Collision Attacks on MD4

COMP 4140: Intro to Cryptography and Cryptosystems
Fall 2015

Zach Havens	**Vanessa Reimer**
7671770	**7691114**
havensz@myumanitoba.ca	**reimerv3@myumanitoba.ca**

Table of Contents

List of Figures

1 Introduction and Motivation

In October 1990, Ronald Rivest proposed a new cryptographic hash function known as the MD4 message-digest algorithm. Taking in a message of any length, the algorithm produces a 128-bit hash value. This is done using three rounds, each performing 16 operations on 4 32-bit word buffers. At the time of proposal, it was conjectured that producing two messages with the same hash value would be computationally infeasible [Rivest, 1991].

The proposal for MD4 came roughly a year after Rivest had first proposed the MD2 message-digest algorithm. MD4 later gave way to the MD5 (proposed in 1992) and MD6 (proposed in 2008) message-digest algorithms. Many other hash functions developed later on were based off of the ideas behind MD4, such as SHA-1, HAVAL, and RIPEMD.

The first collision attack on MD4 was published in 1991 by Bert den Boer and Antoon Bosselaers [Boer & Bosselaers, 1992]. Their attack was partial, only considering the last two rounds of the function. In 1996, Hans Dobbertin published the first full collision attack on MD4, successfully generating a collision with probability 2^{-22} for a single trial, or 2^{20} MD4 operations on average to find a collision over multiple trials [Dobbertin, 1996]. Xiaoyun Wang et al. proposed a much more efficient attack in 2005 that finds a collision with probability 2^{-2} to 2^{-6} in less than 2^{8} MD4 operations [Wang et al., 2005]. The most efficient attack to date was published in 2007 by Yu Sasaki et al., requiring less than 2 MD4 operations to find a collision [Sasaki et al., 2007].

Although MD4 is no longer recommended for use, cryptanalysis of the hashing function remains important. As many newer hashing functions rely on the techniques used in MD4, cryptanalysis of MD4 affects the cryptanalysis of these newer hash functions as well. Additionally, the knowledge gained from exposing security vulnerabilities of MD4 can be applied to ensuring or refuting security in the development of new hash functions.

In section 2, the MD4 message digest algorithm is outlined. Background information on collision attacks and differential cryptanalysis is covered in sections 3 and 4, respectively. Notation used to describe the algorithms is specified in section 5. A closer look at three of the main attacks on MD4 follow; Dobbertin in section 6, Wang et al. in section 7, and Sasaki et al. in section 8. A discussion on the implementations of these three attacks is in section 9, with the paper concluding in section 10.

2 MD4 Algorithm

As outlined in *The MD4 Message Digest Algorithm* [Rivest, 1991] and RFC 1320 [Rivest, 1992], the following steps are performed for a given binary message M_0 with $|M_0| \geq 0$ to produce its 128-bit hash:

Step 1: Append Padding Bits

Pad the message by appending a single '1' bit, followed by a series of '0' bits, until the length of the message modulo 512 is 448. Padding is done regardless of the length of M_0. As few as 1 bit and up to 512 bits are added to the message, with maximum padding occurring when $|M_0|$ % 512 = 448.

Step 2: Append Length

Append the 64-bit representation of $|M_0|$ after the padding applied in step 1. In the case where $|M_0| > 2^{64}$, use only the lower-order 64 bits representing $|M_0|$.

Note that after completing this step, the length of the message is an exact multiple of 512, with each multiple containing 16 32-bit words.

Step 3: Initialize MD Buffer

Initialize a state buffer as the 4 following 32-bits words, corresponding to the MD4 IV:

A:	01	23	45	67
B:	89	AB	CD	EF
C:	FE	DC	BA	98
D:	76	54	32	10

Step 4: Process Message in 16-Word Blocks

Three functions are used when processing the blocks, each taking 3 32-bit words as input and outputting another 32-bit word:

$$\begin{aligned} F(X, Y, Z) &= (X \wedge Y) \vee (\neg X \wedge Z) \\ G(X, Y, Z) &= (X \wedge Y) \vee (X \wedge Z) \vee (Y \wedge Z) \\ H(X, Y, Z) &= X \oplus Y \oplus Z \end{aligned}$$

For each of the 16 32-bit word blocks $M \in M_0$, perform the following compression algorithm:

