0.00
8:	????????????????????????????????	-xx----------------------x----xx	32	0.00	0.00	5.00
9:	????????????????????????????????	--x----------------------x------	32	0.00	0.00	37.00
10:	????????????????????????????????	xxx----------------------x----x-	32	0.00	0.00	69.00
11:	????????????????????????????????	-xx---------------------------x-	32	-32.00	-29.00	101.00
12:	x-------------------------------	x------------------------------x	32	-32.00	-31.00	101.00
13:	x-------------------------------	--------------------------x-----	32	-32.00	-31.00	101.00
14:	--------------------------------	------------------------------xx	32	-32.00	-31.19	101.00
15:	x-----------------------------xx	-x-----------------------x-x--x-	32	-32.00	-27.83	101.00
16:	------------------------------x-	-x-----------------------x------	0	-7.00	-4.00	101.00
17:	x-----------------------------x-	xxx----------------------x-x--x-	0	-7.00	-2.00	94.00
18:	--------------------------------	x-x-----------------------------	0	-5.00	-3.00	87.00
19:	------------------------------x-	x------------------------x------	0	-4.00	-3.00	82.00
	...	...				
49:	------------------------------x-	-------------------------x------	0	-2.00	-1.00	7.00
50:	--------------------------------	x-----------------------------x-	0	-3.00	-2.00	5.00
51:	--------------------------------	--------------------------------	0	-1.00	-1.00	2.00
52:	--------------------------------	x-------------------------------	0	-1.00	-1.00	1.00
53:	--------------------------------	x-------------------------------	0	0.00	0.00	0.00
54:	--------------------------------	--------------------------------	0	0.00	0.00	0.00
	...	...				
60:	--------------------------------	--------------------------------	0	0.00	0.00	0.00
61:	--------------------------------	--------------------------------	0	0.00	0.00	0.00
62:	--------------------------------	--------------------------------	0	0.00	0.00	0.00
63:	--------------------------------	--------------------------------	0	0.00	0.00	0.00
64:	--------------------------------					

Table 6. Propagation of conditions in Example 2

i	∇A_i	∇W_i	F_W	$P_u(i)$	$P_c(i)$	$N_s(i)$
	...					
0:	`01100111010001010010001100000001`	`-xx-----------------------------`	32	0.00	0.00	0.00
1:	`??x-----------------------------`	`xxx-----------------------x-x-x-`	32	0.00	0.00	0.00
2:	`??????????????????????????????x-`	`--x----------------------x----xx`	32	0.00	0.00	0.00
3:	`????????????????????????????????`	`x-xx---------------------x------`	32	0.00	0.00	0.00
	...	...				

reflected in the procedure which we are about to propose: in order to determine where to add which conditions, we will proceed in a number of distinct stages, each of which tries to optimize a specific part of the characteristic.

Stage 1. As observed in Sect. 3.3, the work factor of the collision search algorithm is mainly determined by the shape of the characteristic after step 16. Hence, our first goal is to find a high probability differential characteristic, which can start with any difference in the state variables after step 16, but ends in a zero difference in the last step (later on, when we consider multi-block collisions, this constraint will be removed as well).

In general, the sparser a differential characteristic, the higher its probability, and in the case of the SHA family, it has been shown before that sparse characteristics can easily be found by linearizing all components of the state update transformation, representing the resulting compression function as a linear code, and searching for low-weight vectors (see [5, 12, 13, 20]).

Once a suitable differential characteristic is found for the linearized variant (called an L-characteristic), we determine the corresponding message difference, and impose it to our generalized characteristic. The differences in the state variables after step 16 are copied in the same way, except that we do not directly impose constraints to the most significant and the two least significant bits, but instead determine them by propagation. This will avoid inconsistencies caused in some cases by the nonlinear f-functions.

Stage 2. At this point, the largest part of the work factor is most likely concentrated in steps 12 to 16 (see e.g. Table 5), where the state variables, which are not constraint in any way in the previous steps, are suddenly forced to follow a fixed difference. In order to eliminate this bottleneck, we want to guide the state variables to the target difference by imposing conditions to the first steps as well.

Although the probability of this part of the characteristic is not as critical as before, we still want the differences to be reasonably sparse. Unfortunately, because of the high number of constraints (the message difference and both the differences at the input of the first step and at the output of step 16 are fixed already), suitable L-characteristics are extremely unlikely to exist in this case. In order to solve this problem, we will use a probabilistic algorithm which bears some resemblance to the algorithms used to find low-weight code words, but instead of feeding it with a linear code, we directly apply it to the unmodified (non-linear) compression function.

The basic idea of the algorithm is to randomly pick a bit position which is not restricted yet (i.e., a '`?`'-bit), impose a zero-difference at this position (a '`-`'-bit), and calculate how the condition propagates. This is repeated until all unrestricted bits have been eliminated, or until we run into an inconsistency, in which case we start again. The algorithm can be optimized in several ways, for example by also picking '`x`'-bits once they start to appear, guessing the sign of their differences ('`u`' or '`n`'), and backtracking if this does not lead to a solution. It turns out that inconsistencies are discovered considerably earlier this way.

An interesting property of the proposed procedure is that the sparser a characteristic, the higher the probability that it will be discovered. The number of trials before a consistent characteristic is found, is very hard to predict, though. Experiments show that this number can range from a few hundreds to several hundreds of thousands.

