Hash Functions and the (Amplified) Boomerang Attack

Antoine Joux[1,3] and Thomas Peyrin[2,3]

[1] DGA
[2] France Télécom R&D
thomas.peyrin@orange-ftgroup.com
[3] Université de Versailles Saint-Quentin-en-Yvelines
antoine.joux@prism.uvsq.fr

Abstract. Since Crypto 2004, hash functions have been the target of many attacks which showed that several well-known functions such as SHA-0 or MD5 can no longer be considered secure collision free hash functions. These attacks use classical cryptographic techniques from block cipher analysis such as differential cryptanalysis together with some specific methods. Among those, we can cite the neutral bits of Biham and Chen or the message modification techniques of Wang *et al.* In this paper, we show that another tool of block cipher analysis, the boomerang attack, can also be used in this context. In particular, we show that using this boomerang attack as a neutral bits tool, it becomes possible to lower the complexity of the attacks on SHA-1.

Keywords: hash functions, boomerang attack, SHA-1.

1 Introduction

The most famous design principle for dedicated hash functions is indisputably the MD-SHA family, firstly introduced by R. Rivest with MD4 [16] in 1990 and its improved version MD5 [15] in 1991. Two years after, the NIST publishes [12] a very similar hash function, SHA-0, that will be patched [13] in 1995 to give birth to SHA-1. This family is still very active, as NIST recently proposed [14] 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. Basically, MD-SHA family hash functions use the Merkle-Damgård extension domain and their compression function is build upon a 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.

The first cryptanalysis of a member of this family dates from Dobbertin [7] with a collision attack against MD4. Then, Chabaud-Joux [5] provided the first theoretical collision attack against SHA-0 and Biham-Chen [1] introduced the idea of neutral bits, which led to the computation of a real collision with four blocks of message [2]. Later on, a novel framework of collision attack, using modular difference and message modification techniques, surprised the cryptography community [19,23,24,22]. Those devastating attacks broke a lot of hash functions, such as MD4, MD5, SHA-0, SHA-1, RIPEMD or HAVAL-128.

Even if SHA-1 is theoretically broken (with 2^{69} message modifications), the computational power needed in practice is too important and the question arise that when will someone be able to come up with a real collision. Recently [20,21], it has been claimed that the complexity of this attack can be improved up to 2^{63} message modifications.

In this article we study the application of boomerang attacks, originally introduced by D. Wagner [18] for block ciphers, to the case of hash functions. In particular, we show that this very generic method may improve the already known collision attacks against various hash functions when used with classic improvements such as neutral bits or message modification. Although this method is generic, some aspects are closely related to the particular hash function one is planning to attack. Thus, we give a practical proof of concept by applying this improvement to SHA-1. We provide here the detailed constraints and advantages of this particular case. Finally, we are able to present a novel attack against SHA-1, dividing the work factor by 32 from the previous attacks.

An independent work by Klima, describing tunnels in MD5 was posted on ePrint [10], shortly before our first public presentation of the boomerang attack [8] applied to hash function. Each tunnel in Klima's work can be decomposed into a collection of auxiliary differential in our attack. Note that due to the simple message expansion in MD5, the tunnel can be directly observed in a preexisting attack. In our SHA-1 application, a specific differential attack must be constructed to accommodate the auxiliary differentials.

The paper is structured as follows. In Section 2, we recall the concept of boomerang attack for block ciphers and in Section 3 we show how this concept can be applied to hash functions. In particular, we give two different possible approaches for using this method. Then, in Section 4, we treat a practical example with the case of SHA-1. We explain all the specific aspects of the application of boomerang attacks for SHA-1 and show that this method leads to improvements for a collision attack. Finally, we draw conclusions and give future works in Section 5.

Notations. In the following, $+$ will stand for the addition on 32-bit words (modulo 2^{32}) and $\oplus$ will represent the bitwise exclusive-OR. The left (resp. right) bit rotation will be denoted $\lll$ (resp. $\ggg$), and $\wedge$ (resp. $\vee$) is the bitwise AND (resp. OR). The j-th bit (modulo 32) of a 32-bit word X is denoted X^j and the bitwise complementary of X will be denoted $\overline{X}$.

2 The Boomerang Attack

The boomerang attack was proposed by D. Wagner as a tool for the cryptanalysis of block ciphers in [18]. It allows to weave two partial and independent differential characteristics together into a global attack on the block cipher. The basic idea

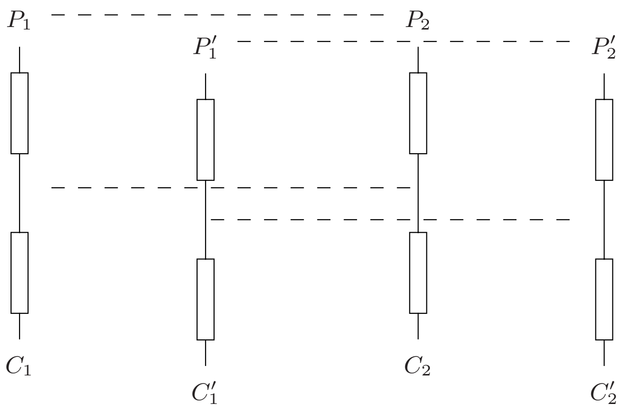

Fig. 1. Schematic view of the boomerang attack on block ciphers

is quite simple. Assume that we are given a first differential characteristic D_1 on the first half of the block cipher which predicts that an input difference Δ leads to an output difference Δ^* with probability p_1. Then, assume a second differential on the second half which predicts that an input difference ∇^* leads to an output difference ∇ with probability p_2. Using these two differentials, we can draw a diagram (see Figure 1) that involves four plaintext/ciphertext pairs.