1) Set M to be the current block
2) Save the state buffer values as $AA = A$, $BB = B$, $CC = C$, $DD = D$
3) Round 1
   - Define R$1(abcd, k, s)$ as:

$$a = (a + F(b, c, d) + m_k) <<< s$$

   - Perform these 16 operations (steps 1 to 16):

$$\begin{array}{llll} R1(ABCD, 0, 3), & R1(DABC, 1, 7), & R1(CDAB, 2, 11), & R1(BCDA, 3, 19), \\ R1(ABCD, 4, 3), & R1(DABC, 5, 7), & R1(CDAB, 6, 11), & R1(BCDA, 7, 19), \\ R1(ABCD, 8, 3), & R1(DABC, 9, 7), & R1(CDAB, 10, 11), & R1(BCDA, 11, 19), \\ R1(ABCD, 12, 3), & R1(DABC, 13, 7), & R1(CDAB, 14, 11), & R1(BCDA, 15, 19) \end{array}$$

4) Round 2

- Define $R2(abcd, k, s)$ as:

$$a = (a + G(b, c, d) + m_k + 5A827999) <<< s$$

- Perform these 16 operations (steps 17 to 32):

$R2(ABCD, 0, 3)$,	$R2(DABC, 4, 5)$,	$R2(CDAB, 8, 9)$,	$R2(BCDA, 12, 13)$,
$R2(ABCD, 1, 3)$,	$R2(DABC, 5, 5)$,	$R2(CDAB, 9, 9)$,	$R2(BCDA, 13, 13)$,
$R2(ABCD, 2, 3)$,	$R2(DABC, 6, 5)$,	$R2(CDAB, 10, 9)$,	$R2(BCDA, 14, 13)$,
$R2(ABCD, 3, 3)$,	$R2(DABC, 7, 5)$,	$R2(CDAB, 11, 9)$,	$R2(BCDA, 15, 13)$

5) Round 3

- Define $R3(abcd, k, s)$ as:

$$a = (a + H(b, c, d) + m_k + 6ED9EBA1) <<< s$$

- Perform these 16 operations (steps 33 to 48):

$R3(ABCD, 0, 3)$,	$R3(DABC, 8, 9)$,	$R3(CDAB, 4, 11)$,	$R3(BCDA, 12, 15)$,
$R3(ABCD, 2, 3)$,	$R3(DABC, 10, 9)$,	$R3(CDAB, 6, 11)$,	$R3(BCDA, 14, 15)$,
$R3(ABCD, 1, 3)$,	$R3(DABC, 9, 9)$,	$R3(CDAB, 5, 11)$,	$R3(BCDA, 13, 15)$,
$R3(ABCD, 3, 3)$,	$R3(DABC, 11, 9)$,	$R3(CDAB, 7, 11)$,	$R3(BCDA, 15, 15)$

6) Add initial values AA, BB, CC, and DD to current values of A, B, C, and D, respectively.

Step 5: Output

The output is defined as the concatenation of A, B, C, D in big endian. The result is 16 bytes in length, regardless of the size of the input message.

3 Collision Attack

One of the pillars of hashing algorithm security is collision resistance. For a given hash function, a collision occurs when two different messages produce the same hash value. For example, consider the MD4 hashes below that were generated using an implementation of Wang's attack from [Stach, 2006].

M = 588f35088d0eb832f54221d6f329f37b3acd0e29bb46585f9181247c3a071478
2f6731995cdf4afe1cd47adb98b09c8a06fee1bf0f233febf8e249740c4e1ed7

M' = 588f35088d0eb8b2f5422146f329f37b3acd0e29bb46585f9181247c3a071478
2f6731995cdf4afe1cd47adb98b09c8a06fee0bf0f233febf8e249740c4e1ed7

MD4(*M*) = *MD4*(*M*') = 4b278534b323500cb3e60e9c7ae523d9

The messages are not equal but result in the same hash value. The reason that it is important for a hash function to be collision resistant is because of the potential uses of colliding messages. Hashing functions are used for things such as digital signatures, HMACs, and simple checksumming. If two messages are found that collide, one could replace the other without the signatures, tags, or checksums becoming invalid, despite the fact that the underlying data has changed. For example, fake web security certificates were created that were indistinguishable from true certificates by exploiting a collision vulnerability in the MD5 algorithm [Sotirov et al., 2008]. Due to these failings, if a hashing function is not collision resistant, it is not considered safe or secure for cryptographic use. Formal collision attacks use knowledge of how a specific hash function works to find messages *M* and *M*' such that $M \neq M'$ but $H(M') = H(M')$. When finding collisions to prove that a hash function is not secure the actual content of the colliding messages is irrelevant. It must simply be proven that colliding messages exist and can be generated in non-trivial polynomial time in order to prove a lack of collision resistance, and therefore a lack of security. For MD4, the brute force attack is 2^{128} hashing operations, but due to the birthday paradox the trivial attack is considered to be $\sim 2^{64}$ hashing operations.

