

Constructing Secure Hash Functions by Enhancing Merkle-Damgård Construction (Full version) *

Praveen Gauravaram¹, William Millan¹, Ed Dawson¹, Matt Henricksen¹, Juanma González Nieto¹ and Kapali Viswanathan²

¹ Information Security Institute (ISI)

Queensland University of Technology (QUT)

2 George Street, GPO Box 2434, Brisbane QLD 4001, Australia.

p.gauravaram@isi.qut.edu.au, {b.millan, e.dawson}@qut.edu.au

² Technology Development Department, ABB Corporate Research Centre
ABB Global Services Limited, 49, Race Course Road, Bangalore - 560 001, India.

kapaleeswaran.v@in.abb.com

Abstract. Recently multi-block collision attacks (MBCA) were performed on the Merkle-Damgård (MD)-structure based hash functions MD5, SHA-0 and SHA-1. In this paper, we first introduce a new generic cryptographic hash construction family called **3CG** devised by enhancing the (MD) construction. We show that one can construct simple to complex modifications to the MD construction by defining instances for the **3CG** construction.

In this paper, we introduce a simple instance for **3CG** called **3C** devised by enhancing the MD construction. We show that the **3C** construction is at least as secure as the MD construction against single-block and multi-block collision attacks. This is the first result of this kind showing a generic construction which is at least as resistant as MD against MBCA. To further improve the resistance of the design against MBCA, we propose the **3C+** design as an enhancement of **3C**. Both these constructions are very simple adjustments to the MD construction and are immune to the straight forward extension attacks that apply to the MD hash function. We also show that these constructions resist some known generic attacks that work on the MD construction. We also propose hybrid constructions for **3C** by combining it with wide-pipe (**3CWP**) and double-pipe constructions (**3CDP**). In addition, we show that several secure variants can be obtained for **3C** and **3C+** introducing generic constructions called **3CG** and **3CG+**. We show constructing variants for **3CG** and **3CG+** constructions. Finally, we compare the security and efficiency features of **3C** with other closely related proposals to the MD based hash functions.

Keywords: Merkle-Damgård construction, multi-block collision attacks, generic attacks, 3C, 3CWP, 3CDP, 3CF, 3CCW+.

1 Introduction

In 1989, Damgård [7] and Merkle [31] independently proposed a similar iterative structure to construct a collision resistant cryptographic hash function $H : \{0, 1\}^* \rightarrow \{0, 1\}^t$ using a fixed length input collision resistant compression function $f : \{0, 1\}^b \times \{0, 1\}^t \rightarrow \{0, 1\}^t$. Since then, this iterated design has been called Merkle-Damgård (MD) construction which influenced the designs of popular dedicated hash functions such as MD5, SHA-0 and SHA-1. The design motivation of the MD construction is that if the compression function f is collision resistant then so is the resultant iterated hash function H .

It is known that, a compression function f secure against the fixed initial value (IV) collisions is necessary but not sufficient to generate a secure hash function H [30, p.373]. The latest multi-block collision attacks (MBCA) on the hash functions MD5, SHA-0 and SHA-1 [4, 37–39] prove this insufficiency. These attacks clearly show that these iterated hash functions do *not properly preserve* the collision resistance property

* The original draft version of **3C** as a PRF, MAC and a hash function design is available at [16]. This is full version of the hash function version of the shortened paper published at ACISP 2006.

of their respective compression functions with the fixed IV. The MBCA on hash functions leave open the questions: is it possible to design collision resistant hash functions relying on the collision resistance of the compression function with fixed IV?, Is it possible to design a simple and efficient structures that offer more resistance to MBCA than the **MD** structure?

In this paper, we attempt to answer these questions. Our motivation is to show that while consecutive iterations of the compression function is necessary for the implementation efficiency of a hash function, **the way** the compression function is iterated or **used** is quite important for the security of the hash function. In this paper, we propose a new mode of operation for the **MD** construction called **3C**. The **3C** hash function processes the intermediate chaining values of the **MD** construction by maintaining a second internal chaining variable. The **3C** construction is the simplest modification of the **MD** construction that one can obtain to improve its security against MBCA.

We show that the **3C** construction is at least as secure as **MD** construction against collision attacks. Against the known multi-block collision attack techniques, **3C** is more secure than **MD**. That is, if there exists an adversary that finds a multi-block collision on **3C** then that adversary would have also found a multi-block collision on the **MD** hash function. In addition, we show that if there exists an adversary that can perform an MBCA on a t -bit **MD** hash function based on a given compression function then the security of **3C** against MBCA instantiated with the same compression function could be as much as 2^t times the security of **MD** against MBCA, depending on the subtle properties of the compression function. We conjecture that this security multiplier of **3C** against MBCA is close to 2^t for any compression function which is secure against single-block collision attacks. We note that a multiplier of at least $2^{t/2}$ is sufficient to provide immunity to MBCA. Next, extra memory is added to the **3C** construction and we call this variant **3C+** and analyse the difficulty in implementing an MBCA on it compared to **3C**. Analysis for **3C** against known generic attacks [9, 18, 22] is given which applies to **3C+** as well. We found that while Joux’s generic attacks [18] work on **3C**, the known generic second preimage attacks [9, 22] found on the **MD** hash function do not work. Also, these constructions prevent the straight forward length extension attacks and 2^{nd} -collision attacks that work on the **MD** hash function.

Finally, we provide hybrid hash function constructions based on **3C** by combining other proposed enhancements to the **MD** construction in the literature. These hybrid constructions provide more resistance against both generic attacks and specific attacks such as differential multi-block collision attacks.

In section 2, we describe **MD** hashing and collision attacks on it. In section 3, new observations on the MBCA are discussed. In section 4, **3C** is introduced and its analysis against MBCA is covered in section 5. In section 6, analysis of **3C** against generic attacks is given and **3C** is compared with some other related hash function proposals in section 7. In section 8, hybrid constructions based on **3C** are introduced with the motivation and **3C+** is introduced and is analysed against MBCA in section 9. The paper is concluded in section 9.

2 MD hashing and collision attacks

A collision resistant cryptographic hash function H following **MD** structure is a function that hashes a message $M \in \{0, 1\}^*$ to outputs of fixed length $\{0, 1\}^t$. The specification of H includes the description of the compression function f , initial state value (IV) and a padding procedure [30, 33]. Every hash function fixes the IV (fixed IV) with an upper bound on the size $|M|$ of the input M . The message M is split into blocks M_1, \dots, M_{L-1} of equal length b where a block M_L containing the length $|M|$ (**MD** strengthening) [24] is added. Each block M_i is iterated using a fixed length input compression function f computing $H_i = f(H_{i-1}, M_i)$ where $i = 1$ to L and finally producing the output $H_{IV}(M) = H_L$ as shown in Fig 1.

Collision attacks on the compression functions:

A hash function H is said to be collision resistant if it is hard to find any two distinct inputs M and N such that $H(M) = H(N)$. For a formal definition see [34]. A hash function H is said to be near-collision resistant if it is hard to find any two distinct inputs M and N such that $H(M) \oplus H(N) = \Delta$ has some small weight. Based on the IV used in finding collisions, collision attacks on the compression functions are classified as follows [30, p.372]:

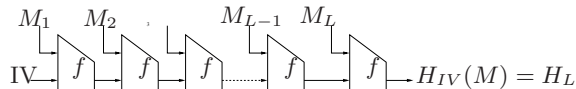


Fig. 1. The Merkle-Damgård (MD) construction

1. Collision attack: collisions using a fixed IV for two distinct messages (e.g. [36]). We call them *Type 1* collisions.
2. Semi-free-start collision attack: collisions using the same random (or arbitrary) IV for two distinct message inputs (e.g. [11]). We call them *Type 2* collisions.
3. Pseudo-collision attack: free-start collision attack using two different IVs for two distinct message inputs (e.g. [10]). We call them *Type 3* collisions.

