Notes on the Wang et al. 2^{63} SHA-1 Differential Path

Martin Cochran *

February 10, 2008

Abstract

Although advances in SHA-1 cryptanalysis have been made since the 2005 announcement of a 2^{63} attack by Wang et al., the details of the attack have not yet been presented or verified. This note does just that. Working from Adi Shamir's 2005 CRYPTO rump session presentation of Wang et al.'s work, this note verifies and presents the differential path and associated conditions. Although the error analysis for the advanced condition correction technique is not verified, a method is presented which yields a two-block collision attack on SHA-1 requiring an estimated 2^{62} SHA-1 computations if the original error analysis by Wang et al. is correct.

The differential path is presented for only the first block of the two-block attack, but the second block path likely differs from the first in only the first 10 steps and could be derived from the information presented here.

Keywords: Cryptographic Hash Functions, Cryptanalysis, SHA-1

1 Introduction

In the rump session at CRYPTO 2005 on behalf of Xiaoyung Wang, Andrew C. Yao and Frances Yao, Adi Shamir gave a short talk detailing a differential path which could be used to find a collision in SHA-1 with an expected 2^{63} SHA-1 compression function invocations [30]. Although some details of their methodology were given, over two years have passed and there seems to be little comprehension of the result. The aim of this document is to verify both the differential path and complexity estimation given in [30]. It is not my intention to "steal" any forthcoming publication by the authors of [30]; I am not planning to publish this note in a peer-reviewed medium but instead hope that it may aid researchers in the area.[1]

Main Results. This note contains details for a first block differential path similar (but not identical) to that given in [30] and a set of sufficient conditions which guarantee the path holds. An almost identical second block path, giving a full collision attack, could be derived easily from

* Department of Computer Science, 430 UCB, Boulder, Colorado 80309 USA. E-mail: Martin.Cochran@colorado.edu WWW: ucsu.colorado.edu/∼cochranm

[1] I contacted the authors of [30] seeking permission to publicly post this note and assume by their lack of reply a tacit approval.

the details given here. The results do not correlate exactly with the data summarized in [30], but the differences are minor and, assuming the original advanced modification error probability is correct, the main figures and estimations are substantiated. Namely, after all conditions are enumerated and advanced condition modification methods and early stopping techniques are taken into account, an expected 2^{62} SHA-1 invocations are needed to find a collision. The apparent complexity improvement is achieved by using an "early stopping" technique used in the analysis from [31]. Additionally, to arrive at this figure many paths related to the main differential path are used which aids the attack complexity by a factor of four. It is possible, although perhaps not likely, that these differential paths were not considered by the authors of [30] and could be combined with their methods to yield a faster attack.

The differential path is given in Figures 2 and 3, the sufficient conditions for which are given in Figures 4, 5 and 6. I also describe an advanced condition modification technique to correct many conditions past step 16 and try to interpret other statements made in [30] regarding degrees of message freedom before and after advanced condition modification. I stress that everything here is a "best guess" at what the authors of [30] intended, but it seems likely that the results are highly correlated.

Previous Cryptanalysis of SHA-1. Starting in 2004 at the CRYPTO rump session, a series of breakthrough attacks on hash functions were presented by Wang and company [29, 31–33]. In the earlier two papers, more efficient attacks were found on MD4, HAVAL, and RIPEMD, among others, and a collision for the widely used MD5 was given. A summary of the methods used in these papers can be found in [4, 5].

Biham et al. [3] provided the first collision in SHA-0, stemming from work in [2] and earlier analysis by Chabaud and Joux [8]. In the same paper they began to chip away at the security of SHA-1, giving a collision in a reduced 40-step version. Shortly thereafter in 2005 Wang et al. published a more efficient collision attack [33] on SHA-0 and at the same conference Wang, Lin and Yu extended the techniques to the full SHA-1 in [31]. In that paper they give a complexity estimate of 2^{69} expected computations of SHA-1 for their attack, besting the general theoretical attack of 2^{80} computations. At that point many considered SHA-1 to be "broken," although performing 2^{69} work is arguably not practical in any reasonable time frame. A day after the talk for [31] was given, Shamir presented the 2^{63} attack, apparently a very recent result at the time, for the figures differed slightly from a lengthier presentation at the NIST hash function workshop a few days later [25]. All specific figures refer to this later presentation.

More recently at the 2007 CRYPTO conference Joux et al. presented a paper [13] on an application of Wagner's boomerang attack [28] to compression functions in general, SHA-1 in particular. At the rump session of the same conference Christian Rechberger presented [16] work done with Vincent Rijmen based on earlier work with Christophe De Cannière [7] where he described the best current known attack, utilizing many different differential paths. The estimate given in [16] was $2^{60.x}$ expected calls to the SHA-1 compression function. There has also been some work by Sugita, Kawazoe, Perret and Imai [26] to provide a theoretical foundation for the advanced condition modification methods of Wang et al.

2 Background, Notation

For a given binary string $S \in \{0,1\}^*$, $|S|$ will denote the length, in bits, of S. For two binary string S and T, $S \parallel T$ will denote their concatenation. Unless otherwise noted, all variables are 32-bit integers and all arithmetic is done modulo 2^{32}. For two 32-bit variables x and y, $x \wedge y$ denotes the logical bitwise 'AND' of x and y, and likewise $x \vee y$ and $x \oplus y$ are used to denote the logical bitwise 'OR' and 'XOR', respectively, of x and y. The left (right) circular shift of x by n bits is denoted as $x \lll n$ ($x \ggg y$), and the bitwise complement of x is denoted as $\bar{x}$. The j-th least significant bit of a variable M_i is denoted by M_i^j.

FOUNDATIONS. Generally, a cryptographic hash function is a function

$$H : \{0,1\}^* \rightarrow \{0,1\}^n$$

for some n.

