Constructing Secure Hash Functions by Enhancing Merkle-Damgård Construction

Praveen Gauravaram[1], William Millan[1] and Ed Dawson[1] and Kapali Viswanathan[2]

[1] Information Security Institute (ISI), QUT
2 George Street, Brisbane, QLD, 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. The classic Merkle-Damgård (**MD**) structure provides a popular way of turning a fixed-length compression function into a variable-length input cryptographic hash function. However, the multi-block collision attacks (MBCA) on the **MD**-style hash functions MD5, SHA-0 and SHA-1 demonstrate the weakness of the **MD** construction in extending the collision resistance property of a single compression function to its iterations. In this paper, we investigate a recently proposed cryptographic construction (called **3C**) devised by enhancing the **MD** construction, and prove it provides quantitatively more resistance against MBCA than does the **MD**-style. Specifically, we prove that it requires at least $2^{t/2}$ computational effort to perform any MBCA on the t-bit **3C** hash function when the same attack on a t-bit **MD** hash function (using the same compression function) requires an effort not less than $2^{t/4}$. This is the first result showing a generic construction with resistance to MBCA. We further improve the resistance of the **3C** design against MBCA and propose the new **3C+** hash function construction. We prove that **3C+** is completely *immune* to MBCA since it costs at least $2^{t/2}$ effort to perform any MBCA on the **3C+** construction. This reduces the collision security of **3C+** to the collision security of the underlying compression function, hence restoring the paradigm that one only needs to design a secure compression function to obtain a secure iterated hash function. Both the **3C** and **3C+** constructions are very simple adjustments to the **MD** construction and they are immune to the straight forward extension attacks which apply to the **MD** hash functions. The second preimage attacks on t-bit hash functions also do not work on the constructions presented in this paper.

Key words: Merkle-Damgård construction, multi-block collision attacks (MBCA), hash function, 3C, 3C+.

1 Introduction

In 1989, Damgård [4] and Merkle [17] independently proposed a similar iterative structure to construct a collision resistant cryptographic hash function $H : \{0,1\}^* \rightarrow \{0,1\}^t$ using a fixed 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 current dedicated hash functions such as MD4, MD5, SHA-1, SHA-256 and SHA-512. The motivation of the **MD** construction is:

> "If there exists a computationally collision free function f from b bits to t bits where $b > t$, then there exists a computationally collision free function H mapping messages of arbitrary polynomial lengths to t-bit strings." [4]

Until recently[1], it had been believed that the problem of designing a collision resistant H reduces to the problem of designing a collision resistant fixed-input compression function, f. 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 [16, p.373]. The existence of multi-block collision attacks (MBCA) on the hash functions MD5, SHA-0 and SHA-1 [2, 25, 26] prove this insufficiency. These attacks show that these iterated hash functions do *not properly preserve* the collision resistance property of their respective compression functions with fixed IV.

The multi-block collision attacks on hash functions leave open the following questions:

1. Is it possible to design a collision resistant hash function relying on the collision resistance of the compression function with its fixed IV?
2. Is it possible to design an efficient structure resistant to multi-block collision attacks?

In this paper, we attempt to answer both these questions. Our motivation is to show that while the consecutive iterations of the compression function is necessary for the implementation efficiency of a hash function, **the way** the compression function is iterated is important for the security of the hash function. In this paper, we propose a new variant to the **MD** construction called **3C**. The **3C** hash function processes the intermediate chaining values of the **MD** construction maintaining a second internal chaining variable. The **3C** construction is the simplest secure variation of the **MD** construction that one can obtain.

The first result of this paper is that a multi-block collision attack on the t-bit **3C** hash function iterated over a compression function f requires an effort of at least $2^{t/2}$ when the same attack on an **MD** hash using the same compression function f requires at least $2^{t/4}$ computational effort. That is, a multi-block collision attack on **3C** based on f succeeds with a complexity less than $2^{t/2}$ *only if* the multi-block collision attack works on an **MD** hash function with the same f with a complexity less than $2^{t/4}$. This provides an exponential increase of security for an iterated hash function against multi-block collision attacks, and is the first result of its kind.

