Collisions on SHA-0 in one hour

Stéphane Manuel[1] and Thomas Peyrin[2,3,4]

[1] INRIA
stephane.manuel@inria.fr
[2] Orange Labs
thomas.peyrin@orange-ftgroup.com
[3] AIST
[4] Université de Versailles Saint-Quentin-en-Yvelines

Abstract. At Crypto 2007, Joux and Peyrin showed that the *boomerang attack*, a classical tool in block cipher cryptanalysis, can also be very useful when analyzing hash functions. They applied their new theoretical results to SHA-1 and provided new improvements for the cryptanalysis of this algorithm. In this paper, we concentrate on the case of SHA-0. First, we show that the previous perturbation vectors used in all known attacks are not optimal and we provide a new 2-block one. The problem of the possible existence of message modifications for this vector is tackled by the utilization of auxiliary differentials from the boomerang attack, relatively simple to use. Finally, we are able to produce the best collision attack against SHA-0 so far, with a measured complexity of $2^{33,6}$ hash function calls. Finding one collision for SHA-0 takes us approximatively one hour of computation on an average PC.

Key words: hash functions, SHA-0, boomerang attack.

1 Introduction

Cryptographic hash functions are an important tool in cryptography. Basically, a cryptographic hash function H takes an input of variable size and returns a hash value of fixed length while satisfying the properties of preimage resistance, second preimage resistance, and collision resistance [11]. For a secure hash function that gives an n-bit output, compromising these properties should require 2^n, 2^n, and $2^{n/2}$ operations respectively.

Usually, hash functions are built upon two components: a *compression function* and a *domain extension algorithm*. The former has the same security requirements that a hash function but takes fixed length inputs. The latter defines how to use the compression function in order to handle arbitrary length inputs. From the early beginning of hash functions in cryptography, designers relied on the pioneering work of Merkle and Damgård [8, 17] concerning the domain extension algorithm. Given a collision resistant compression function, it became easy to build a collision resistant hash function. However, it has been recently shown that this iterative process presents flaws [9, 12, 13, 15] and some new algorithms [1, 4] with better security properties have been proposed. One can distinguish

The first author is supported in part by the French Agence Nationale de la Recherche under the project designation EDHASH, DPg/ANR-CI FA/VB 2007-010. The second author is supported by the Japan Society for Promotion of Science and the French RNRT SAPHIR project (http://www.crypto-hash.fr).

three different methods for compression function designs: block cipher based, related to a well studied hard problem and from scratch.

The most famous design principle for dedicated hash functions is indisputably the `MD-SHA` family, firstly introduced by R. Rivest with `MD4` [24] in 1990 and its improved version `MD5` [23] in 1991. Two years after, the NIST publishes [19] a very similar hash function, `SHA-0`, that will be patched [20] in 1995 to give birth to `SHA-1`. This family is still very active, as NIST recently proposed [21] a 256-bit new version `SHA-256` in order to anticipate the potential cryptanalysis results and also to increase its security with regard to the fast growth of the computation power. All those hash functions use the Merkle-Damgård extension domain and their compression function, even if considered conceived from scratch, is built upon a dedicated block cipher in Davies-Meyer mode: the output of the compression function is the output of the block cipher with a feed-forward of the chaining variable.

Dobbertin [10] provided the first cryptanalysis of a member of this family with a collision attack against `MD4`. Later, Chabaud-Joux [7] published the first theoretical collision attack against `SHA-0` and Biham-Chen [2] introduced the idea of neutral bits, which led to the computation of a real collision with four blocks of message [3]. Then, a novel framework of collision attack, using modular difference and message modification techniques, surprised the cryptography community [26–29]. Those devastating attacks broke a lot of hash functions, such as `MD4`, `MD5`, `SHA-0`, `SHA-1`, `RIPEMD` or `HAVAL-128`. In the case of `SHA-0` the overall complexity of the attack was 2^{39} message modification processes. Recently, Naito *et al.* [18] lower this complexity down to 2^{36} operations, but we argue in this paper that it is a theoretical complexity and not a measured one.

At Crypto 2007, Joux and Peyrin [14] published a generalization of neutral bits and message modification techniques and applied their results to `SHA-1`. The so-called *boomerang attack* was first devoted for block ciphers cryptanalysis [25] but their work showed that it can also be used in the hash functions setting. Used in parallel with the automated tool from De Cannière and Rechberger [5] that generates non-linear part of a differential path, this method turns out to be quite easy to use and handy for compression functions cryptanalysis.

This article presents a new attack against the collision resistance of `SHA-0` requiring only 2^{33} hash computations and the theoretical analysis is confirmed by experimentation. First, we show that the previously used perturbation vector, originally found by Wang *et al.*, is not optimal. We therefore introduce a new vector, allowing ourselves to use two iterations of the compression function. In order to compensate the loss of the known message modifications due to the perturbation vector change, we use the boomerang attack framework in order to accelerate the collision search. Finally, this work leads to the best collision attack against `SHA-0` from now on, now requiring only one hour of computation on an average PC.

We organized the paper as follows. In Section 2, we recall the previous attacks and cryptanalysis techniques for `SHA-0`. Then, in Section 3, we analyze the perturbation vector problem and give new ones that greatly improve the complexity of previous attacks. We then apply the boomerang technique as a speedup technique in Section 4 and provide the final attack along with its complexity analysis in Section 5. Finally, we draw conclusions in Section 6.

2 Previous collision attacks on SHA-0

2.1 A short description of SHA-0

SHA-0 [19], is a 160-bit dedicated hash function based on the design principle of MD4. It applies the Merkle-Damgård paradigm to a dedicated compression function. The input message is padded and split into k 512-bit message blocks. At each iteration of the compression function h, a 160-bit chaining variable H_t is updated using one message block M_{t+1}, *i.e* $H_{t+1} = h(H_t, M_{t+1})$. The initial value H_0 (also called IV) is predefined and H_k is the output of the hash function.

The SHA-0 compression function is build upon the Davis-Meyer construction. It uses a function E as a block cipher with H_t for the message input and M_{t+1} for the key input, a feed-forward is then needed in order to break the invertibility of the process:

$$H_{t+1} = E(H_t, M_{t+1}) \boxplus H_t,$$

where $\boxplus$ denotes the addition modulo 2^{32} 32-bit words by 32-bit words. This function is composed of 80 steps (4 rounds of 20 steps), each processing a 32-bit message word W_i to update 5 32-bit internal registers (A, B, C, D, E). The feed-forward consists in adding modulo 2^{32} the initial state with the final state of each register. Since more message bits than available are utilized, a message expansion is therefore defined.