Stage 3. In the final stage, we try to further improve the work factor corresponding to the characteristic by performing local optimizations. To this end, we run through all bit positions of every state variable and every expanded message word, check which conditions can be added to improve the total work factor, and finally pick the position and corresponding condition which yields the largest gain. By repeating this many times, we can gradually improve the work factor. The example in Table 7 shows how our previous characteristic looks like after applying this greedy approach for a number of iterations.

An interesting issue here, is when to stop adding new conditions. In order to answer this question, we first notice that every additional condition reduces the size of the search tree, but at the same time lowers the expected number of surviving leaves at step N. In general, the work factor will improve as long as the search tree is reduced by a larger factor than the number of surviving leaves. At some point, however, the expected number of leaves will drop below one, meaning that message pairs which actually follow the characteristic are only expected to exist with a certain probability. This is not necessarily a problem if we are prepared to repeat the search for a number of different characteristics, and in fact, that is exactly how we constructed the second block of the 64-step collision presented in the next section. In this case, three different characteristics were used, the third of which is shown in Table 10 (notice that the expected number of characteristics needed to find one surviving leave can directly be read from $N_s(0)$, in this example $2^{1.24} \approx 3$). Coming back to our original question, we can conclude that we should in principle continue adding conditions as long as the gain in work factor justifies the cost of generating additional characteristics.

5 Applications

To illustrate our method, we give a characteristic for a two-block collision of SHA-1 reduced to 64 steps with the standard IV. Note that for different initial

Table 7. Example 3, after adding conditions to minimize workfactor

i	∇A_i	∇W_i	F_W	$P_u(i)$	$P_c(i)$	$N_s(i)$
	...					
0:	01100111010001010010001100000001	0uu01010110011010000111101110101	0	0.00	0.00	0.00
1:	n0n01010100000011010100000101000	unn00001000110100010110111u1u0n0	0	0.00	0.00	0.00
2:	00u1unnnnnnnnnnnnnnnnnnnnnnn01u0	00n11101001100111111111011n1011uu	0	0.00	0.00	0.00
3:	100010100110010010011 1u11100u111	n0un01100001101011001101 0u111100	0	0.00	0.00	0.00
4:	u000u01n11uu010u11u10100101010u0	un0n0110100100001000101 10n1u01uu	0	0.00	0.00	0.00
5:	n01001000n100011n1n000101uu0n010	uu1n101011111001110111 0110n000u0	0	0.00	0.00	0.00
6:	010100110n0101u00100001000001100	10n10000111111000000000000010011	0	0.00	0.00	0.00
7:	1011111unnnnnnnnnn100000nu101n10	1nu0100000010111001----001nu01u1	4	-1.00	0.00	0.00
8:	n1100110111000000101---00110nu00	0nu1101110111------------u0011nu	12	-8.00	0.00	0.00
9:	n01010010000111101110----n10111n	11u1100001111-----------0u100111	11	-0.13	0.00	0.00
10:	n011010010111-----------000000n0	nnn111101----------------n1010u0	16	-4.00	-0.68	0.68
11:	u0110101011-------------n1100100	1un1001-0-----------------0011u1	18	-6.00	-1.68	5.36
12:	u0010100101-------------0-110001	u10110-0-0----------------11000u	18	-11.00	-2.96	17.36
13:	u11100101110010----------0100000	0010010100000-------------u00101	13	-4.00	-2.42	24.36
14:	01110011011111-------------11000	1001000111111-------------1001uu	11	-3.00	-2.00	33.36
15:	u1010110101-1-------------1001uu	0n110--0-----------------n0n00n0	19	-10.14	-0.14	41.36
16:	1100011000000000-----------110n0	1u0100101000-------------u100100	0	0.00	0.00	50.22
17:	u000111011------------------11u1	unn11101000000-----------n0n10n1	0	-0.22	-0.21	50.22
18:	11101-----------------------1001	n1u0--1-----------------01100101	0	-1.00	-0.48	50.00
19:	--0--------------------------1u1	u00110-0-----------------n101011	0	-1.00	-0.54	49.00
20:	----0------------------------1--	10u00-1-1----------------011100n	0	0.00	0.00	48.00
21:	-------------------------------u	00n--0-------------------nu01010	0	0.00	0.00	48.00
22:	--------------------------------	n1000-0------------------010010u	0	-1.00	-1.00	48.00
	...	...				
60:	--------------------------------	------------------------0-------	0	0.00	0.00	0.00
61:	--------------------------------	-----------------------1-0------	0	0.00	0.00	0.00
62:	--------------------------------	------------------------1-1-----	0	0.00	0.00	0.00
63:	--------------------------------	-----------------------0--------	0	0.00	0.00	0.00
64:	--------------------------------					

chaining variables, different characteristics might be needed. This is in contrast to MD4 or MD5 where good characteristics are possible without having conditions on the chaining variables. In addition to the characteristic, we also give a message pair which follows the described characteristic and collides. Note that, to the best of our knowledge, not a single second block characteristics for SHA-0 or SHA-1 has been presented so far, neither in the literature nor in informal public talks. Hence the example we give is the first of its kind. Additionally, it is a collision for SHA-1 with the highest number of steps published so far (previously known collisions covered up to 58 steps).