Next, we add extra memory to the **3C** construction to achieve a higher level of security against multi-block collision attacks and call this variant **3C+**. We show that **3C+** is *immune* to any MBCA conducted against the internal **MD**-style structure. Specifically, we show that performing any MBCA on **3C+** requires effort not less than $2^{t/2}$, even if the MBCA can be conducted for free on the **MD**-style! Hence the collision security of the **3C+** construction reduces to the collision security of the underlying compression function.

Our results suggest the existence of a fundamental trade-off between internal memory of a hash function and its resistance to MBCA. Our result for **3C+** is optimal since complete immunity to MBCA is achieved; further increase in internal memory can give no further resistance to MBCA and using less internal memory (as does **3C**) gives only partial MBCA resistance.

Related work: Lucks [15] has proposed wide-pipe and double-pipe hash constructions as failure-friendly variants to the **MD** hash functions improving the resistance of **MD** structure against generic attacks such as multicollisions in iterated hash functions [10]. The **3C** and **3C+** hashes also work as failure-friendly variants to the **MD** hash functions which improve the resistance against MBCA. One could see our proposed multiple-chain structure as a special case of the wide-pipe hash, however our proposal is optimally efficient as no new compression function needs to be designed. Our proposal is the minimum adjustment to **MD**-style that could be imagined, and we offer a new proof of security.

[1] New kinds of collision attacks have been proposed in which the differences between the two messages extend over more than one message block.

Coron *et al.* [3] have provided four hash functions (all are modifications to the plain **MD** construction) that work as random oracles when the underlying compression functions work as random oracles. We note that following the assumptions of [3], one can show that when the underlying compression function works as a random oracle, the **3C** also works as a random oracle. Ferguson and Schneier [8] proposed double hashing to prevent straight forward extension attacks and **3C** prevents straight forward extension attacks using single hashing. While there has been work [11, 12, 20] on improving compression functions against known techniques of differential cryptanalysis, our work shows that a minimal variation of the **MD** structure provides more resistance against multi-block collision attacks.

The 3C construction was initially proposed in [9], where it was proven than 3C works as a Pseudo-Random Function (PRF) (and hence is suitable as a Message Authentication Code (MAC)), so long as the compression function is a PRF with no additinal assumptions required. Due to the similarity of design, clearly 3C+ is also a PRF and a MAC. In this paper we focus on the security of 3C and 3C+ as a cryptographic hash function, and prove their resistance to multi-block collision attacks.

Outline: In Section 2, we describe **MD** hashing and collision attacks on it. In Section 3, new observations on multi-block collision attacks are discussed. In section 4, the **3C** hash function is introduced and its security analysis against multi-block collision attacks is covered in Section 5. In Section 6, analysis of **3C** against generic attacks is given and it is compared with some other hash function proposals. In Section 7 we introduce the **3C+** structure and prove its security. The paper is concluded in Section 8.

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 [16, 18]. 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 $M_1, \ldots, M_{L-1}$ of equal length b where a block M_L containing the length $|M|$ (**MD** strengthening) 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 outputting $H_{IV}(M) = H_L$ as shown in Fig 1.

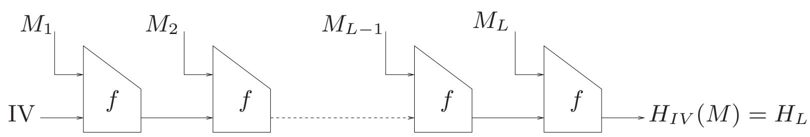

Fig. 1. The Merkle-Damgård (**MD**) construction

Collision attacks on hash 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 the formal definition see [19]. A hash function 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 [16, p.372]:

1. Collision attack: collisions using a fixed IV for two distinct messages (e.g. [23]). 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. [7]). 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. [6]). We call them *Type 3* collisions.

Multi-block collision attacks on hash functions:

A multi-block collision attack (MBCA) finds two colliding messages which differ in more than a single 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 protecting against MBCA is very significant. Although all the MBCA attacks reported so far use some special structure, in this paper we use the term MBCA to refer to any collision attack where the message differences extend over more than one message block. In our later security analysis, we make no further assumptions about the nature of the MBCA we consider and hence the security analysis is completely generic and applies equally well to (as yet) undiscovered MBCA styles.