A formal discussion of hash function security notions and the relations between them can be found in [23], but the following are the three informal properties desired by designers of hash functions:

- **Pre-image Resistance** Given $y \in \{0,1\}^n$, it is "difficult" to find $x \in \{0,1\}^*$ such that $H(x) = y$.
- **Second Pre-image Resistance** Given $x \in \{0,1\}^*$, it is "difficult" to find $x' \in \{0,1\}^*$, $x' \neq x$, such that $H(x) = H(x')$.
- **Collision Resistance** It is "difficult" to find any two $x, x' \in \{0,1\}^*$ such that $x \neq x'$ and $H(x) = H(x')$.

There are some problems encoding these requirements in a complexity-theoretic framework (see [22] for a thorough discussion). However, some guiding upper bounds for expected attack complexity of candidate hash functions can be obtained by examining theoretical attacks. A simple probabilistic attack can be used to obtain chosen pre-images and second pre-images in time $\mathcal{O}(2^n)$ and $\mathcal{O}(1)$ space, and a general collision attack based on Pollard's method [21] can be mounted with $\mathcal{O}(2^{n/2})$ invocations and a constant amount of space. Most designers of hash functions try to achieve the strongest security property, collision resistance, and thus any proposed hash has its security judged against the $\mathcal{O}(2^{n/2})$ attack.

Although there are many possible ways to construct a hash function, SHA-1 and most others are iterated via a compression function, utilizing the well-known result simultaneously discovered by Merkle [17] and Damgård [10] that a collision-resistant hash function accepting strings of arbitrary length may be constructed from a collision-resistant compression function which has two inputs of fixed length.

With the additional structure imposed by the Merkle-Damgård paradigm, improvements to the above theoretical attacks are possible. Kelsey and Schneier [15] have shown a second pre-image attack on any Merkle-Damgård hash function that requires about $k2^{n/2+1} + 2^{n-k+1}$ hash function calls to find a second pre-image on a message of block-length 2^k. Similarly, because most

compression functions are based on blockciphers more efficient brute force attacks discussed in [1] may be employed in pre-image attacks (essentially a key search when the IV is fixed). Various time-memory tradeoffs and parallel methods for Pollard's Rho-based collision attacks are discussed further in [27]. There have been other recent interesting results on Merkle-Damgård hash functions [12, 14], but they are mostly unrelated to this note.

SHA-1. SHA-1 (Secure Hash Algorithm-1), based on the MD function family due to Ron Rivest and using the Merkle-Damgård paradigm, was standardized by NIST in 1995 [19]. It replaced a similar hash, SHA-0, then known simply as "SHA."

The SHA-1 compression function, denoted as $\texttt{SHA}_\texttt{c}$, takes as input a 160-bit chaining value CV, broken into five 32-bit values such that $\text{CV} = \text{CV}_0 \parallel \text{CV}_1 \parallel \text{CV}_2 \parallel \text{CV}_3 \parallel \text{CV}_4$, and a 512-bit message input M, with $M = M_0 \parallel M_1 \parallel \ldots \parallel M_{15}$, $|M_i| = 32$.

The full SHA-1 on input message M is computed by using Merkle-Damgård strengthening [10, 17] and padding to obtain a message M' such that $|M'|$ is a multiple of 512. Let $M' = M'_0 \parallel M'_1 \parallel \ldots \parallel M'_\ell$, where $|M'_i| = 512$, for some ℓ and let

$$\text{IV}_0 = \texttt{0x67452301} \parallel \texttt{0xefcdab89} \parallel \texttt{0x98badcfe} \parallel \texttt{0x10325476} \parallel \texttt{0xc3d2e1f0}.$$

Define $\text{IV}_{\text{i}+1} = \texttt{SHA}_\texttt{c}(\text{IV}_\text{i}, M'_i)$. Then the SHA-1 output on input M is $\text{IV}_{\ell+1}$.

$\texttt{SHA}_\texttt{c}$. The compression function of SHA-1 consists of 80 *steps* producing 80 intermediate *step values* A_i for $1 \le i \le 80$. Additionally, the values $A_0, A_{-1}, A_{-2}, A_{-3},$ and A_{-4} are initialized with $\text{CV}_0, \text{CV}_1, (\text{CV}_2 \ll 2), (\text{CV}_3 \ll 2)$ and $(\text{CV}_4 \ll 2)$, respectively.

The 80 step values are computed iteratively by the following recurrence:

$$A_{i+1} \leftarrow (A_i \ll 5) + f_i(A_{i-1}, (A_{i-2} \gg 2), (A_{i-3} \gg 2)) + (A_{i-4} \gg 2) + W_i + K_i$$

where f_i is called the *round function*, K_i is a step-dependent constant and W_i is a variable computed from the input message. The message expansion for determining the W_i for $0 \le i < 80$ is defined by the following.

$$W_i = \begin{cases} M_i & : \text{ if } i < 16 \\ (W_{i-3} \oplus W_{i-8} \oplus W_{i-14} \oplus W_{i-16}) \ll 1 & : \text{ otherwise} \end{cases}$$

The function f_i is divided into four *rounds* (I will sometimes call it a round function) and is defined as follows:

$$f_i(x, y, z) = \begin{cases} (\text{IF}) : (x \wedge y) \vee (\bar{x} \wedge z) & : \text{ if } 0 \le i < 20 \\ (\text{XOR}) : x \oplus y \oplus z & : \text{ if } 20 \le i < 40 \\ (\text{MAJ}) : (x \wedge y) \vee (x \wedge z) \vee (y \wedge z) & : \text{ if } 40 \le i < 60 \\ (\text{XOR}) : x \oplus y \oplus z & : \text{ if } 60 \le i < 80 \end{cases}$$