Multi-block collision attacks on hash functions:

A multi-block collision attack (MBCA) technique on an iterated hash function finds two colliding messages each of at least two blocks in length. The recent collision attacks on MD5 [39], SHA-0 [4,37] and SHA-1 [38] are multi-block collision attacks where collisions were found by processing more than one message block. Since, by far, most of the possible messages are more than a single block and collisions are distributed randomly, it is fair to say that most collisions that could exist are in fact multi-block collisions. Hence, any result improving the resistance of hash functions against MBCA is very significant.

3 New Observations on multi-block collision attacks

This section aims at developing the understanding of the multi-block collision attacks on hash functions by linking several parts of the literature. We observed that multi-block collision attacks on hash functions can further be classified into three categories based on the manner in which the compression functions are attacked. This leads to choosing particular message block formats that result in an MBCA. For example, 2-block collision attacks can be classified into three types as shown in Table 1 based on the message formats chosen.

Table 1. Classification for 2-block collision attacks

MBCA Type	Message formats
MBCA-1	(M_1, M_2) and (M_1, N_2)
MBCA-2	(M_1, M_2) and (N_1, M_2)
MBCA-3	(M_1, M_2) and (N_1, N_2)

While the 2-block collision attacks on MD5 [39] and SHA-1 [38] belong to MBCA-3 category, the 2-block collision attack on SHA-0 [37] belongs to an MBCA-1 category¹. In an MBCA-3, near-collisions found after processing the first message blocks were converted to full collisions as was demonstrated on MD5 and SHA-1. The *Type 1* collisions were (reportedly) hard to find for the compression functions of these hash algorithms based on their initial states. For example, the attacks on MD5 and SHA-1 use near-collisions obtained after processing the first distinct message blocks (M_1, N_1) as a tool to find collisions for the second distinct message blocks (M_2, N_2) . This technique can be generalized to more than two blocks as the 4-block collision attack on SHA-0 [4]. Similarly, 2-block MBCA-1 and MBCA-2 attack techniques can be generalized to more than two blocks. For example, in an MBCA-1 technique [37], a few initial message blocks to be processed can be

¹ The first full 4-block collision attack on SHA-0 [4] also belongs to an MBCA-3 category except that differences in the message blocks span over more than two blocks.

chosen to be the same to satisfy certain conditions required in the attack followed by the processing of two different message blocks that give a collision.

We note that in a 2-block MBCA, collisions found on the second blocks are basically a *special case* of *Type-3* collisions for the compression function as these collisions require processing of two equal (MBCA-1) or different blocks (MBCA-2 and MBCA-3) using the fixed IV of the hash function. That is, a 2-block MBCA-3 on the **MD** hash function is a combination of a near-collision and a *special Type-3* on the compression function. The collision on the second compression function is a *special Type-3* as it requires a particular nearly collided value obtained based on certain conditions essential for the attack as inputs for the second block messages. In addition, near-collisions do not need to begin after processing message blocks based on the fixed IV of the hash function. They can also be due to an arbitrary chaining value when the attacker chooses the same blocks initially and starts an MBCA-3 after processing those same initial blocks. Hence these collisions, whether they start from the fixed IV or arbitrary chaining values are clearly a chain of *special Type 3* collisions.

From these observations on MBCA, it is clear that the designers of MD5, SHA-0 and SHA-1 *have not considered security of the compression functions of these hash functions against special Type-3 collisions in their design criteria*. Preneel pointed out more than a decade back [33] that most hash functions are not designed to meet this criteria. Note that SHA-1 did not exist then. Even Damgård’s [7] proof implicitly notes the necessity of *special Type-3* collision resistance for the compression functions. In addition, to attain *Type-3* collisions, the two IVs do not have to be significantly different as suggested in [30, p.372]. For example, the two IVs in the *Type-3* collision attack on the compression function of MD5 [10]² differ in *only* 6 bits. From the known attacks on hash functions, we derived Table 2 assuming that if the compression function is not *Type-1* collision resistant then it is neither *Type-2* nor *Type-3* collision resistant. The sign “-” in the Table 2 indicates does not apply.

Table 2. Resistances of some compression functions

Compression function	<i>Type-1</i>	<i>Type-2</i>	<i>Type-3</i>	<i>Special Type-3</i>
MD4	NO [36]	NO	NO	-
MD5	YES	NO [11]	NO [10]	NO [39]
SHA-0	YES	NO [39]	YES	NO [4]
SHA-1	YES	YES	YES	NO [38]
RIPEMD	NO [36]	NO	NO	-
HAVAL-128	NO [36]	NO	NO	-

4 The 3C construction: An Enhanced MD construction

The **3C** construction is shown in Fig. 2 and 3. This structure has an *accumulator* XOR function iterated in the *accumulation chain* (whose chaining value is denoted by u_i in Fig. 3) and a compression function f (f , for example, is the compression function of MD5 or SHA-1) iterated in the *cascade chain* (whose chaining value is denoted by w_i in Fig. 3) exactly as in the **MD** construction. Clearly, **3C** is a very simple and efficient modification to the **MD** construction. One economic benefit of our proposal is that any software currently implementing an **MD**-style hash function can be very simply altered to adopt the **3C** structure, without altering the compression function.

3C hashing process: For $i = 1$ to L , let w_i and u_i be the chaining values in the *cascade chain* and *accumulation chain* respectively. Then, as in the **MD** hash, for $i = 1$ to L , $w_i = f(w_{i-1}, M_i)$ where $w_0 = IV$ and $u_1 = w_1$. In the *accumulation chain*, for $i = 2$ to L , $u_i = u_{i-1} \oplus w_i$. The result u_L in the *accumulation*

² To be precise, the way this attack was defined contradicts the definition of pseudo collision given in [30] as it takes two different IVs and the same message block to produce a collision.

chain is denoted with Z . An extra compression function f , denoted by g , is added at the end and the hash result of **3C** is $g(\overline{Z}, w_L)$. To process data of one block or less than a block, the compression function is executed three times; first to process the data block, next to process the padded block (**MD** strengthening) and finally the block \overline{Z} formed in the *accumulation chain* as shown in Fig 2. If the size of data is less than block size b of f then zeros are appended to the data to make a b -bit data block. The block \overline{Z} is the padded result of Z obtained by appending a bit 1, some 0's and the binary encoded format of length of the message M to make it a b -bit block for the function g .

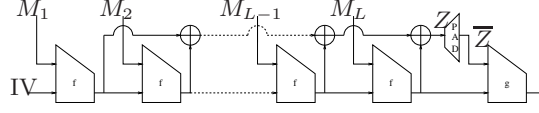


Fig. 2. The **3C**-hash function

5 Analysis of **3C** against collision attacks

In this section, we investigate the security of **3C** against single-block and multi-block collision attacks. We conclude that the security of **3C** against single-block collision attacks is upper bounded by the collision security of the compression function and its security against MBCA is not less than that on **MD**. Fig 3 is used to explain the analysis.

