Collision Resistant Usage of SHA-1 via Message Pre-processing

Michael Szydlo

RSA Security

Yiqun Lisa Yin

Independent Consultant

Recent Advances in Hash Collision Attacks

- Efficient collisions found for MD4, MD5
  - Improved techniques include differential, message modification approaches
  - Other hash functions affected
- Wang, Yin, Yu focus on full SHA-1 (2005)
  - Complexity of collision 2^{69} now improved to 2^{63}
  - Compare to design goal of 2^{80}
- Security community planning response

Standard Track Response

- Option #1: Upgrade hash function
  - Completely new hash function
  - Use SHA-256
  - Truncate to SHA-256 output to 160 bits
- Option #2: Re-design affected protocols
  - Incorporate randomness into hashing
  - Randomized Hashing (Halevi, Krawczyk)
    - $H_r(m) = H(m \text{ XOR } r||r||r\ldots r)$
    - $\text{RSASign}(m) = (r, \text{RSA}(r, H_r(m))$

Considerations

- Upgrade Option
  - New hash function design and standardization takes years
  - Larger output of SHA-256 inconvenient
  - Security of “Truncated SHA-256” must be explicitly studied
- Randomized Hashing Option
  - Randomness is required and needs to be managed
  - Possible changes in signature size
  - Alter protocols such as those in PKCS#1

Message Pre-processing

- A simple message transformation
  - Intuition: Add redundancy to message
  - M' = Φ(M), Φ is very simple function
  - New derived hash function is
    - SHApp(m) = SHA-1(Φ(M))
- Effects on applications
  - Prevents all known collision attacks
  - Φ stretches message length 33-100%

Two Candidate Transformations

- Message Whitening (word-wise)
  - $m_1\ m_2\ m_3\ m_4\ m_5$ … becomes
  - $m_1\ m_2$ … m_{12} 0 0 0 0 $m_{13}\ m_{14}$ … m_{24} 0 0 0 0 m_{25}…
  - Each block contains *whitened* words
- Message Interleaving
  - $m_1\ m_2\ m_3\ m_4\ m_5$ … becomes
  - $m_1\ m_1\ m_2\ m_2\ m_3\ m_3$ …
  - Each block contains *duplicated* words

Implementation Options

- Pre-processing within SHA-1 Function
  - Change SHAUpdate() to SHAppUpdate()
  - New function SHAppUpdate()
    - expands m via Φ
    - calls usual SHAUpdate() as black box
- Pre-processing outside SHA-1 Function
  - Processing occurs first and then calls usual SHA-1 as black box
- Two options are interoperable
  - Which option is better depends on the application

Implementation and Security Features

- Zero “API signature” change
  - Output of SHApp(m) is automatically 160-bit
- Almost zero change to protocol
  - Only need a new algorithm identifier for SHApp
- Security analysis
  - Leverages on existing analysis of SHA-1
  - Effects of pre-processing techniques can be quantified

Comparing Approaches

	Truncate SHA-256	Random Hash	Preprocess
Hash Output Truncation	√		
Change Signature Size		√	
Randomness Required		√	
Replace SHA1 Code	√		
Change Message before Hashing		√	√
Execution Cost (time increase)	50-200% Depends on SHA-256 slowdown on platform	(not %) Depends on random generation	33-100% Depends whitening parameter

Components in Security Analysis

- Understand differential attack on SHA1
  - Very specific message differentials employed
  - Message modification changes message bits
- 1) Message redundancy reduces flexibility
  - Expanded message as a code word
  - Message whitening & interleaving changes code
  - Fewer low hamming weight codes
- 2) Message modification don’t work
- *Existing* attacks can’t beat 2^{80}

Conclusions

- Message preprocessing is viable solution to increasing secure life of SHA-1
- Technique can also be applied to MD5
- Long term solutions involve design of new hash function from the ground up
- See paper for additional detail including security analysis
  - Available online at: http://eprint.iacr.org/2005/248

Collision-Resistant usage of MD5 and SHA-1 via Message Preprocessing

Michael Szydlo[1] and Yiqun Lisa Yin[2]

[1] RSA Laboratories, Bedford, MA 01730.
mszydlo@rsasecurity.com
[2] Independent Security Consultant
yiqun@alum.mit.edu

Abstract. A series of recent papers have demonstrated collision attacks on popularly used hash functions, including the widely deployed MD5 and SHA-1 algorithm. To assess this threat, the natural response has been to evaluate the extent to which various protocols actually depend on collision resistance for their security, and potentially schedule an upgrade to a stronger hash function. Other options involve altering the protocol in some way. This work suggests a different option. We present several simple message pre-processing techniques and show how the techniques can be combined with MD5 or SHA-1 so that applications are no longer vulnerable to the known collision attacks. For some applications, this may a viable alternative to upgrading the hash function.

Key words: SHA-1, MD5, padding, hash collision, signature

1 Introduction

The recent advances in cryptanalysis of hash functions have been spectacular, and the collision attacks on MD5 and SHA-1 are of particular practical importance since these algorithms are so widely deployed. To assess the threat, the first step is to re-examine which protocols actually depend on collision resistance for their security. The most common type of vulnerable application is the use of standard signatures to provide non-repudiation or certification services.