5.1 On the Choice of the Message Difference

The choice of the message difference determines the high-probability characteristics L_1 that is followed in the later part of the compression function. This is illustrated in Fig. 2. In a first step, only '`-`' and '`x`' conditions are needed, *i. e.* we only allow XOR-differences. The signs of the differences as well as some values of bits are determined in a later stage of the attack.

As previous work shows [5, 12, 13, 20], it turns out that interleaving so-called local collisions (a disturbing and a set of correcting differences) is the best way to construct these high-probability characteristics in the case of SHA-1. It turns out that these characteristics are L-characteristics. In order to allow for a small work factor, we do not put restrictions on the output difference of the compression function. Thus, δh_1 will be nonzero. Good L-characteristics for variants of SHA-1 with other than 80 steps are usually shifted versions of each other. These effects have also been considered in previous work, thus we do not expand on this issue here. In order to turn such high probability characteristics, which actually describe a pseudo-near-collision, into a collision-producing characteristic, NL-characteristics are needed. As illustrated in Fig. 2, a first NL-characteristic (NL_1) is needed to connect from a zero-difference in the chaining variables to L_1. After the feed-forward of the first block, we expect to have a modular difference $+d$ in the chaining variables.

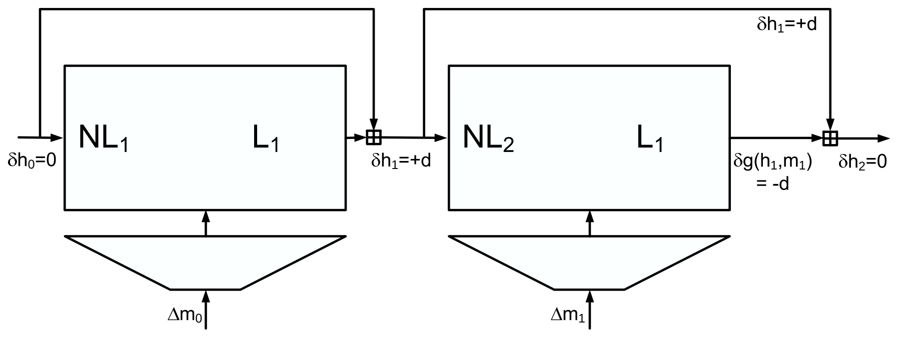


Fig. 2. Two-block approach to produce collisions

However, this difference does not fit to the difference needed to directly connect to the same L-characteristic used in the first block. Regardless of that, we want to follow this L-characteristics in the second block again (with the exception of different signs for some differences). The reason is that we want to cancel out the expected low-weight difference after the last step of the second block with the difference that is fed forward. We require

$$\delta g(h_1, m_1) + \delta h_1 = 0.$$

Thus, a new NL-characteristic (NL_2) for the second block is needed, taking into account the difference between δh_0 and δh_1 and the actual values at the chaining input of the second block. Note that with the ability to find these general NL-characteristics NL_1 and NL_2, collision-producing characteristics covering more than two blocks do not improve the work factor.

In [20, 22], examples for NL-characteristics are given which connect to a previously selected L-characteristic in the first block. It is commonly assumed that finding these NL-characteristics was based on experience and intuition, and done manually. Based on Sect. 3 and 4, we describe in the following an application for the *automatical* search for suitable NL-characteristics, which succeeds for the first and the second block.

5.2 A Two-block Collision for 64-step SHA-1

Herein we present a collision for 64-step SHA-1 using two message blocks. Table 9 and 10 detail the used characteristic for the first block and the second block respectively (see Sect. 3.2 for an explanation of the notation). Using our current (unoptimized) methods, we have an expected work factor of about 2^{35} compression function evaluations to find it. This compares favorably to the estimate of 2^{36} given in [20].

The number of nodes in the tree visited in the search, N_w, is given as the sum of all N_s in Tables 9 and 10. N_w relates to the expected work factor in the following way. We measured the cost of visiting a node in the search tree to be about 2^{-5} compression function evaluations. For that, we used as a means of comparison the SHA-1 implementation of OpenSSL 0.9.7g, which can do about 2^{19} compression functions per second on our PC. Note that the work factor for both blocks is lower than estimated. The reason is that carry differences in the last steps can be ignored and that the characteristic of the second block can be adjusted to allow additional deviations in the last steps of the first block.