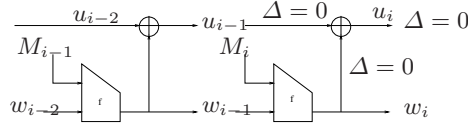


Fig. 3. Creating an internal collision for **3C**

Consider a **3C** hash function H . Consider two distinct messages $M \neq N$ of same length L (including padding) such that $H(M) = H(N)$ is the result of a collision on **3C**. The messages M and N are expanded to sequences $(M_1, \dots, M_L) \neq (N_1, \dots, N_L)$ where the last data blocks are the padded blocks containing the length L of the messages. We denote by (H_i^M, H_i^N) and (u_i, v_i) (for $i = 1$ to L), the internal hash values obtained on the *cascade chain* and *accumulation chain* while computing $H(M)$ and $H(N)$ respectively. We denote (u_L, v_L) by (Z_M, Z_N) and $\overline{Z}_M = \text{PAD}(Z_M)$, $\overline{Z}_N = \text{PAD}(Z_N)$. All possible types of collisions on H are given in Definition 1.

Definition1:

Every collision on H takes one of the following forms:

1. Terminal/Final collisions on H : They involve one of the following cases:

- $H_L^M \neq H_L^N$ and $\overline{Z}_M \neq \overline{Z}_N$ with $g(H_L^M, \overline{Z}_M) = g(H_L^N, \overline{Z}_N)$
- $H_L^M = H_L^N$ and $\overline{Z}_M \neq \overline{Z}_N$ with $g(H_L^M, \overline{Z}_M) = g(H_L^N, \overline{Z}_N)$
- $H_L^M \neq H_L^N$ and $\overline{Z}_M = \overline{Z}_N$ with $g(H_L^M, \overline{Z}_M) = g(H_L^N, \overline{Z}_N)$

2. Internal collisions on H : $H_L^M = H_L^N$ and $\overline{Z}_M = \overline{Z}_N$ implies $g(H_L^M, \overline{Z}_M) = g(H_L^N, \overline{Z}_N)$. □

Definition2:

A compression function $f : \{0, 1\}^b \rightarrow \{0, 1\}^t$ is Type-1 (resp. Type-2, Type-3) collision resistant if the best possible collision attack on it using fixed IV (resp. arbitrary IV, different IVs) is the birthday attack which takes about $2^{t/2}$ operations of f . For sufficiently large t , it is computationally infeasible to perform this attack.

Lemma 1. *Against single block collision attacks, the security of $\mathbf{3C}$ is exactly equal to the security of \mathbf{MD} when both the constructions are instantiated with the same f .*

Proof: By inspection of $\mathbf{3C}$ structure in Fig 3, it is clear that it contains \mathbf{MD} construction in it. Hence, an adversary that is able to find a single block collision for the f -function is able to construct a collision for the hash function with minimal additional effort. It follows that the collision security of $\mathbf{3C}$ is upper bounded by the collision security of the f -function.

Apart from single block collision attacks, the only other approach to find collisions for $\mathbf{3C}$ is to use multi-block messages. This invites the opportunity to use messages with different lengths. For two messages with the same length, an internal collision for $\mathbf{3C}$ gives an actual collision for $\mathbf{3C}$. However, for two messages of different lengths, in general, this is not the case due to the different padding strings used as a virtual message block in the second last iteration of the compression function. Thus the security analysis of $\mathbf{3C}$ can be restricted to considering internal collisions generated by pairs of messages with the same length. We now examine the nature of internal collisions for $\mathbf{3C}$.

Lemma 2. *To get an internal collision on $\mathbf{3C}$ at iteration i , it is required that a collision in the accumulation chain exists at iteration $i - 1$.*

Proof: An internal collision in $\mathbf{3C}$ at iteration i is a simultaneous collision in the *accumulation chain* and *cascade chain* at iteration i . For messages M and N to collide on the *cascade chain* at iteration i , the condition that $H_i^M \oplus H_i^N = 0$ must be satisfied. Now for an internal collision on $\mathbf{3C}$, the condition that $(H_i^M \oplus H_i^N) \oplus (u_{i-1} \oplus v_{i-1}) = 0$ must be satisfied. This condition will occur only when $u_{i-1} \oplus v_{i-1} = 0$, which is a collision in the *accumulation chain* at iteration $i - 1$.

Remark: A collision in the accumulation chain at iteration $i - 1$ is achieved by creating a sequence of \mathbf{MD} chain differences where the chaining difference in the \mathbf{MD} chain at iteration $i - 1$ is the XOR sum of all the previous differences in the \mathbf{MD} chain until the iteration $i - 2$.

Lemma 3. *Assuming the existence of a collision in the accumulation chain at iteration $i - 1$, it requires a single-block special Type-3 collision attack on f to create an internal collision in $\mathbf{3C}$ at iteration i .*

Proof: By the inspection of the $\mathbf{3C}$ structure in Fig 3, a special Type-3 collision attack must be performed on the f -function at iteration i . It is a *special Type-3* collision attack as the attacker must use the internal chaining values of the *cascade chain* at iteration $i - 1$ that created a collision in the *accumulation chain* as inputs to get a collision at iteration i . This is equivalent to performing a single-block *special Type-3* collision attack on f at iteration i .

Now we can consider the above process in two ways: as two separate single block collision attacks, or as a multi-block collision attack on the \mathbf{MD} -chain with an extra t -bit requirement. We ignore the first option as the single block collision security of the f -function is already an upper bound for the collision security of $\mathbf{3C}$ from Lemma 1. The second case can be achieved in either of the following two ways:

1. Assume the existence of an MBCA on the \mathbf{MD} -chain when it is instantiated with some given compression function. Then the MBCA on the \mathbf{MD} -chain has to be repeated until the internal chaining differences on the \mathbf{MD} -chain happen to produce the required collision on both the accumulation and cascade chains. This option requires repeating the attack at most 2^t times under an assumption that the internal chaining differences are uniformly distributed.
2. Devise an entirely new MBCA for the $\mathbf{3C}$ when instantiated with some given compression function satisfying some conditions on the differences.

Now, the above two cases result in the following theorems:

Theorem 1 *If there is an MBCA on the \mathbf{MD} construction instantiated with a given f then the security of $\mathbf{3C}$ instantiated with the same f against an MBCA is at most 2^t times the security of \mathbf{MD} against MBCA.*

Proof: To obtain a collision in the *accumulation chain* required by Lemma 2, an MBCA on the **MD** chain must be repeated, where each attempt succeeds with probability 2^{-t} . That is, the security of **3C** against MBCA is some multiple ($[1, 2^t]$) of the security against MBCA for the **MD**.

The difficulty of providing a tight quantitative analysis for **3C** against MBCA prevents a more precise formal proof for the practical collision security of **3C** at this stage leading to the following conjecture.

Conjecture: From the above analysis, we conjecture that the improvement in the security of **3C** against MBCA is close to 2^t over the security of **MD** against MBCA.

Theorem 2 *The security of 3C against an MBCA is not less than the security of MD against MBCA.*

Proof: Every internal collision for **3C** contains within it a collision for **MD**. There exist collisions for **MD** that are not internal collisions for **3C**. Thus the security of **3C** against MBCA is lower bounded by the security of **MD** against MBCA.