4 Differential Cryptanalysis

All three of the attacks discussed in this paper rely on differential cryptanalysis to find weaknesses in the MD4 hashing algorithm. In this section, we will give a brief outline of what differential cryptanalysis is and how it can be related to attacking MD4 in general terms.

Differential cryptanalysis is centered around the differences between the input messages and resulting output messages of a cryptographic function. This is often because there are some series of steps in the algorithm that can be predicted or adjusted for by crafting specific inputs for the algorithm to process. The process of determining whether or not the probabilities that given input results in a specific output are non-uniform, and if so, how they can be exploited, is the core concept of differential analysis [Heys, 2002].

This technique is very valuable when it comes to evaluating hashing functions like MD4. In order for hashing functions in this family to create short hash values for variable length input strings, they rely on their internal compression functions which use small internal buffers. By looking at these compression functions and performing differential analysis on the individual steps or groups of steps in them, it may be shown to have predictable results for specific inputs [Dobbertin, 1996]. If the difference between an input message and an output message can be shown to have non-uniform probability, then the output for messages with those characteristics can be predicted. Because collision attacks are interested in finding a pair of messages with no difference in their hashes, differential analysis is used to analyze *relative* differences between two messages and their outputs. The difference between two input messages to a function is called the input difference and the difference in the resulting outputs is called the output difference. If an input difference can be shown to result in an output difference of zero with some non-negligible probability in polynomial time, then a collision attack has been found. The input difference that was used for the attack can then be used to craft pairs of input messages that collide. These attacks only make use of one block long messages and ignore padding. Using one block long messages is done to make analysis easier and is valid as it merely restricts inputs to a subset of the message space for MD4. Ignoring padding can also be done because padding is uniform for two messages of the same length, and is appended to the message, therefore it has no impact on differential attack as it will be applied uniformly to already colliding messages.

5 Notation

In order to help describe these collision attacks, we will define a standardized set of notation for various aspects of MD4 used in attacks. All words are big endian at the byte level. Let the following be defined:

1. $M = (m_0, m_1, ..., m_{15})$, $M' = (m'_0, m'_1, ..., m'_{15}) \in \{0, 1\}^{512} \mid M \neq M'$ are input messages
2. $\Delta M = M' - M$ is the input difference
3. a_i, b_i, c_i, d_i and a'_i, b'_i, c'_i, d'_i are the values of the A, B, C, D buffer words after the ith step for inputs M and M' respectively
4. $\Delta_i = \{a_i - a'_i, b_i - b'_i, c_i - c'_i, d_i - d'_i\}$ is the difference in the M and M' buffer words after the ith step
5. q_i, q'_i are the results of the operation performed during the ith step for inputs M and M' respectively (i.e, for Step 1 this is the result of: $(A + F(B, C, D) + X[0]) <<< 3$)
6. $\Delta q_i = q_i - q'_i$ is the difference in the operation results of step i for M and M'

6 Dobbertin

The first collision attack against all three rounds of MD4 was introduced by Dobbertin in 1996. In this section, we discuss the specifics of this attack and the theory behind it as presented in his paper [Dobbertin, 1996].

Dobbertin's differential analysis of the MD4 algorithm revealed a weakness in the differential paths using only a small difference in input messages. The message difference that he used to cause a collision was:

$$\begin{aligned} \Delta m_{12} &= 1 \\ \Delta m_i &= 0, \forall i \neq 12 \end{aligned} \tag{1}$$

The reason that the 12th message word was chosen to differ is because m_{12} appears exactly once in each round of the compression function, in the 13th, 20th, and 36th steps [Dobbertin, 1996]. Since the message blocks that are applied for all steps past 36 will be the same for both messages, if the buffer states are the same after step 36, the states will be the same after step 48. The resulting output hashes will also be the same because the final addition of the initial values is uniform for both messages. Therefore, if Δ_{36} = {0, 0, 0, 0} and equation (1) holds, it follows that Δ_{48} = {0, 0, 0, 0} and the hashes for M and M' will collide.