Table 8. Example of a 64-step collision using the standard IV

i	Message 1 (m_0), first block	Message 1 (m_1), second block
1–4	63DAEFDD 30A0D167 52EDCDA4 90012F5F	3B2AB4E1 AAD112EF 669C9BAE 5DEA4D14
5–8	0DB4DFB5 E5A3F9AB AE66EE56 12A5663F	1DBE220E AB46A5E0 96E2D937 F3E58B63
9–12	D0320F85 8505C67C 756336DA DFFF4DB9	BE594F1C BD63F044 50C42AA5 8B793546
13–16	596D6A95 0855F129 429A41B3 ED5AE1CD	A9B24128 816FD53A D1B663DC B615DD01
i	Message 2 (m_0^*), first block	Message 2 (m_1^*), second block
1–4	63DAEFDE 70A0D135 12EDCDE4 70012F0D	3B2AB4E2 EAD112BD 269C9BEE BDEA4D46
5–8	ADB4DFB5 65A3F9EB 8E66EE57 32A5665F	BDBE220E 2B46A5A0 B6E2D936 D3E58B03
9–12	50320F84 C505C63E B5633699 9FFF4D9B	3E594F1D FD63F006 90C42AE6 CB793564
13–16	596D6A96 4855F16B 829A41F0 2D5AE1EF	A9B2412B C16FD578 11B6639F 7615DD23
i	XOR-difference are the same for both blocks	
1–4	00000003 40000052 40000040 E0000052	00000003 40000052 40000040 E0000052
5–8	A0000000 80000040 20000001 20000060	A0000000 80000040 20000001 20000060
9–12	80000001 40000042 C0000043 40000022	80000001 40000042 C0000043 40000022
13–16	00000003 40000042 C0000043 C0000022	00000003 40000042 C0000043 C0000022
i	The colliding hash values	
1–5	A750337B 55FFFDBB C08DB36C 0C6CFD97	A12EFFE0

In Table 8, we give the colliding messages. Note that we do not consider padding rules in our example, which would simply mean adding a common block to both messages after the collision. At this point we stress that this example serves as a proof of concept for the unified approach to searching for complex characteristics and optimizing the characteristic for the final search phase. Hence it does not rule out other, probably more efficient ways to speed up the search for colliding pairs using the given characteristic.

6 Comparison with Previous Work

In order to put our contribution into perspective, we compare it with related previous work.

On finding suitable characteristics. In 1998, the pioneering work of Chabaud and Joux [4] resulted in a collision-search attack on an earlier version of SHA-1 (termed SHA-0). Their attack is based on L-characteristics they found. The Hamming weight of these characteristics (or a part of them) was used as a rough estimate of the attack complexity. However, the details depend on the positions of all differences. For each difference, the sign, the step in which it occurs, the bit-position within the word as well as its relative position to neighboring differences influence its impact on the attack complexity. A general and practical way to calculate this impact was described in Sect. 3.3.

In 2005, Rijmen and Oswald reported an attack on step-reduced SHA-1 [13], which is based on L-characteristics as well. Also the complexity of a collision search on SHA-0 was improved by Biham and Chen using the neutral-bit technique [1], and by Biham *et al.* using a multi-block approach [2]. Note that the attack on SHA-0 [2] employed four message blocks. Using the presented method of automatically finding complex characteristics, we eliminate the need for more than two blocks for an efficient collision-search attack.

Recent results of Wang *et al.* [20, 22] describe further major improvements. By employing the multi-block technique as described in Sect. 5.1, together with the ability to manually find NL-characteristics, attack costs are improved by many orders of magnitude. As shown in Sect. 5, our method can be used to automatically reach the same goal. This also answers the question left open in [16]. Since the NL-characteristic for the second block (NL_2) depends on the chosen message pair for the first block, this also prevents a manual search for new characteristics in the middle of a collision search.

The only related work which also aims for automatic search for complex characteristics is by Schläffer and Oswald [14] on MD4. Their method is very different from ours. It assumes a fixed differential behavior of the function f and limits carry extensions to only a few bit positions to reduce the search space. Thus it is not easy to extend it to more complex hash functions since these restrictions are too strict. Our method is not restricting anything, but is still practical.

On the cost of the final search. In previous work, the cost of the attack is further improved by a technique called message modification. The ideas developed in Sect. 3 and 4 can also be used for similar improvements. Both the originally published results by Wang *et al.* [20] as well as work by Sugita *et al.* [16] give rough estimates for the cost of message modification: 2^1 and 2^2 compression function evaluations(c_g), respectively. Sugita *et al.* also give a different trade-off. By using Gröbnerbasis-methods they reduce the number of trials significantly at the cost of increased message modification costs. Overall, this method does not lead to improvements in practice.

Note that for the recently announced but to the best of the authors knowledge unpublished improvements of the complexity of the collision search for full SHA-1 [18] (from 2^{69} to 2^{63}), no message modification costs are given, thus we lack comparability here.

Our approach can be seen as a trade-off towards very fast trials without the overhead of expensive message modification. As mentioned in Sect. 5.2, the cost of visiting one node in our search is only in the order of $2^{-5}c_g$. Note that the neutral-bit technique [1, 2] can also be seen as a trade-off in this direction. However, as reported in [1], only a small fraction (one out of eight in the simpler case of SHA-0) of the trials conforms to a previously selected characteristic. Comparing the neutral-bit technique to our method, we observe two differences. Firstly, instead of a small fraction, we can be sure that every trial will conform to the characteristic we select. Secondly we don't rely on randomly generating message pairs which conform to a previously selected characteristic to bootstrap the final search. Instead we can exploit the available degrees of freedom in a sensible way.

On exploiting degrees of freedom. In Sect. 3.3, we described a method to calculate the expected number of collisions given a particular characteristic. Thus we can make a sensible use of degrees of freedom up to the point where we expect to find only one suitable message pair. In fact, also this distinguishes our approach from all previous work.

7 Conclusions and Future Work

We described, for the first time, a computer-implementable method to search for complex characteristics as needed in the effective cryptanalysis of hash functions of the MD4 family like SHA-1. As a proof of concept, we gave the characteristics needed for a 64-step two-block collision of SHA-1. Furthermore, for the first time an actual collision for 64-step SHA-1 is produced, with an expected work factor of 2^{35} compression function computations.