Applications which do not require collision resistance are unlikely to require changes in the near future as a result of these recent collision results. For those that do, changing the hash function is the simplest response, and the standardized SHA-2[21] family (which includes SHA-256) is the leading candidate for an upgrade. Although it has not received the same amount of analysis as earlier hash functions, SHA-256 is expected to be significantly stronger. There has been some progress analyzing SHA-256, for example [10] and [8]. These papers show that SHA-256 also has “local collisions” (defined in [5]) with probability between 2^{-9} and 2^{-39}. This implies that the security of SHA-256 is mainly hinged on its message pre-processing.

A second alternative is to re-design the protocols themselves, so they no longer rely on collision resistance of the hash function. This can be done on a case by case basis or in a more uniform fashion. For example, a recent Internet Draft [9] proposes to change signature scheme protocols by use of a primitive called “randomized hashing”. There are some architectural advantages to changing the signature scheme in such a modular way, replacing each hash invocation with a random member of the hash family. Any protocol employing this solution will require a good source of randomness, and will also need to specify and manage the random hash family member. This kind of solution can certainly be considered viable when the additional resource of randomness is readily available.

This paper points out a third option: There are simple, alternate modes of using MD5 or SHA-1, in a manner which renders them no longer susceptible to the known collision attacks. These approaches essentially involve some light message pre-processing code to effectively *derive* a new hash function from an old one. Although the exact same standardized hash function is used, this technique can be viewed as an indirect but convenient way of effectively upgrading the hash function. Advantages include the fact that no additional resource of randomness is needed and no change to the output length or truncation is required. In the short term, some implementations might find this to be a compelling alternative which will serve to extend the useful life of MD5 or SHA-1.

Organization

In Sections 2 we present some background material on the hash functions under consideration, and in and Section 3 review the nature of the recent collision attacks. In Section 4 we further motivate and present the basic message preprocessing technique. In Section 5 we present the details of the construction. Focusing primarily on SHA-1, we analyze the security in light of the known collision attacks in Section 6. An alternate approach to message preprocessing is described in Section 8. In Section 9, we provide analysis specific to MD5. Final conclusions and recommendations are made in Section 10.

2 Background

2.1 The MDx Family of Hash Functions

The MDx family of hash functions includes MD4 (1990) and MD5 (1991), which were designed by Ron Rivest to be one-way and collision resistant. SHA-0 (1993), SHA-1, (1995) and SHA-2 (2001) were produced by the NSA and standardized by NIST and follow similar design principles as Rivest’s algorithms. SHA-1 is currently the FIPS Secure hash standard [20], and is the most widely deployed hash function. An earlier version of this algorithm was SHA-0 [19], while the SHA-2 family is intended for higher security levels. Until recently, SHA-1 was considered to be as secure as its 160 bit output would allow, and MD5 also still enjoys significant deployment.

The general approach behind the design of these hash function involves the Merkle-Damgård *iterative structure*, (see [6, 15]), to allow arbitrary length messages. The algorithms divide the input into fixed length blocks and process the blocks sequentially by updating an initial state variable. Each block is combined with the previous state in a *compression function* to calculate an updated state, or *chaining variable*. When the entire message has been processed, the output is the final state. The state vectors for MD5 and SHA-1 are 128 bits and 160 bits, respectively.

Coron, et. al. [4] suggest a modification of the Merkle-Damgård chaining method. However, their work is orthogonal to ours since we focus on the compression function, except in the IV message dependent approach of Section 8.

The compression functions consist of two basic components, *message expansion* and *round operations*. The compression function of SHA-1 operates on 512-bit message blocks, and utilizes a 160 bit state variable, represented by five 32-bit words, denoted A, B, C, D, E. The block of 512 bits is expanded to 2560 bits, represented by 80 words of 32 bits. Each of these words is used to update the internal state in a *round update function*. MD5 follows a similar structure, but uses a 128 bit state variable, and has 64 rounds instead of 80.

2.2 Collision Attacks on MD5 and SHA-1

Successful cryptanalysis of these hash functions has generally focused on finding collisions, rather than on inverting the hash functions. Wang, et. al. announced real collisions for MD4, MD5, RIPEMD HAVAL-128 in 2004 and 2005 in [22, 23, 25], and also introduced *message modification techniques*. These results have been improved by Klima [13], and Naito et. al. [17], and as of writing, the complexity of locating a collision in MD4 and MD5 are approximately 2^2 and 2^{30}.

Regarding SHA-0 and SHA-1, early analysis in 1998 by Chabaud and Joux used differential methods (local collisions and disturbance vectors) to find a collision attack on SHA-0 of complexity 2^{61} [5]. Biham and Chen found near collisions on SHA-0 in complexity 2^{40} [1]. The work of Biham, Joux, and Chen included the first real collision of SHA-0 in [11, 3]. Additional work on reduced round versions of SHA-1 appeared in [2, 18, 16]. Recently Wang, Yin, and Yu described an improved attack on SHA-0 in [26], and finally, in [24] presented the first attack on the full SHA-1, where they show that finding collisions is at most of complexity 2^{69}. Improvements to these attacks were announced in [27] where the attack complexity has been reduced to 2^{63}.

2.3 From Random Collisions to Meaningful Collisions

An early critique had been the initial collisions found by researchers results have involved just a few message blocks or short binary strings, without enough structure to be considered "meaningful" collisions. However, meaningful collisions *can* be found for these hash functions, and regardless, general collision resistance is a real design goal of hash function construction. For example, Lenstra et. al. [14] have found collisions between two distinct X.509 certificates, and collisions between two properly formatted postscript documents has been exhibited in [7]. Each of these examples involved the MD5 hash function. Examining these two examples, we see that they exploit the relative freedom in the form of certificates and postscript documents, and one may still argue that collisions are likely difficult to produce among messages of a suitably restricted form.