6.1 Inner-Almost Collisions

In order to ensure that Δ_{36} = {0, 0, 0, 0}, Dobbertin outlines what he calls an "inner-almost collision". He defines this as the execution of steps from 13 to 20 where the resulting difference Δ_{20} is equal to {0, 2^{24}, -2^{4}, 0} [Dobbertin, 1996]. This is a specially chosen difference that enables satisfying Δ_{36} = {0, 0, 0, 0}. It is effectively controlling the differential path between the first two uses of m_{12} so that its final use will result in the all zero difference needed for a full collision. The inner collision is ensured by creating a system of equations representing the different operations performed in these steps and the Δq_i ($13 \le i \le 20$), that result in $\Delta q_{19} = -2^4$ and $\Delta q_{20} = 2^{24}$ (which satisfies the inner-almost collision requirement). q_{13} and q'_{13} are set to predefined values and the other necessary q_i are randomized, then the system is solved. Once all the q_i values are calculated, then the m_i, m'_i that would be used to generate those q_i are calculated using the inverse of the step operations. These values are m_{12}, m'_{12}, m_{13}, m_{14}, m_{15}, m_0, m_4, and m_8. At this point, the inner-almost collision is satisfied.

6.2 Completing the Messages

Once an inner-almost collision has been found, the rest of the of the nine m_i not already defined need to be calculated. In order for the collision to be valid, we need to ensure that the values of a_{12}, b_{12}, c_{12}, d_{12} that are inputs for step 13 need to be the actual values computed after step 12. To do this, the attack works backwards from round 12 towards the IV so that the assumptions made for all previous calculations remain valid. The first step in this reversal is randomly assigning m_1, m_2, m_3, and m_5. From there, m_{11}, m_{10}, and m_9 can be assigned values by reversing steps 12, 11, and 9 respectively using the value of Δq_{13} as a starting point. Finally, the previously fixed value of m_8 can be managed by setting specific values for Δq_8, Δq_7, and Δq_6, then continuing to work backwards, applying each step in reverse to get the remaining m_8, m_7, and m_6. At this point, all m_i are defined such that $\Delta_{48} = \{0, 0, 0, 0\}$. We can apply the message difference to get the colliding message $M' = M + \Delta M$, and are thus able to output (*M*, *M'*) which forms a collision pair. This attack does not succeed with a probability of 1, however, which is discussed in the next section.

6.3 Success Probability and Theoretical Complexity

If the attack is against the compression algorithm alone, the attacker is allowed to use any initial value (IV) instead of the one defined by MD4 [Dobbertin, 1996]. In this scenario the nine m_i that were not fixed as part of the inner-almost collision get assigned randomly, because it is trivial to work backwards from step 12 to step 1 to determine the an arbitrary initial value. This creates uncertainty in the attack success due to the previous assumption that the difference from the inner-almost collision propagates correctly through to the end of step 36. This is not a valid assumption because of the fact that these randomized m_i values are factored into the buffers during steps 21 through 36. For each of these steps, the probability that Δ_i results in the correct values assuming Δ_{i-1} is correct is influenced by the properties of the functions *G* and *H* as well as the values of the randomized message words. The overall probability that the attack succeeds is therefore defined as:

$$\prod_{i=21}^{36} Pr[\Delta_i \textit{ is correct} \mid \Delta_{i-1} \textit{ is correct}] = 2^{-30.11}$$ [Dobbertin, 1996]

However, this is based on the assumption that we do not need to work backwards towards the true MD4 IV. Dobbertin does not outline what the theoretical probability is if reconciling the IV is necessary in the context of the complete MD4 scheme, therefore it is not the probability of a full attack. Experimentally, Dobbertin found that the probability was actually closer to 2^{-22} for a single trial successfully generating a collision using the full MD4 scheme and the proper IV [Dobbertin, 1996].

In terms of complexity, Dobbertin [1996] defines one operation as the evaluation of a single equation as all of the equations used in the attack are an arithmetic reordering of the functions used by MD4, and therefore equivalent to one MD4 step. He determined that a single trial would take approximately 16 steps worth of computation, equivalent to one third of a single MD4 compression. When factoring in the probability of a single trial successfully generating a collision, he posed that the runtime complexity of the attack is approximately equivalent to 2^{20} MD4 compression operations [Dobbertin, 1996].