Remarks:

1. While at least two blocks must be processed to find a multi-block collision on **MD**, at least three blocks must be processed to create a multi-block collision on **3C**. For example, the difference pattern $(0, \Delta, \Delta, 0)$ which creates a collision on **MD** based on a given f , will also create a collision on **3C** based on the same f . But note that its reduced pattern $(0, \Delta)$ would create a collision for **MD** but not for **3C**. Hence in this case, the complexity of performing an MBCA on **3C** based on some f might be more than the complexity of performing an MBCA on an **MD** hash based on the same f .
2. As an another example, if a special difference pattern for the chaining values like $(0, \Delta_1, \Delta_2, \Delta_1 + \Delta_2, 0)$ is required to create a collision on the **MD** based on some f , then this pattern would also create a collision for **3C** based on the same f [23]. In this case, the complexities of performing MBCA on both hash functions is the same.
3. We note that, the MBCA-1 Type collisions as found on SHA-0 work on **3C** instantiated with the compression function of a hash function susceptible to *Type-3* collision attacks thus justifying the above theorem.

In Section 8, we propose a construction called **3C+** with similar properties to **3C** as an improvement of **3C** for more protection against MBCA.

6 Security analysis of 3C against known generic attacks:

6.1 Analysis against Joux’s attacks:

Joux [18] described a generic multicollision attack on the **MD** hash where constructing 2^d -collisions costs d times as much as building ordinary 2-collisions. This attack can be used as a tool to find multi (2^{nd}) preimages very effectively on the **MD** hash. We note that these attacks work on **3C** as effectively as they are on the **MD** hash. Following [27], our adversaries are probabilistic algorithms and we focus on the expected running time. Running time is described asymptotically. We use the symbol O for the “expected running time is asymptotically at most”.

In a multicollision attack on **3C**, the attacker finds collisions on every function f in the *cascade chain* (for example using the birthday attack) that would result in a collision at the subsequent point of the XOR operation in the *accumulation chain*. If the function f in the *cascade chain* of **3C** is modeled as a random oracle, as an upper bound, the total complexity to find 2^d -collisions on **3C** is $O(d * 2^{t/2})$.

We note that the attack technique used to find D -way (2^{nd}) preimages on the **MD** hash for a given hash value works on **3C** as well. For example on **3C**, the attacker first finds D -collisions on d -block messages M^1, \dots, M^{2^d} with $H_d = H(M^1) = \dots = H(M^{2^d})$ with a complexity of $O(d * 2^{t/2})$. Then she finds the block M_{d+1} such that the execution of the last two compression functions would result in the given digest Y . The later task takes time $O(2^t)$ as the last two compression functions are treated as a single component. Hence the total cost of finding D -preimages for **3C** is $O(d * 2^{t/2} + 2^t)$. To find D - 2^{nd} preimages for a given message M , the attacker first computes the hash $H(M)$ of the message M and then finds D -preimages as explained above that all collide to $H(M)$.

6.2 Analysis against second-preimage attacks:

Dean [9] has demonstrated that for hash functions with fixed point compression functions, it would cost less than 2^t effort to find second preimages. Kelsey and Schneier [22] have expanded this result using Joux multi-collision finding technique to find second preimages for hash functions based on any compression function for an effort less than 2^t . Both these attacks use the notion of *expandable messages*- patterns of messages of different lengths that all process to internal hash values without considering **MD** strengthening. Following [22], an (a, b) - expandable message will take on any length between a and b message blocks.

For a compression function $H_i = f(H_{i-1}, M_i)$, a fixed point is a pair (H_{i-1}, M_i) such that $H_{i-1} = f(H_{i-1}, M_i)$. The compression functions of many hash functions such as MD5 and SHA-1 are Davies-Meyer designs with a block cipher operating in a feed-forward mode. For these compression functions, there exists one and only one fixed point for every message block. For a t -bit hash function with a maximum of 2^d blocks in its messages, using fixed points it costs about $2^{t/2+1}$ compression function computations to find $(1, 2^d)$ -expandable message [22]. In the **3C** design, since the chaining state is twice as large as the hash value, a fixed point is defined for both the chains and this is obtained for any message block M_i , *only when* $f(0, M_i) = 0$ and this occurs with a probability of 2^{-t} . Hence, having fixed points for the compression functions will not assist in finding second preimages for less than 2^t work on the **3C** design.

It was demonstrated in [22] that finding a $(d, d + 2^d - 1)$ expandable message for any compression function with t -bit state takes only $d \times 2^{t/2+1}$ effort. The procedure involves first finding colliding pair of messages, one of one block and the other of $2^{d-1} + 1$ blocks starting from the initial state of the hash function. Then using the collided state as the starting state, collision pair of length either 1 or $2^{d-2} + 1$ is found and this process is continued until a collision pair of length 1 or 2 is reached. It was shown in [22] that applying this generic expandable message finding algorithm to find the second preimage for a message of $2^d + d + 1$ -block length message costs $d \times 2^{t/2+1} + 2^{n-d+1}$ compression function computations. When this attack technique is applied on **3C**, a collision for both the chains is required and this costs an effort of 2^t at every stage as the size of the internal state is twice that of the hash size³.

6.3 Analysis against length extension attacks:

In the cryptographic hash function literature on the **MD** construction, authors have defined straight-forward length extension attacks (or extension attacks) in two ways with the same nomenclature. In this section, we categorise extension attacks into two types following some examples in the literature. Finally, we show that the **3C** construction prevents both types of straight-forward length extension attacks.

We treat the extension attack on the **MD** hash function H by Ferguson and Schneier [14] as follows: Consider a message M split into blocks M_1, \dots, M_{L-1} hashing to a value $H(M)$. We make two broad assumptions on the format of this message M as follows:

1. The message M completely specifies the actual data, meaning all blocks contain the data that gets hashed and there is no length encoded padding. If the last block is incomplete, 0's would have been appended to make it a data block. The hash result $H(M)$ of such an M is the intermediate or internal hash value.
2. The message M does not contain the actual data to be hashed, meaning the last block (in some cases even last before block depending on $|M|$) contains padding bits which contain the binary encoded representation of the length of the message M . The hash result $H(M)$ of such an M is the final hash value or hash value.

It seems that [14] uses the first assumption. Now the attacker is given $H(M)$ and $|M|$ but not M itself. That is, the attacker is given intermediate or internal hash value $H(M)$ which is not computed using the complete hash algorithm which in general includes the length padding. With this information, the attacker uses $H(M)$ to process at least one new data block. Let there is a new data block represented as M_L . Since most hash functions use **MD** strengthening, it is straight forward for the attacker to encode the length of

³ If different parts of the internal state of **3C** are attacked separately, **3C** might not resist the second preimage attack [21]

the message in the additional last block M_{L+1} and compute new hash value for the message $M||M_L$ without knowing the message M . We call this attack as *Type-1* extension attack. This *Type-1* extension attack is also discussed in [26, 27].

The second type of extension attack is based on the second assumption where the attacker is given the final hash value $H(M)$ and $|M|$ but not M . As explained in the *Type-1* extension attack, the attacker uses $H(M)$ and the new message block, say M_L , to compute the new hash value for the message $M||M_L$. An additional block M_{L+1} containing the length encoded format of total length of the message $M||M_L$ is also processed. So, in total, padding is performed twice for the hash function. We call this attack as *Type-2* extension attack. This *Type-2* extension attack is discussed in [12, 27].