Message expansion. First, the message block M_t is split into 16 32-bit words $W_0, \ldots, W_{15}$. These 16 words are then expanded linearly, as follows:

$$W_i = W_{i-16} \oplus W_{i-14} \oplus W_{i-8} \oplus W_{i-3} \text{ for } 16 \leq i \leq 79.$$

State update. First, the chaining variable H_t is divided into 5 32-bit words to fill the 5 registers $(A_0, B_0, C_0, D_0, E_0)$. Then the following transformation is applied 80 times:

$$STEP_{i+1} := \begin{cases} A_{i+1} = (A_i \lll 5) + f_i(B_i, C_i, D_i) + E_i + K_i + W_i, \\ B_{i+1} = A_i, \\ C_{i+1} = B_i \ggg 2, \\ D_{i+1} = C_i, \\ E_{i+1} = D_i. \end{cases}$$

where K_i are predetermined constants and f_i are boolean functions defined in Table 1.

Feed-forward. The sums modulo 2^{32}: $(A_0 + A_{80})$, $(B_0 + B_{80})$, $(C_0 + C_{80})$, $(D_0 + D_{80})$, $(E_0 + E_{80})$ are concatenated to form the chaining variable H_{t+1}.

Note that all updated registers but A_{i+1} are just rotated copies, so we only need to consider the register A at each step. Thus, we have:

$$A_{i+1} = (A_i \lll 5) + f_i(A_{i-1}, A_{i-2} \ggg 2, A_{i-3} \ggg 2) + A_{i-4} \ggg 2 + K_i + W_i. \qquad (1)$$

round	step i	$f_i(B, C, D)$	K_i
1	$1 \leq i \leq 20$	$f_{IF} = (B \wedge C) \oplus (\overline{B} \wedge D)$	`0x5a827999`
2	$21 \leq i \leq 40$	$f_{XOR} = B \oplus C \oplus D$	`0x6ed6eba1`
3	$41 \leq i \leq 60$	$f_{MAJ} = (B \wedge C) \oplus (B \wedge D) \oplus (C \wedge D)$	`0x8fabbcdc`
4	$61 \leq i \leq 80$	$f_{XOR} = B \oplus C \oplus D$	`0xca62c1d6`

Table 1. Boolean functions and constants in `SHA-0`.

2.2 First attacks on `SHA-0`

The first published attack on `SHA-0` has been proposed by Chabaud and Joux in 1998 [7]. It focused on finding linear differential paths composed of interleaved 6-step *local collisions*, which have probability 1 to hold in a linearized version of `SHA-0`. However, in the standard version of `SHA-0`, a local collision only has a certain probability to hold. The overall probability of success of the attack is the product of the holding probability of each local collision.

The core of the differential path is represented by a perturbation vector which indicates where the 6-step local collisions are initiated. The probability of success of the attack is then related to the number of local collisions appearing in the perturbation vector. In their paper, Chabaud and Joux have defined 3 necessary conditions on perturbation vectors in order to permit the differential path to end with a collision for the 80-step compression function. Such a perturbation vector should not have truncated local collisions, should not have two consecutive local collisions initiated in the first $16th$ steps and should not start a local collision after step 74. Under these constraints they were able to find a perturbation vector (so-called L-characteristic) with a probability of success of 2^{68}. The running complexity of their attack is decreased to 2^{61} by a careful implementation of the collision search. As the attacker has full control on the first 16 message blocks, those blocks are chosen such that the local collisions of those early steps hold with probability 1. See [7] for more details.

In 2004, Biham and Chen have improved the attack of Chabaud and Joux by introducing the neutral bit technique. The idea is to multiply the number of conformant message pairs up to a certain step s (message pairs that verify the main differential path up to step s) for a cost almost null. This is done by looking for different sets of small modifications in the message words such that each set will have very low impact on the conformance of the message pair up to step s. Basically, the attacker can effectively start the collision search at a higher step than in a normal setting, and this improvement finally led to the computation of the first real collision for `SHA-0` with four blocks of message [3] with an overall complexity of 2^{51} functions calls.

2.3 The Wang approach

The attack on `SHA-0` of Wang *et al.* is derived from the approach of Chabaud and Joux. The principle of this attack consists in relaxing two of the three conditions on the perturbation vectors defined by Chabaud and Joux, namely no truncated local collision allowed and no consecutive local collisions in the $16th$ first steps. Relaxing those conditions permits to search for better perturbation vectors, *i.e.* higher probability linear differential paths.

However, the main drawback of this approach is that non corrected perturbations inherited from truncated local collisions appear in the first steps. In order to offset these unwanted perturbations, they had to construct a non linear differential path (so-called NL-characteristic) which connects to the desired linear differential path. Said in other words, they kept the same linear differential mask on the message, but computed a new and much more complex differential mask on the registers for the early steps of `SHA-0`. A NL-characteristic presents also the advantage that consecutive local collisions in the early steps are no more a problem. Using modular subtraction as the differential, the carry effect (a property of the powers of 2, i.e. $2^j = -2^j - 2^{j+1} \ldots - 2^{j+k-1} + 2^{j+k}$) and by carefully controlling the non linearity of the round function IF, they succeeded to build their NL-characteristic by hand. A NL-characteristic holds only if specific conditions are verified step by step by the register values. In their paper, Wang *et al.* denoted these conditions as *sufficient conditions*. These sufficient conditions are described with respect to the register A into one general form $A_{i,j} = v$ where $A_{i,j}$ denotes the value of bit j of the register A at the step i and where v is a bit value fixed to be 0 or 1 or a value that has been computed before step i.

The NL-characteristic found presents conditions on the initial value of the registers[5]. However, since the initial value is fixed, Wang *et al.* have build their collision with two blocks of message. The first block is needed in order to obtain a chaining variable verifying the conditions, inherited from the NL-characteristic, on the initial values of the register. This is detailed in Figure 1.

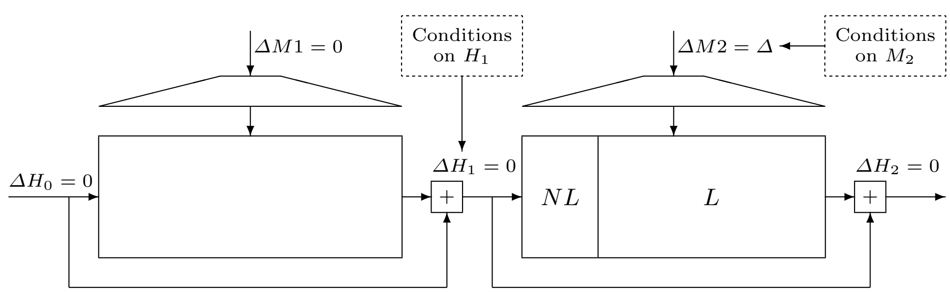

Fig. 1. Attack of Wang *et al.*

The attack of Wang *et al.* is thus divided into two phases. The first one is the precomputation phase:

1. search for a higher probability L-characteristic by relaxing conditions on the perturbation vectors,
2. build a NL-characteristic which connects to the L-characteristic by offsetting unwanted perturbations,
3. find a first block of message from which the incoming chaining variable verifies the conditions inherited from the NL-characteristic.