In light of the fact that the runtime complexity of the attack is well under the 2^{64} hashing operations that are required for the birthday attack and that it has a high probability of success within that time, Dobbertin was able to conclude that MD4 is not collision resistant.

7 Wang et al.

In 2005, Wang et al. put forth a paper that proposed a new message difference for MD4 that aimed to reduce the runtime complexity and increase the probability of finding a collision when compared to Dobbertin's attack. In this section, we discuss the specifics of this attack and the theory behind it as presented in their paper [Wang et al., 2005].

7.1 Message Difference

Wang's attack differs from Dobbertin's in that it uses differences in multiple input words to take advantage of two complete collisions within the MD4 compression algorithm, and uses specific modification of the messages to fulfill certain conditions. These two characteristics allow for an increased probability of success as it does not rely on randomness to resolve inner collisions into true collisions. The message difference that they proposed that meet these characteristics is as follows:

$$\begin{aligned} \Delta m_1 &= 2^{31} \\ \Delta m_2 &= 2^{31} - 2^{28} \\ \Delta m_{12} &= -2^{16} \\ \Delta m_i &= 0 \qquad \forall i = 0, 3, 4, 5, 6, 7, 8, 9, 10, 11, 13, 14, 15 \end{aligned}$$

The two inner collisions exploited are one between steps 2 and 25, and another between steps 36 and 41. This implies that $\Delta_{25} = \Delta_{41} = \{0, 0, 0, 0\}$. For steps 42 to 48, the message words m_i that are used are m_9, m_5, m_{13}, m_3, m_{11}, m_7, and m_{15}, all of which are equal for both *M* and *M'*. Therefore, all operations on the buffer after step 41 are uniform for both *M* and *M'*, and the hashes will collide with a high probability. The differential path for the second inner collision is below:

Second Inner Collision						
Step	$\Delta_i = \{\Delta a_i, \Delta b_i, \Delta c_i, \Delta d_i\}$				Δm_k	Δq_i
36	0	2^{31}	0	0	-2^{16}	2^{31}
37	0	2^{31}	2^{31}	0	$-2^{28} + 2^{31}$	2^{31}
38	0	2^{31}	2^{31}	0	0	0
39	0	2^{31}	2^{31}	0	0	0
40	0	0	2^{31}	0	0	0
41	0	0	0	0	2^{31}	0

Figure 1. Second inner collision for attack by Wang et al. [Sasaki et al., 2007]

In order to ensure that the inner collisions are satisfied, Wang et al. provide a table of bit conditions for the values of Δq_i during the intermediate steps (see Table 6 in [Wang et al., 2005]). These conditions are based on the differential path and the characteristics of the functions *F*, *G*, and *H*. Initially, *M* is chosen randomly and the pair (*M*, *M'* = *M* + Δ*M*) is calculated. The probability that the conditions are satisfied and (*M*, *M'*) form a colliding pair given a uniform M is 2^{-122}. In order to use this message difference to proper effect, Wang et al. use two types of message modification that they call "single-step modification" and "multi-step modification" in order to increase the probability of a collision [Wang et al., 2005].

7.2 Message Modification

The first optimization made by this attack to increase success probability is modifying the message M in order to satisfy all the bit conditions for the first round chaining variables Δq_2 through Δq_{16}. These modifications are simply performing the bit operations required to set the necessary bit conditions to those chaining variables, or "single-step modification". Once those conditions are satisfied, a significant portion of the randomization has been controlled, increasing the probability of the new resulting pair (M, $M' = M + \Delta M$) colliding to 2^{-25} [Wang et al., 2005].

In order to increase the probability even further, multi-step modification is done. This is a process outlined to satisfy some of the bit conditions for the second round chaining variables, Δq_{17} through Δq_{32}. These modifications require more operations due to the fact that the changes they make to M can cause the bit conditions for the first round to become unsatisfied. Therefore, in order to account for changes, multiple m_i need to be modified directly, and the impacted Δq_i must be recalculated in order to satisfy these complex conditions.

Many of the multi-step modifications are outlined as part of the attack, but not all. Wang et al. [2005] claim that almost all conditions in rounds 1 and 2 can be satisfied through approximately 22 multi-step modifications. These adjustments increase the probability of generating a successful collision (M, $M' = M + \Delta M$) to somewhere in the approximate range of 2^{-6} to 2^{-2} [Wang et al., 2005]. The exact value is be determined by the round 3 conditions that are not satisfied manually, as they must be satisfied randomly when generating the unmodified M. The reason that these conditions are not set manually is because they are computationally expensive, as changing them requires modifying many more m_i and Δq_i in order to maintain previously satisfied conditions.