This diagram can be turned into an attack as follows. First, the attacker choses a random plaintext and asks for the encryption of both this plaintext P_1 and of the plaintext P_2 obtained by xoring P_1 with Δ. The resulting ciphertexts are denoted by C_1 and C_2. After that, the attacker computes C'_1 by xoring C_1 with ∇ and C'_2 by xoring C_2 with ∇. Then, he asks for the decrypted plaintext P'_1 and P'_2. The key idea of the attack is to remark that when the pair (P_1, P_2) follows the Δ differential path and both decryptions follow the ∇ differential path, then the intermediate values corresponding to P'_1 and P'_2 have the correct difference Δ^*. If in addition (P'_1, P'_2) is also a correct pair for Δ then the attacker finds that $P'_1 \oplus P'_2$ is Δ.

Assuming independence between the four instances of differential paths, we obtain a probability of success $p_1^2 p_2^2$. Basically, this yields a distinguisher that allows us to make the difference between the block cipher and a random permutation.

3 Adapting the Boomerang Attack to Hash Functions

At first, since many hash functions are based on block ciphers, it seems tempting to directly apply the boomerang attack to these hash functions, however several obstructions are quickly encountered and prevent this straightforward approach from working. In particular, the need for decryption, which is an essential part of the boomerang attack, can not be available in the context of hash functions.

Yet, we now show that the boomerang attack, and more specifically its chosen plaintext variant (so-called amplified boomerang attack [9]), can be adapted to the hash function setting and yields improvements compared to previously known differential attacks. The basic idea to adapt the boomerang attack is to use, in addition to the good global differential path used in the now classical differential attacks, several partial differential paths which are very good on a limited number of steps but fail to cover the complete compression function. In order to combine these differential paths together, we use the same basic diagram as with the boomerang attack against block ciphers. However, some specific obstructions appear and need to be removed. The first problem, that we already described when considering the direct application, is the fact that in order to obtain collisions, we cannot use the compression function in the backward direction. The second problem is that we no longer have a nice symmetry with two characteristics playing almost the same role. Instead, there is a main differential path which is our target and some auxiliary paths which help in applying the main one.

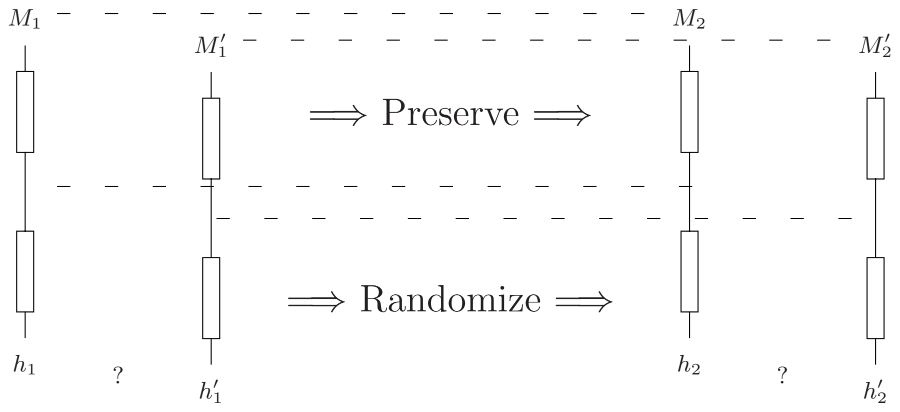
M_1
M'_1
M_2
M'_2
$\Longrightarrow$ Preserve $\Longrightarrow$
$\Longrightarrow$ Randomize $\Longrightarrow$
h_1
?
h'_1
h_2
?
h'_2

Fig. 2. Schematic view of the boomerang attack on hash functions

Our adapted boomerang attack on iterated hash functions is based on a simple basic block, which we now describe. We start from a basic differential path on an iterated hash function. For the sake of simplicity, we assume that this differential path is of the simple type which yields a collision after a single iteration. Generalizing this description to near-collisions or multiple iterations is a straightforward matter. The basic differential path consists in a message difference Δ, possibly completed by a list of restrictions on acceptable messages, such that the two single block messages M and $M \oplus \Delta$ collide, with probability p_Δ. As usual, this probability do not take into account the so-called early steps where parts of the message M can be chosen independently of each others. From now on, we split the rest of the steps into two main parts, the middle steps and the late steps[1]. To each part, we associate a corresponding probability p_M for

[1] In some multi-block attacks, some of the final steps can be treated specifically, ignoring partial misbehaviors which can be corrected in the subsequent blocks.

the middle steps and p_F for the late steps. Under a classical step independence assumption, we have $p_\Delta = p_M \cdot p_F$. The goal of the boomerang based attack is to improve p_M and thus the total complexity of the process. For this, we use an auxiliary differential path that covers both the early and the middle steps as a tool. Assume such an auxiliary differential path, that predicts that, with probability p_δ two messages M and $M \oplus \delta$ yield, after the middle steps, two intermediate internal states with some prescribed (not necessarily null) difference. Take a message pair M and $M' = M \oplus \Delta$ that conforms to the main differential path on the early and middle steps. Assume that both $(M, M \oplus \delta)$ and $(M', M' \oplus \delta)$ conform to the auxiliary differential path. Then, we see that the internal states differences cancel out, and that the pair $(M \oplus \delta, M' \oplus \delta)$ also conforms to the main differential up to the beginning of the late steps (see Figure 2). Assuming independence, this pair yields a collision with probability $p_\delta^2 \cdot P_F$.

The basic block we just described is quite promising. Indeed, when $p_\delta^2 < p_M$ we can expect an improved attack. However, matters are not that simple. Indeed, unless we are given a first pair (M, M'), we cannot construct the second pair. Thus, the basic block, by itself, at best doubles the number of candidate pairs. Luckily, when a large number of auxiliary differential paths can be found, which is a reasonable hypothesis since we are dealing with a small number of steps, we can apply the basic block many times. Assuming that $p_\delta = 1$, for each of t auxiliary differentials, we amplify a single candidate pair into 2^t pairs. Of course, we need to arrange the auxiliary differentials to make sure that they do not overlap or present other similar incompatibilities. When p_δ is smaller than 1 (but not too small), we still amplify a single pair into many.