The recent collision attacks on MD5 [26], SHA-0 [2] and SHA-1 [25] are multi-block collision attacks where near-collisions found after processing a few message blocks were converted to full collisions. The *Type 1* collisions were (reportedly) hard to find for the single compression functions of these hash algorithms. 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) as shown in Fig 2 where $h_1 \oplus h'_1 = \Delta$ and $h_2 = h'_2$. This technique can be generalized to more than two blocks as the 4-block collision attack on SHA-0 [2]. (This attack was later improved [24] to the collision format $H(M_1, N_1) = H(M_1, N_2)$ which is not a multi-block collision attack).

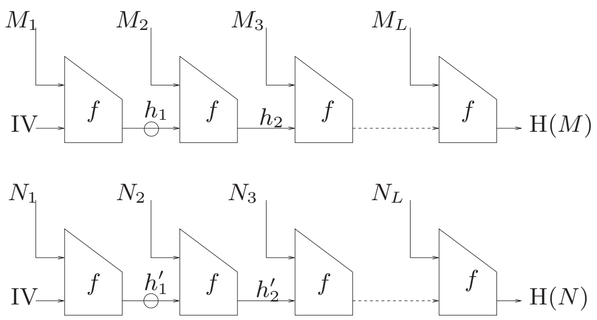

Fig. 2. 2-Block collision in a hash function H (H, is for example, MD5)

These cryptanalytical results show that the establishing full collisions in hash functions using near collisions is easier than attacking compression functions themselves by exploiting the **MD** iterative structure on which these hash functions are based on. Generating collisions in hash functions using this tool is particularly useful when no characteristic exists that predicts a full collision in the first block and this technique reduces the complexity of the attack when the complexity required to find a collision for the first block is really large [2]. For example, it was demonstrated on SHA-1 [25] that due to the freedom available to the attacker in generating first block near-collisions, one can maintain essentially twice the search complexity while converting those near-collisions to full collisions on the second block.

3 New Observations on multi-block collision attacks

We note that while establishing full collisions in hash functions using near-collisions is easy using differential cryptanalysis, in principle, there are other ways of generating multi-block collisions on **MD** hash functions instead of using near-collisions (see Appendix A for the other way of generating multi-block collisions). In addition, we note that collisions on the second block are basically a *special case* of *Type-3* collisions where a near-collision is required as input. So a *multi-block collision on the* **MD** *hash function is a combination of a near-collision and a special Type-3 collision*. In addition, near-collisions do not need to begin from the fixed IV of the hash function. They can also be due to arbitrary IV when the attacker chooses the same blocks initially and starts the multi-block collision attack after processing those initial blocks. The multi-block collision attacks on MD5, SHA-0 and SHA-1 belong to the former case.

From these observations 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 Type-3 collisions in their design criteria.* Preneel pointed out a decade back [18] that most hash functions are not designed to meet this criteria. Note that SHA-1 did not exist then. Even Damgård's [4] proof implicitly notes that the necessity of *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 [16, p.372]. For example, the two IVs in the *Type 3* collision attack on the compression function of MD5 [6] differ in *only* 6 bits. Hence multi-block collisions, whether they start from fixed IV or arbitrary IV are clearly a chain of *special Type 3* collisions.

Finally, consider the following statement from [16, p.373]:

> "A compression function secure against fixed IV collision attacks is necessary and sometimes, but not always, sufficient for a secure iterated hash; and security against pseudo collision attacks is desirable, but not always necessary for a secure hash function in practice".

These attacks prove the insufficiency of *Type 1* collision resistance of the compression functions of MD5, SHA-0 and SHA-1 for a secure hash supporting the first part of the above statement. They also show that *Type 3* collision resistance of the compression function is a necessary property for a secure hash contradicting the second part of the above statement. From the known attacks on hash functions, we derived the Table 1 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 1 shows does not apply.