[5] The conditions given by Wang *et al.* in their article are incomplete. In fact, two more conditions need to be verified [16, 18].

The second phase is the collision search phase. It consists in searching for a second block of message for which the sufficient conditions on the register values are fulfilled for a maximum number of steps. In order to achieve that goal, they use both *basic modification technique* and *advanced modification technique*. The main idea of the former is simply to set $A_{i,j}$ to the correct bit by modifying the corresponding bit of the message word W_{i-1}. This is only possible for the first 16 steps, where the attacker has full control on the values W_i. The advanced modification technique are to be applied to steps 17 and higher, where the message words W_i are generated by the message expansion. The idea is to modify the message words of previous steps in order to fulfill a condition in a given step. Wang *et al.* claimed in their article that using both basic modification and advanced modification techniques, they are able to fulfill all the sufficient conditions up to step 20. However very few details can be found on advanced modification technique in their article. Finally, their attack has a claimed complexity of 2^{39} `SHA-0` operations.

Remark. Wang *et al.* optimized the choice of their perturbation vector taking into account their ability to fulfill the conditions up to step 20.

2.4 Naito *et al.*

Naito *et al.* recently proposed [18] a new advanced modification technique so-called *submarine modification*. Its purpose is to ensure that the sufficient conditions from steps 21 to 24 are fulfilled. The main idea of submarine modifications is to find *modification* characteristics which will permit to manipulate bit values of registers and message words after step 16. Each parallel characteristic is specifically built to satisfy one target condition. In order to construct such a characteristic, Naito *et al.* use two different approaches the *cancel method* and the *transmission method*. The former is based on the local collision principle. Whereas the transmission method combines the recurrence properties of the message expansion and of the step update transformation.

Those modification characteristics define new sets of conditions on register values and message words. The new conditions should not interfere with the already pre-computed sufficient conditions. Naito *et al.* detailed their submarine modifications up to step 17 (see [18] proof of Theorem 1). They remarked that the probability that one of these modifications can satisfy a target condition without affecting the other sufficient conditions is almost 1. No detail is given about the impact of the submarine modifications after step 24.

The claimed complexity of the attack described in [18] is 2^{36} `SHA-0` operations. This is a theoretical complexity that will be further discussed in section 5. The given collision example is based on the same NL-characteristic and perturbation vector that Wang *et al.* used to produce their own collision. Taking into account that their submarine modifications permit to fulfill the sufficient conditions up to step 24, Naito *et al.* proposed a new perturbation vector which would therefore minimizes the complexity of the attack. However, in order to effectively build an attack based on the proposed vector, a new NL-characteristic and new submarine modifications should be found.

3 A new perturbation vector

In order to lower the complexity of a collision search on `SHA-0`, high probability L-characteristics are needed. In the previous attacks on `SHA-0`, the authors have proposed perturbation vectors which do not have local collisions starting after step 74. By relaxing this last condition, it may be possible to find better perturbation vectors. Note that those vectors do not seem to be eligible for a collision search, since they would lead to a near-collision (two compression function outputs with very few bits of difference) instead of a collision. However, this problem can be tackled by using the *multi-block technique* as in [3]: the attacker can take advantage of the feed-forward operation inherited from the Davis-Meyer construction used in the compression function of `SHA-0`. Said in other words, we allow ourselves to use several message blocks with differences, whereas the previous known attacks on `SHA-0` only use one of such blocks of message. Thanks to the new automatic tool from De Cannière and Rechberger [5] that can generate NL-characteristics on `SHA-1`, computing non linear parts for `SHA-0` is relatively easy. Indeed, `SHA-1` and `SHA-0` only differ on a rotation in the message expansion, which has no effect on the validity of this tool. Moreover, the ability to generate NL-characteristics reduce the multi-block problem to the use of only two blocks. More precisely, we start with a L-characteristic L_1 (defined by a new perturbation vector), and modified on the early steps by a generated NL-characteristic NL_1. We thus obtain a specific near-collision $\Delta H_1 = +d$ after this first block. We then apply the same L-characteristic modified on the early steps by a second generated NL-characteristic NL_2, that takes in account the new incoming chaining variable H_1. Finally, before the feed-forward on this second block, we look for the opposite difference $\Delta E(H_1, m_1) = -d$ and the two differences cancel each over $\Delta H_2 = 0$. This is detailed in Figure 2.

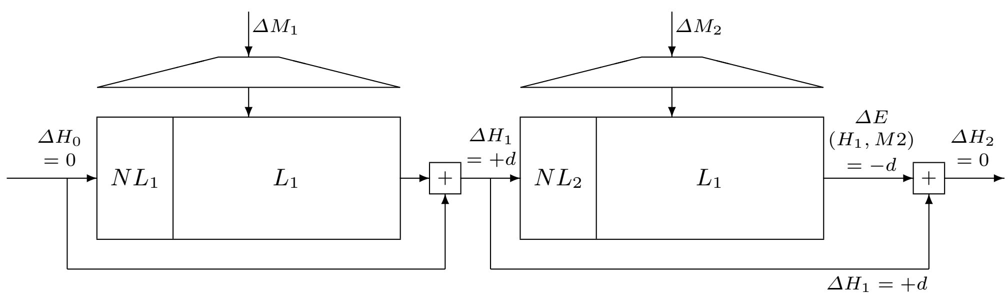

Fig. 2. Multi-block collision on `SHA-0`.

Now that all the conditions on the perturbation vectors are relaxed, we need to define what are the criteria for good perturbation vectors. In order to fulfill the sufficient conditions inherited from NL_1, NL_2 and L_1, we will use basic message modifications and boomerang techniques. In consequence, we focused our search on perturbation vectors that have the smallest number of sufficient conditions to fulfill in steps 17 to 80. Namely, a characteristic L_1 for which the probability of success of the attack is maximized. We used the same approach of Chabaud and Joux in order to evaluate the probability of holding of each local collision involved. See [7] and particularly Section 2.2 (Tables 4 and 5) for detailed examples. There are a lot of perturbation vectors with an evaluated probability of success around 2^{-40} for the steps between 17 and 80. However, this probability can be affected by the NL-characteristic. Thus, we build the NL-characteristics corresponding to each matching perturbation vector in order to compute the exact probability of success.

This aspect of the search will be further detailed in the next section. The perturbation vector we chose, which has 42 conditions between step 17 and 80, is given in Table 2.