3 Analyzing the Recent Collision Attacks

In this section, we analyze the nature of the recent collision attacks on the MDx family of the hash functions and motivate techniques that would be useful to thwart such attacks. Throughout this paper, we will use M to denote a message to be hashed. Both MD5 and SHA-1 break the message to be hashed into 512 bit blocks. When we need to refer to a single block we denote it m, and m is often partitioned into sixteen 32-bit message words denoted by $m_0, m_1, ..., m_{15}$. When referring to the expansion function internal to the compression function, we denote the expanded message block by w. In the case of SHA-1, w is partitioned into eighty 32-bit message words, denoted by $w_0, \ldots w_{79}$. In the case of MD5, w is expanded into 64 32-bit message words. We use C to denote the compression function, and H to denote the complete hash function.

3.1 Basic Ideas in the Collision Attacks

We first briefly review some of the basic ideas behind these attacks. Focusing on a single block, the general common strategy behind these collision attacks involves finding a message difference $\Delta(w) = w - w'$ between two expanded messages such that the probability that $C(m)$ equals $C(m')$ is higher than expected. This is possible when it can be arranged such that during the round computations of the blocks m and m' the state vectors never deviate significantly, and can be "corrected" with high enough probability.

The basic tool is the *local collision*, a series of a few rounds in which certain small differences in the expanded message words will be absorbed with reasonable probability. Due to the message expansion there will be many differing words of m and m', so these local collisions must be strung together. *Disturbance vectors* describe how the local collisions are joined. The entire sequence of differences in the state vectors is called a *differential path.* The overall success probability depends on the simultaneous satisfaction of a set of conditions for each local collision.

The structure of the various attacks consist of analysis of the local collisions, search for a low Hamming weight disturbance vector, a brute force search on input messages, and a variety of methods are used to boost the success probability, including specifying concrete conditions for the differential path, *message* *modification* so that some conditions always hold, and usage of two blocks to construct collisions from near collisions. We remark that the above summary most accurately describes the approaches for SHA-1, and the analysis of MD5 differs slightly.

3.2 Thwarting the Collision Attacks

From the summary of the attacks above we can see that there are several strategies which one might employ to attempt to prevent the success of these approaches. The most obvious approach is to attempt to prevent the existence of any "good" differential – a differential path that leads to (near) collisions and holds with probability greater than $2^{-n/2}$. An additional precaution would be to restrain the power of the message modification techniques, thereby significantly reducing the success probability of the attack. A third possibility is to consider situations in which the Merkle-Damgård iterative structure can not be exploited; for example if single message bits were to affect multiple blocks.

4 Message Pre-processing Techniques

In this section we describe the general message preprocessing framework, and discuss the *streaming* requirement that some applications may have.

4.1 Message Pre-processing Framework

The working assumption behind the general techniques we suggest for improving the collision resistance is that the underlying hash function itself will not be changed. Let M be a message string to be hashed, and let H be a standard hash function, such as MD5, or SHA-1. Our objective is to define a *derived* hash function H^* which calls H as a subroutine. Our proposal is simply to preprocess the message before it is hashed in a standard way. Formally, let $\phi : M \mapsto M^*$ be a *preprocessing* function mapping strings to strings. For each such function, a derived hash function H^* may be defined by

$$H^*(M) = H(\phi(M)).$$

Of course, we are interested in cases where ϕ is a relatively simple function, and the derived hash function H^* is collision resistant with respect to known attacks, even if H is not. The function ϕ must be chosen appropriately for a particular H to ensure that H^* is secure.

4.2 Streaming Data Requirement

Many applications are set up architecturally to incrementally digest a large message as it becomes available. For example, with SHA-1, applications can repeatedly make a *SHA-1Update* function call as portions of the message stream in.

This requirement can be satisfied when the message pre-processing can also be performed in a streaming fashion, for example, by dividing the message into blocks and expanding each one. Formally, we call a ϕ a *local expansion* if ϕ can be defined by $\phi(m_0, m_1, \ldots m_k) = m_0^*, m_1^*, \ldots m_k^*$ where each m_i is of fixed length and $m_i^* = f(m_i)$ for some expansion function $f : \{0,1\}^l \rightarrow \{0,1\}^{l^*}$, where $l^* > l$. It is clear that when ϕ is a local expansion, the state of the preprocessing function can be stored in the message digest context, so that a derived update function could also call *SHA-1Update* as a subroutine.

5 Local Expansion Approaches

We now discuss two local expansion approaches to message preprocessing: message whitening and message interleaving.

5.1 Message Whitening

In this approach, the basic idea is to alter the message by inserting fixed characters at regular intervals. The motivation here is to decrease the flexibility in finding good message differentials. These fixed characters can be taken to be words filled with all zero bits, so we call the approach *whitening*. For a hash function with at 512-bit block size, sequential chunks of fewer than 512 bits can be expanded into a full 512 bits. For example each sequence of $(16-t)$ 32-bit words $m = (m_0, m_1, \ldots m_{15-t})$ could be expanded to $m = (m_0, m_1, \ldots m_{15-t}, 0, \ldots, 0)$, where the last t words would be fixed as zeros. Each execution of the compression function effectively only process $(16-t)$ message words rather than 16 message words, so it is easy to calculate the performance slowdown. This approach is also easy to implement, since such a preprocessing function ϕ is a local expansion, the streaming requirement would be met. From a security standpoint, the intuition is that processing fewer bits of message should allow the message to be better mixed within the calculation.