After this overview of our adapted boomerang attack, the reader may rise two important objections. The first one is the fact that the independence hypothesis is extremely unnatural, because all these messages pairs are extremely correlated. Experimentally, this hypothesis is *false*, however, we remarked that for well-chosen differential characteristic, the bias induced by the dependencies is playing for the attacker and not against him. The main gain is that, since M and $M \oplus \Delta$ gives similar computations, the overall success probability of the two copies of each auxiliary differential is usually nearer to p_δ than p_δ^2. The second objection is that, at first, the early steps do not seem to come for free for the auxiliary differentials. This would be a major problem, since we want p_δ to be much better than p_M. In fact, we propose two different ways of putting together the message construction and the auxiliary differentials choice in order to effectively overcome this objection. Depending on the hash function under consideration and the properties of the differential characteristics in use, each has its own advantages.

3.1 Neutral Bits Approach

The first way to use the adapted boomerang attack is to note its similarity with the neutral bit technique proposed by Biham and Chen [1] at Crypto'04. There, the authors remarked in the case of `SHA-0` that given a differential path, corresponding to our main path, it is possible to find so-called *neutral bits*. For a message pair that conforms to the differential characteristic up to some reference step, a neutral bit[2] is a bit of the message which when its value is flipped yields a new message pair that still conforms to the main path up to the reference step. In [1], the neutral bits are found using a guided exhaustive search technique. We argue that using auxiliary differential paths in place of or in addition to these neutral bits, leads to a better attack. Otherwise, this way of implementing our attack closely follows the method of Biham and Chen. The first step is to identify among a large list of candidate auxiliary differential paths those which works for the current message pair. Once this is done, we check, one pair at a time, whether the acceptable differentials are mutually compatible. Even without writing down the explicit algebraic conditions which need be satisfied for each differential, it is clear that this pairwise compatibility check only works for pairs of differential which do not strongly interact[3]. Then, build a large clique of mutually compatible differentials in the graph of pairwise compatible ones.

Once this clique is build, assume that it contains t auxiliary differentials and, using the basic technique presented above, construct the 2^t pairs of messages obtained by adding any subset of these differentials to the original message. We expect that a good proportion of the derived pairs conforms to the main characteristic up to the start of the final steps.

The main drawback of this technique is that the auxiliary differentials do not take advantage of the free early steps. Indeed, the original message pair is chosen independently of them, thus some probability must be paid for the early steps. This prevents us from using auxiliary differentials which are very good in the middle range but have a low probability of success in the early steps. It can be improved by trying to use the free steps both on the main characteristic path and on the auxiliary paths. However, if too many auxiliary paths are considered during a single step, the probability of making a correct choice becomes too low and no initial message pair can be constructed. The second approach given below gives a way out of this dilemma.

3.2 Explicit Conditions Approach

In order to get a good set of auxiliary characteristics, it is preferable to construct the first message pair carefully, forcing it to conform both to the main differential path and to the chosen auxiliary paths in the early steps. In order to do this, we should write down explicit conditions on bit values that are sufficient for each auxiliary characteristic to hold (or at least such that p_δ is increased). Once this is done, we can check whether the condition of the various characteristics are mutually compatible and, if so, we can choose the message values for each of the early steps, except the last one or two, in order to satisfy these explicit

[2] Here the term bit is taken in its information theoretic sense and may be a group of several elementary message bits which are all flipped simultaneously.

[3] Some long range interaction, such as carry propagation over several bits may be overlooked. However, they rarely occur anyway and can be ignored in a first approximation.

conditions. In the sequel, we call the partial message resulting from these choices a message seed. Note that, while simple as a principle, this approach requires a lot of specific work for each hash function in order to find a good way of writing and satisfying these explicit conditions.

After building a message seed, we can complete it in many ways on the one or two missing blocks to get an initial message pair. If the pair conforms to the main differential path far enough, we can use the neutral bit technique described above on the message pair, using as neutral bits the set of auxiliary paths that we have forced into the message. Compared to the straight neutral bit approach, the resulting auxiliary paths on the message pair are much more effective. Moreover, with this approach, we may be able to build auxiliary differential paths remaining conformant for more steps than in the case of neutral bits. In other words, the final steps will contain less steps than in the classical attacks such as neutral bits or message modification, and the total complexity will therefore decrease. To resume, while more complicated to set up in practice, this approach yields much better attacks.

For this method to succeed, we have to be able to build a main differential path containing all the sufficient conditions needed for every auxiliary differential paths we are planing to use. In order to make the approach efficient in practice, it would be very useful to have an automated tool that generates a main differential path satisfying those conditions. The availability and efficiency of such a tool greatly depend on the hash function we are considering. In the sequel, we show that the path generator proposed by De Cannière and Rechberger in [3] for SHA-1 can be used together with our boomerang approach.

4 Application to SHA-1

In this section, we show how our new attack applies for the case of SHA-1. After a short description of the algorithm and the state-of-the-art attacks, we explain how to build auxiliary differential paths, place them in a main differential path and use them during the collision search.

4.1 A Short Description of SHA-1

SHA-1 is a 160-bit dedicated hash function based on the design principle of MD4. Like most hash functions, SHA-1 uses the Merkle-Damgård paradigm [6,11] and thus only specifies a compression function. After a padding process, the message is divided into k blocks of 512 bits. At each iteration of the compression function h, a 160-bit chaining variable cv_i is updated using one message block m_{i+1}, i.e. $cv_{i+1} = h(cv_i, m_{i+1})$. The initial value cv_0 (also called IV) is predefined and cv_k is the output of the hash function.