Steps 1 to 40	
vector	1 1 1 1 0 1 0 1 1 0 1 1 1 0 0 0 1 0 0 0 1 0 1 0 1 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0
♯ conditions	- - - - - - - - - - - - - - - - 2 0 2 1 1 0 2 0 2 0 1 0 0 0 0 0 0 0 1 0 1 1 0 1

Steps 41 to 80	
vector	0 0 1 0 0 0 0 0 1 0 0 0 0 1 0 0 1 0 1 1 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 1 0 0 0 0
♯ conditions	1 1 1 0 1 1 1 0 1 0 1 1 1 1 0 1 2 1 2 2 1 1 0 0 0 0 1 0 1 0 1 0 1 0 0 1 0 1 0 0

Table 2. A new perturbation vector for `SHA-0`, along with the number of conditions at each steps (the conditions before step 16 have been removed since not involved in the complexity during the collision search).

4 Boomerang attacks for `SHA-0`

Now that we found good perturbation vectors by relaxing certain conditions, a problem remains. Indeed, no message modification and no NL-characteristic are known for those vectors, and this makes the attack complexity drastically increase. This is the major drawback of Wang *et al.* collision attack on `SHA-0` and other hash functions: is not easily reusable and we are stuck with their perturbation vector. Hopefully, some work have been done recently in order to theorize Wang *et al.* major improvements. Recently, De Cannière and Rechberger [5] introduced an automated tool that generates non linear part of a differential path, thus resolving the NL-characteristic problem. Then, Joux and Peyrin [14] provided a framework that generalizes message modifications and neutral bits. Thanks to the so-called *boomerang attack*, they describe techniques that allows an attacker to easily use neutral bits or message modifications, given only a main perturbation vector. In fact, boomerang attacks and NL-characteristic automated search are exactly the two tools we need for our attack to be feasible. Finally, we replace the loss of the message modifications because of the new vector by the gain of the boomerang attack, which is a much more practical technique and fit for our constraints.

Boomerang attacks for hash functions can be seen as a generalization of collision search speed up techniques such as neutral bits or message modification. However, new possibilities were also suggested. In the usual setting, the attacker first sets the differential path and then tries to find neutral bits or message modifications if possible. In the *explicit conditions approach* from boomerang attacks, the attacker first set some constraints on the registers and the message words and then tries to find a differential path taking in account those constraints. One can see that for the latter part the NL-characteristic automated search tool becomes really handy. The constraints are set in order to provide very good message modifications or neutral bits that would not exist with a random differential path, or with very low probability. More generally, this can be seen as an *auxiliary characteristic*, different from the main one, but only fit for a few steps and this auxiliary characteristic can later be used as a neutral bit or a message modification, with very high probability (generally probability equal to 1) thanks to the preset constraints. Obviously the complexity of the collision search will decrease by adding as much auxiliary characteristics as possible.

	W_0 to W_{15}	W_{16} to W_{31}
perturbation mask	0000001000000000	
differences on W^j	0000001000000000	0000101101100111
differences on W^{j+5}	0000000100000000	0000010110110011
differences on W^{j-2}	0000000000010000	000$\underline{1}$001000000010

i	A_i	W_i
-1:	`--------------------------------`	
00:	`--------------------------------`	`--------------------------------`
01:	`--------------------------------`	`--------------------------------`
02:	`--------------------------------`	`--------------------------------`
03:	`--------------------------------`	`--------------------------------`
04:	`--------------------------------`	`--------------------------------`
05:	`---------------------------b----`	`--------------------------------`
06:	`---------------------------b----`	`-----------------------------a--`
07:	`-----------------------------a--`	`------------------------ā-------`
08:	`-------------------------------0`	`--------------------------------`
09:	`-------------------------------1`	`--------------------------------`
10:	`--------------------------------`	`--------------------------------`
11:	`--------------------------------`	`-------------------------------ā`
12:	`--------------------------------`	`--------------------------------`
13:	`--------------------------------`	`--------------------------------`
14:	`--------------------------------`	`--------------------------------`
15:	`--------------------------------`	`--------------------------------`

Fig. 3. Auxiliary differential path AP_1 used during the attack. The first table shows the 32 first steps of the perturbation vector (with the first uncontrolled difference on registers at step 20) and the second gives the constraints forced in order to have probability one local collisions in the early steps in the case where the auxiliary path is positioned at bit $j = 2$. The MSB's are on the right and "-" stands for no constraint. The letters represent a bit value and its complement is denoted by an upper bar on the corresponding letter (see [14] for the notations).

Building an auxiliary path requires the same technique as for a main path, that is the local collisions. We refer to [14] for more details on this construction for `SHA-1`, since the technique is identically applicable to `SHA-0`. In our attack, we will consider two different types of auxiliary paths and we will use them as neutral bits (and not message modifications). Informally, we define the range of an auxiliary path to be the latest step where the uncontrolled differences from the auxiliary path (after the early steps) do not interfere in the main differential path. The first one, AP_1, will have very few constraints but the range will be low. On contrary, the second type, AP_2, will require a lot of constraints but the range will be much bigger. A trade-off among the two types needs to be considered in order not to have to many constraints forced (which would latter makes the NL-characteristic automated search tool fail) but also have a good set of auxiliary differential paths. More precisely, AP_1 and AP_2 are detailed in Figures 3 and 4 respectively. AP_1 is build upon only one local collision but the first uncontrolled difference appears at step 20. AP_2 is build upon three local collisions but the first uncontrolled difference appears at step 25. Note that, as remarked in the original paper from Joux and Peyrin, an auxiliary differential path used as a neutral bit with the first uncontrolled difference at step s is a valid neutral bit for step $s+3$ with a very high probability (the uncontrolled difference must first propagate before disrupting the main differential path). Thus, in our attack, we will use AP_1 and AP_2 as neutral bits for steps 23 and 28 respectively; that is as soon as we will find a conformant message pair up to those step during the collision search, we will trigger the corresponding auxiliary path in order to duplicate the conformant message pair. This will directly provide new conformant message pairs for free.

	W_0 to W_{15}	W_{16} to W_{31}
perturbation mask	1010000000100000	
differences on W^j	1010000000100000	00000000$\underline{1}$0110110
differences on W^{j+5}	0101000000010000	0000000001011011
differences on W^{j-2}	0001111100000011	0000000000001110

```
 i    A_i                                 W_i
-1:   ---------------------------d----
00:   ---------------------------d----    -----------------------------a--
01:   ---------------------------e-a--    ------------------------ā-------
02:   ---------------------------e---1    -----------------------------b--
03:   -----------------------------b-0    ------------------------b̄------ā
04:   -------------------------------0    -------------------------------ā
05:   -------------------------------0    -------------------------------ā
06:   --------------------------------    -------------------------------b̄
07:   --------------------------------    -------------------------------b̄
08:   --------------------------------    --------------------------------
09:   ---------------------------f----    --------------------------------
10:   ---------------------------f----    -----------------------------c--
11:   -----------------------------c--    ------------------------c̄-------
12:   -------------------------------0    --------------------------------
13:   -------------------------------0    --------------------------------
14:   --------------------------------    -------------------------------c̄
15:   --------------------------------    -------------------------------c̄
```

Fig. 4. Auxiliary differential path AP_2 used during the attack. The first table shows the 32 first steps of the perturbation vector (with the first uncontrolled difference on registers at step 25) and the second gives the constraints forced in order to have probability one local collisions in the early steps in the case where the auxiliary path is positioned at bit $j = 2$. The MSB's are on the right and "-" stands for no constraint. The letters represent a bit value and its complement is denoted by an upper bar on the corresponding letter (see [14] for the notations).