A variant of this approach may select specific words to whiten to further increase the difficulty of known attacks. Below, we discuss how whitening the middle two words of SHA-1 significantly reduces the effect of message modification techniques.

5.2 Message Self Interleaving

In this approach, the basic idea is to duplicate each message word so that each bit appears twice after the preprocessing. Assuming the entire message M is broken up into some number of 32-bit words: $M = (m_0, m_1, \ldots m_k)$, then the preprocessed message would be $\phi(m) = (m_0, m_0, m_1, m_1, \ldots m_k, m_k)$ where each word appears twice. As with the message whitening approach, message interleaving causes fewer message bits to be fed into each message block, causing better mixing. As ϕ is a local expansion, the streaming requirement is also met.

5.3 Generalized Local Expansion

The whitening and interleaving approaches discussed above have obvious minor variations, such as choice of which bits to whiten. The frequency of message interleaving could also be chosen word by word, rather than character by character. Both of these approaches, as well as the minor variants have the property that the local expansion is a linear function. Thus, one way to generalize is to consider an arbitrary linear function. Although we prefer simpler pre-processing functions, one could certainly consider non-linear functions as well, effectively using an arbitrary local expansion. Regardless of the specific function, these approaches all attempt to increase security by increasing the structure of each message block. This can make finding good differentials more difficult for the attacker, as well as disrupt message modification techniques.

6 Security Analysis of Local Expansion Approaches

In this section, we further discuss why the message pre-processing techniques described in the preceding section help prevent existing attacks. We focus our discussions on SHA-1.

6.1 Intuition

The message whitening and message interleaving both operate by increasing the structure within each block. For these approaches and their variants, we can simply view the derived hash function as a modification of the original hash function, except with a different message expansion rule.

Concretely, in the case of SHA-1, the message interleaving approach effectively takes as input 256 bits of data instead of 512, and expands them to the 80 words required by the SHA-1 round operations. The amount of data required by the whitening approach would depend on its calibration, i.e., how many bits or words were whitened. Intuitively, this means that fewer data bits are processed for each execution of the compression function, and hence the derived hash function can offer a better mixing of the data bits.

In the following, we provide more quantitative analysis of the two message pre-processing approaches by considering how they affect constructing good differentials and performing message modification, both of which are critical in existing collision attacks.

6.2 Insights from Coding Theory

One way to understand the effect of message whitening or message interleaving is to study the *code of expanded message words*. For hash functions which employ a linear message expansion rule the space of expanded messages is a linear code, so we have a tool to reason about the existence of low Hamming weight vectors.

For example, for SHA-1 each block expands 16×32 bits into 80×32 bits. The expansion function $E\{0,1\}^{512} \rightarrow \{0,1\}^{2560}$ is defined word-wise by the recurrence relation

$$w_t = (w_{t-3} \oplus w_{t-8} \oplus w_{t-14} \oplus w_{t-16}) <<< 1. \tag{1}$$

For MD5, the original message is simply repeated 3 times, so the expanded message words of both MD5 and SHA1 can be viewed as linear codes of dimension 512. In either case, the code is generated by the 512 basis vectors $E(1, 0, \ldots, 0)$, $E(0, 1, \ldots, 0)$, ..., $E(0, \ldots, 0, 1)$.

Both the interleaving and whitening approaches work by restricting the form of the 512-bit input message block, thus restricting full code of expanded message words. The form of the whitened message is $m^* = (m_0, m_1, \ldots m_{15-t}, 0, \ldots 0)$ so the restricted code is $512 - 32t$ dimensional, generated by basis vectors corresponding to the non-whitened bits. The form of an interleaved message is $m^* = (m_0, m_0, m_1, m_1, ..., m_7, m_7)$, so this code is only 256 dimensional, generated by vectors of the form $(1, 0, \ldots ; 1, 0, \ldots ; 0; \ldots)$, where each generator consists of zeros except for two matching 1 bits. When we view the collision attacks as attempts to piece together local collisions in a manner consistent with this linear code, it becomes clear that reducing the dimension of the code will make these attacks less feasible.

Reducing Solutions to Linearized Hash Function: Another way to understand the whitening and message interleaving is in terms of the set of solutions to a linearized version of the hash function. This is the approach followed by Oswald and Rijmen in [18]. Rather than focus on local collisions, they analyze the difference between the linearized and actual SHA-1, so that each difference in the expanded message word yields one or more conditions which will be only probabilistically satisfied in the actual SHA-1. They search for low Hamming weight code words $\Delta(w)$ which are also solutions to the linearized SHA-1 equation. The solutions yielding an output of 160 zeros are defined by an additional 160 linear constraints (see [18] for details), so it is natural to consider the *code of linear solutions*, consisting of expanded message words which also satisfy these 160 constraints. This restricted code has dimension 512-160=352, and the collision attack first seeks a low Hamming weight code words, then a message pair such that the conditions will be satisfied.