The SHA-1 compression function is build upon the Davies-Meyer construction. It uses a function E as a block cipher with cv_i for the message input and m_{i+1} for the key input, a feed-forward is then needed in order to break the invertibility of the process: $cv_{i+1} = E(cv_i, m_{i+1}) \oplus cv_i$. 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 $(A_i, B_i, C_i, D_i, E_i)$. Since more message bits than available are utilized, a message expansion is therefore defined.

Message expansion. First, m_i is split into 16 32-bit words $M_0, \ldots, M_{15}$. These 16 words are then expanded linearly into 80 32-bit words W_i, as follows:

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

State update. First, the chaining variable cv_i is divided into 5 32-bit words to fill the 5 registers $(A_i, B_i, C_i, D_i, E_i)$. Then we apply 80 times the following transformation:

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

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

Table 1. Boolean function and constants in SHA-1

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

We refer to [13] for a more exhaustive description. Note that all updated registers but A_{i+1} are just rotated copies, so we only need to consider the register A at each iteration. Thus, we have:

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

4.2 Previous Attacks on SHA-1

Lots of research on SHA-1 has been conducted recently, but the major breakthrough has been published by Wang *et al.* [22]. They provided the first collision attack against the full SHA-1 algorithm, requiring only 2^{69} message modifications, which is lower than the 2^{80} hash computations expected for an ideal 160-bit hash function. This attack is possible thanks to a non-linear main differential path for SHA-1, given in the original paper. They also use a tool called *message modification technique* that allows to build a message pair of messages conforming to the main differential on the early and middle steps (approximatively step 22), thanks to clever modifications of message bits. Later, an unpublished result [20,21] claimed that using another non-linear main differential with more complex message modification techniques, one can keep the conformance up to step 25 approximatively and thus lower the complexity down to 2^{63} message modifications. The main problem with this approach is that message modifications can be costly during the collision attack and only the ones for the differential path in [22] are known. Note however that some recent work [17] tried to theorize this method.

Very recently, an interesting approach has been published by De Cannière and Rechberger [3] in order to find non-linear main differential paths in an automatic way. By introducing a sharp method to compute the probability of conformance and the number of messages (called nodes) one has to deal with at each step, they can use a heuristic algorithm to converge to a valid non-linear main differential path (prebuild from the Wang *et al.*'s disturbance vector). This algorithm allowed to compute a 2-block collision on a 64-step reduced version of SHA-1 (and more recently on a 70-step reduced version [4]). Note that this automatic tool did not improve the complexity of the previously explained collision attack against full SHA-1, since a non-linear main differential path was already known for that case.

We will show that using the boomerang attack for hash functions, we can improve the collision attacks for SHA-1 of a factor 32. We managed to place five auxiliary differentials maintaining conformance up to step 28 or even further (compared to step 25 approximatively for neutral bits or message modification). Another advantage of our new method is that once an auxiliary differential path is settled, the cost for using it is null, unlike the message modification case which can be quite demanding in terms of complexity [17]. The complex part of the boomerang attack in the explicit conditions approach only takes place during the main differential construction.

4.3 Building Auxiliary Differential Paths

In this section, our goal is to give an insight on how to build auxiliary differential paths for SHA-1. We want those paths to conform to the main differential one as far as possible. Since in the explicit conditions approach the main path is not yet known at this stage, a natural method would be to find auxiliary differentials leading to a collision on a late step. We also want the auxiliary differential paths to be as light as possible. If not so, the number of necessary conditions to have $p_\delta \simeq 1$ would quickly grow and this would be a problem while using the main path automated generator, which needs a lot of degrees of freedom in the message and in the registers.

Building a good auxiliary differential path is very close to building a main differential one. As observed in the latter case, the sparser the better. So in order to find good paths, we will use a well known tool for SHA-1 or SHA-0, introduced in [5]: the local collisions. This technique seems to make the attacker's job much more easier and minimize the number of differences one has to deal with. The idea is to avoid the inserted differences (called perturbations or disturbance vector) to spread among the registers by applying the necessary corrections on the expanded message (the perturbations and the corrections will therefore define the difference in the message). The problem arise that since the message is expanded, we do not have full control over the disturbance vector and thus this vector must respect the expansion as well. This is important when one has to build a main differential path, but here the problem is much more relaxed as we only deal with a few number of steps.

Local collisions. By inserting a difference on W_i^j at step $i+1$, another difference will appear on A_{i+1}^j. Note that a propagation of the difference to other bits of A_{i+1} may occur due to carry effect. To avoid this, we can set $W_i^j = A_{i+1}^j$. Then, at step $i+2$, the difference in A_{i+1}^j needs to be corrected and this can be done by setting $W_{i+1}^{j+5} = \overline{A_{i+1}^j}$. For steps $i+3$ to $i+5$, the behaviour highly depends on the boolean function f_i we are using (and thus the round we are into). Finally, at step $i+6$, we set $W_{i+5}^{j-2} = \overline{A_{i+1}^j}$ to correct the difference in A_{i+1}^j. At this point, we achieve the local collision: no more difference will appear in the next steps. We give in Table 2 all the constraints corresponding to the first round case ($f_i = f_{IF}$). If one respects all those constraints, the local collision occurs with probability 1.