The next step is now to build a main differential path with the tool from De Cannière and Rechberger, containing as much auxiliary paths as possible (of course while favoring AP_2 instead of AP_1, the latter being less powerful). We refer to [5] for the details of this algorithm. The tool works well for SHA-0 as for SHA-1 and given a random chaining variable, it is easy to find a main differential path containing at least five auxiliary paths, with at least three AP_2 characteristics. Note that this part, as the automated tool, is purely heuristic and often more auxiliary paths can be forced[6]. However, the behavior of the automated tool is quite dependant of the perturbation vector. Thus, among the possible good ones, we chose a perturbation vector (depicted in Table 2) that seemed to work well with the automated search tool. Note that since the perturbation vector remains the same during the two parts of the attack, this property will be true for the two blocks of message.

5 The final collision attack

At this point, we have all the required elements to mount the attack and analyze it. Its total complexity will be the addition of the complexities of the collision search for both blocks[7]. Note that, unlike for the 2-block collision attacks for SHA-1 where the first block complexity is negligible compared to the second block one, here our perturbation vector imposes the same raw complexity for both blocks.

5.1 A method of comparison

As observed in [6, 14], there are many different collision search speeding techniques for the SHA family of hash functions. However, their real cost is often blurry and it becomes hard to compare them. Thus, it has been advised to measure the efficiency of those tools with an efficient implementation of the various hash functions attacked, for example by using OpenSSL [22]. For any computer utilized, one can give the complexity of the attack in terms of number of function calls of the hash function with an efficient implementation, which is relatively independent of the computer used.

In their paper [18], Naito *et al.* claimed a complexity of 2^{36} SHA-0 calls for their collision attack. However, their implementation required approximatively 100 hours on average in order to find one collision on a PentiumIV 3,4 GHz. 100 hours of SHA-0 computations on this processor would correspond to $2^{40,3}$ SHA-0 calls approximatively with OpenSSL, which is far from the claimed complexity.

Thus, in this paper, we chose to handle the complexities in terms of number of SHA-0 calls with OpenSSL in order to allow an easy comparison. The time measurements have been done on a single PC with an AMD Opteron 2, 2 GHz processor.

[6] The auxiliary path AP_2 has one condition on the IV (the bit d in Figure 4) and this harden the task of the attacker to place a lot of them in the main differential trail. However, this is already taken in account in the evaluation. There remains space for improvements on this part but we chose to describe an easy-to-implement attack instead of the best possible but hard to implement one.

[7] One can argue that the cost of the NL-characteristics automated search tool has also to be counted. However, unlike in the SHA-1 case, for SHA-0 the number of possible collisions that can be generated with only one full differential path construction is really big. Thus, this cost becomes largely negligible compared to the collision search.

5.2 Without collision search speedup

Without even using boomerang attacks, our new differential paths already provides an improvement on the best known collision attack against SHA-0. Indeed, in our perturbation vector, 42 conditions remain after step 17. However, by refining the differential path utilized (i.e. by forcing some conditions just before step 17, without any impact on the complexity since located in the early steps), one can easily take care of the two conditions from step 17 before beginning the collision search. Therefore, we are finally left with 40 conditions per message block. Since for each basic message pair tested during the collision search only one quarter of a whole SHA-0 is computed in average, we expect a complexity of $2^{40}/2^{2} = 2^{38}$ SHA-0 evaluations for one block and thus a total complexity of 2^{39} SHA-0 evaluations for one complete collision. This theoretical complexity is fully confirmed by practical implementation ($2^{37,9}$ and $2^{37,8}$ SHA-0 evaluations on average for the first and second blocks respectively). Our computer needs 39 hours on average in order to find a collision, which is much faster than Naito *et al.*'s attack and this with a less powerful processor.

5.3 Using the boomerang improvement

As one can use many collision search speedup techniques, a good choice can be the boomerang attack for its simplicity of use. We give in the appendix a possible differential path for the first block (Tables 5 and 6) and the second block (Tables 7 and 8). The notations used are also given in the appendix in Table 4. The chaining variable of the second block is the output of the first valid message pair found (i.e. conformant to the whole differential path) for the first block. As for the previous subsection, we are left with 40 conditions for each blocks since the two conditions from step 17 can be easily cancelled before the collision search. The differential path for the second block possesses 5 auxiliary paths. However, as explained before, it is possible to build NL-characteristics containing more auxiliary paths but we only take in account the average case and not the best case of behavior of the automated NL-characteristics search tool. For example, one can check that there are 3 AP_2 auxiliary paths in bit positions $j = \{17, 22, 30\}$ and 2 AP_1 auxiliary paths in bit positions $j = \{9, 11\}$ for the second block. The first block case is particular since the chaining variable is the predefined IV which is highly structured. This particular structure greatly improves our capability to place auxiliary paths since the condition on the chaining variable for AP_2 is verified on much more bit positions than what someone would expect in the random case. Thus, for the first block one can place 2 more AP_2 on average and one can check in Table 5 that there are 5 AP_2 auxiliary paths in bit positions $j = \{10, 14, 19, 22, 27\}$ and 2 AP_1 auxiliary paths in bit positions $j = \{9, 11\}$.