We note that if at all there are applications where one can employ these attacks, *Type-1* extension attacks will be of more practical importance than *Type-2* extension attacks. The reason being in *Type-2* extension attack, the continuity of meaningful message is lost due to the first padding and the sequence of meaningful data is maintained in the second message as padding is performed only at the end. We are unsure about the applications where one could use these attack attacks. Nevertheless, the property of exploiting the iterative structure of **MD** construction used in the above extension attacks, can be used to break the naive authentication scheme based on **MD** hash functions [14, p.90] where an attacker computes authentication tags of messages without the knowledge of the secret key.

We note that both forms of extension attack do not work on **3C**. In the application of *Type-1* extension attack on **3C**, the attacker is given the hash value of M and $|M|$ with no padding on M . Assume that the attacker is given the hash value after the iteration i . To perform *Type-1* extension attack, the attacker must know the data on the *accumulation chain* after iteration $i - 1$. The latter requires the knowledge of all the *cascade* chaining state values prior to the iteration i ⁴. Similarly, the application of the extra compression function at the end prevents *Type-2* extension attack on **3C**.

6.4 Analysis against 2nd-collision attacks:

One can use the principles of extension attacks on the **MD** hash function H in performing 2nd-collision attacks. If $H(M) = H(N)$ for any two different messages M and N , then $H(M||X) = H(N||X)$ [27]. That is, given a free collision for a hash function, the attacker finds a second collision by exploiting the **MD** iterative structure. To obtain meaningful 2nd-collisions [8, 25], the first collision should be an internal collision with no padding. Hence, if the attacker is provided with a simultaneous internal collision on both the chains of **3C** then the attacker can perform 2nd-collision attack on **3C**. As pointed out above, it does not make sense in providing the *accumulation chain* values to the attacker.

The other case is where the attacker is given final or terminal collisions $H(M) = H(N)$ for $M \neq N$ (see Definition 1 for terminal collisions on **3C**). Since the given free collision is a terminal collision, the attacker must first get a simultaneous collision on both the chains of **3C** before she performs 2nd-collision attack by appending the same or different message blocks to the messages that result in a simultaneous collision. Even to find a simultaneous collision, the attacker has to find additional message blocks that would give a simultaneous collision on both the chains. This is equivalent to performing an MBCA on the **3C** hash function.

If the compression function f (also g) in **3C** is modeled as a random oracle, finding 2nd-collisions for **3C** for a given final collision would take time $\Omega(2^{t/2})$. This case is analysed by considering all the possible cases of final collisions on **3C** given in Definition 1 as follows:

- $H_L^M \neq H_L^N$ and $Z_M \neq Z_N$ with $g(H_L^M, \bar{Z}_M) = g(H_L^N, \bar{Z}_N)$. In this case, for f modeled as a random oracle, the attacker has to get a simultaneous collision on both the chains to get 2nd collisions. This would take time about 2^t as the attacker has to get a collision on the *cascade chain* which costs $\Omega(2^{t/2})$ provided she gets a collision on the *accumulation chain* which also costs $\Omega(2^{t/2})$.

⁴ If the attacker is provided with the values of both the chains after iteration i then she can perform extension attack on **3C**. But it does not make sense in providing the attacker the value of the *accumulation chain* as it is not internal hash digest.

- $H_L^M = H_L^N$ and $\bar{Z}_M \neq \bar{Z}_N$ with $g(H_L^M, \bar{Z}_M) = g(H_L^N, \bar{Z}_N)$. Like the above case, this case also requires time about 2^t to get a simultaneous collision on both the chains.
- $H_L^M \neq H_L^N$ and $\bar{Z}_M = \bar{Z}_N$ with $g(H_L^M, \bar{Z}_M) = g(H_L^N, \bar{Z}_N)$. Since there is a collision on the *accumulation chain* the attacker has to find a collision on the *cascade chain* which costs $\Omega(2^{t/2})$ for f modeled as a random oracle.

To sum up, it would take time $\Omega(2^{t/2})$ to perform a 2nd-collision attack on **3C** for the compression function modeled as a random oracle.

7 Comparison of 3C with related hash function proposals

Ferguson and Schneier [14] proposed double-hashing scheme $H_{IV}(H_{IV}(x))$ to prevent straight-forward length extension attacks discussed in section 6.3. This double-hashing scheme is a key-less or fixed IV variant of NMAC and HMAC MAC constructions proposed by Bellare, Canetti and Krawczyk [2]. It is obvious that multi block collision attacks on this nested construction work as effectively as they are on **MD**. That is, the attacker finds multi-block collisions on the inner hash function first and the application of the outer function does not prevent the propagation of the collision to the whole hash function. As on the **MD** hash, 2^d -collisions can be found on their scheme with a complexity of $O(d \cdot 2^{t/2})$ and finding 2^d -(2^{nd}) preimages would take time $O(d \cdot 2^{t/2} + 2^t)$.

Gauravaram *et al.* [15] proposed CRUSH hash function based on iterated length halving technique to design rate 1 hash functions that can be instantiated with any secure 128-bit block cipher reduced to half the number of rounds as an alternative for the **MD** hash function. CRUSH was designed well before the MBCA on MD5 by Wang *et al.* [36] anticipating the single point of failure of hash functions in the MD family. CRUSH is immune against extension attacks and is conjectured that MBCA type attacks do not work on this structure. Unlike for **MD** hash functions, CRUSH requires buffering of data to be hashed which may not be the case with many applications that use hash functions. While the CRUSH structure is completely different from **MD**, **3C** structure is a tiny enhancement to **MD**.

Lucks [27] proposed wide-pipe and its special case double-pipe hash designs as failure-tolerant designs showing that they are more resistant against generic attacks explained in section 6 than the **MD** hash functions. While wide-pipe hash maintains more internal state than the hash size t using larger compression functions, double-pipe hash maintains twice the hash size as the internal state size by employing one single t -bit compression function used twice in parallel for each message block. In contrast, one could see **3C** structure as a special case of wide-pipe hash and is optimally efficient as no new large compression function needs to be designed for its execution. The wide-pipe, double-pipe, **3C** and double-hashing proposals resist the straight-forward length extension attack and 2nd-collision attacks discussed in section 6. All these hash functions provide $t/2$ -bit level of security against these attacks as long as their design criteria is satisfied; for example, wide-pipe hash requires processing of the compression function with an internal state at least twice the size of the hash value, **3C** requires at least three calls to the compression function.

Interestingly, we note that **3C** provides the same security levels as wide-pipe and double-pipe against most of the generic attacks on **MD** hash functions (attacks in sections 6.2, 6.3 and 6.4) in a more economical way. That is, one does not have to use wideish compression functions to prevent these attacks. Unlike the wide-pipe and double-pipe hash functions, **3C** does not provide more resistance and is as good as **MD** hash against Joux’s generic attacks discussed in section 6.1. While wide-pipe and double-pipe are designed for more protection against known generic attacks on **MD** hash functions discussed in section 6, **3C** is an enhancement of **MD** resisting recent multi-block collision attacks on the **MD** based hash functions.

In the wake of differential attacks on MD5, SHA-0 and SHA-1 [3, 5, 6, 38, 39] due to poor message expansion of the compression functions of these hash functions, Jutla and Pathak [20] proposed a new provably secure collision resistant compression function for SHA-1 hash function against the known differential collision attacks. In a related work [19], the same authors have shown that their new SHA-1 compression function also resists the natural extensions to the differential attacks. While their work focuses on improving the collision resistance of the compression functions against known collision attacks, our work aims at providing

efficient modifications to the popular **MD** structure for an improved protection against MBCA and some generic attacks. We note that one can also accommodate the new compression functions [20] in our design for more resistance against any future attacks from the point of view of design and also underlying compression functions.