7.3 Theoretical Complexity

All of the non-randomization operations needed to perform this attack are within the message modification portion of the process. The modifications for Δq_2 to Δq_{16} are considered trivial due to the single operation to perform them (some combination of bitwise operators). According to Wang et al. [2005], each of the ~22 multi-step modifications only needs "about a few" step operations, and they claim that the total modification time is ≈2 MD4 computations worth of operations for a single trial. Given that, and the worst case 2^{-6} probability of a trial succeeding, this attack should generate a colliding pair (M, M') in 2^{8} operations on average [Wang et al., 2005]. This represents a 2^{13} hashing operation decrease in complexity over Dobbertin's attack.

8 Sasaki et al.

In 2007, Sasaki et al. proposed a new inner collision and new message difference for finding MD4 collisions. These improvements enabled an even more efficient attack to be produced, as prior attacks relied on Wang et al.'s inner collision and message difference. In this section, we discuss the specifics of this attack and the theory behind it as presented in their paper [Sasaki et al., 2007].

8.1 New Inner Collision and Message Difference

An efficient message difference of a collision attack on MD4 is dependent on an efficient inner collision in the third round. This is because following the differential path during the third round is computationally expensive in comparison to following the differential path during the first and second rounds [Sasaki et al., 2007]. At minimum, one message difference must occur in the third round since the attack requires colliding messages to be different. Sasaki et al. constructed an inner collision that uses just one message difference in the third round in the MSB of Δq_i, summarized in Figure 2. Steps 34, 35, 36, and 37 cancel out the difference introduced by step 33.

New Inner Collision						
Step	$\Delta_i = \{\Delta a_i, \Delta b_i, \Delta c_i, \Delta d_i\}$				Δm_k	Δq_i
33	2^{31}	0	0	0	2^{28}	2^{31}
34	2^{31}	0	0	0	2^{31}	0
35	2^{31}	0	0	0	2^{31}	0
36	2^{31}	0	0	0	2^{31}	0
37	0	0	0	0	2^{31}	0

Figure 2. New inner collision proposed by Sasaki et al. [Sasaki et al., 2007]

The message difference used to exploit this is as follows:

$$\begin{aligned} \Delta m_0 &= 2^{28} \\ \Delta m_2 &= 2^{31} \\ \Delta m_4 &= 2^{31} \\ \Delta m_8 &= 2^{31} \\ \Delta m_{12} &= 2^{31} \\ \Delta m_i &= 0 \qquad \forall i = 1, 3, 5, 6, 7, 9, 10, 11, 13, 14, 15 \end{aligned}$$

Sasaki et al. also proposed a differential path construction algorithm that they used to determine the path for their new message difference. The full differential path found, the conditions generated to follow the path, and the message modification procedures for each condition are found in [Sasaki et al., 2007] in Tables 7, 8, and 9 through 20, respectively.

As with Wang et al.'s attack, message modification is applied to randomly generated messages in an attempt to satisfy the sufficient conditions. Because of the reduced number of steps involved with the third round collision, only 1 bit difference is needed in the third round instead of 2, and thus fewer conditions are needed that cannot be guaranteed to be satisfied. Therefore, a single trial succeeds in finding a colliding message pair (M, $M' = M + \Delta M$) with higher probability than with Wang et al.'s attack due to decreased reliance on randomization. Sasaki et al. [2007] do not provide a numerical success probability for which their algorithm finds a collision, only stating that it is "with high probability".

8.2 Theoretical Complexity

Based on the pseudocode provided in their paper, it appears a collision is guaranteed with enough repetition of the final message modification to satisfy the last sufficient condition. This pseudocode has three main components, each of which is outlined below along with the presented complexity.

First, for $1 \leq i \leq 16$, the value of q_i is randomly generated and changed to satisfy all its conditions. Once satisfied, m_{i-1} is computed as $(q_i >>> s_i) - q_{i-4} - F(q_{i-1}, q_{i-2}, q_{i-3})$. Both random number generation and modifying q_i to satisfy all conditions take O(1) time. Calculating m_{i-1} requires almost 1 MD4 step, therefore this component has a total runtime of 16 MD4 steps [Sasaki et al., 2007].