In theory, with k auxiliary paths, one would expect an improvement of a factor 2^k. However, this slightly depends on the type and the number of auxiliary paths used. Obviously, compared to the AP_1 auxiliary path, using the AP_2 type is better during the collision search. Thus, we expect an improvement of the attack of a factor approximatively $2^7 = 128$ for the first block and $2^5 = 32$ for the second one. We get something close in practice with a measured complexity of $2^{32,2}$ and 2^{33} function calls for the first and second block respectively. This leads to a final complexity for a collision on the whole SHA-0 of $2^{33,6}$ function calls, which compares favourably in theory and practice to the best known collision attack on this hash function: our computer can generate a 2-block collision for SHA-0 in approximatively one hour on average (instead of 100 hours of computation on

	1^{st} block		2^{nd} block	
	M_1	M'_1	M_2	M'_2
W_0	0x4643450b	0x46434549	0x9a74cf70	0x9a74cf32
W_1	0x41d35081	0x41d350c1	0x04f9957d	0x04f9953d
W_2	0xfe16dd9b	0xfe16dddb	0xee26223d	0xee26227d
W_3	0x3ba36244	0x3ba36204	0x9a06e4b5	0x9a06e4f5
W_4	0xe6424055	0x66424017	0xb8408af6	0x38408ab4
W_5	0x16ca44a0	0x96ca44a0	0xb8608612	0x38608612
W_6	0x20f62444	0xa0f62404	0x8b7e0fea	0x0b7e0faa
W_7	0x10f7465a	0x10f7465a	0xe17e363c	0xe17e363c
W_8	0x5a711887	0x5a7118c5	0xa2f1b8e5	0xa2f1b8a7
W_9	0x51479678	0xd147963a	0xca079936	0x4a079974
W_{10}	0x726a0718	0x726a0718	0x02f2a7cb	0x02f2a7cb
W_{11}	0x703f5bfb	0x703f5bb9	0xf724e838	0xf724e87a
W_{12}	0xb7d61841	0xb7d61801	0x37ffc03a	0x37ffc07a
W_{13}	0xa5280003	0xa5280041	0x53aa8c43	0x53aa8c01
W_{14}	0x6b08d26e	0x6b08d26c	0x90811819	0x9081181b
W_{15}	0x2e4df0d8	0xae4df0d8	0x312d423e	0xb12d423e

A_2	B_2	C_2	D_2	E_2
0x6f84b892	0x1f9f2aae	0x0dbab75c	0x0afe56f5	0xa7974c90

Table 3. Message instance for a 2-block collision: $H(M_1, M_2) = H(M'_1, M'_2) = A_2||B_2||C_2||D_2||E_2$, computed according to the differential path given in Tables 5, 6, 7 and 8 from appendix.

a faster processor for the best known attack). For proof of concept, we provide in Table 3 a 2-block message pairs that collides with SHA-0.

6 Conclusion

In this paper, we introduced a new attack against SHA-0. By relaxing the previously established constraints on the perturbation vector, we managed to find better candidates. Then, as a collision search speedup technique, we applied on those candidates the boomerang attack which provides a good improvement with a real practicality of use. This work leads to the best collision attack on SHA-0 so far, requiring only one hour of computation on an average PC. Yet, there stills space for further improvements as some parts of the attack are heuristic. Moreover, this work shows the efficiency of the dual use of the boomerang attack for hash functions combined with a differential path automated search tool.

References

1. M. Bellare and T. Ristenpart. Multi-Property-Preserving Hash Domain Extension and the EMD Transform. In X. Lai and K. Chen, editors, *Advances in Cryptology – ASI-*

ACRYPT 2006, volume 4284 of *Lecture Notes in Computer Science*, pages 299–314. Springer-Verlag, 2006.
2. E. Biham and R. Chen. Near-Collisions of SHA-0. In M.K. Franklin, editor, *Advances in Cryptology – CRYPTO 2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 290–305. Springer-Verlag, 2004.
3. 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*, volume 3494 of *Lecture Notes in Computer Science*, pages 36–57. Springer-Verlag, 2005.
4. E. Biham and O. Dunkelman. A Framework for Iterative Hash Functions: HAIFA. In Proceedings of *Second NIST Cryptographic Hash Workshop*, 2006 . Available from: `www.csrc.nist.gov/pki/HashWorkshop/2006/program_2006.htm`.
5. C. De Cannière and C. Rechberger. Finding SHA-1 Characteristics: General Results and Applications. In X. Lai and K. Chen, editors, *Advances in Cryptology – ASIACRYPT 2006*, volume 4284 of *Lecture Notes in Computer Science*, pages 1–20. Springer-Verlag, 2006.
6. C. De Cannière, F. Mendel and C. Rechberger. Collisions for 70-step SHA-1: On the Full Cost of Collision Search. to appear in C. Adams and A. Miri and M. Wiener, editors, *Selected Areas in Cryptography – SAC 2007*, *Lecture Notes in Computer Science*. Springer-Verlag, 2007.
7. F. Chabaud and A. Joux. Differential Collisions in SHA-0. In H. Krawczyk, editor, *Advances in Cryptology – CRYPTO'98*, volume 1462 of *Lecture Notes in Computer Science*, pages 56–71. Springer-Verlag, 1998.
8. I. Damgård. A Design Principle for Hash Functions. In G. Brassard, editor, *Advances in Cryptology – CRYPTO'89*, volume 435 of *Lecture Notes in Computer Science*, pages 416–427. Springer-Verlag, 1989.
9. R.D. Dean. *Formal Aspects of Mobile Code Security*. PhD thesis, Princeton University, 1999.
10. H. Dobbertin. Cryptanalysis of MD4. In D. Gollmann, editor, *Fast Software Encryption – FSE'96*, volume 1039 of *Lecture Notes in Computer Science*, pages 53–69. Springer-Verlag, 1996.
11. A.J. Menezes, S.A. Vanstone, and P.C. Van Oorschot. *Handbook of Applied Cryptography*. CRC Press, Inc., Boca Raton, FL, USA, 1996.
12. J.J. Hoch and A. Shamir. Breaking the ICE - Finding Multicollisions in Iterated Concatenated and Expanded (ICE) Hash Functions. In M.J.B. Robshaw, editor, *Fast Software Encryption – FSE 2006*, volume 4047 of *Lecture Notes in Computer Science*, pages 179–194. Springer-Verlag, 2006.
13. A. Joux. Multi-collisions in Iterated Hash Functions. Application to Cascaded Constructions. In M. Franklin, editor, *Advances in Cryptology – CRYPTO 2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 306–316. Springer-Verlag, 2004.
14. A. Joux and T. Peyrin. Hash Functions and the (Amplified) Boomerang Attack. In A. Menezes, editor, *Advances in Cryptology – CRYPTO 2007*, volume 4622 of *Lecture Notes in Computer Science*, pages 244–263. Springer-Verlag, 2004.
15. J. Kelsey and B. Schneier. Second Preimages on n-bit Hash Functions for Much Less Than 2^n Work. In R. Cramer, editor, *Advances in Cryptology – EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 474–490. Springer-Verlag, 2005.
16. S. Manuel. Cryptanalyses Différentielles de SHA-0. Mémoire pour l'obtention du Mastère Recherche Mathematiques Applications au Codage et à la Cryptographie. Université Paris 8, 2006. Available from: `http://www-rocq.inria.fr/codes/Stephane.Manuel`
17. R.C. Merkle. One Way Hash Functions and DES. In G. Brassard, editor, *Advances in Cryptology – CRYPTO'89*, volume 435 of *Lecture Notes in Computer Science*, pages 428–446. Springer-Verlag, 1989.
18. Y. Naito, Y. Sasaki, T. Shimoyama, J. Yajima, N. Kunihiro and K. Ohta. In X. Lai and K. Chen, editors, *Advances in Cryptology – ASIACRYPT'06*, volume 4284 of *Lecture Notes in Computer Science*, pages 21–36. Springer-Verlag, 2006.
19. National Institute of Standards and Technology. FIPS 180: Secure Hash Standard, May 1993 . Available from: `http://csrc.nist.gov`.