In this framework, our message interleaving approach corresponds to the addition of 256 additional constraints, and the whitening approach corresponds to the addition of $32t$ additional constraints. Although there is no simple way to locate low Hamming weight codewords in an arbitrary code, the existence and number of lower weight words decreases as the *minimum relative distance* goes up. This ratio is simply the ratio of the code length to the code dimension, and equals $352/2560 = 7.27$ for the original code. This code, restricted with whitening parameter t, has dimension $352 - 32t$, so in case $t = 4$, the minimum relative distance is increase to 8.88. If, instead, the code is restricted by the interleaving approach, the dimension is reduced to $352 - 256 = 96$, so the minimum relative distance increases to 26.66. This heuristic does not preclude the existence of good differentials, but it does provide a useful metric for how restricting the form of messages will increase the difficulty of the known collision attacks.

6.3 Preventing Good Differentials

We now address more concretely the best known attacks on SHA-1. As discussed earlier, a major step for constructing a good differential path for SHA-1 is to find a disturbance vector with low Hamming weight. In this section, we consider how message pre-processing affects constructing good differentials.

First, we review some basic facts of the SHA-1 disturbance vectors. A disturbance vector dv is a set of 80 32-bit words dv_i ($i = 0, ..., 79$), and $dv_{i,j} = 1$ iff a local collision starts in step i bit j. Each local collision consists of 5 additional changes in the expanded message word, called *correction vectors* (See [5]). The correction vectors wc_1, wc_2, wc_3, wc_4, wc_5 are automatically linearly determined from dv, and the difference in the expanded messages is simply the sum $\Delta(w) = dv + \Sigma wc_i \pmod 2$. Although, only $\Delta(w)$ must be a code word (i.e. satisfy the recurrence relation), in practical attacks dv itself is taken to be a code word, so that the five $\{wc_i\}$ and $\Delta(w)$ are automatically code words. The Hamming weight of dv, denoted by $HW(dv)$ is the central important factor in determining the success of the collision attacks, an estimate of the complexity of an attack on SHA-1 is about $2^{3HW(dv)}$.[1]

In the attacks on SHA-1 [24, 27], disturbance vectors of low Hamming weight were found by a heuristic search algorithm, and it is based on the following intuition: If we view dv as an 80-by-32 0-1 matrix, then the non-zero entries in a low Hamming dv are likely to be concentrated in one column. The search algorithm proceeds by first choosing a 16-bit column in the matrix and expanding backwards and forwards with message expansion. The best vector is then chosen among all possible choices for the column. Using this heuristic search, the lowest Hamming weight is reached when the 16-bit column takes the value $L = (100...000)$.

Now we are ready to analyze how the two pre-processing techniques affect finding low Hamming weight disturbance vectors. For the message whitening technique, each whitened message word m_i^* would yield an extra condition on the differential path, namely

$$\Delta m_i^* = 0. \tag{2}$$

For the interleaving techniques, the extra conditions on Δm^* are

$$\Delta m_{2i}^* = \Delta m_{2i+1}^*, \text{ for } i = 0, 1, ..., 7. \tag{3}$$

Experiments: We used the same heuristic search algorithm to find disturbance vectors for "SHA-1 with message pre-processing". Our assumption is that a good

[1] It was the introduction of message modification techniques in [24] that allowed the initial conditions in steps 1-20 to be automatically satisfied so that the limiting factor was actually the Hamming weight in the final 60 words of dv.

disturbance vector follows similar patterns as the ones for the original SHA-1. Starting with L, we computed 150 words of dv by expanding L forwards and backwards with the recurrence relation E and compute many words of Δw from dv. The words of this extended dv may be found in the rows of Table 5 in reference [24]. The next step is to pick 80 words from the computed Δw such as the above conditions due to whitening (or interleaving) are satisfied while keeping the Hamming weight as small as possible. Using the numbering of [24], and focusing on whitening with $t = 2$, we examined the values of Δw, and see that there are no two consecutive zero words before step 55, and there are no two consecutive words that are the same before step 53. This means that we have to shift down by 40 words when choosing a good disturbance vector, in order for Δw to satisfy the message pre-processing conditions. This would cause a significant increase in the Hamming weight of the vector, so these experimental results suggest that the Hamming weight of the disturbance vector (restricted to steps 21-80) would go from 25 (for SHA-1) to over 80. Even if advanced message modification such as that announced in [27] progresses to 32 steps, the hamming weight for the remaining 48 steps would be sufficient.

We remark that the conditions on Δm_i^* given in the above two equations are *necessary* conditions for the differential path to be constructed, since they are derived from the pair of input message words m_i and m_i'. This is in contrast to the three conditions on the disturbance vectors in the original attack on SHA-0 [5] as well several works on SHA-1 [3, 24]. Those conditions are for easier construction of a valid path from the disturbance vector, and so they are *not necessary* conditions. That's why these conditions can be removed as in the attack on the full SHA-1 [24]. However, the above conditions, due to message pre-processing, cannot be removed. Finally, the techniques of Jutla and Patthak [12] could be adapted to provide rigorous bounds on the hamming weights of the codes associated to whitened or interleaved message blocks.

6.4 Weakening Message Modification

In addition to preventing good differentials, the whitening and interleaving approaches also render the message modification techniques less effective, thereby increasing the complexity of existing collision attacks.