We also tackled issues like work factors or degrees of freedom and put them into a precise framework. Thus an optimal exploitation of available degrees of freedom gets possible for goals like fast collision search.

Future work includes optimization of the found characteristics for different final search strategies, or the application of the described technique to other hash functions. Given the increased design complexity of members of the SHA-2 family compared to SHA-1, an automatic approach as described in our article seems to be highly beneficial for the analysis of these hash functions.

Given the ability to automatically incorporate some differences from the chaining variables at the start of the compression function, applications such as meaningful collisions or speeding up techniques like herding attacks [6] are also future work.

Acknowledgements

We thank Florian Mendel and Vincent Rijmen for many insightful discussions, and the anonymous reviewers for their helpful comments. The work described in this paper has been supported by the European Commission through the

IST Programme under Contract IST-2002-507932 ECRYPT. The first author is supported by the Austrian Science Fund (FWF) project P18138.

Disclaimer

The information in this document reflects only the authors' views, is provided as is and no guarantee or warranty is given that the information is fit for any particular purpose. The user thereof uses the information at its sole risk and liability.

References

1. E. Biham and R. Chen. Near-Collisions of SHA-0. In M. K. Franklin, editor, *Advances in Cryptology - CRYPTO 2004, 24th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 2004, Proceedings*, volume 3152 of *LNCS*, pages 290–305. Springer, 2004.
2. E. Biham, R. Chen, A. Joux, P. Carribault, C. Lemuet, and W. Jalby. Collisions of SHA-0 and Reduced SHA-1. In R. Cramer, editor, *Advances in Cryptology - EUROCRYPT 2005: 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005. Proceedings*, volume 3494 of *LNCS*, pages 36–57. Springer, 2005.
3. J. Black, M. Cochran, and T. Highland. A Study of the MD5 Attacks: Insights and Improvements. In M. Robshaw, editor, *Proceedings of Fast Software Encryption - FSE 2006, Graz, Austria, March 15-17, 2006*, volume 4047 of *LNCS*, 2006. To appear.
4. F. Chabaud and A. Joux. Differential Collisions in SHA-0. In H. Krawczyk, editor, *Advances in Cryptology - CRYPTO '98, 18th Annual International Cryptology Conference, Santa Barbara, California, USA, August 23-27, 1998, Proceedings*, volume 1462 of *LNCS*, pages 56–71. Springer, 1998.
5. C. S. Jutla and A. C. Patthak. Provably Good Codes for Hash Function Design. In R. Cramer, editor, *Proceedings of SAC 2006*, LNCS. Springer, 2006. to appear.
6. J. Kelsey and T. Kohno. Herding Hash Functions and the Nostradamus Attack. In S. Vaudenay, editor, *Advances in Cryptology - EUROCRYPT 2006, 25th Annual International Conference on the Theory and Applications of Cryptographic Techniques, St. Petersburg, Russia, May 28 - June 1, 2006, Proceedings*, volume 4004 of *LNCS*, pages 183–200. Springer, 2005.
7. V. Klima. Tunnels in Hash Functions: MD5 Collisions Within a Minute. Cryptology ePrint Archive, Report 2006/105, 2006. `http://eprint.iacr.org/`.
8. J. Liang and X. Lai. Improved Collision Attack on Hash Function MD5. Cryptology ePrint Archive, Report 2005/425, 2005. `http://eprint.iacr.org/`.
9. Y. Naito, Y. Sasaki, N. Kunihiro, and K. Ohta. Improved Collision Attack on MD4. Cryptology ePrint Archive, Report 2005/151, 2005. `http://eprint.iacr.org/`.
10. Y. Naito, Y. Sasaki, T. Shimoyama, J. Yajima, N. Kunihiro, and K. Ohta. Message Modification for Step 21-23 on SHA-0. Cryptology ePrint Archive, Report 2006/016, 2006. `http://eprint.iacr.org/`.
11. National Institute of Standards and Technology (NIST). FIPS-180-2: Secure Hash Standard, August 2002. Available online at `http://www.itl.nist.gov/fipspubs/`.