The output of $\texttt{SHA}_\texttt{c}$ is

$$A_{79} + \text{CV}_0 \parallel A_{78} + \text{CV}_1 \parallel (A_{77} \gg 2) + \text{CV}_2 \parallel (A_{76} \gg 2) + \text{CV}_3 \parallel (A_{75} \gg 2) + \text{CV}_4$$

XOR Differential vs. Subtraction Differential. The notation in the differential path uses a combination of the XOR differential and the subtraction differential. That is, for two 32-bit unsigned integers x, x', consider the function $\Delta_X(x, x') = x \oplus x'$. This defines the XOR differential for x, x'. Alternatively, define $\Delta_S(x, x')$ as $x' - x \bmod 2^{32}$. This is the subtraction differential. The value of one differential does not imply the value of the other. For example, let $\Delta_S(x, x') = 2^2$. There are many possibilities for $\Delta_X(x, x')$:

- $\Delta_X(x, x') =$ `0x00000004` (there is only one bit different between x and x', in index 2)
- $\Delta_X(x, x') =$ `0x0000000c` (bit 3 is set in x' but is not set in x, bit 2 is not set in x' but is set in x)
- $\Delta_X(x, x') =$ `0x0000fffc` (bit 15 is set in x' but is not set in x, bits 2 through 14 are not set in x' but are set in x

The differential used here in Figures 2 and 3 (and originally in [29, 32]) resolves this problem with the following notation. Let x be in $[0, 2^{31} - 1]$. Then $x' = x[a_1, a_2, \ldots, a_n, -b_1, -b_2, \ldots, -b_m]$ denotes $x' = x + 2^{a_1} + 2^{a_2} + \cdots + 2^{a_n} - 2^{b_1} - 2^{b_2} \cdots - 2^{b_m} \bmod 2^{32}$. From this information one can compute both $\Delta_X(x, x')$ and $\Delta_S(x, x')$ if and only if for every bit index i for which x and x' differ $i \in \{a_1, a_2, \ldots, a_n, b_1, b_2, \ldots, b_m\}$.

3 Methods

The Wang et al. collision attacks are differential attacks, whereby a pair of messages M, M' is chosen such that $\Delta(M', M) = \delta$ for some prescribed function (or set of functions) Δ and some prescribed value (vector) δ. Let A_i, $1 \leq i \leq 80$ be the step values computed on input M and define A'_i likewise for M'. The attacker then tracks the values $\Delta(A'_i, A_i)$ throughout the computation of the compression function, but is mostly uninterested in the values of A'_i and A_i themselves. A differential path specifies target values $\Delta^t A_i$ for all i and is usually associated with some set of conditions on the A_i, CV, and M which guarantee that if A_i, CV, and M satisfy those conditions, then given the message $M' \leftarrow M + \delta$ and the associated values A'_i, $\Delta^t A_i = A'_i - A_i$ for all relevant i. The recent attacks all use the value of CV which is of interest, the specified initial value, so the task then becomes efficiently finding a message M such that the sufficient conditions are satisfied.

The Wang et al. attack on SHA-1 described in this note uses a two-block differential path. I will provide details only for the first block differential path, but the path for the second block is very similar and likely only differs from the first block path for the first ten steps, in order to accommodate the differences introduced by the chaining value.

The differential path presented here has 305 total conditions, all but 85 of which are easily satisfied deterministically. Of those remaining conditions, 20 may be satisfied with advanced methods with probability around 1/2 according to [30], leaving effectively 65 conditions which are satisfied probabilistically. The heuristic for determining the attack complexity, useful in practice, is to assume that each condition is satisfied with probability 1/2. Typically, the differential path is chosen so that as many conditions may be satisfied deterministically as possible, which in practice means

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

Table 1: A local collision with associated conditions in round 1. A difference is introduced by a change in W_i^j, and conditions can be chosen to absorb differences in W_{i+2}^j, W_{i+3}^{j-2}, and W_{i+4}^{j-2}. Regardless, a change in W_{i+5}^{j-2} is required to correct the possible change in step $i+6$.

the differential paths have very few conditions in the later rounds. The actual attack complexity considers several other factors and is discussed later.

Local Collisions. A theme among collision attacks on the SHA family is the idea of 'local collisions' [8]. A local collision specifies how a difference introduced in step i may be absorbed by other differences introduced in the following five steps. In this way a local collision is self-contained; the differences do not continue to propagate after those five steps. There are many ways to construct local collisions, but the one first mentioned in [8] and used in most attacks on SHA-1 was chosen for the relatively few number of conditions required for the local collision to occur. Table 1, summarizing a first round local condition and its related sufficient conditions, was taken from [13].

As can be seen from the table, depending on the round and the whether or not corrections are needed, there are four conditions on A_i^j and up to 5 conditions on M_i^j to ensure a local collision occurs in the first round. In [31] and [30], a specialized differential path is used in the first round and the local collision differential paths are only employed in later rounds, and the same is true with the path presented here. Note that the correction for step $i+2$, for example, is a condition only involving the message-dependent values W_i. As such this may be easily deterministically satisfied, even in later rounds, by computing via the message expansion function the relevant set of linear equations over $\mathbb{GF}_2$ for the target bits (W_{i+1}^{j+5} and W_i^j in this case), and using Gaussian elimination.

Disturbance Vector. A major component of the Wang et al. SHA-1 attack is their choice of a *disturbance vector*, which is simply a choice of where to introduce local collisions. We can think of a disturbance vector as an 80×32 binary array, indexed by i, j, such that if i, j is 1 in the disturbance vector, then a local collision is introduced in step i, bit j. By the message expansion function, once

a disturbance vector has been chosen for 16 consecutive steps, the other 64 steps are determined. Another handy property is that the corrections needed by differences in message bits for any local collision (see Table 1) are satisfied by the same recurrence. Thus, from the disturbance vector, the appropriate message difference may be computed.