4 The 3C construction: An Enhanced MD construction

The **3C** construction is shown in Figure 3 and Figure 4. This structure has an *accumulator* XOR function iterated in the *accumulation chain* (denoted by u_i in Figure 4) and a compression function f (f, for example, is the compression function of MD5 or SHA-1) iterated in the *cascade chain* (denoted H_i in Figure 4) 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

Table 1. Resistances of some compression functions

Compression function	*Type-1*	*Type-2*	*Type-3*	*Special Type-3*
MD4	NO [23]	NO	NO	-
MD5	YES	NO [7]	NO [6]	NO [26]
SHA-0	YES	NO [26]	YES	NO [2]
SHA-1	YES	YES	YES	NO [25]
RIPEMD	NO [23]	NO	NO	-
HAVAL-128	NO [22]	NO	NO	-

software currently implementing an **MD**-style hash function can be very simply altered to adopt the **3C** structure, without altering the underlying 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 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 one block data, 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* by padding Z with 0 bits as shown in Fig 3. If the size of the data is less than block size b of f then zeros are appended to the data to fill the b bit data block.

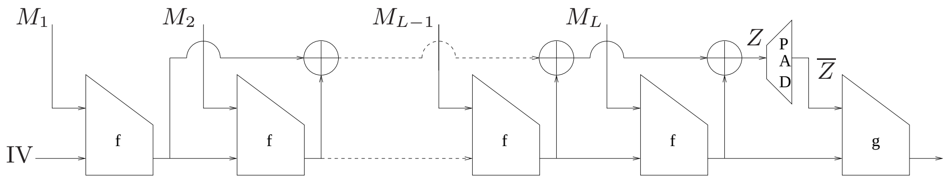

Fig. 3. The **3C**-hash function

5 Security analysis of the 3C hash function

Security analysis of the **3C** hash construction can be given in several ways based on the assumptions on the compression function f. By assuming f as a random oracle, one can show that **3C** works as a random oracle following the assumptions and proof techniques of [3]. Then any application proven secure assuming the hash function as a random oracle would remain secure if one plugs in **3C** assuming that f works as a random oracle.

In this Section, we will provide security analysis of **3C** against generic multi-block collision attacks. We will use Fig 4, to explain the analysis.

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 $(M_1, \ldots, M_L) \neq (N_1, \ldots, N_L)$ 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

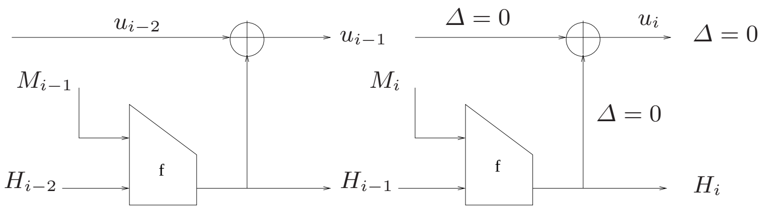

Fig. 4. Creating an internal collision for **3C**

$H(M)$ and $H(N)$ respectively. We denote (u_L, v_L) by (Z_{LM}, Z_{LN}) and $\overline{Z}_{LM} = \text{PAD}(Z_{LM})$, $\overline{Z}_{LN} = \text{PAD}(Z_{LN})$.
Collisions on H can be obtained internally or finally as given in Definition 1

Definition 1. *Collisions on H can be obtained internally or finally as below:*

1. *Terminal/Final collisions: They involve one of the following cases:*
   - $H_L^M \neq H_L^N$ *and* $\overline{Z}_{LM} \neq \overline{Z}_{LN}$ *with* $g(H_L^M, \overline{Z}_{LM}) = g(H_L^N, \overline{Z}_{LN})$
   - $H_L^M = H_L^N$ *and* $\overline{Z}_{LM} \neq \overline{Z}_{LN}$ *with* $g(H_L^M, \overline{Z}_{LM}) = g(H_L^N, \overline{Z}_{LN})$
   - $H_L^M \neq H_L^N$ *and* $\overline{Z}_{LM} = \overline{Z}_{LN}$ *with* $g(H_L^M, \overline{Z}_{LM}) = g(H_L^N, \overline{Z}_{LN})$
2. *Internal collisions:* $H_L^M = H_L^N$ *and* $\overline{Z}_{LM} = \overline{Z}_{LN}$ *implies* $g(H_L^M, \overline{Z}_{LM}) = g(H_L^N, \overline{Z}_{LN})$. □