Recently, Szydlo and Yin [35] have shown that when some simple message pre-processing techniques are combined with MD5 or SHA-1, the applications based on these hash functions are no longer vulnerable to known collision attacks. **3C**, **3C+** (see section 9) and the message pre-processing solutions of Szydlo and Yin all attempt to defeat the multi-block collision attacks by removing control of message differentials from the attacker. This occurs either through chaining (**3C**, **3C+**) or through message redundancy caused by interleaving message blocks with fixed strings or duplicates [35]. Both solutions are minimally invasive to existing deployments of the affected hash functions. However, Szydlo and Yin have the slight advantage in that their solutions do not require hardware implementations of the hash functions to be changed; only the inputs require additional processing. **3C** and **3C+** are only effective when the hash function can be modified to incorporate the chaining between compression functions, so may not be applicable in legacy hardware environments. For long messages, both solutions are exceptionally efficient, although **3C** and **3C+** have the advantage that their overheads diminish proportionally to the length of the message being hashed. After one hundred blocks have been hashed, the overhead of **3C/3C+** has fallen to less than 1%. Conversely, the overhead of the Szydlo-Yin schemes remains constant in the length of the message, chosen by the user as a parameter of between 12% and 50% reduction in throughput to match security expectations.

From the performance point of view, **3C** is slightly more expensive than **MD** especially when it is used to process short messages as the former requires at least three iterations of the compression function to process an arbitrary length message. To process 1-block (resp. 2-block) message, the running time of the **3C** is twice (resp. 1.5 times) that of **MD**. On an Intel Pentium 4 3.2GHz processor, **3C** based on MD5, incurs about 0.36% overhead and **3C** based on SHA-1 incurs about 0.27% overhead when these functions are used to process long messages. **3C** requires an extra iteration of the compression function similar to the double hashing (DHASH) proposal [14] and is as efficient as this scheme for the processing of long messages and unlike the DHASH scheme, **3C** is a single hashing scheme.

8 Hybrid hash constructions using **3C**

We note that as far as our knowledge is concerned the only way to increase the security of hash functions against Joux’s generic attacks [18] is by using the wide-pipe hash using largish compression functions or double-pipe hash function with two compressions to process every message block. For example, SHA-224 is a wide-pipe hash of SHA-256 [13] and SHA-384 is a wide-pipe hash of SHA-512 [13]. So far, there are no full collision attacks on any of these standard hash functions. Nevertheless, considering some active research in the analysis of SHA-256 [17, 29, 32, 40], we note that they would apply to its variant SHA-224 with no additional effort. For example, a near-collision on the reduced version of SHA-256 [29] became a full collision on the reduced version of SHA-224. Since attacks always get better, we see that having large compression functions is necessary to protect only against generic attacks but may not be against any future specific attacks such as differential collision attacks.

Due to the above reasons, we propose hybrid variants for **3C** that offer additional protection against both generic attacks and multi-block differential collision attacks. We combine the wide-pipe hash and the **3C** construction to attain a hybrid construction called 3CWP (see Fig 4) attaining additional protection against both the generic attacks and MBCA. The security analysis of 3CWP against generic attacks follows from [27] and against multi-block collision attacks follows from section 5. Similarly, one can combine **3C** with double-pipe hash function to obtain one more hybrid construction called **3CDP**.

9 The **3C+** construction: An enhanced **3C** construction

Fig 5 shows the **3C+** construction where a third internal chain called *final chain* has been added on top of the *cascade* and *accumulation chains* of **3C**. In **3C+**, we call *accumulation chain* as the *middle chain*. The

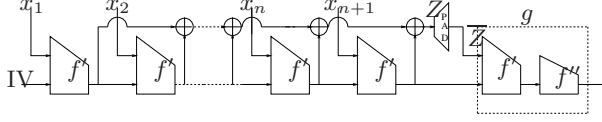


Fig. 4. The **3CWP** hybrid construction

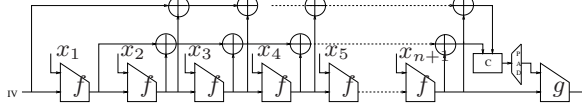


Fig. 5. The **3C+** hash construction

final chain in **3C+** slightly differs in the way it accumulates data from the *accumulation chain* of **3C** as it accumulates data from the *cascade chain* but the accumulation starts after processing the second message block. The final compression function f (denoted by g in Fig 5) takes as “message” the concatenation of the accumulated data from the *middle* and *final* chains, appropriately padded.

To find a multi-block collision on **3C+** the attacker has to get collisions simultaneously on all the three chains. To create a simultaneous collision on all the three chains at iteration i , the chaining difference on the *cascade chain* at iteration $i - 1$ (say Δ_{i-1}) must cancel the differences accumulated in the *middle* and *final* chains till the iteration $i - 2$. That is, the *middle* and *final* chains must maintain an equal difference Δ_{i-1} until the iteration $i - 2$ of the function f for a cancellation at iteration $i - 1$. This is impossible if *middle* and *final* chains do not start with the same difference before iteration $i - 2$.

For example, consider the pattern $(0, \Delta_1, \Delta_2, \Delta_3, \Delta_4, \Delta_5)$ of differences on the *cascade chain* obtained from the first to fifth iterations of f in **3C+**. This creates the pattern $(\Delta_1, \sum_{j=1}^2 \Delta_j, \sum_{j=1}^3 \Delta_j, \sum_{j=1}^4 \Delta_j, \sum_{j=1}^5 \Delta_j)$ on the *middle chain* until the fifth iteration and the pattern $(\Delta_2, \sum_{j=2}^3 \Delta_j, \sum_{j=2}^4 \Delta_j, \sum_{j=2}^5 \Delta_j)$ on the *final chain* until the fifth iteration. Now the difference $\sum_{j=2}^5 \Delta_j$ in the *cascade chain* after the sixth iteration of f cancels the difference accumulated in the *final chain* until the fifth iteration and creates a difference Δ_1 on the *middle chain*. To cancel the difference Δ_1 on the *middle chain*, the attacker must get the same difference Δ_1 on the *cascade chain* after the seventh iteration of f and again creating a difference of Δ_1 on the *final chain*. So canceling a difference in the *final chain* creates a difference in the *middle chain* and this process repeats.

Now, if an attacker finds two colliding messages that have identical first message blocks, then these two chains begin with a difference zero. This implies that a minimum of four message blocks need to be processed to find an MBCA on **3C+**. For example, the pattern $(0, 0, \Delta, \Delta, 0)$ creates a simultaneous collision on all the chains after processing four blocks. From this discussion, it is clear that **3C+** structure demands crafting and maintaining the same difference in both the *final* and *middle* chains until a multi-block collision is found.

Finally, we note that if the input to the *final chain* is taken from the *middle chain* rather than from the *cascade chain*, the difference pattern $(0, \Delta, \Delta, 0)$ that creates a collision on **MD** and **3C** will also create a collision on this modified construction of **3C+** and the MBCA security of this construction is lower bounded by **3C** and **MD**. Note this pattern does not create a collision on **3C+**. The analysis given for **3C** against generic attacks in section 6 also applies to **3C+**.