The disturbance vector in [30] (and here) is the same as the disturbance vector in [31], but shifted by 2 steps. In [31], 71 conditions (not including message conditions) occurred in steps 16-79, and in the path presented here there are 85 such conditions (83 if considering the extra differential paths), but there is also a way to easily correct 20 of them.

Although the disturbance vector provides a map of the differential path in later rounds, a fact that greatly aided the reconstruction of the path, in the first round the differential path ignores the disturbance vector. Motivation for why this was done is presented in [31].

A trick to save a few conditions is mentioned in [31], where if within one step there are disturbances in both bits 0 and 1 and the signs of those two differences are opposite, the introduced difference can be confined to bit zero. The message differences still can be used to form a local collision, and two fewer conditions are required. In the path presented here, this trick is employed in steps 22 and 34.

Alternate Differential Paths. The differences introduced in steps 22, 26, 30, 34, 66, 69, 72, 74 and 75 have multiple differential paths associated with them. For instance, in step 26 if the introduced difference in A_{27}^0 causes a carry, then with only one extra message condition and slight alterations to existing conditions the same local collision occurs. The net result is an alternate differential path with one extra message condition and one extra regular condition. All alternate paths involve this same trick of one carry, and each path has up to four extra conditions. When all of these alternate paths and their combinations are taken into account, it has the effect of reducing the complexity by a factor of 4. By my count, there are 85 conditions in steps 16-79, compared to 83 given by [30]; perhaps these alternate paths are factored into that figure. If there is another means of reducing the 85 conditions to 83 which does not interfere with the alternate differential paths, it is possible that a combination of methods can result in an attack with 2^{60} complexity.

Advanced Condition Modification. There are 85 conditions in steps 16-79, which can be reduced to 83 by the multiple differential paths. In order to obtain an attack complexity of 2^{63}, 20 conditions must be satisfied deterministically via a process called *advanced condition modification.*

There are several key ideas to Wang's advanced condition modification techniques, perhaps best explained by the following series of insights.

- The message conditions can be expressed as equations over $\mathbb{GF}_2$ with variables in W_i^j for $i < 16,\ 0 \leq j < 32$ by simply expanding the desired bit W_i^j via the recurrence formula. There are 43 such equations, which can be easily solved deterministically before advanced condition modification.

- There are 50 variables in W_i^j for $10 \leq i < 16,\ 0 \leq j < 32$ which are involved in not one of these equations. When changes in W_9 are allowed, this number increases to 59.

- A *control path* is a possible bit propagation chain from W_i^j to some bit $A_{i'}^{j'}$. It is easy to generate all possible control paths from the 50 free message bits to conditions on A_i^j for $17 \leq i < 26$. In order to simplify the analysis, I ignored carry effects in finding control paths. [30] did the analysis with the carry effects intact, which gave a probability of 1/2 that the methods cause some earlier satisfied condition to be upset by the modifications.
- After generating the above graph, it is possible to find a set of free message bits M which can be used to flip all desired condition bits through step 23.
- Some message bits will flip multiple bits with associated conditions, so it is necessary to determine an order for flipping which won't undo previous corrections. I found that there are eight sets of flips and associated conditions, explained further in Figure 1.
- Now consider the carry effects and find an order for flipping conditions which minimizes the probability that a previously fixed condition is undone. I have not done this here, but [30] claims that this probability is about 1/2 for their "topological order." Their technique uses multiple message differences, instead of the single bit flipping presented here. In the example from [30], three differences are introduced which give a local collision starting at step 11; the message differences don't produce uncontrolled differences in the step values until step 19, which reduces the probability that some earlier difference in a step variable undoes a prior condition.
- The extra conditions for the advanced condition modification in [30] are derived from round 1 conditions which guarantee a change in the message bit will (or will not) produce a change in Δf_i in subsequent steps.

The (simplified) corrections presented in Table 1 will satisfy all conditions through step 23, and five of the conditions in steps 24-26. In addition, if both of the conditions $\{W_{24}^0 = A_{25}^1,\ W_{25}^1 = A_{26}^1\}$ are unsatisfied, a change in the bit A_{22}^3 will correct both conditions, effectively eliminating one condition. Perhaps this is what is meant in [30] by "searching for two conditions in steps 25-26 by one computation," although this is only a guess.

With an average of 10 needed condition corrections per iteration, and simple bit-flipping as the message modification method, an average of less than 40 step computations are needed to correct all conditions through step 25, thereby canceling the effect of that with probability 1/2 the advanced modification method undoes itself. This early stopping technique was used for the estimate in [31], so it seems reasonable to use it here to lower the attack complexity.

Complexity Estimation Justification. The total complexity is calculated as follows. After all differential paths are considered, there are an equivalent of 83 conditions which need to be satisfied in steps 16-79, but the carry condition in step 78 can be safely dropped with negligible error probability (2^{-25}). Advanced condition modification will satisfy an equivalent of 20 conditions with probability 1/2, which is counted as balancing the dropped condition in step 78. The early stopping technique then brings the attack complexity to 2^{62}.