12. N. Pramstaller, C. Rechberger, and V. Rijmen. Exploiting Coding Theory for Collision Attacks on SHA-1. In N. P. Smart, editor, *Cryptography and Coding, 10th IMA International Conference, Cirencester, UK, December 19-21, 2005, Proceedings*, volume 3796 of *LNCS*, pages 78–95. Springer, 2005.
13. V. Rijmen and E. Oswald. Update on SHA-1. In A. Menezes, editor, *Topics in Cryptology - CT-RSA 2005, The Cryptographers' Track at the RSA Conference 2005, San Francisco, CA, USA, February 14-18, 2005, Proceedings*, volume 3376 of *LNCS*, pages 58–71. Springer, 2005.
14. M. Schläffer and E. Oswald. Searching for Differential Paths in MD4. In M. Robshaw, editor, *Proceedings of Fast Software Encryption - FSE 2006, Graz, Austria, March 15-17, 2006*, volume 4047 of *LNCS*, 2006. To appear.
15. M. Stevens. Fast Collision Attack on MD5. Cryptology ePrint Archive, Report 2006/104, 2006. `http://eprint.iacr.org/`.
16. M. Sugita, M. Kawazoe, and H. Imai. Gröbner Basis Based Cryptanalysis of SHA-1. Cryptology ePrint Archive, Report 2006/098, 2006. `http://eprint.iacr.org/`.
17. X. Wang, X. Lai, D. Feng, H. Chen, and X. Yu. Cryptanalysis of the Hash Functions MD4 and RIPEMD. In R. Cramer, editor, *Advances in Cryptology - EUROCRYPT 2005: 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005. Proceedings*, volume 3494 of *LNCS*, pages 1–18. Springer, 2005.
18. X. Wang, A. Yao, and F. Yao. Cryptanalysis of SHA-1. Presented at the Cryptographic Hash Workshop hosted by NIST, October 2005.
19. X. Wang, A. Yao, and F. Yao. New Collision Search for SHA-1, August 2005. Presented at rump session of CRYPTO 2005.
20. X. Wang, Y. L. Yin, and H. Yu. Finding Collisions in the Full SHA-1. In V. Shoup, editor, *Advances in Cryptology - CRYPTO 2005, 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings*, volume 3621 of *LNCS*, pages 17–36. Springer, 2005.
21. X. Wang and H. Yu. How to Break MD5 and Other Hash Functions. In R. Cramer, editor, *Advances in Cryptology - EUROCRYPT 2005: 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005. Proceedings*, volume 3494 of *LNCS*, pages 19–35. Springer, 2005.
22. X. Wang, H. Yu, and Y. L. Yin. Efficient Collision Search Attacks on SHA-0. In V. Shoup, editor, *Advances in Cryptology - CRYPTO 2005, 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings*, volume 3621 of *LNCS*, pages 1–16. Springer, 2005.
23. Y. L. Yin. Personal Communication, March 2006.