First, we briefly review the basics of the message modification techniques [25, 23]. For the MD4-family of hash functions, including MD5 and SHA-1, the round function has the following general form:

$$a_i = G(\text{input chaining variables}) + m_{i-1},$$

where a_i is the output chaining variable and m_{i-1} is the message word used in step i. Once the differential path has been constructed, it is easy to derive a set of sufficient conditions on a_i that ensure that all conditions on path hold. The conditions are of the form $a_{i,j} = v$, where v is 0 or 1. The main idea of the message modification techniques is simply to set $a_{i,j}$ to the correct bit v and then recompute $m_{i-1} = a_i - G()$. In other words, we can modify the message word in step i to make the condition on a_i to hold. This basic technique can be used for the first 16 steps since the message words are all independent of each other up this point. A simple variation of the basic technique is to modify the message words used in the two steps *before* step i (i.e., m_{i-2} or m_{i-3}) to achieve the same goal. This is particular useful when m_{i-1} cannot be modified due to other constraints. In addition, more advanced techniques, called multi-step message modification techniques, were introduced for dealing with computation beyond the first 16 steps. The improvements announced in [27] are achieved with such advanced message modification techniques.

Next, we analyze how message interleaving affects the effectiveness of message modification. Since $m^*_{2i} = m^*_{2i+1}$ (for $i = 0, 1, ..., 7$), the two consecutive message words have to be modified simultaneously, making it almost impossible to change any single bit. Now suppose a differential path P has already been chosen, and conditions on a_i have been determined. Since most of these conditions can no longer be made to hold through message modification, the complexity of the attack using path P would go up significantly.

In the case of whitening, the t whitened message words $m^*_i, m^*_{i+1}, ...m^*_{i+t-1}$ cannot be modified, since these message words are simply zero and independent of the input message. It is possible to modify a couple of message words immediately before the whitening step so that some of the conditions on a_i and a_{i+1} can still hold, but the effect can be weakened if we choose $t \geq 4$.

Targeted Whitening: For a given path P, it is good to choose the t consecutive message words that maximize the total number of conditions s in those steps. In the attack on SHA-1 [24], the conditions on a_i are given in Table 12. From the table, it is easy to see that if $t = 4$, and we whiten words 7 to 10, the total number of conditions is $s = 83$.

It is possible that the attacker could select a new differential path P' other than what was used in existing attacks on SHA-1 and MD5, and he could try to minimize the number of conditions associated with P' in the specified whitening steps. However, such an approach would likely not be very effective for the following reason: One special feature of the differential paths in existing attacks is that they are "front-loading" (with a lot of conditions in the first 20 steps) in order to minimize the number of conditions after step 20, which is directly related to the complexity of the attack. Hence, if the attacker selects P' that has fewer conditions in the first 20 steps, then it is very likely that P' would have more conditions later. This observation applies even more strongly when considering the improved attacks of [27] which extend message modification to additional steps.

7 Implementation Consideration

In this section, we consider practical implementation issues related to the message pre-processing proposal. For ease of discussion, we refer to the derived new hash function as SHApp, where “pp” stands for pre-processing[2]. We will consider issues related to programming implementation of SHApp as well as upper layer protocols that call SHApp as subroutines.

7.1 Programming Implementation

We propose two possible implementation options for SHApp. They vary only in terms of where pre-processing occurs in the code, and they are suitable for different applications.

Option 1: Pre-processing within SHA-1 Function For most existing implementation of SHA-1, the hash computation on a given input is generally carried out by three functional calls as described below.[3]

```
SHAInit(context)
SHAUpdate(context, input, inputLen)
SHAFinal(digest, context)
```

We can implement the new hash function SHApp with the same sequence of functional calls as follows:

```
SHAppInit(context)
  // same as SHAInit
SHAppUpdate(context, input, inputLen)
{
    newInput = SHAppPreProcess (input)
    newInputLen = Length (newInput)
    SHAUpdate(context, newInput, newInputLen)
}
SHAppFinal(digest, context)
  // same as SHAFinal
```

Note that `SHAppUpdate` has exactly the *same* i/o interface as the original `SHAUpdate` in existing implementation. The pre-processing step is done as a private function that is invisible to upper layer protocols using SHApp. Due to the simplicity of whitening and interleaving, only a small amount of code is needed for implementing the `SHAppPreProcess` function.

[2] A more accurate name would be SHA1pp, but we omit the “1” so that it can be pronounced as “shap.”

[3] The naming for the functions may vary slightly among implementations. For example, `SHAUpdate` may be called `SHAadd` etc. Despite this name variation, the functions accomplish essentially the same thing: the first one initializes the IV; the second one does proper padding and the main loop; the third one finalizes the computation and writes output.

Option 2: Pre-processing outside SHA-1 Function For some applications, implementation of SHA-1 may be hard-coded, and hence it can be difficult to make internal changes to the code as described in option 1. In this case, pre-processing can be done entirely prior to calling the function SHA-1 as below.

```
SHApp(message)
{
    newMessage = SHAppPreProcess (message)
    SHA-1(newMessage)
}
```

Note that the original implementation of SHA-1 is used as a "black box" without changing anything inside. Again, there is no impact on the interface.

Interoperability We remark that for both options, the result of the hash computation is the same for the same message. There is no interoperability issue between the two options. Hence implementers can simply choose the option that best suits their applications.

7.2 Protocols

From the discussions on programming implementation, we can see that SHApp have exactly the same input and output interface as the original SHA-1. Hence, replacing SHA-1 with SHApp in a protocol would not cause any upper layer changes other than replacing the *Algorithm Identifier*.

Newer digital signature schemes (e.g., RSA-PSS) have a "hierarchical" identifier, where the hash function is a parameter. For those schemes, the algorithm identifier for SHApp is sufficient.