For the first block, three more conditions in steps 78-79 can be dropped, which has the effect of changing the values of ΔA_{79} and ΔA_{80} from those presented in Figure 3. These differences can

Group 1	Group 2	Group 3	Group 4
$M_{15,8} \rightarrow A^1_{21}$, $M_{15,10} \rightarrow A^0_{22}$, $M_{15,12} \rightarrow A^3_{23}$	$M^{10}_{13} \rightarrow A^3_{19}$, $M^{11}_{14} \rightarrow A^{31}_{19}$, $\{M^{12}_{13}, M^7_{14}, M^{17}_{14}\} \rightarrow A^3_{22}$, $M^{12}_{14} \rightarrow A^0_{23}$	$M^{10}_{11} \rightarrow A^3_{17}$, $\{M^{12}_{10}, M^7_{12}, M^{17}_{12}$, $M^7_{13}, M^{10}_{14}\} \rightarrow A^3_{20}$, A^{31}_{20}	$M^{13}_{11} \rightarrow A^1_{18}$, $\{M^7_{10}, M^7_{11}\} \rightarrow A^3_{18}$
Group 5	Group 6	Group 7	Group 8
$\{M^{10}_{12}, M^{15}_{13}\} \rightarrow A^{30}_{17}$, $M^2_{12} \rightarrow A^2_{22}$	$\{M^{10}_{10}, M^1_{11}\} \rightarrow A^1_{19}$, $M^8_{10} \rightarrow A^2_{21}$, $\{M^9_{10}, M^9_{12}, M^{14}_{13}$, $M^9_{14}, M^{14}_{15}\} \rightarrow A^{29}_{21}$	$\{M^8_{11}, M^{13}_{12}\} \rightarrow A^1_{17}$, $M^{16}_{14} \rightarrow A^{31}_{18}$, $\{M^{13}_{13}, M^9_{15}\} \rightarrow A^{29}_{20}$	$M^{11}_{12} \rightarrow A^{31}_{17}$, $\{M^{14}_{10}, M^9_{11}, M^9_{13}$, $M^{14}_{14}\} \rightarrow A^{29}_{18}$

Figure 1: Restrictions on the ways 22 conditions may be modified by flipping free message bits when carry effects are ignored. All conditions may be satisfied by processing all the conditions within each group at the same time, and processing the groups in reverse order.

be easily accommodated via suitable changes in the first 10 steps of the second-block differential path. Thus, the total attack complexity is $2^{59} + 2^{62} = 1.125 \times 2^{62}$.

Differences from Wang et al. Path. There are some differences from the path presented by Wang et al. in [30]:

- The differentials for the first 16 steps do not match. However, it is simple to check that the example 320-bit message prefix given in [30] does not follow their own differential path, but instead follows the path given here.
- I found 43 message conditions instead of 42, plus one extra condition for use with the associated differential paths. This is probably because the classification of one of the message conditions in step 19 is open to interpretation.
- The set of unrestricted message bits after satisfying message conditions is somewhat different. I have no plausible explanation.
- [30] claims to have a free message space of size 2^{55} after advanced modifications. I'm not sure what is meant by this, but calculating a rough estimate of message freedom is straightforward: there are 20 conditions in steps 10-16, plus 43 message conditions and 22 message bits used for advanced condition modification. Thus, with $7 \times 32 = 224$ original bits, there is essentially a message space of about 2^{139}, for which 63 conditions need to be satisfied probabilistically. The "message space available for direct modification" is probably related to the 59 free message bits in message words M_9 through M_{15} available to change after all message conditions are satisfied.

- I have not investigated the scenario where, once all advanced modifications have been performed and all relevant conditions satisfied, message bits may be flipped with small probability of disturbing already satisfied conditions (the "neutral bits" of [2]). Perhaps sets of message bits, giving first-round local collisions, may be a good strategy for finding neutral modifications.
- I can only speculate if my advanced condition modification techniques are what the original authors had in mind. My methods seem to provide correction of an equal number of conditions but, judging by the cleverness inherent in all other aspects of their attack for which they provided details, I suspect their methods are more efficient or have some other beneficial properties.

4 The Differential Path

The relevant information for the differential path is contained entirely in Figures 2, 3, 4, 5 and 6. All conditions listed for a step i are used for some correction or condition needed for that step value computation. Conditions are color-coded for clarity: blue conditions are conditions on message bits, red conditions are needed to produce an output difference or some desired property thereof from the step function f_i, green signifies conditions which are needed so that no differences are produced via f_i, and conditions which are not colored are required to satisfy desired carry effects.