Now that we know how to build local collisions, how do we use them ? We want the number of perturbations inserted in the early steps to be as low as

Table 2. Constraints for a local collision with a perturbation on W_i^j for the first round of `SHA-1`

step	**type**	**constraints**
$i+1$	no carry	$W_i^j = a$, $A_{i+1}^j = a$
$i+2$	correction	$W_{i+1}^{j+5} = \overline{a}$
$i+3$	no correction	$A_{i-1}^{j+2} = A_i^{j+2}$
	correction	$A_{i-1}^{j+2} \neq A_i^{j+2}$, $W_{i+2}^j = \overline{a}$
$i+4$	no correction	$A_{i+2}^{j-2} = 0$
	correction	$A_{i+2}^{j-2} = 1$, $W_{i+3}^{j-2} = \overline{a}$
$i+5$	no correction	$A_{i+3}^{j-2} = 1$
	correction	$A_{i+3}^{j-2} = 0$, $W_{i+4}^{j-2} = \overline{a}$
$i+6$	correction	$W_{i+5}^{j-2} = \overline{a}$

possible (at most 5 in practice), in order to minimize the number of constraints on the message and the registers. Moreover, we want the auxiliary path to collide at some middle step k (with $k \geq 25$ in practice). It seems pretty clear that one will achieve this minimization by setting all the corrected perturbations in the 16 first message blocks on the same bit position, as remarked for the main differential path. Our goal is thus to have the first uncorrected perturbation as late as possible. By brute-forcing all the possibilities of this 16-bit mask and all the possibilities of propagation and corrections into the local collisions (corresponding to the f_{IF} case for round 1), we managed to find a lot of candidates (i.e. no difference in registers A_{k-4} to A_k). However, we added a filter: no perturbation should occur from W_{15} to W_{k-1} (note that a perturbation on W_k necessarily exists since we have the first difference on A_{k+1}). Indeed, a corrected perturbation introduced after step 14 would force some constraints on the message and the register outside the early steps where we have degrees of freedom. This would harden the final search of colliding messages. We even sharpen the filter by setting no perturbation from W_{11} to W_{k-1} to avoid problems with wrong bit position corrections due to the rotation in the expansion for the case `SHA-1` (if the perturbation would occur on a bit j, some corrections would apply on bit $j+1$ and thus introduce unwanted differences). Finally, our auxiliary differential path will have no difference from register A_{12} to A_k. Note that a general rotation on the bit position does not change the validity of an auxiliary path.

We give in Table 3 the disturbance vector and the differences on the message for an auxiliary differential path with only three perturbations. In this example, the first uncorrected perturbation (underlined) comes on W_{24}^j and thus we get a collision at step 24. Here the three perturbations apply on step 1, 3, 11 and the corresponding constraints to force $p_\delta = 1$ are depicted in Table 4. Each bit a, b, c, d, e, f can take any value, as long as the $\overline{a}$, $\overline{b}$, $\overline{c}$, $\overline{d}$, $\overline{e}$, $\overline{f}$ constraints are fulfilled.

Table 3. Example of an auxiliary differential path, with the perturbation mask and its corresponding message differences for the 32 first steps. The rotation in the expansion is not taken in account.

	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

We previously claimed that we were looking for an auxiliary differential path with $k \geq 25$, so why do we presented a $k = 24$ one ? In fact, even if a perturbation appears at step 25, there is a great probability, depending on the main differential path, that our pair remains conformant for some more steps[4]. We experimentally

[4] Said in other words, our auxiliary differential path will have a non-zero output difference.

Table 4. Example of an auxiliary differential path in the case $j = 2$: the constraints on the registers and on the message blocks. The MSB's are on the right and "-" stands for no constraint.

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

observed that this greatly depends on the bit position j where we plan to apply our auxiliary path, and the perturbation vector of the main path. For some very few values of j, the auxiliary differential has a small probability to succeed. However, in general, we have a good probability that a first perturbation at step n does not change the main differential conformance of a pair of messages up to step $n + 4$. We just have to choose the j values by avoiding critical positions.

4.4 Placing Auxiliary Differential Paths

We set ourselves in the case of a 2-block collision attack for SHA-1. For more details, we refer to [3]. The first part is thus to find a valid main path for the first block (with no difference on the IV). At this stage, our goal is to get the biggest clique of auxiliary differential paths by placing them in a main one. Since the main differential path automated tool from De Cannière and Rechberger is a heuristic algorithm, placing auxiliary paths in a main one is not a formal science. We tried different techniques but the best one seemed to be to force as much space between the constraints as we could. Note that when placing several auxiliary differential paths, some of them may have constraints in common. Even if not dramatic, we preferred to avoid this situation and strengthen the independence between the auxiliary paths (and thus use them as a clique as for neutral bits). Moreover, some positions are forbidden as the constraints on the message must apply on no-difference bits of the message only (otherwise, one of the message pair would not follow the auxiliary path). Lots of parameters are available when implementing or using the main differential automated tool and they highly influence the number of auxiliary constraints one can force. However, due to space restrictions, we omitted those details here.

We quickly recall in Table 5 the notations used in [3], but we encourage the reader to glance through the original paper. The final main path presented in Tables 8 and 9 contains the constraints of five independent auxiliary paths given in Table 3 and Table 4 at positions $j = \{9, 12, 15, 18, 21\}$.

Note that the auxiliary differential path used here has constraints on the IV ($A_{-1}^{j+2} = A_0^{j+2}$ in Table 4). It expresses the equality between two bits and thus happens with probability 1/2 for each auxiliary differential. The prepended message computed to get the IV used in Table 8 is given in Table 6.

4.5 Using Auxiliary Differential Paths

Once a differential path is settled, one can easily generate a message instance conformant up to the end of the early steps since at this point the message blocks can be fixed independently. De Cannière and Rechberger use this fast generating technique coupled with a refinement of the differential path. Advanced approaches such as neutral bits or message modifications can decrease the complexity of the final attack, even if their power is reduced for `SHA-1` compared to the `SHA-0` case[5]. The boomerang attack for hash functions can be viewed as a generalization of those techniques and thus can be used as a neutral bits or message modification tool:

Neutral bit based. The easiest approach is to use a generalization of Biham and Chen neutral bit implementation guidelines together with two levels of message diversification. First, one constructs a base message with a large clique of simultaneously neutral bits which are in addition compatible with the auxiliary differential path. Then, one launches an enumeration that starts from this initial message and applies the neutral bits (using a Gray code encoding for efficiency). This yields many message pairs that follow the main differential path quite far. When the enumeration finds a message conformant up to round 25, a second level of enumeration diversifies this message using the auxiliary paths. The advantage of this technique is that it is quite easy to implement and that the neutral bits and the auxiliary paths can be addressed using very similar treatments. The main drawback is the gap between the range of ordinary neutral bits and the range of the auxiliary paths, which is a bit too wide and thus wastes degree of freedom in the message, compared to the theoretic complexity gain.