Second, for $17 \leq i \leq 29$, q_i is computed as $q_i = (q_{i-4} + G(q_{i-1}, q_{i-2}, q_{i-3}) + m_k + 5A827999) <<< s_i$. Then for each bit position j of q_i, message modification is applied if the condition on $q_{i,j}$ is not satisfied. Computation of each q_i requires 1 MD4 step. 11 conditions and their corresponding message modification procedures exist over these values of i, each requiring less than 3 MD4 steps [Sasaki et al., 2007]. Each of these conditions is already satisfied with probability 1/2 due to the randomization of M, therefore $(3 \cdot 11) / 2 = 16.5$ MD4 steps are needed [Sasaki et al., 2007]. Thus, a total of 29.5 MD4 steps are computed in this stage.

Third, q_{30}, q_{31}, and q_{32} are computed as $q_i = (q_{i-4} + G(q_{i-1}, q_{i-2}, q_{i-3}) + m_k + 5A827999) <<< s_i$, and $q_{33} = (q_{29} + H(q_{32}, q_{31}, q_{30}) + m_0 + 6ED9EBA1) <<< 3$. If the condition of $q_{33,31} = 0$ is not satisfied, message modification is applied and this component is repeated. Again, the computation of each q_i requires 1 MD4 step. Then, the condition of $q_{33,31} = 0$ can either be true or false. If true, no further steps are taken. If false, message modification of less than 1 MD4 step occurs and this component is repeated. Sasaki et al. claim this condition can be expected to be met with at most 2 attempts [Sasaki et al., 2007]. Therefore, the complexity of this component is the average of these two scenarios, i.e., $(4 + 9) / 2 = 6.5$ MD4 steps [Sasaki et al., 2007].

Finally, the resulting message is defined as M, and $M' = M + \Delta M$. This computation is O(1). Bringing it all together, this attack is expected to take $16 + 29.5 + 6.5 = 52$ MD4 steps in total. As one MD4 operation takes 48 MD4 steps, the complexity is less than 2 MD4 operations. Even if all conditions in the second component of the attack are not satisfied and require message modification, the complexity is still below 2 MD4 operations at 68.5 MD4 steps.

9 Implementation of Algorithms

To gain a more thorough understanding of these three attacks and enable us to compare their actual performance, we set out to implement each algorithm. A publicly available implementation of Wang et al.'s attack was found online at [Stach, 2006]. Using this code as a template, we developed implementations of Dobbertin's and Sasaki et al.'s attacks based off of the pseudocode found in their respective papers. All three implementations are in C. A bash script was also written to easily confirm collisions of the MD4 hashes of outputted message pairs.

Unfortunately, due to various factors, we were unable to successfully generate colliding message pairs within the timeline of this project. We discuss the implementation of each algorithm separately in the following sub-sections.

9.1 Dobbertin

The primary issue encountered with the implementation of the Dobbertin attack is finding an inner-almost collision. Dobbertin's outline of the attack process defines the generation of these collisions as a "continuous approximation" towards the necessary $\Delta_{20} = \{0, 2^{24}, -2^{4}, 0\}$. Step 2 of this process involves randomizing bits of the chaining variables q_{15} through q_{20}, using these temporary values to recalculate the other chaining variables in the inner-almost collision, and testing the 4 most significant bits of an equation representing Δq_{15} to see if they equal 0. If they do, the new values of q_{15} through q_{20} are set, and the process is repeated for the next 8, 12, 16, …, 32 most significant bits until the equation is entirely satisfied [Dobbertin, 1996]. Throughout all of our testing and various interpretations of the pseudocode, including randomizing the same bit in all chaining variables and restarting the process every time the left x bits of the equation were not satisfied, we were unable to complete this step of the process. Allowing the program to run for long periods of time (>8 hours) did not solve the issue either. When the program is run in its current state the output will indicate when a certain number of bits in the equation are satisfied. It will also indicate when it restarts the entire collision generation process after a certain number of attempts to solve the equation for Δq_{15}, printing the values of the state buffers for M and M' at that time before restarting.

We believe that this issue is because of a misinterpretation of the vague instructions given in the pseudocode, which is hard to decipher as the need for, and value of, the continuous approximation is not given in any detail. This information would allow us to get a better understanding of how it fits in the context of the attack and discern the true interpretation. With that information, or time for more trial and error for various interpretations of the relevant step, we believe we would be able to successfully implement Dobbertin's attack.