Step	Δx_i	Δw_i	$\Delta a_i \ll 5$	Δf_i	$\Delta a_{i-4} \gg 2$	Δa_{i+1}
0	80000001	[0,−1,−29,−31]				[−0,29,−30,31]
1		[−4,5,29]	[2,−3,4,−5]			[−2,29]
2	40000001	[29,30]	[2,−7]			[2,7,8,...,−22,−30,31]
3	2	[-1,-3,-5,-29,30,-31]	[−3,4,7,−12]	[−29]		[-1,5,-6,7,12,-13,31]
4	2	[−0,1,6,29]	[4,-6,10,-11,12,17,-18]	[-0,-7,10,-12,17,-22,27,29]		[−4,5,7,−8,−22,27]
5	80000002	[−6,28,−29,−31]	[0,9,−10,12,−13,−27]	[−0,6,9,12,29]	[27,−28,29,−30]	[−10,11,31]
6	1	[−0,1,−4,6,28,30,31]	[4,−15,16]	[0,−6,27,29]	[−0,27]	[0,−15,−16,...,27]
7		[−1,5,28,30,31]	[5,20]	[3,−5,−20,29]	[0,−5,−28,29]	[3]
8	80000001	[−29]	[8]	[−0,3,−5,9]	[3,−4,5,10,−11,29,−31]	[−0,8,−9,31]
9	2	[−1,4,5,29,−30]	[4,−5,13,−14]	[13,20,−25,29]	[−2,3,5,−6,−20,25]	[1]
10	2	[0,−1,−6,29,30]	[6]	[−0,1,9,−31]	[−8,9,29]	[8,−9]
11	2	[6,−29]	[13,−14]	[1,−6,−29]	[13,30]	[1]
12		[−1,−6,−29,30,31]	[6]	[−29,31]	[1]	
13		[1,−29,−30]		[−1,6,31]	[6,−7,29,−30]	
14	1	[0,31]			[31]	[0]
15		[5]	[5]		[6,−7]	
16	80000002	[±0,±1]		[±0]	[31]	[±1,±31]
17	2	[±1,±4,±6,−30]	[±4,±6]	[30]		[±1]
18	80000002	[±6,30]	[±6]	[±1,30]		[±1,31*]
19		[±1,±4,±6,±29,−30,31*]	[±4,±6]	[±1,±29,31*]	[30]	
20	2	[±29,31*]		[±1,±29,31*]		[±1]
21		[±6,31*]	[±6]	[±29]	[±29,31*]	
22	3	[±0,±29]		[±1,±29,31*]	[31*]	[±0]
23		[±5,±6,±29]	[±5]	[31*]	[±29,31*]	
24	2	[±0,31*]		[±0,31*]		[±1]
25	2	[±1,±6,±30]	[±6]	[±30]	[31*]	[±1]
26	1	[±0,±1,±6,±30,31*]	[±6]	[±1,±30]		[±0]
27		[±1,±5,±30]	[±5]	[±1,31*]	[±30]	
28	2	[±0,±1]		[±0]		[±1]
29	2	[±1,±6,±30]	[±6]	[±30,31*]	[31*]	[±1]
30	1	[±0,±1,±6,±30,31*]	[±6]	[±1,±30]	[31*]	[±0]
31		[±1,±5,±30,31*]	[±5]	[±1,31*]	[±30]	
32		[±0]		[±0]		
33	2	[±1,±30]		[±30]	[31*]	[±1]
34	3	[±0,±1,±6,±30,31*]	[±6]	[±30]	[31*]	[±0]
35		[±1,±5,±6,±30]	[±5]	[±1]	[±30]	
36	2	[±0,31*]		[±0,31*]		[±1]
37	2	[±1,±6,±30]	[±6]	[±30,31*]		[±1]
38		[±1,±6,±30]	[±6]	[±1,±30]	[31*]	
39		[±1,±30]		[±1,31*]	[±30]	

Figure 2: The differential path. Columns one and two denote the step index and disturbance vector for that step, respectively. Columns three through six denote the input differences to that step value computation with column seven indicating the final output value. An asterisk denotes that the sign of the difference is irrelevant. A plus-minus sign indicates that the difference can be either sign, but may be related to prior or future differences.

Step	Δx_i	Δw_i	$\Delta a_i \ll 5$	Δf_i	$\Delta a_{i-4} \gg 2$	Δa_{i+1}
40	2	[1]				[1]
41		[6]	[6]	[31]	[31]	
42		[1,31]		[1]	[31]	
43		[31]		[31]		
44	2	[1,31]		[31]		[1]
45		[6,31]	[6]		[31]	
46	2			[1]		[1]
47		[6,31]	[6]	[31]		
48	2	[31]		[1,31]		[1]
49		[6]	[6]	[31]	[31]	
50	2	[31]		[1,31]		[1]
51		[6]	[6]	[31]	[31]	
52		[1,31]		[1,31]		
53				[31]	[31]	
54		[31]		[31]		
55		[31]			[31]	
56						
57						
58						
59						
60						
61						
62						
63						
64						
65						
66	4	[2]				[2]
67		[7]	[7]			
68		[2]		[2]		
69	8	[0,3]		[0]		[3]
70		[0,8]	[8]	[0]		
71		[0,3]		[3]	[0]	
72	10	[1,4]		[1]		[4]
73		[1,9]	[9]	[1]		
74	8	[1,3,4]		[4]	[1]	[3]
75	20	[2,5,8]	[8]	[2]		[5]
76		[2,3,10]	[10]	[2,3]		
77		[1,2,5]		[1,5]	[2]	
78	40	[1,3,6]		[1,3]		[6]
79		[1,3,11]	[11]	[3]	[1]	

Figure 3: The second half of the differential path.