20. National Institute of Standards and Technology. FIPS 180-1: Secure Hash Standard, April 1995 . Available from: `http://csrc.nist.gov`.
21. National Institute of Standards and Technology. FIPS 180-2: Secure Hash Standard, August 2002 . Available from: `http://csrc.nist.gov`.
22. OpenSSL. The Open Source toolkit for SSL/TLS, 2007. . Available from: `http://www.openssl.org/source`.
23. Ronald L. Rivest. RFC 1321: The MD5 Message-Digest Algorithm, April 1992 . Available from: `http://www.ietf.org/rfc/rfc1321.txt`.
24. Ronald L. Rivest. RFC 1320: The MD4 Message Digest Algorithm, April 1992 . Available from: `http://www.ietf.org/rfc/rfc1320.txt`.
25. D. Wagner. The Boomerang Attack. In L.R. Knudsen, editor, *Fast Software Encryption – FSE'99*, volume 1636 of *Lecture Notes in Computer Science*, pages 156–170. Springer-Verlag, 1999.
26. 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*, volume 3494 of *Lecture Notes in Computer Science*, pages 1–18. Springer-Verlag, 2005.
27. X. Wang and H. Yu. How to Break MD5 and Other Hash Functions. In R. Cramer, editor, *Advances in Cryptology – EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 19–35. Springer-Verlag, 2005.
28. X. Wang, H. Yu and Y.L. Yin. Efficient Collision Search Attacks on SHA-0. In V. Shoup, editor, *Advances in Cryptology – CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 1–16. Springer-Verlag, 2005.
29. X. Wang, Y.L. Yin, and H. Yu. Finding Collisions in the Full SHA-1. In V. Shoup, editor, *Advances in Cryptology – CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 17–36. Springer-Verlag, 2005.

Appendix

(x, x^*)	$(0,0)$	$(1,0)$	$(0,1)$	$(1,1)$
?	✓	✓	✓	✓
-	✓	-	-	✓
x	-	✓	✓	-
0	✓	-	-	-
u	-	✓	-	-
n	-	-	✓	-
1	-	-	-	✓
#	-	-	-	-

(x, x^*)	$(0,0)$	$(1,0)$	$(0,1)$	$(1,1)$
3	✓	✓	-	-
5	✓	-	✓	-
7	✓	✓	✓	-
A	-	✓	-	✓
B	✓	✓	-	✓
C	-	-	✓	✓
D	✓	-	✓	✓
E	-	✓	✓	✓

Table 4. Notations used in [5] for a differential path: x represents a bit of the first message and x^* stands for the same bit of the second message.

i	A_i	W_i
-4:	00001111010010111000011111000011	
-3:	01000000110010010101000111011000	
-2:	01100010111010110111001111111010	
-1:	11101111110011011010101110001001	
00:	01100111010001010010001100000001	0100111001001011000001010n0010u1
01:	1110110111111111100111011n1111u0	0100000011011011010100001n000000
02:	0110011101111111101n10101101010	1111011000011110100111011n011011
03:	001101010010100n00001100n0uuuuu0	0011100010101001011100100u000101
04:	111100000nu000001010110001110000	u11001000100000000101000000u0101n1
05:	00111n000000100110001000000u0n1u1	n0010100110010000101010010100000
06:	10110101110110110000101u100u1001	n0100010111101000011110000u000100
07:	100unnnnnnnnn0100nu0100101u11001	00010010111101000101011001011010
08:	1000011100001n000n100u0n010nn001	0101101001110001000110001n0001u1
09:	0010000000000010un00nu1u1un01100	n1010001010001111001011000u1110n0
10:	11100110100101000nu01u10un00n100	01111010011000100100011100011000
11:	011110001110001101nuu10101000101	0111000100110111010110011u1110u0
12:	010011010110100000010u0000n110000	10110111110101-----1-----u000001
13:	010110011100000----010-0-01001u0	101001010----------------n0000u1
14:	10111100--------------1--110u011	01101-0----0--1----0---0-1-011u0
15:	10100------------------0-1-u0100	n0101-0----0--1----0---0-1-11000
16:	--01-----------------------n0011	010001110----------------00101n0
17:	------------------------------1n-	n1000-0----1--1----1---0-u-10011
18:	1----------------------------0--	01000-0----1--1----0---0-0-011u0
19:	--------------------------------	n00110100----------------0001011
20:	--------------------------------	n0110-0----1-------0-----0-000u1
21:	------------------------------n-	u1100-1------------------u-10111
22:	--------------------------------	00001-1------------------0-00110
23:	------------------------------n-	n1011-1----0-------0-----u-11001
24:	--------------------------------	u0000-0------------------1-11100
25:	------------------------------n-	01101-1------------------u-10111
26:	--------------------------------	u1010-1----0-------1-----0-011u0
27:	--------------------------------	01001-1------------------0-01110
28:	--------------------------------	u0000-0------------------1-11011
29:	--------------------------------	u0111-0------------------0-00010
30:	--------------------------------	01101-1------------------1-10010
31:	--------------------------------	10110-1------------------0-01001
32:	--------------------------------	00111-1------------------1-00100
33:	--------------------------------	01011-1------------------1-11101
34:	--------------------------------	00010-0------------------0-010u0
35:	------------------------------u-	10001-0------------------n-10110
36:	--------------------------------	11100-0------------------0-000u1
37:	--------------------------------	n0010-0------------------0-001u0
38:	------------------------------u-	n1101-0------------------n-11110
39:	--------------------------------	n1100-1------------------0-001n0
	...	...