Message modification based. From a theoretical point of view, a message modification approach seems better. Indeed, the current best attack is message modification based and using it avoids the initial loss seen with the neutral bit approach. However, in addition to the implementation difficulties, using message

[5] some conditions on the message words coming from the late steps have to be fulfilled and in `SHA-1` the rotation in the message expansion greatly increase the number of impacted message bits. This harden the neutral bits or message modification work since one has to check that the conditions remain valid after their use.

modification involves a much higher cost per message pair than the neutral bit approach. As a consequence, the apparent theoretical gain is less clear in practice.

Right now, our implementation of these ideas is not fast enough to allow full scale attacks. However, once an initial pair is found, the multiplicative effect works very well. For example, in Table 7 is the first message of a pair conformant until step 29, following the differential path from Table 8. Using the auxiliary differential paths provide 2^5 new conformant messages, the conformance limit is always between step 27 and 29. Note that this group of message words was generated using the neutral bit technique. This has the side effect of slightly changing the main characteristic during the message generation. More precisely, some bits (a, b or c in Table 4) of the auxiliary characteristics are flipped. Bit a is changed for the characteristic in positions 9,12,15; bit b for 12,15,18 and bit c for 15,18. The flipped bits are underlined in the given message. Of course, the slightly modified characteristic is still correct and compatible with the auxiliary ones (the 5 auxiliary differential paths remain valid).

4.6 Complexity Analysis for a Full Collision Attack

The literature has provided two ways of computing the complexity of a 2-block collision attack against `SHA-1` : the number of conditions introduced by Wang *et al.* or the number of nodes introduced by C. De Cannière and C. Rechberger. Whatever the original collision attack we are using, our improvement decreases the complexity of a factor 32 since no message modification technique nor neutral bit can keep the conformance later than step 25. Moreover, the probability that a message being valid at step 25 is also valid at step 28 is lower than 2^{-5}.

We do not provide here any main path with auxiliary differentials for the second block since one needs the first block output values. However, experiments showed the same behaviour as for the first block case and the authors believe that the same technique can apply for the second part of the 2-block collision attack.

The reader could argue that we gave a differential path for the first block with a prepended message leading to an IV with chosen properties, and this will not be available for the second block stage of the attack. First, one has to note that the IV defined by the specifications of `SHA-1` is strongly structured. Moreover, in a 2-block collision for `SHA-1` the first block part costs much less than the second one (about a factor of 8), due to the possible misbehaviour of the final steps for the first block. Thus, by executing several times a first block research, the general complexity is not increased and we have enough degrees of freedom to start properly the second block: assuming the positions where we are placing the auxiliary differentials paths for the second block, the probability of satisfying the 5 constraints is 2^{-5} and 32 trials are required. However, this is not the case here since when reaching the end of the first block, the idea is to look at the available positions for including auxiliary differential. If enough positions are available, we try to construct a compatible main path. Thus, instead of having a single possibility with probability 2^{-5}, we have many. Experimentally, less than 4 tests of prepended messages are needed to apply the boomerang attack with five auxiliary paths.

5 Conclusion

In this paper, we showed that the boomerang attack which was initially devised as a cryptanalytic tool for block ciphers can be adapted to apply on iterated hash functions. Since the attacker model is quite different, due to the absence of keys and the impossibility to use a chosen ciphertext attack, the adaptation is not straightforward. Nonetheless, this new method leads to an improved cryptanalytic technique.

In order to illustrate this technique, we applied it to `SHA-1` and obtained a significant improvement for collision attacks on this hash function. We believe that this method would also be powerful against other hash functions. Applying boomerang attack against `SHA-0` or `MD5` would be an interesting research topic. It may also be worth looking for more general auxiliary differential paths, for example by letting some local collisions slightly behave in a non-linear manner. Another future work could be to find a way to place more auxiliary differential paths in the main differential one, and thus lower the final complexity.

Acknowledgements

The authors would like to thank Christophe De Cannière and Christian Rechberger for their helpful advices when implementing their non-linear differential path automatic search tool.

References

1. Biham, E., Chen, R.: Near-Collisions of SHA-0. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 290–305. Springer, Heidelberg (2004)
2. Biham, E., Chen, R., Joux, A., Carribault, P., Lemuet, C., Jalby, W.: Collisions of SHA-0 and Reduced SHA-1. In: Cramer, R.J.F. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 36–57. Springer, Heidelberg (2005)
3. De Cannière, C., Rechberger, C.: Finding SHA-1 Characteristics: General Results and Applications. In: Lai, X., Chen, K. (eds.) ASIACRYPT 2006. LNCS, vol. 4284, pp. 1–20. Springer, Heidelberg (2006)
4. Rechberger, C., De Cannière, C., Mendel, F.: In: Rump Session of Fast Software Encryption – FSE 2007 (2007)
5. Chabaud, F., Joux, A.: Differential Collisions in SHA-0. In: Krawczyk, H. (ed.) CRYPTO 1998. LNCS, vol. 1462, pp. 56–71. Springer, Heidelberg (1998)
6. Damgård, I.: A Design Principle for Hash Functions. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 416–427. Springer, Heidelberg (1990)
7. Dobbertin, H.: Cryptanalysis of MD4. In: Gollmann, D. (ed.) Fast Software Encryption. LNCS, vol. 1039, pp. 53–69. Springer, Heidelberg (1996)
8. Joux, A., Peyrin, T.: Message modification, neutral bits and boomerangs. In: Proceedings of *NIST 2nd Cryptographic Hash Workshop* (2006)
9. Kelsey, J., Kohno, T., Schneier, B.: Amplified Boomerang Attacks Against Reduced-Round MARS and Serpent. In: Schneier, B. (ed.) FSE 2000. LNCS, vol. 1978, pp. 75–93. Springer, Heidelberg (2001)