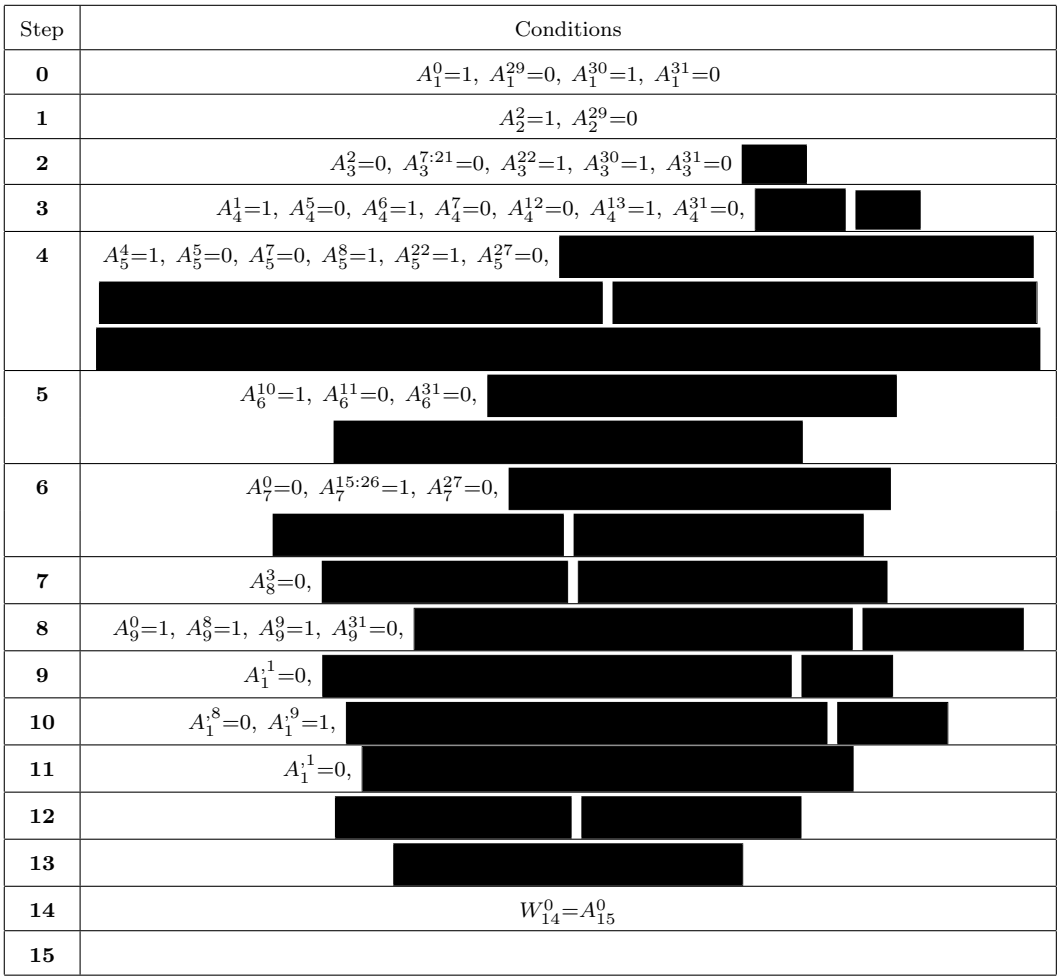

Figure 4: The set of sufficient conditions for the first 16 steps of the differential path. Conditions to prevent carry effects are uncolored, message conditions are blue, conditions required to produce or correct a change in Δf_i are colored with red, and conditions required to prevent a change in Δf_i are green. Conditions enclosed in parentheses are redundant and conditions followed by an '$(*)$' can be affected by advanced condition correction. The notation $A_i^{j:k}$ is used as shorthand for $A_i^j = A_i^{j+1} = \cdots = A_i^k$.

Step	Conditions
16	$W_{16}^{1}{=}A_{17}^{1}(*)$,
17	$W_{17}^{4}{\neq}A_{17}^{31}(*)$, $W_{17}^{1}{=}A_{18}^{1}(*)$,
18	$A_{19}^{1}{\neq}A_{17}^{1}{+}A_{16}^{3}(*)$,
19	$W_{19}^{30}{=}A_{18}^{31}(*)$, $W_{19}^{4}{\neq}A_{19}^{31}(*)$,
20	$A_{21}^{1}{=}A_{19}^{1}{+}A_{18}^{3}{+}A_{17}^{3}(*)$,
21	$W_{21}^{6}{\neq}A_{21}^{1}(*)$, $A_{17}^{31}{\neq}A_{18}^{31}{+}A_{19}^{31}{+}A_{20}^{29}(*)$
22	$W_{22}^{0}{\neq}A_{23}^{0}(*)$,
23	
24	$W_{24}^{0}{=}A_{25}^{1}(*)$,
25	$W_{25}^{1}{=}A_{26}^{1}(*)$,
26	$W_{26}^{0}{\neq}A_{27}^{0}(*)$,
27	
28	$W_{28}^{0}{\neq}A_{27}^{0}{+}A_{26}^{2}{+}A_{25}^{2}$, $W_{28}^{1}{=}A_{29}^{1}$
29	$W_{29}^{1}{\neq}A_{28}^{1}{+}A_{27}^{3}{+}A_{26}^{3}$, $W_{29}^{30}{=}A_{28}^{30}{+}A_{27}^{0}{+}A_{26}^{0}$,
30	$W_{30}^{0}{=}A_{31}^{0}$, $W_{30}^{1}{\neq}A_{29}^{1}{+}A_{28}^{3}{+}A_{27}^{3}$, $W_{30}^{30}{\neq}A_{29}^{30}{+}A_{28}^{0}{+}A_{27}^{0}$,
31	$W_{31}^{1}{\neq}A_{30}^{1}{+}A_{29}^{3}{+}A_{28}^{3}$,
32	$W_{32}^{0}{\neq}A_{31}^{0}{+}A_{30}^{2}{+}A_{29}^{2}$
33	$W_{33}^{1}{=}A_{34}^{1}$, $W_{35}^{30}{=}A_{32}^{30}{+}A_{31}^{0}{+}A_{30}^{0}$
34	$W_{34}^{0}{\neq}A_{35}^{0}$,
35	
36	
37	$W_{37}^{1}{=}A_{38}^{1}$,
38	
39	

Figure 5: The set of conditions for steps 16-39.

Step	Conditions
40	$W_{40}^{1}{=}A_{41}^{1}$
41	
42	
43	
44	$W_{44}^{1}{=}A_{45}^{1}$
45	
46	$A_{47}^{1}{=}W_{44}^{1}$,
47	
48	$W_{44}^{1}{=}A_{49}^{1}$,
49	
50	$W_{44}^{1}{=}A_{51}^{1}$,
51	
52	
53	
54	
55–65	
66	$W_{66}^{2}{=}A_{67}^{2}$
67	
68	
69	$W_{69}^{3}{=}A_{68}^{3}$,
70	
71	
72	$W_{72}^{4}{=}A_{73}^{4}$,
73	
74	$W_{74}^{3}{=}A_{75}^{3}$,
75	$W_{75}^{5}{=}A_{75}^{5}$,
76	
77	
78	$W_{78}^{6}{=}A_{79}^{6}$,
79	

Figure 6: The set of conditions for steps 40-79.

References

[1] Bernstein, D. J. Understanding brute force. Draft available as `http://cr.yp.to/snuffle/bruteforce-20050425.pdf`.