Table 5. Steps 1 to 39 of the main differential path of the first block.

i	A_i	W_i
	...	...
40:	--------------------------------	n1111-0------------------0-10000
41:	--------------------------------	n0010-1------------------0-11010
42:	--------------------------------	n0100-0------------------1-110u1
43:	------------------------------u-	00000-1------------------n-01010
44:	--------------------------------	00011-0------------------0-100n0
45:	--------------------------------	n0111-1------------------1-10110
46:	--------------------------------	n0111-1------------------0-00010
47:	--------------------------------	u0010-1------------------1-00000
48:	--------------------------------	01101-0------------------0-010n0
49:	------------------------------n-	11111-1------------------u-10011
50:	--------------------------------	01000-1------------------0-100u0
51:	--------------------------------	u1110-1------------------0-10010
52:	--------------------------------	n1101-1------------------1-11110
53:	--------------------------------	n0001-1------------------1-001u0
54:	------------------------------u-	11011-0------------------n-11110
55:	--------------------------------	10001-0------------------0-000n0
56:	--------------------------------	n0111-1------------------0-001n1
57:	------------------------------n-	n0110-1------------------u-11101
58:	--------------------------------	u1110-1------------------1-11001
59:	------------------------------n-	u1110-0------------------u-010u1
60:	------------------------------u-	n1111-1------------------n-100n1
61:	--------------------------------	01010-0------------------0-010n1
62:	--------------------------------	01111-1------------------1-11111
63:	--------------------------------	10011-1------------------0-00010
64:	--------------------------------	n1000-0------------------0-10110
65:	--------------------------------	01000-0------------------1-00011
66:	--------------------------------	01000-0------------------0-101u1
67:	------------------------------u-	01001-0------------------n-01001
68:	--------------------------------	10001-0------------------0-100u0
69:	--------------------------------	u0010-1------------------1-11000
70:	--------------------------------	u1010-0------------------1-011n1
71:	------------------------------n-	u0101-0------------------u-01101
72:	--------------------------------	00011-1------------------0-100u0
73:	--------------------------------	n1010-1------------------0-11000
74:	--------------------------------	n1100-0------------------0-10010
75:	--------------------------------	u1110-1------------------1-110n1
76:	------------------------------n-	11011-1------------------u-00100
77:	--------------------------------	00111-0------------------1-000n1
78:	--------------------------------	n0011-0------------------1-11101
79:	--------------------------------	u0101-0------------------1-01000
80:	--------------------------------	

Table 6. Steps 40 to 80 of the main differential path of the first block.

i	A_i	W_i
-4:	1110110100001010100011101010 10u1	
-3:	01000110011100010110101100101000	
-2:	10010000011011111010001110100111	
-1:	11100110001011011100010100001001	
00:	01011110101010111100001100111101	10011010011101101100111 10u1100n0
01:	u0111011010011100010111unn1010n1	00000100101110011001010 10u111101
02:	1101001100111001101 1u000110u0111	11101110001001000010001 00n111101
03:	111001111000010u000unnnnnn000100	10011010010001100110010 01n110101
04:	u0100101unn01010000u100011110110	u0111000010000000000101 01u1101u0
05:	n000un001000011u00100000000nn0n0	u0111000011000000000011 000010010
06:	nnn00100010111100111 00n1nu1u011u	u0001011011111101000010 11u101010
07:	10nuuuuuuuuuuuu11100n00un0u1001	11100001011111111111011 000111100
08:	0001111100000000unnn11010001n001	10100010111100011011100 01u1001n1
09:	0000011111111111111110001n111un111	u1001010000001111001100 10n1101u0
10:	1110110110111111110100nu111uu011	00000010111100001010011 111001011
11:	00111111001000101001101 1uu0n000u0	11110111011001001110101 01n1110n0
12:	0100011010001110001111 11nuu1u011	001101111111-------------n111010
13:	101010000000----0--------01111u0	01010--------------------u0000u1
14:	00110001-----------------1010010	10010------0----1--------00110n1
15:	10011--------------------10101n0	n0110------0----1--------0111110
16:	0-------------------------011000	10000--------------------11010u1
17:	1-----------------------------n-	u1000------1----1--------u100111
18:	0-------------------------------	01100------1----0--------01111u0
19:	--------------------------------	n1010--------------------1110100
20:	--------------------------------	u1000------1-------------10000n1
21:	------------------------------n-	n1101--------------------u010011
22:	--------------------------------	11101--------------------1100010
23:	------------------------------n-	u1011------0-------------u110101
24:	--------------------------------	n1001--------------------0010110
25:	------------------------------u-	00010--------------------n001011
26:	--------------------------------	u0001------1-------------01110u0
27:	--------------------------------	10111--------------------0011001
28:	--------------------------------	n1110--------------------1101001
29:	--------------------------------	u0000--------------------0010100
30:	--------------------------------	01000--------------------0001001
31:	--------------------------------	01011--------------------1000101
32:	--------------------------------	00101--------------------1010111
33:	--------------------------------	11000--------------------0010001
34:	--------------------------------	01110--------------------00000n0
35:	------------------------------n-	10101--------------------u101001
36:	--------------------------------	10011--------------------10110u1
37:	--------------------------------	n1000--------------------01100u0
38:	------------------------------u-	n1001--------------------n010100
39:	--------------------------------	n0001--------------------11000n0
	...	...

Table 7. Steps 1 to 39 of the main differential path of the second block.

i	A_i	W_i
	...	...
40:	--------------------------------	u0101--------------------1001001
41:	--------------------------------	n0100--------------------0010111
42:	--------------------------------	u0000--------------------01100u1
43:	------------------------------u-	00111--------------------n101101
44:	--------------------------------	10001--------------------01011n0
45:	--------------------------------	n0011--------------------1010000
46:	--------------------------------	n0011--------------------1100111
47:	--------------------------------	n0011--------------------0011000
48:	--------------------------------	11101--------------------10011u0
49:	------------------------------u-	01010--------------------n001000
50:	--------------------------------	01110--------------------11100n0
51:	--------------------------------	n0111--------------------0111000
52:	--------------------------------	n0001--------------------1101011
53:	--------------------------------	n0100--------------------11100u0
54:	------------------------------u-	11000--------------------n000010
55:	--------------------------------	00111--------------------00001n0
56:	--------------------------------	u1100--------------------10001u0
57:	------------------------------u-	u0001--------------------n110000
58:	--------------------------------	n1000--------------------1101011
59:	------------------------------u-	u1111--------------------n0000u1
60:	------------------------------u-	n0010--------------------n0100n0
61:	--------------------------------	01100--------------------10100n1
62:	--------------------------------	11001--------------------0101000
63:	--------------------------------	01100--------------------0000100
64:	--------------------------------	n0011--------------------0101001
65:	--------------------------------	00101--------------------0101000
66:	--------------------------------	01011--------------------11101n0
67:	------------------------------n-	11111--------------------u100000
68:	--------------------------------	11110--------------------10100n1
69:	--------------------------------	n0100--------------------1010011
70:	--------------------------------	n0010--------------------00011n0
71:	------------------------------n-	n0100--------------------u100001
72:	--------------------------------	10011--------------------10101u1
73:	--------------------------------	n1001--------------------0010111
74:	--------------------------------	n0101--------------------1101110
75:	--------------------------------	u1111--------------------11001n1
76:	------------------------------n-	01100--------------------u111110
77:	--------------------------------	00001--------------------11010n0
78:	--------------------------------	n0111--------------------1101000
79:	--------------------------------	n0001--------------------0110011
80:	--------------------------------	

Table 8. Steps 40 to 80 of the main differential path of the second block.