9.1 Variants for **3C** and **3C+**

We note that one can construct many variants for our **3C** and **3C+** designs by replacing XOR functions with any function in such a way that the new construction is at least as secure as **MD**. The generic construction for **3C** type of structures is called **3CG** as shown in Fig 6. The XOR function in the *accumulation chain* of **3C** is replaced with a generic function f' which can be instantiated with any non-linear function. The function ZPAD in Fig 6 denotes the padding of chaining values with 0's or any other formatted bits to use

them as a block for the function f' in the *accumulation chain*. The use of ZPAD depends on the choice of the function used for f' in **3CG**. So **3C**, in a way, is an instance of **3CG** design and is the simplest variant of **3CG** design.

Similarly, we call the generic variant for **3C+** as **3CG+**.

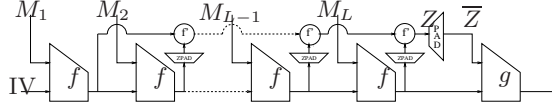


Fig. 6. The generic **3CG**-hash function

Here we provide two other variants for **3CG** and **3CG+** constructions. If the function f' in **3CG** is replaced with the compression function f then this modified construction resembles double pipe hash in some way and is less efficient than **3C**. We call this construction as **3CF**. The *cascade chain values* after the second iteration of f will be used as data blocks by applying the ZPAD function and the padded blocks are the inputs for the f functions in the accumulation chain. Clearly the amount of control that an attacker can have on these blocks to create an MBCA is much less than **3C**. Alternatively, one can use reduced versions of the compression function f in the *accumulation chain* in the place of f' . In such a case, an appropriate number of padded bits need to be included in the ZPAD function to meet the requirements of f' . A slight variant of **3C+** whose cost relative to **3C** bound to be nearly as small as XOR can be designed by interpreting the t -bit chaining value in the *final chain* as an element of $GF(2^t)$ and multiplying it by 2 at each step [1]. If the *final chain* accumulation process starts after the first iteration of f unlike in **3C+** then this structure results in a *final chain* accumulation equation that resembles Galois-Carter-Wegman structure of GHASH in [28]. We call this construction as **3CCW+**. Similarly, one can construct several variants for both **3CG** and **3CG+** designs.

10 Implementation issues

Tables 3 and 4 compare the throughputs of MD5 and SHA1 respectively, against their double-hashing, **3C** and **3C+** versions. The metrics are obtained on a Pentium-M machine with a CPU speed of 2 GHz, by hashing messages of the given payload repeatedly until one gigabit of digest material has been acquired.

The results are unsurprising. It is well known that SHA-1 is less efficient than MD5 because it executes more rounds per message block and has a more complex message expansion than MD5. For each compression function, DHASH outperforms **3C** and **3C+** for short messages. **3C** and **3C+** suffer from the requirement that a minimum of three blocks (ie. 192 bytes) be hashed, irrespective of the payload. For payloads in excess of this amount, none of the schemes shown in the tables are especially distinguishable via their throughputs.

An increase in the state size of **3C**, **3C+** and their associated hash functions is required to hold the additional chaining variables. For MD5, these increases in size amount to 17% and 34% respectively. For SHA-1, the chain represent an overhead of 4% and 8% respectively.

11 Some discussion

As said earlier, while the plain **MD** iterative structure provides implementation efficiency for a hash function, the way it is iterated is important for the security of the hash function from both generic attacks and short cut differential attacks. **3C** is a fundamental concept and design aimed at improving the **MD** construction slightly. Hence, it should be taken as a first step in improving the general hash function design. It is important to construct variants for the **3C** construction that become instances of **3CG** as suggested in the paper by replacing the linear XOR sum with various non-linear functions that provide more security to the evolved designs. Such designs would take away the control from the attacker in constructing message blocks with specific differentials that produce a collision on both the chains.

Table 3. Time in secs to hash one gigabyte of data using MD5 on Pentium-M machine with a CPU speed of 2GHz

Payload (bytes)	MD5	MD5-DHASH	MD5-3C	MD5-3C+
16	4.53	8.91	11.72	11.67
32	2.11	4.32	5.33	5.56
64	1.95	3.05	2.99	3.17
128	1.61	2.17	1.85	1.97
256	1.43	1.71	1.55	1.64
1K	1.30	1.37	1.33	1.40
1M	1.25	1.26	1.25	1.33
1G	1.25	1.25	1.25	1.34

Table 4. Time in secs to hash one gigabyte of data using SHA-1 on Pentium-M machine with a CPU speed of 2GHz

Payload (bytes)	SHA-1	SHA-1-DHASH	SHA1-3C	SHA1-3C+
16	7.73	18.13	24.34	26.55
32	3.61	8.51	11.99	12.67
64	3.37	5.81	6.51	6.79
128	2.67	3.92	3.7	3.82
256	2.33	2.95	2.58	2.63
1K	2.08	2.22	2.14	2.17
1M	1.99	1.99	1.99	2.02
1G	1.99	2.00	2.02	2.03

12 Conclusion

The recent cryptanalysis of hash functions MD5, SHA-0, SHA-1 exploited the **MD** iterative structure of these hash functions using multi-block collision search techniques. The proposed **3C** and **3C+** variants to the **MD** construction are at least as resistant as **MD** against MBCA. The constructions can be implemented by simple adjustments to the existing **MD**-style implementations. This paper is the first paper to introduce solutions to MBCA on the **MD** based hash functions since they were first identified by Wang *et al* on MD5 [36]. Several constructions presented in this paper are relatively new and all are variants for the fundamental construction **3C**.

Acknowledgments:

Many thanks to Suganya Annadurai, Paulo Barreto, John Kelsey, Lars Knudsen, Stefan Lucks, Adrian McCullagh, David McGrew, Vincent Rijmen and Søren Thomsen for their encouragement and valuable comments on the analysis, design and performance aspects presented in the earlier drafts. Many thanks to anonymous reviewers of ACISP 2006 for many useful comments on several aspects on the shortened version of this paper which is to be published at ACISP 2006.

References

1. Paulo Barreto, 2006. Personal Communication.
2. Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Keying hash functions for message authentication. In Neal Koblitz, editor, *Advances in Cryptology—CRYPTO '96*, volume 1109 of *Lecture Notes in Computer Science*, pages 1–15. Springer-Verlag, 18–22 August 1996. Full version of the paper is available at <http://www-cse.ucsd.edu/users/mihir/papers/hmac.html>. Last access date: 9th of July 2005.
3. Eli Biham and Rafi Chen. Near-Collisions of SHA-0. In Matt Franklin, editor, *Advances in Cryptology-CRYPTO 2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 290–305. Springer, August 15–19 2004.
4. Eli Biham, Rafi Chen, Antoine Joux, Patrick Carribault, Christophe Lemuet, and William Jalby. Collisions of SHA-0 and Reduced SHA-1. In Ronald Cramer, editor, *Advances in Cryptology - EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 36–57. Springer, 2005.