Definition 2. *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.*

Definition 3. *A fixed-difference collision attack on a function finds two inputs which generate outputs that have a pre-specified XOR difference.*

When the fixed difference is zero, then the fixed-difference attack is identical to a traditional collision attack.

Lemma 1. *The expected complexity of performing a fixed-difference collision attack on t-bit vectors is $2^{\frac{t}{2}}$.*

Proof: The well-known birthday paradox based collision search can be modified slightly to find fixed-difference collisions. In the general case of finding a match between elements of two tables, each of size $2^{\frac{t}{2}}$, simply XOR the required fixed difference with each element in one of the tables, then sort and find the match normally. □

Definition 4. *A difference-set collision attack is a variation of the fixed-difference collision attack, where the two values must have an XOR difference of any value which is an element of a pre-specified set.*

It is easier to perform a difference-set collision attack when the size of the targeted difference-set is large. This is quantified in Lemma 2.

Lemma 2. *The expected complexity of performing a difference-set collision attack on t-bit values when the target difference set has 2^Q values is given by $2^{\frac{t-Q}{2}}$.*

Proof: Consider a two-table collision search, where the tables are of size $2^x \leq 2^{\frac{t}{2}}$. There exist about 2^{2x} differences between elements of these tables. With high probability, one of these differences is contained in the required difference-set (of size 2^Q) when $2x + Q = t$, which implies the complexity of the difference-set attack is given by $2^x = 2^{\frac{t-Q}{2}}$. □
Remark: When the size of the set is 1, so that $Q = 0$, the set-difference attack collapses to the fixed-difference collision attack.

The above results are used in the proof of our main result regarding the security of 3C against multi-block collision attacks.

Main Results

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

Proof: By inspection of **3C** structure in Fig 4 and properties of XOR in iteration i. □

Lemma 4. *Assuming the existence of a collision in the accumulation chain at iteration $i - 1$, it requires effort equal to a single-block Type-3 collision attack on f to create an internal collision in* **3C** *at iteration i.*

Proof: By inspection of **3C** structure in Fig 4, a fixed-difference Type-3 collision attack must be performed on the f-function at iteration $i - 1$. By Lemma 1, this is equivalent to the effort of performing a single-block Type-3 collision attack on f. □

Lemma 5. *Creating a collision in the accumulation chain of* **3C** *requires a fixed-difference Type-3 collision attack on f, with complexity $2^{\frac{t}{2}}$ when f is Type-3 collision resistant.*

Proof: By structure of **3C** in Fig 4 and XOR properties, the difference at the *cascade chain* in iteration $i - 1$ must be equal to the difference in the *accumulation chain* at iteration $i - 2$, so that these differences cancel to produce the desired collision in the *accumulation chain* at iteration $i - 1$. From Lemma 1, the complexity of this process is $2^{\frac{t}{2}}$ for Type-3 collision resistant f. □

Lemma 6. *To create a collision for* **3C** *requires one of the following:*
(i) a terminal collision, or
(ii) a single-block Type-1 collision attack on the first f-function, or
(iii) Type-2 collision attack on f or
(iv) a pair of independent single block Type-3 collision attacks on consecutive internal f-functions, or
(v) a multi-block attack on the cascade chain that is compatible with the differences in the accumulation chain.

Proof: By Definition 1, a collision for **3C** results from either a terminal collision (case (i)) or an internal collision. The four ways of generating an internal collision for **3C** are summarized as cases (ii), (iii) and (iv).