For various older digital signature schemes, a new algorithm identifier is needed for both SHApp itself as well as the combination of SHApp with the specific signature scheme. The relevant standards organizations need to take care of the assignment for combinations of DSA, ECDSA, etc. For example, RSA Security can assign identifiers for SHApp and its combination with PKCS #1 v1.5. Depending on the standards, it may take little time or some amount of time for such assignments.

8 IV Message Dependence Approaches

In this section we describe a completely different approach does not involve a local expansion, but instead works by effectively ensuring that the initialization vector (IV) is message dependent.

8.1 Message Duplication

One way to cause the IV to be message dependent is to concatenate the message with itself before hashing. To simplify the explanation, we suggest first padding M so that it is a whole number of blocks. With this assumption, the pre-processing is simply $\phi(M) = M||M$, where $||$ denotes string concatenation. Let us examine the calculation halfway through, just after all the blocks of the first M have been processed. Notice that the full original message M is left to be processed, except that the intermediate IV chaining variable is a function of the message itself. This illustrates that an equivalent way to view this construction is as a regular hash of M where the starting IV chaining variable is a function of the message itself rather than constant.

8.2 Security Analysis

The IV message dependence approach increases security in a way completely different than the local expansion approach. Instead of affecting the blockwise compression function, they rely on the fact that the entire message must be processed twice within the framework of the Merkle-Damgård iterative chaining. Since each message bit is input to separate blocks, the previous attack strategies simply can not be applied. Instead, attacks on this variant would have to be of a completely different sort, and would not be able to focus on a single compression function, or on a few adjacent message blocks. This, or any other variant of the IV message dependent approach would also present an additional obstacle to automatically constructing collisions on long messages from single block collisions.

This approach is interesting because it is an extremely simple way of thwarting the known collision attacks for MD5 and SHA-1. However, a disadvantage with this approach is that the preprocessing function ϕ is not a local expansion, so it can not be effectively used with streaming data.

One might also consider alternate methods of achieving IV message dependence, for example by setting the initial starting IV value to be the first 160 bits of $H(M)$. However, this would not be not a "pure" preprocessing technique, and would require accessing the internals of the hash function itself, to set the IV value.

9 Analysis for MD5

SHA-1 was designed based on MD4 and MD5, and hence MD5 and SHA-1 are quite similar in terms of their structure and choices of mathematical operations. Consequently, the latest collision attacks on MD5 [25] and SHA-1 [24] also share some similarities. Therefore, most of the security analysis in preceding section also directly applies to MD5, including the general insight from coding theory, effects on message modification, and the IV-message dependency.

Here we point out some differences between the two hash functions and how they would affect the analysis. The main difference lies in the message expansion. For MD5, each message block expands 16×32 bits into 64×32 bits. The expansion function E operates by repeating and *re-ordering* the 16 message words 3 times. So the MD5 message expansion is much simpler than SHA-1, and hence offers less mixing.

The differential path P used in the latest attack on MD5 is also different from SHA-1, other than they are both "front-loading", and the MD5 analysis does not make use of an explicit disturbance vector. In the recent attack on MD5, the path was constructed by first finding a near collision that only involves the MSB in the second half and then deriving a more complicated collision path in the first half. For the chosen path, Δm_i is non-zero in steps 5, 12, and 15.

Message interleaving would result in 6 of the Δm_i to be non-zero, which would make the particular path P invalid. More importantly, interleaving would make message modification almost impossible. Note that there are over 200 conditions associated with P in the first 16 steps, and *all* these conditions need to be set true through message modification in order to reduce the complexity of the attack to about 2^{30+}. Therefore, message interleaving can significantly increase the complexity of existing attacks.

A similar argument can be carried out for message whitening, although a higher parameter of t would be required to rule out the availability of low Hamming weight Δm_i vectors. In this case, it seems more difficult to have a rigorous argument that the attacker cannot find a completely new path that would effectively target the particular whitening techniques.

10 Conclusions

In this paper we have considered several techniques to use SHA-1 and MD5 in a more collision resistant manner. The simplest approach which we have discussed in this paper is the message whitening approach. The word-wise message interleaving is also quite simple, and has very similar security properties. These approaches are both easy to implement, support streaming message digesting, and are amenable to analysis with respect to the known differential attacks. The IV message dependent approaches are appealing due to their immunity to single-block collision attack approaches, but have the drawback that they are not convenient for message streaming.

For practical applications wishing to improve SHA-1 use, we suggest the use of message whitening pre-processing with parameter $t \geq 4$, so that 12 words of the message are expanded into 16. This results in a performance slowdown of 25 percent. An even more secure alternative would be the message interleaving, although it results in a slowdown of 50 percent. For MD5, our recommendation is to use the message interleaving approach, or in case the application does deal with small data items (such as certificates), the IV message dependence approach.

Our solutions can be viewed as a general purpose, safer, collision resistant way of using MD5, and SHA-1. Due to their simplicity, we contend that such an approach can be appealing for practitioners who wish to increase security in the short term, without changing the underlying hash function at all.

Relationship to Hash Function Design: The solution in this paper is not intended to be a complete replacement for an appropriate, timely hash function update nor for improved hash function design. On the other hand, our proposal has something in common with proposals for enhancing the security of SHA-like hash designs such as [12] in that we also focus on the code of expanded message words.