Table 9. Characteristic used for the first block of the 64-step collision

i	∇A_i	∇W_i	F_W	$P_u(i)$	$P_c(i)$	$N_s(i)$
-4:	00001111010010111000011111000011					
-3:	01000000110010010101000111011000					
-2:	01100010111010110111001111111010					
-1:	11101111110011011010101110001001					
0:	01100111010001010010001100000001	011000111101101011101111110111nu	0	0.00	0.00	1.07
1:	0000001110001111100010001001000n	0n1100001010000011010-010u1n01u1	1	0.00	0.00	1.07
2:	0n00100101000010110-00011u0un0	0u01001011101101----11011n100100	4	-3.00	0.00	2.07
3:	1u10100001110010100-1un110nuu110	unn10000000000-1001----10u0u11u1	5	-4.00	0.00	3.07
4:	1un0010110011110un1100-0n1n11nu1	n0n01101101101001-01111-10110101	2	-2.00	0.00	4.07
5:	n1u10110101un00010nu10u111000010	u1100101101000111111----1n101011	4	-4.00	0.00	4.07
6:	100u100u01111nu00u1110nu111u1un1	10u01110011001101-1------101011n	7	-5.00	0.00	4.07
7:	nn1100101n1101011-1111-11u1001u0	00n100101010-101------100nu11111	7	-5.00	0.00	6.07
8:	01110111001100u00010--0n11110u11	u1010000001100---00---11-000010u	7	-6.00	0.00	8.07
9:	1n1u000101uuuu0uu1110-1010n110n0	1n00010100000101-100--10-u1111n0	4	-3.00	0.00	9.07
10:	1011000101n11111n111u-01n00un100	nu1101010110001--011----1u0110un	6	-5.00	0.00	10.07
11:	nnnnnnnnnnnnnnnnnnnnnnn-nnnnn0n1	1u01111111111111---------0u110n1	9	-9.00	0.00	11.07
12:	00110100000011110110000110011000	010110010110110101101---1-0101nu	4	-3.00	0.00	11.07
13:	01000000000010000000111 00-011000	0n001000010101-----------n1010n1	11	-4.00	0.00	12.07
14:	10011000100011000-0------0110101	nu00001010011-----------1n1100uu	11	-2.00	0.00	19.07
15:	1101101011111--1----------00010n	uu101101010-1-1--------1-1n011n1	11	-0.07	0.00	28.07
16:	11111100------------------0-0111	1101001010100-----------1010101u	0	-1.00	-1.00	39.00
17:	0000----------------------1-1111	1u0011100111------------111011u0	0	-1.00	-0.99	38.00
18:	----0-----------------------01u-	un00111011-0-0----------0n0011nu	0	0.00	0.00	37.00
19:	-------------------------------n	1u1100011111------------1un011n0	0	0.00	0.00	37.00
20:	--------------------------------	n1101001100--------------011000n	0	-1.00	-1.00	37.00
21:	------------------------------n-	1u1000110-1-0-----------0u1000n0	0	-2.00	-2.00	36.00
22:	------------------------------n-	1n011010011-------------0u0110n1	0	-2.00	-2.00	34.00
23:	--------------------------------	0n10011011--------------011111n0	0	-1.00	-1.00	32.00
24:	--------------------------------	00101001-0-0------------001010n1	0	-1.00	-1.00	31.00
25:	------------------------------n-	0001110111--------------1u100100	0	0.00	0.00	30.00
26:	--------------------------------	n00010000--------------0-11111n1	0	-1.00	-1.00	30.00
27:	--------------------------------	n001111-1-1-------------11101010	0	0.00	0.00	29.00
28:	--------------------------------	u10111110----------------11001n0	0	-1.00	-1.00	29.00
29:	------------------------------n-	n0011100----------------1u110010	0	0.00	0.00	28.00
30:	--------------------------------	001010-1-1-------------101010110	0	-2.00	-2.00	28.00
31:	------------------------------n-	u0110101----------------0u110111	0	0.00	0.00	26.00
32:	--------------------------------	u101001----------------011111010	0	-2.00	-2.00	26.00
33:	------------------------------u-	00010-1-0-------------110n100000	0	0.00	0.00	24.00
34:	--------------------------------	u011010----------------001101110	0	-2.00	-2.00	24.00
35:	------------------------------n-	101111----------------010u111001	0	0.00	0.00	22.00
36:	--------------------------------	n111-1-1---------------1010110u0	0	-1.00	-1.00	22.00
37:	--------------------------------	110110-----------------100000000	0	0.00	0.00	21.00
38:	--------------------------------	n1001------------------010111110	0	0.00	0.00	21.00
39:	--------------------------------	u11-0-1----------------101101011	0	0.00	0.00	21.00
40:	--------------------------------	01010-------------------01011100	0	0.00	0.00	21.00
41:	--------------------------------	1011-------------------100100000	0	0.00	0.00	21.00
42:	--------------------------------	00-0-0-----------------100111001	0	0.00	0.00	21.00
43:	--------------------------------	1101-------------------001111011	0	0.00	0.00	21.00
44:	--------------------------------	011---------------------10010000	0	0.00	0.00	21.00
45:	--------------------------------	1-1-0------------------101111000	0	0.00	0.00	21.00
46:	--------------------------------	110-------------------1011010010	0	0.00	0.00	21.00
47:	--------------------------------	01---------------------101011000	0	0.00	0.00	21.00
48:	--------------------------------	-0-0-------------------101100001	0	0.00	0.00	21.00
49:	--------------------------------	10--------------------1101010111	0	0.00	0.00	21.00
50:	--------------------------------	0---------------------1010101n11	0	-1.00	-1.00	21.00
51:	-----------------------------n--	0-1-------------------10u100011-	0	0.00	0.00	20.00
52:	--------------------------------	1----------------------001000u11	0	-1.00	-1.00	20.00
53:	--------------------------------	----------------------110111n00u	0	-2.00	-2.00	19.00
54:	----------------------------n---	-1---------------------u111011-u	0	-1.00	-1.00	17.00
55:	--------------------------------	----------------------101010u00u	0	-1.00	-1.00	16.00
56:	--------------------------------	-----------------------0111n10u-	0	-2.00	-1.91	15.00
57:	---------------------------n----	0---------------------u111000-u-	0	-1.00	-1.00	13.00
58:	--------------------------------	---------------------0-1010un1u-	0	-2.00	-1.83	12.00
59:	----------------------------u---	----------------------1n01n11u--	0	-2.00	-1.87	10.00
60:	--------------------------n-----	---------------------u-11000xu-0	0	-2.00	-1.00	8.00
61:	--------------------------------	----------------------0000n01ux-	0	-2.00	-1.00	6.00
62:	--------------------------------	---------------------1000n00n-x-	0	-3.00	-1.89	4.00
63:	-------------------------n------	--------------------u-10010-n-n-	0	-1.00	-1.00	1.00
64:	--------------------------------					

Table 10. Third characteristic used for the second block of the 64-step collision