The possibility for case (ii) is obvious. Case (iii) results from the observation of the collision format of two streams (M_1, N_1) and (M_1, N_2) where $N_1 \neq N_2$. There can be more than one similar blocks in the two streams initially before the different message blocks. The combination of Lemma 3, Lemma 4 and Lemma 5 shows that two consecutive single-block Type-3 collision attacks (one of which is a fixed-difference Type-3 collision attack) are required to obtain a simultaneous internal collision for both the chains of **3C** in case (iv). This leaves the possibility of using multi-block attacks as case (v), where the non-trivial compatibility requirement is seen from Lemma 5. □

Theorem 1. *Using any multi-block collision attack on* 3C *(which is based on a compression function f) requires effort at least $2^{\frac{t}{2}}$, whenever the same multi-block collision attack on a* MD *hash function (based in the same f) requires effort of $2^{\frac{t}{4}}$ or more.*

Proof: Consider an generic MBCA on the MD-structure within **3C**, and consider the conditions under which it also creates a full (2-chain) internal collision at iteration i. By **3C** structure in Fig 4 and properties of the XOR at iteration $i-1$, it is clear that, in order to generate the *accumulation chain* collision at iteration $i-1$ (as required by Lemma 3), the multi-block collision attack must generate a difference in the *cascade chain* at iteration $i-1$ which is *exactly the same* as the *accumulation chain* difference already present in iteration $i-2$. Hence, performing the multi-block attack on the *cascade chain* is greatly hampered by this additional requirement. The attacker *is not free* to use the result of an individual multi-block collision attack, as the last non-zero difference it generates (in iteration $i-1$) must be equal to the *accumulation chain* difference obtained at iteration $i-2$. The probability that these independently generated differences being the same is only 2^{-t}. Assume an attacker generates data providing a set of 2^Q differences in the *accumulation chain* at iteration $i-2$. Then the subsequent multi-block collision attack on the *cascade chain* must be repeated a sufficient number of times to obtain a match between the *cascade chain* difference at iteration $i-1$, and an element of this set.

Thus a difference-set collision attack must be performed as part of any attack using an MBCA on **MD** to create an MBCA on **3C**. By Lemma 2, the attacker must repeat the multi-block collision attack at least $2^{\frac{t-Q}{2}}$ times. The total effort by the attacker is $2^Q + 2^{C+\frac{t-Q}{2}}$, where the complexity of a multi-block collision attack is 2^C. In the case where $Q \geq \frac{t}{2}$, clearly **3C** has security versus the attack of at least $2^{\frac{t}{2}}$, for any positive value of C. Now consider the requirements in the case when $Q \leq \frac{t}{2}$. Ignoring the first term (which is small when Q is small) and taking logs to base 2, we need $C + \frac{t-Q}{2} \geq \frac{t}{2}$ to hold for the effort to break **3C** to be at least $2^{t/2}$. Now, the assumption that $Q \leq \frac{t}{2}$ is equivalent to saying that $C + \frac{t-Q}{2} \geq C + \frac{t}{4}$. Combining these two expressions we conclude that **3C** has at least $2^{\frac{t}{2}}$ security versus the multi-block collision attack whenever $C \geq \frac{t}{4}$. □

The above theorem is completely generic and it applies to any attack which seeks to generate a collision using messages that differ in more than one message block. The next theorem shows that the collision security of **3C** reduces to the collision security of the underlying compression function f (assuming that the best MBCA on **MD** (using f) has complexity not less than $2^{\frac{t}{4}}$).

Theorem 2. *Given a t-bit underlying f function which is Type-3 and Type-2 collision resistant and has security against multi-block collision attacks of at least $2^{t/4}$, the best collision attack on* **3C** *is either (i) a traditional birthday paradox based Type-1 collision attack on the entire* **3C***, or (ii) a traditional single-block Type-1 collision attack on the first f-function or (iii) a Type-2 collision attack with the same few initial blocks. Overall, the collision security of* **3C** *with t-bit output is $2^{\frac{t}{2}}$.*

Proof: The complexity of finding a collision for **3C** is the minimum complexity among the four cases from Lemma 6. Clearly cases (i),(ii),(iii) of Lemma 6 have complexity $2^{\frac{t}{2}}$. Case (iv) of Lemma 6 requires effort of $2^{\frac{t}{2}+1}$. By Theorem 1 the complexity of case (v) of Lemma 6 is not less than $2^{\frac{t}{2}}$. The minimum of these efforts is $2^{\frac{t}{2}}$. □