10. Klima, V.: Tunnels in Hash Functions: MD5 Collisions Within a Minute. ePrint archive (2006), `http://eprint.iacr.org/2006/105.pdf`
11. Merkle, R.C.: One Way Hash Functions and DES. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 428–446. Springer, Heidelberg (1990)
12. National Institute of Standards and Technology. FIPS 180: Secure Hash Standard (May 1993) available from `http://csrc.nist.gov`
13. National Institute of Standards and Technology. FIPS 180-1: Secure Hash Standard (April 1995) available from `http://csrc.nist.gov`
14. National Institute of Standards and Technology. FIPS 180-2: Secure Hash Standard (August 2002) available from `http://csrc.nist.gov`
15. Rivest, R.L.: RFC1321: The MD5 Message-Digest Algorithm (April 1992) available from `http://www.ietf.org/rfc/rfc1321.txt`
16. R.L. Rivest. RFC 1320: The MD4 Message Digest Algorithm (April 1992), `http://www.ietf.org/rfc/rfc1320.txt`
17. Sugita, M., Kawazoe, M., Imai, H.: Gröbner Basis based Cryptanalysis of SHA-1. In: Fast Software Encryption – FSE'07. LNCS, Springer, Heidelberg (2007), `http://eprint.iacr.org/2006/098.pdf` (to appear)
18. Wagner, D.: The Boomerang Attack. In: Knudsen, L.R. (ed.) FSE 1999. LNCS, vol. 1636, pp. 156–170. Springer, Heidelberg (1999)
19. Wang, X., Lai, X., Feng, D., Chen, H., Yu, X.: Cryptanalysis of the Hash Functions MD4 and RIPEMD. In: Cramer, R.J.F. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 1–18. Springer, Heidelberg (2005)
20. Wang, X., Yao, A.C., Yao, F.: Cryptanalysis on SHA-1. In: Proceedings of NIST Cryptographic Hash Workshop (2005)
21. Wang, X., Yin, Y.L., Yu, H.: New Collision Search for SHA-1. In: Rump Session of CRYPTO (2005)
22. Wang, X., Yin, Y.L., Yu, H.: Finding Collisions in the Full SHA-1. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 17–36. Springer, Heidelberg (2005)
23. Wang, X., Yu, H.: How to Break MD5 and Other Hash Functions. In: Cramer, R.J.F. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 19–35. Springer, Heidelberg (2005)
24. Wang, X., Yu, H., Yin, Y.L.: Efficient Collision Search Attacks on SHA-0. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 1–16. Springer, Heidelberg (2005)

Appendix

Table 5. Notations used in [3] for a differential path: x represents a bit of the first message and x^* stands for the same bit of the second message

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

Table 6. Message prepended to start with the IV used in Table 8

W_0	0x63e045ce	W_8	0x24b67e5d
W_1	0x362a3ed8	W_9	0x3898e2dd
W_2	0x5c333351	W_{10}	0x18be4543
W_3	0x76481862	W_{11}	0x60746d11
W_4	0x71a360ab	W_{12}	0x4cd56e7c
W_5	0x25e16eb9	W_{13}	0x1589d326
W_6	0x0419a9c2	W_{14}	0x19bab19c
W_7	0x5977272f	W_{15}	0x5fa6c656

Table 7. First message (in binary and hexadecimal) of an example pair following differential path from Table 8, conformant until step 29. Bits underlined are the bits flipped in the the main differential path from Table 8 due to the neutral bits technique.

M_0	1111110110011111$\underline{1}$11$\underline{1}$01$\underline{1}$111111011	0xfd9ff7fb
M_1	01110101000$\underline{0}$01$\underline{0}$10$\underline{0}$11111101110001	0x75053f71
M_2	0001110000011$\underline{1}$01$\underline{0}$11$\underline{1}$001100011111	0x1c1d731f
M_3	00000111$\underline{0}$01$\underline{1}$10$\underline{0}$000$\underline{0}$00$\underline{0}$10$\underline{0}$1111001	0x07380279
M_4	111101011010111010$\underline{0}$01$\underline{0}$00$\underline{0}$0101001	0xf5ae8829
M_5	001101011111101011$\underline{0}$01$\underline{0}$11$\underline{0}$1010011	0x35facb53
M_6	000100000111110$\underline{0}$10$\underline{1}$01$\underline{0}$1100011001	0x107cab19
M_7	101001101111111$\underline{0}$01$\underline{1}$00$\underline{0}$1101101001	0xa6fe6369
M_8	01001000001100111010100101011101	0x4833a95d
M_9	01100000000110110110100111101100	0x601b69ec
M_{10}	1010001101001$\underline{0}$10$\underline{0}$100111001100100	0xa34a4e64
M_{11}	01011100$\underline{1}$00$\underline{1}$11101011111100100111	0x5c9ebf27
M_{12}	10111011010000110101001001110111	0xbb435277
M_{13}	10100101011101110100110011010100	0xa5774cd4
M_{14}	111111100111101$\underline{1}$10$\underline{1}$1010000000000	0xfe7bb400
M_{15}	101101010011101$\underline{1}$10$\underline{1}$0110101101011	0xb53bad6b