i	∇A_i	∇W_i	F_W	$P_u(i)$	$P_c(i)$	$N_s(i)$
-4:	11110011111100010000010000n10011					
-3:	01101110111000001010001110011101					
-2:	11001011101100100011110111000100					
-1:	1001011011110100100111001n110101					
0:	10100000000111101110010101101000	0011101100101010101101-0111000nu	1	0.00	0.00	1.24
1:	1111001001110010110010-10000n1nu	1n1010101101000---0100101u1n11u1	3	-3.00	0.00	2.24
2:	uu10001001100001000001nu01un01u0	0u1001101001110010011--11n101110	2	-2.00	0.00	2.24
3:	0u10010110111100000nnnn01011u1nn	nun1110111101010010011010n0u01n0	0	0.00	0.00	2.24
4:	0nu1110110110n0010uuuu0uunuu1u10	n0n11101101111100010001000001110	0	0.00	0.00	2.24
5:	1000111nu111u0001n11111100100001	u01010110100011010-00101-u100000	2	-1.00	0.00	2.24
6:	u11110un101n0u0111-1011n010u1010	10n1011011100-10110----10011011u	5	-2.00	0.00	3.24
7:	u001u11nn0101011100n---0u011n111	11u10011111001----00-0-10uu00011	6	-4.00	0.00	6.24
8:	1n010101001u01n10000-0-11000u011	u0111110010110----0----1-001110n	9	-7.00	0.00	8.24
9:	01001u1n10100110100101-1-uu10100	1n11110101100------------u0001n0	12	-10.00	0.00	10.24
10:	uuuuuuuuuuuuuuuuuuuuuu--1100u011	nu01000011000100---------n1001nu	9	-8.00	0.00	12.24
11:	0100111011111100011111un-0111100	1n00101101111001001------1n001u0	6	-6.00	0.00	13.24
12:	1100000010111111111111111111u110	101010011011001001000---001010nn	3	-2.00	-1.00	13.24
13:	0110000101111111111111--0110110n	1n000001011011111--------n1110u0	8	-2.24	0.00	14.24
14:	0101111110011010110--------010u0	uu01000110110-----------1u0-11nn	12	-4.00	0.00	20.00
15:	01010010010000010-----------00nu	un110110000-0-0--------1-0n000n1	11	-1.00	0.00	28.00
16:	001001001011---------------10010	1100010100000-----------1101001n	0	0.00	0.00	38.00
17:	100000----------------------1000	0n1101111101------------11-001u1	0	-1.00	-0.99	38.00
18:	----0------------------------0u1	nn11101111-0-1----------0n0010nu	0	0.00	0.00	37.00
19:	-------------------------------n	0u1100011010------------1un000n1	0	-1.00	-1.00	37.00
20:	-0------------------------------	n0101010011------------11-10110n	0	-1.00	-1.00	36.00
21:	------------------------------n-	1u0001000-0-0-----------0u1000n1	0	-1.00	-1.00	35.00
22:	------------------------------n-	0n010001010-------------0u-011n1	0	-2.00	-2.00	34.00
23:	--------------------------------	1n10010111---------------00101n1	0	-1.00	-1.00	32.00
24:	--------------------------------	11011111-0-1------------000101n1	0	-1.00	-1.00	31.00
25:	------------------------------n-	0010000100--------------0u010000	0	0.00	0.00	30.00
26:	--------------------------------	u10011101---------------001000u0	0	-1.00	-1.00	30.00
27:	--------------------------------	n100100-0-0-------------01010001	0	0.00	0.00	29.00
28:	--------------------------------	u11001101--------------0-0-100n0	0	-1.00	-1.00	29.00
29:	------------------------------n-	n1111011----------------1u110000	0	0.00	0.00	28.00
30:	--------------------------------	100110-1-1-------------00--00100	0	-2.00	-2.00	28.00
31:	------------------------------u-	u0000101----------------1n000111	0	0.00	0.00	26.00
32:	--------------------------------	u011010---------------0001111100	0	-2.00	-2.00	26.00
33:	------------------------------n-	11111-0-0--------------0-u100101	0	0.00	0.00	24.00
34:	--------------------------------	u011010----------------0-0000000	0	-2.00	-2.00	24.00
35:	------------------------------u-	100100----------------010n011010	0	0.00	0.00	22.00
36:	--------------------------------	n100-0-1-------------0-1-11010n0	0	-1.00	-1.00	22.00
37:	--------------------------------	010111-----------------100001001	0	0.00	0.00	21.00
38:	--------------------------------	u0001------------------0-0001101	0	0.00	0.00	21.00
39:	--------------------------------	u00-0-0----------------101010100	0	0.00	0.00	21.00
40:	--------------------------------	11110------------------010000101	0	0.00	0.00	21.00
41:	--------------------------------	0011-------------------011010010	0	0.00	0.00	21.00
42:	--------------------------------	00-1-0-----------------01-001100	0	0.00	0.00	21.00
43:	--------------------------------	1010-------------------001111100	0	0.00	0.00	21.00
44:	--------------------------------	000-------------------11-0011100	0	0.00	0.00	21.00
45:	--------------------------------	0-1-0------------------1-1100101	0	0.00	0.00	21.00
46:	--------------------------------	000-------------------0---010010	0	0.00	0.00	21.00
47:	--------------------------------	11---------------------001010101	0	0.00	0.00	21.00
48:	--------------------------------	-0-0------------------1100111001	0	0.00	0.00	21.00
49:	--------------------------------	10---------------------0-1111110	0	0.00	0.00	21.00
50:	--------------------------------	0---------------------11-1100n10	0	-1.00	-1.00	21.00
51:	-----------------------------n--	1-0-------------------10u010110-	0	0.00	0.00	20.00
52:	--------------------------------	1---------------------0000001u10	0	-1.00	-1.00	20.00
53:	--------------------------------	----------------------011011n10u	0	-2.00	-2.00	19.00
54:	----------------------------n---	-1---------------------u-11011-u	0	-1.00	-1.00	17.00
55:	--------------------------------	----------------------111011u01u	0	-1.00	-1.00	16.00
56:	--------------------------------	----------------------01-00n10u-	0	-2.00	-1.91	15.00
57:	---------------------------n----	1---------------------u101111-u-	0	-1.00	-1.00	13.00
58:	--------------------------------	----------------------10-00un0u-	0	-2.00	-1.83	12.00
59:	----------------------------u---	----------------------0n01u11u--	0	-2.00	-1.87	10.00
60:	--------------------------u-----	---------------------n-0-111xu-0	0	-2.00	-1.00	8.00
61:	--------------------------------	----------------------0100u01ux-	0	-2.00	-1.00	6.00
62:	--------------------------------	-----------------------0-u11n-x-	0	-3.00	-1.89	4.00
63:	-------------------------u------	--------------------n-10110-u-n-	0	-1.00	-1.00	1.00
64:	--------------------------------					