9.2 Wang et al.

As mentioned above, a publicly available implementation of Wang et al.'s attack was found at [Stach, 2006]. It has been modified only to match the notation found in this paper, and has been included along with our implementations.

9.3 Sasaki et al.

The main obstacle in completing the Sasaki et al. implementation is clarity in components of the algorithm, due to typos or breverity.

There are two separate occasions where a condition specified for the new inner collision in Table 8 of [Sasaki et al., 2007] does not match the correlating modification procedure provided in the paper. The first case of this is for the chaining variable $b_{18,28}$. In the inner collision table, the condition specified is 'c', or $b_{18,28} = b_{16,28}$. However, the correlating modification procedure (Table 12 of [Sasaki et al., 2007]) specifies the condition as $b_{18,28} = 0$ instead. The second case of this is for the chaining variable $b_{22,31}$. In the inner collision table, the condition specified is 'c', i.e., $b_{22,31} = b_{20,31}$. However, the correlating modification procedure (Table 18 of [Sasaki et al., 2007]) specifies the condition $b_{22,31} = b_{21,31}$.

The final modification procedure for the condition $b_{33,31} = 0$ is provided in Table 20 of [Sasaki et al., 2007]. It is unclear as to whether all extra conditions for the various values of i should be set all at once, or one at a time. In the actual procedure steps, it is also unclear as to if the notation for $i - 3$ and $i + 16$ is mod 32, or if values outside of 0 - 31 are ignored.

With more time to experiment and figure out the intentions of these components, we are confident a successful implementation of Sasaki et al.'s attack could be achieved.

10 Conclusion

In March 2011, RFC 6150 stated RFC 1320 as obsolete [Turner & Chen, 2011]. This is due in part to fact that MD4 is not collision resistant, as shown by the three differential attacks studied in this paper. Such attacks continue to gain efficiency through iterative improvements. The implementation of Wang et al. provided by [Stach, 2006] generates a collision in mere seconds on a standard personal computer. A complete implementation of

Sasaki et al.'s algorithms would produce a colliding message pair in even less time based on its theoretical complexity. As a result, MD4 should not be used when a collision resistant hashing function is needed. Despite MD4 being of little use today, the differential cryptanalysis of these attacks is beneficial and can be related to more recently developed hashing functions.

References

Boer, Bert Den, and Antoon Bosselaers. "An Attack on the Last Two Rounds of MD4." *Advances in Cryptology — CRYPTO '91 Lecture Notes in Computer Science* (1992): 194-203. Print.

Dobbertin, Hans. "Cryptanalysis of MD4." *Fast Software Encryption Lecture Notes in Computer Science* (1996): 53-69. Print.

Heys, Howard M. "A Tutorial on Linear and Differential Cryptanalysis."*Cryptologia* 26.3 (2002): 189-221. Print.

Rivest, Ronald. "RFC 1320 - The MD4 Message-Digest Algorithm." *RFC 1320 - The MD4 Message-Digest Algorithm*. Network Working Group, 1992. Web. 17 Nov. 2015. <https://tools.ietf.org/html/rfc1320>.

Rivest, Ronald. "The MD4 Message Digest Algorithm." *Advances in Cryptology-CRYPT0' 90* (1991): 303-11. Print.

Sasaki, Yu, Lei Wang, Kazuo Ohta, and Noboru Kunihiro. "New Message Difference for MD4." *Fast Software Encryption, Lecture Notes in Computer Science* (2007): 329-48. Print.

Sotirov, Alexander, et al. "MD5 Considered Harmful Today, Creating a Rogue CA Certificate." *25th Annual Chaos Communication Congress*. No. EPFL-CONF-164547. 2008. Print.

Stach, Patrick. "MD5 and MD4 Collision Generators." Bishop Fox, 2006. Web. 25 Nov. 2015. <http://www.bishopfox.com/resources/tools/other-free-tools/md4md5-collision-code/>.

Turner, S., and L. Chen. "RFC 6150 - MD4 to Historic Status." *RFC 6150 - MD4 to Historic Status*. Internet Engineering Task Force, 2011. Web. 9 Dec. 2015. <https://tools.ietf.org/html/rfc6150>.

Wang, Xiaoyun, Xuejia Lai, Dengguo Feng, Hui Chen, and Xiuyuan Yu. "Cryptanalysis of the Hash Functions MD4 and RIPEMD." *Lecture Notes in Computer Science Advances in Cryptology – EUROCRYPT 2005* (2005): 1-18. Print.