Future Improvements on Collision Attacks: With respect to the attacks of [24, 27], both whitenening with parameter $t \geq 4$ and message interleaving techniques still yield a derived hash function for which collisions can not be found with effort below 2^{80}. Although it is impossible to predict the improvements in collision attacks we make a few comments on the robustness of our techniques. In general, the message pre-processing we propose makes will apply to other attacks of the same genre, because: (1) message modification in general is much harder, (2) multi-step message modification techniques are almost impossible, and (3) the constraints on Δm are targeted at preventing effective "front-loading" of differential paths.

11 Acknowledgments

The authors would like to thank Scott Contini, Russ Housley, Burt Kaliski, Jim Randall, Ron Rivest, Moti Yung, and the anonymous reviewers for helpful comments. Special thank to Paul Hoffman for discussions on implementation issues.

References

1. E. Biham and R. Chen. *Near Collisions of SHA-0.* In Advances in Cryptology – Crypto'04 , Springer-Verlag, August 2004.
2. E. Biham and R. Chen. *New Results on SHA-0 and SHA-1.* In Crypto'04 Rump Session, August 2004.
3. E. Biham, R. Chen, A. Joux, P. Carribault, W. Jalby and C. Lemuet. *Collisions in SHA-0 and Reduced SHA-1.* In Advances in Cryptology – Eurocrypt'05 , Springer-Verlag, May 2005.
4. J. Coron, Y. Dodis, C. Malinaud, and P Puniya *Merkle-Damgrd Revisited : How to Construct a Hash Function* In Advances in Cryptology – Crypto'05, Springer-Verlag, 2005.
5. F. Chabaud and A. Joux. *Differential Collisions in SHA-0.* In Advances in Cryptology – Crypto'98, Springer-Verlag, August 1998.
6. I. Damgård. *A Design Principle for Hash Functions*, In Advances in Cryptology – Crypto'89, Springer-Verlag, 1990.
7. M. Daum and S. Lucks. *The Story of Alice and her Boss* In Rump session of Eurocrypt'05. http://www.cits.rub.de/MD5Collisions/.

8. H. Handschuh and H. Gilbert *Security Analysis of SHA-256 and Sisters*. Proceedings of the Workshop on Selected Areas in Cryptography - SAC'03, Springer-Verlag, 2003.
9. S. Halevi and H. Krawczyk *Strengthening Digital Signatures via Randomized Hashing*, Internet-Draft, May 12, 2005. http://www.ietf.org/internet-drafts/draft-irtf-cfrg-rhash-00.txt.
10. P. Hawkes and M. Paddon and G. Rose. *On Corrective Patterns for the SHA-2 Family*. http://eprint.iacr.org/2004/207
11. A. Joux. *Collisions for SHA-0*. In Rump session of Crypto'04, August 2004.
12. C. Jutla and A. Patthak *A Simple and Provably Good Code for SHA Message Expansion*, IACR Eprint archive, Report 2005/247, http://eprint.iacr.org/2005/247.
13. V. Klima: *Finding MD5 Collisions on a Notebook PC Using Multi-message Modifications*, IACR Eprint archive, Report 2005/102, http://eprint.iacr.org/2005/102.
14. A. Lenstra and X. Wang and B. de Weger. *Colliding X.509 Certificates*, IACR Eprint archive, Report 2005/067. http://eprint.iacr.org/.
15. R. Merkle. *One Way hash Functions and DES*, In Advances in Cryptology – Crypto'89, Springer-Verlag, 1990.
16. K. Matusiewicz and J. Pieprzyk. *Finding Good Differential Patterns for Attacks on SHA-1*. IACR Eprint archive, December 2004.
17. Y. Naito and Y. Sasaki and N. Kunihiro and K. Ohta. *Improved Collision Attack on MD4* IACR Eprint archive, Report 2005/151.
18. V. Rijmen and E. Oswald. *Update on SHA-1*. In Topics in Cryptology – CT-RSA 2005, Springer-Verlag, 2005.
19. NIST. *Secure hash standard*. Federal Information Processing Standard, FIPS 180, May 1993.
20. NIST. *Secure hash standard*. Federal Information Processing Standard, FIPS 180-1, April 1995.
21. NIST. *Secure hash standard*. Federal Information Processing Standard, FIPS 180-2, August 2002.
22. X. Wang, F. Guo, X. Lai, and H. Yu. *Collisions for Hash Functions MD4, MD5, HAVAL-128 and RIPEMD*. In Rump session of Crypto'04 and IACR Eprint archive, August 2004.
23. X. Wang, X. Lai, F. Guo, H. Chen, X. Yu. *Cryptanalysis for Hash Functions MD4 and RIPEMD*. In Advances in Cryptology – Eurocrypt'05, Springer-Verlag, May 2005.
24. X. Wang and Y.L. Yin and H. Yu. *Finding Collisions in the full SHA-1*. In Advances in Cryptology – Crypto'05, Springer-Verlag, 2005.
25. X. Wang and H. Yu. *How to Break MD5 and Other Hash Functions*. In Advances in Cryptology – Eurocrypt'05, Springer-Verlag, May 2005.
26. X. Wang and H. Yu and Y.L. Yin. *Efficient Collision Search Attacks on SHA-0*. In Advances in Cryptology – Crypto'05, Springer-Verlag, 2005.
27. X. Wang, A. Yao, and F. Yao, *New Collision search for SHA-1*, Rump Session Crypto'05.