Table 8. Steps 1 to 39 of the main differential path of the first block. The constraints needed for the first auxiliary differential path (in position j=9) are underlined.

i	A_i	W_i
-4:	00101001010011011100100101000111	
-3:	00000111100001000110010101100010	
-2:	11011000010000101001111101011111	
-1:	01011011110111101101$\underline{1}$01111010001	
00:	01000010101101110111$\underline{1}$01110011011	1uu11101100111110110--$\underline{0}$111111011
01:	n1n01011100101100100$\underline{1}$-$\underline{0}$100100110	nuu101-10001011--$\underline{1}$11111101u1n0n1
02:	1nu11--0111110111110$\underline{1}$101$\underline{1}$11111u1	--n11-----0-10-1111000$\underline{1}$10n0111uu
03:	nnu00-----0-00-0110000$\underline{1}$1$\underline{0}$111110n	x-nn-1--1--010100$\underline{0}$1001--$\underline{1}$u111001
04:	u010u11-0--00010010110-1$\underline{0}$10un0u1	uu-u0-------11-0--101100$\underline{1}$n1n10nu
05:	1001u00-0--000000000001u$\underline{0}$0011010	nn-u0------11010111--1--$\underline{1}$1n100u1
06:	011unnnnnnnnnnnnnnn1---110n001uu	00n-------1-1--1--001111$\underline{0}$0011001
07:	u110-01000000u010110nu111uu1010n	1nu001------1--1-100-1-1$\underline{0}$-un-0n-
08:	1111010111111---011unu110-0--nu1	-un0----------11---------u0111nu
09:	-0010---1--1--01-0u-$\underline{1}$0nnnnu01010	--u0-------------1--1001-u1--100
10:	--------1--1--0--01-$\underline{1}$01nu1111u10	xxu00-----0--1--1--0--$\underline{1}$--u----n-
11:	0---------0--1--1--0n-$\underline{1}$00nn0u1n0	-xn--1--0--0--1--$\underline{0}$---11-0010--x-
12:	0---0-------0--0--0--01-$\underline{0}$10n1-nn	x------------------------------u
13:	00----------0--0--0--001$\underline{0}$0n0n-00	--10------------------0--1n1----
14:	-0--0-----------------10001u0un-	---1--------1--0--0--1--$\underline{0}$00---xn
15:	n-----------------------unnn1101	-x-10-------1--0--0--1--$\underline{0}$u-n--u-
16:	--1---------------------1--nu001	-n0---------------------1u0-----
17:	n-0-----------------------111-0n	xxn-----------------1---1u-x--n-
18:	-11-------------------------101-	x-u1--------------------0----0--
19:	------------------------------u-	x----------------------11n------
20:	--------------------------------	--x----------------------------x
21:	-------------------------------x	--n----------------------xx-----
22:	--------------------------------	x-----------------------1------x
23:	------------------------------x-	-x-----------------------x----x-
24:	------------------------------x-	xu-----------------------x----xx
25:	-------------------------------x	-x------------------------x---x-
26:	--------------------------------	------------------------------xx
27:	------------------------------x-	-x-----------------------x----x-
28:	------------------------------x-	xx-----------------------x----xx
29:	-------------------------------x	xx------------------------x---x-
30:	--------------------------------	-------------------------------x
31:	--------------------------------	-x----------------------------x-
32:	------------------------------x-	xx-----------------------x----xx
33:	-------------------------------x	-x-----------------------xx---x-
34:	--------------------------------	x------------------------------x
35:	------------------------------x-	-x-----------------------x----x-
36:	------------------------------x-	-x-----------------------x----x-
37:	--------------------------------	-x----------------------------x-
38:	--------------------------------	------------------------------x-
39:	------------------------------x-	-------------------------x------
	. . .	. . .

Table 9. Steps 40 to 80 of the main differential path of the first block

i	A_i	W_i
	. . .	. . .
40:	--------------------------------	x-----------------------------x-
41:	--------------------------------	x-------------------------------
42:	--------------------------------	x-----------------------------x-
43:	------------------------------x-	x------------------------x------
44:	--------------------------------	--------------------------------
45:	------------------------------x-	x------------------------x------
46:	--------------------------------	x-------------------------------
47:	------------------------------x-	-------------------------x------
48:	--------------------------------	x-------------------------------
49:	------------------------------x-	-------------------------x------
50:	--------------------------------	x-----------------------------x-
51:	--------------------------------	--------------------------------
52:	--------------------------------	x-------------------------------
53:	--------------------------------	x-------------------------------
54:	--------------------------------	--------------------------------
55:	--------------------------------	--------------------------------
56:	--------------------------------	--------------------------------
57:	--------------------------------	--------------------------------
58:	--------------------------------	--------------------------------
59:	--------------------------------	--------------------------------
60:	--------------------------------	--------------------------------
61:	--------------------------------	--------------------------------
62:	--------------------------------	--------------------------------
63:	--------------------------------	--------------------------------
64:	--------------------------------	-----------------------------x--
65:	-----------------------------x--	------------------------x-------
66:	--------------------------------	-----------------------------x--
67:	--------------------------------	----------------------------x--x
68:	----------------------------x---	-----------------------x-------x
69:	--------------------------------	----------------------------x--x
70:	--------------------------------	---------------------------x--x-
71:	---------------------------x----	----------------------x-------x-
72:	--------------------------------	---------------------------xx-x-
73:	----------------------------x---	-----------------------x--x--x--
74:	--------------------------x-----	---------------------x------xx--
75:	--------------------------------	--------------------------x--xx-
76:	--------------------------------	-------------------------x--x-x-
77:	-------------------------x------	--------------------x-------x-x-
78:	--------------------------------	-------------------------xx-----
79:	--------------------------x-x---	---------------------x-xx--x----
80:	------------------------x-------	