5. Eli Biham, Rafi Chen, Antonie Joux, Patrick Carribault, Christophe Lemuet, and William Jalby. Collisions of SHA-0 and reduced SHA-1. In Ronald Cramer, editor, *Advances in Cryptology - EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 36–57. Springer, 2005.
6. F. Chabaud and A. Joux. Differential collisions in SHA-0. In Hugo Krawczyk, editor, *Advances in Cryptology—CRYPTO '98*, volume 1462 of *Lecture Notes in Computer Science*, pages 56–71, 1998.
7. Ivan Damgård. A Design Principle for Hash Functions. In Gilles Brassard, editor, *Advances in Cryptology: CRYPTO 89*, volume 435 of *Lecture Notes in Computer Science*, pages 416–427. Springer-Verlag, 1989.
8. Magnus Daum and Stefan Lucks. Hash Collisions (The Poisoned Message Attack) "The Story of Alice and her Boss". Presented at the Rump Session of EUROCRYPT 2005, May 2005. The result is available at <http://th.informatik.uni-mannheim.de/people/lucks/HashCollisions/>. Last access date: 10 November 2005.
9. Richard Drews Dean. *Formal Aspects of Mobile Code Security*. PhD thesis, Princeton University, 1999.
10. Bert den Boer and Antoon Bosselaers. Collisions for the compression function of MD5. In T. Hellesest, editor, *Advances in Cryptology – EUROCRYPT '93*, volume 765 of *Lecture Notes in Computer Science*, pages 293–304. Springer-Verlag, 1994.
11. Hans Dobbertin. Cryptanalysis of MD5 compress. Presented at the Rump Session of EuroCrypt'96, 1996.
12. Don B. Johnson. Improving Hash Function Padding. Technical report, National Institute of Standards and Technology NIST, October 2005. The paper and slides of this work are available at <http://www.csrc.nist.gov/pki/HashWorkshop/2005/program.htm>. Last access date: 12th of May 2006.
13. Federal Information Processing Standard (FIPS). *Secure Hash Standard*. National Institute for Standards and Technology, August 2002. This document is available at <http://csrc.nist.gov/publications/fips/fips180-2/fips180-2.pdf>. Last access date: 15th of May 2006.
14. Niels Ferguson and Bruce Schneier. *Practical Cryptography*, chapter 6, pages 83–96. John Wiley & Sons, 2003.
15. Praveen Gauravaram, William Millan, and Lauren May. CRUSH: A New Cryptographic Hash Function using Iterated Halving Technique. In *Proceedings of the workshop on Cryptographic Algorithms and their uses*, pages 28–39, Goldcoast, Australia, July 4–5 2004.
16. Praveen Gauravaram, William Millan, Juanma Gonzalez Nieto, and Edward Dawson. 3C- A Provably Secure Pseudorandom Function and Message Authentication Code. A New mode of operation for Cryptographic Hash Function. Cryptology ePrint Archive, Report 2005/390, 2005. <http://eprint.iacr.org/>.
17. Henri Gilbert and Helena Handschuh. Security Analysis of SHA-256 and Sisters. *Lecture Notes in Computer Science*, 3006:175–193, 2004.
18. Antoine Joux. Multicollisions in Iterated Hash Functions. Application to Cascaded Constructions. In Matt Franklin, editor, *Advances in Cryptology-CRYPTO 2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 306–316, Santa Barbara, California, USA, August 15–19 2004. Springer.
19. Charanjit S. Jutla and Anindya C. Patthak. Is SHA-1 conceptually sound? Cryptology ePrint Archive, Report 2005/350, 2005. <http://eprint.iacr.org/>.
20. Charanjit S. Jutla and Anindya C. Patthak. A simple and provably good code for SHA message expansion. Cryptology ePrint Archive, Report 2005/247, 2005. <http://eprint.iacr.org/>.
21. John Kelsey, 2006. Personal Communication.
22. John Kelsey and Bruce Schneier. Second Preimages on n-bit Hash Functions for Much Less than 2ⁿ Work. In Ronald Cramer, editor, *Advances in Cryptology - EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 474–490. Springer, 2005.
23. Lars Knudsen. Personal Communication, June 2006.
24. Xuejia Lai and James L. Massey. Hash Functions Based on Block Ciphers. In R. A. Rueppel, editor, *Advances in Cryptology—EUROCRYPT 92*, volume 658 of *Lecture Notes in Computer Science*, pages 55–70. Springer-Verlag, 24–28 May 1992.
25. Arjen Lenstra, Xiaoyun Wang, and Benne de Weger. Colliding x.509 certificates. Cryptology ePrint Archive, Report 2005/067, 2005. <http://eprint.iacr.org/>.
26. Stefan Lucks. Design principles for iterated hash functions. Cryptology ePrint Archive, Report 2004/253, 2004. <http://eprint.iacr.org/>.
27. Stefan Lucks. A Failure-Friendly Design Principle for Hash Functions. In Bimal Roy, editor, *Advances in Cryptology - ASIACRYPT 2005*, volume 3788 of *Lecture Notes in Computer Science*, pages 474–494. Springer-Verlag, 2005.
28. David McGrew and John Viega. The Galois/Counter Mode of Operation (gcm). NIST special publication, National Institute for Standards and Technology.
29. Florian Mendel, Norbert Pramstallar, Christian Rechberger, and Vincent Rijmen. Analysis of a step-reduced SHA-256, 2006. This paper is to be published in the Proceedings of FSE'2006.

30. Alfred J. Menezes, Paul C. Van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*, chapter 9, pages 321–383. The CRC Press Series on Discrete Mathematics and its Applications. CRC Press, 1997.
31. Ralph Merkle. One way Hash Functions and DES. In Gilles Brassard, editor, *Advances in Cryptology: CRYPTO 89*, volume 435 of *Lecture Notes in Computer Science*, pages 428–446. Springer-Verlag, 1989.
32. Norbert Pramstallar, Christian Rechberger, and Vincent Rijmen. Preliminary analysis of the SHA-256 Message Expansion. Technical report, National Institute of Standards and Technology NIST, October 2005. This paper is available at <http://www.csrc.nist.gov/pki/HashWorkshop/program.htm>. Last access date: 6 December 2005.
33. Bart Preneel. *Analysis and design of Cryptographic Hash Functions*. PhD thesis, Katholieke Universiteit Leuven, 1993.
34. Phillip Rogaway and Thomas Shrimpton. Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance. In Bimal K. Roy and Willi Meier, editors, *Fast Software Encryption (FSE)*, volume 3017 of *Lecture Notes in Computer Science*, pages 371–388. Springer-Verlag, 2004.
35. Michael Szydlo and Yiqun Lisa Yin. Collision-resistant usage of MD5 and SHA-1 via message preprocessing. In David Pointcheval, editor, *CT-RSA*, volume 3860 of *Lecture Notes in Computer Science*, pages 99–114. Springer, 2006.
36. Xiaoyun Wang, Dengguo Feng, Xuejia Lai, and Hongbo Yu. Collisions for Hash Functions MD4, MD5, HAVAL-128 and RIPEMD. Cryptology ePrint Archive, Report 2004/199, 2004. <http://eprint.iacr.org/>.
37. Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. Efficient collision search attacks on SHA-0. In Victor Shoup, editor, *Advances in Cryptology—CRYPTO '05*, volume 3621 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 2005, 14–18 August 2005.
38. Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. Finding collisions in the full SHA-1. In Victor Shoup, editor, *Advances in Cryptology—CRYPTO '05*, volume 3621 of *Lecture Notes in Computer Science*, pages 17–36. Springer, 2005, 14–18 August 2005.
39. Xiaoyun Wang and Hongbo Yu. How to Break MD5 and Other Hash Functions. In Ronald Cramer, editor, *Advances in Cryptology - EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 19–35. Springer, 2005.
40. Hirotaka Yoshida and Alex Biryukov. Analysis of a SHA-256 variant. In *Selected Areas in Cryptography*, volume 3897 of *Lecture Notes in Computer Science*, pages 245–260. Springer, 2005.