[2] Biham, E., and Chen, R. Near-collisions of SHA-0. In Franklin [11], pp. 290–305.

[3] Biham, E., Chen, R., Joux, A., Carribault, P., Lemuet, C., and Jalby, W. Collisions of SHA-0 and reduced SHA-1. In Cramer [9], pp. 36–57.

[4] Black, J., Cochran, M., and Highland, T. A study of the MD5 attacks: Insights and improvements. Full Version. `http://www.cs.colorado.edu/ jrblack/papers.html`.

[5] Black, J., Cochran, M., and Highland, T. A study of the MD5 attacks: Insights and improvements. In *FSE* (2006), M. J. B. Robshaw, Ed., vol. 4047 of *Lecture Notes in Computer Science*, Springer, pp. 262–277.

[6] Brassard, G., Ed. *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings* (1990), vol. 435 of *Lecture Notes in Computer Science*, Springer.

[7] Cannière, C. D., and Rechberger, C. Finding SHA-1 characteristics: General results and applications. In *ASIACRYPT* (2006), X. Lai and K. Chen, Eds., vol. 4284 of *Lecture Notes in Computer Science*, Springer, pp. 1–20.

[8] Chabaud, F., and Joux, A. Differential collisions in SHA-0. In *CRYPTO* (1998), H. Krawczyk, Ed., vol. 1462 of *Lecture Notes in Computer Science*, Springer, pp. 56–71.

[9] Cramer, R., Ed. *Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings* (2005), vol. 3494 of *Lecture Notes in Computer Science*, Springer.

[10] Damgård, I. A design principle for hash functions. In Brassard [6], pp. 416–427.

[11] Franklin, M. K., Ed. *Advances in Cryptology - CRYPTO 2004, 24th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 2004, Proceedings* (2004), vol. 3152 of *Lecture Notes in Computer Science*, Springer.

[12] Joux, A. Multicollisions in iterated hash functions. application to cascaded constructions. In Franklin [11], pp. 306–316.

[13] Joux, A., and Peyrin, T. Hash functions and the (amplified) boomerang attack. In *CRYPTO* (2007), A. Menezes, Ed., vol. 4622 of *Lecture Notes in Computer Science*, Springer, pp. 244–263.

[14] Kelsey, J., and Kohno, T. Herding hash functions and the nostradamus attack. In *EUROCRYPT* (2006), S. Vaudenay, Ed., vol. 4004 of *Lecture Notes in Computer Science*, Springer, pp. 183–200.

[15] Kelsey, J., and Schneier, B. Second preimages on n-bit hash functions for much less than 2^n work. In Cramer [9], pp. 474–490.

[16] Mendel, F., Rechberger, C., and Rijmen, V. Update on SHA-1. Presented by Christian Rechberger at the rump session of CRYPTO 2007. Slides may be found currently at `http://rump2007.cr.yp.to/09-rechberger.pdf`.

[17] Merkle, R. C. One way hash functions and DES. In Brassard [6], pp. 428–446.

[18] National Institute of Standards and Technology. FIPS 180: Secure Hash Standard, May 1993. Available from `http://csrc.nist.gov`.

[19] National Institute of Standards and Technology. FIPS 180-1: Secure Hash Standard, April 1995. Available from `http://csrc.nist.gov`.

[20] National Institute of Standards and Technology. FIPS 180-2: Secure Hash Standard, August 2002. Available from `http://csrc.nist.gov`.

[21] Pollard, J. M. A Monte Carlo method for factorization. *BIT 15* (1975), 331–334.

[22] Rogaway, P. Formalizing human ignorance: Collision-resistant hashing without the keys. In *VIETCRYPT* (2006), N. P. Quang, Ed., vol. 4341 of *Lecture Notes in Computer Science*, Springer, pp. 221–228.

[23] Rogaway, P., and Shrimpton, T. Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance. In *FSE* (2004), B. K. Roy and W. Meier, Eds., vol. 3017 of *Lecture Notes in Computer Science*, Springer, pp. 371–388.

[24] Shoup, V., Ed. *Advances in Cryptology - CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings* (2005), vol. 3621 of *Lecture Notes in Computer Science*, Springer.

[25] NIST hash function workshop, October 31 - November 1, 2005.

[26] Sugita, M., Kawazoe, M., Perret, L., and Imai, H. Algebraic cryptanalysis of 58-round SHA-1. In *FSE* (2007), A. Biryukov, Ed., vol. 4593 of *Lecture Notes in Computer Science*, Springer, pp. 349–365.

[27] van Oorschot, P. C., and Wiener, M. J. Parallel collision search with cryptanalytic applications. *J. Cryptology 12*, 1 (1999), 1–28.

[28] Wagner, D. The boomerang attack. In *Fast Software Encryption* (1999), L. R. Knudsen, Ed., vol. 1636 of *Lecture Notes in Computer Science*, Springer, pp. 156–170.

[29] Wang, X., Lai, X., Feng, D., Chen, H., and Yu, X. Cryptanalysis of the hash functions MD4 and RIPEMD. In Cramer [9], pp. 1–18.

[30] Wang, X., Yao, A. C., and Yao, F. Cryptanalysis on SHA-1. Presented by Adi Shamir at the rump session of CRYPTO 2005. Slides may be found currently at `http://csrc.nist.gov/groups/ST/hash/documents/Wang_SHA1-New-Result.pdf`.

[31] Wang, X., Yin, Y. L., and Yu, H. Finding collisions in the full SHA-1. In Shoup [24], pp. 17–36.

[32] Wang, X., and Yu, H. How to break MD5 and other hash functions. In Cramer [9], pp. 19–35.

[33] Wang, X., Yu, H., and Yin, Y. L. Efficient collision search attacks on SHA-0. In Shoup [24], pp. 1–16.