These results show that the security of **3C** against MBCA is greatly improved over the classic **MD**-style hash functions. The underlying compression function can possess a mild weakness versus MBCA in **MD**-style and yet still offer ideal protection versus MBCA for the **3C** structure. Thus currently available compression functions can be utilized in **3C** to obtain a hash function with ideal security, despite the corresponding **MD**-style hashes being vulnerable to known kinds of MBCA. Further security improvements are obtained by the **3C+** structure, at the cost of some additional internal memory, see Section 7.

6 Security analysis of 3C against generic attacks:

Analysis against Joux attacks:

Joux [10] described a generic multicollision attack on the **MD** hash where constructing 2^d collisions costs d times as much as building ordinary 2-collisions. The multicollision attack can also be used as a tool to find multiple (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 [15], 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 $M^1, \ldots, M^{2^d}$ with $H_d = H(M^1) = \ldots = H(M^{2^d})$ 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 $O(d*2^{t/2}+2^t)$. For the 2^{nd} preimage attack with the given target message M, assign $Y = H(M)$.

Analysis against second-preimage attacks:

Dean [5] has demonstrated that for hash functions with fixed point compression functions, it would cost less than 2^t effort to find second preimages. At EuroCrypt'2005 [13], Kelsey and Schneier 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 of these attacks use the notion of *expandable messages*- patterns of messages of different lengths that all process to internal hash values without considering the length padding (Merkle-Damgård strengthening) at the end. Following [13], 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 does exist a 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 messages [13]. 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 won't help in finding second preimages for less than 2^t work on the **3C** design.

It was demonstrated in [13] 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, where is 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 [13] 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 at 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 size.

6.1 Comparison of 3C with other hash function proposals

Ferguson and Schneier [8] proposed double-hashing scheme $H_{IV}(H_{IV}(x))$ which is basically the NMAC construction [1] with secret keys replaced by the initial states of the hash functions to prevent straight-forward length extension attacks. It is obvious that multi block collision attacks work on this nested construction as effectively as they are on **MD** based hash functions. As on the **MD** hash, $D = 2^d$ collisions can be found on their scheme with a complexity of $O(d.2^{t/2})$. As on the **MD** hash, finding 2^d (2^{nd}) preimages would take time $O(d.2^{t/2} + 2^t)$.

Lucks [15] has proposed wide-pipe and double-pipe hash designs as failure-tolerant functions showing that these designs provide more resistance against these generic attacks than the **MD** hash. The double-pipe hash is a special case of wide-pipe hash function. The wide-pipe, **3C** and the double-hashing proposals resist the straight-forward length extension attacks which is a well-known weakness of the **MD** hash function. Informally, given the digest H of the message M, it is straightforward to compute N and H' such that $H' = H(M||N)$ even for unknown M but for known $|M|$. The attack uses $H(M)$ as the internal hash value to compute $H(M||N)$. All these hash functions provide $t/2$-bit level of security against straight forward extension 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 and **3C** requires processing of at least three compression functions. While the wide-pipe and double-pipe hash functions are designed to provide more resistance against generic attacks, the **3C** is an enhancement to the **MD** hash function resisting the recent multi-block collision attacks on the **MD** based hash functions. In addition, one can combine the wide-pipe hash and the **3C** construction to attain a hybrid construction (see Fig 5) attaining additional protection against both Joux generic attacks and the multi-block collision attacks.

From the performance point of view, **3C** is slightly more expensive than **MD** hash functions 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. On an Intel Pentium 4 3.2GHz processor, **3C** based on the compression of MD5, incurs about 0.36% overhead and **3C** with the compression function of SHA-1 incurs about 0.27% overhead when these functions are used to process large messages. **3C** requires an extra iteration of the compression function similar to the double hashing proposal [8] of Ferguson and Schneier and is as efficient as their scheme for processing messages of at least one block assuming that the computational effort involved in accumulating the

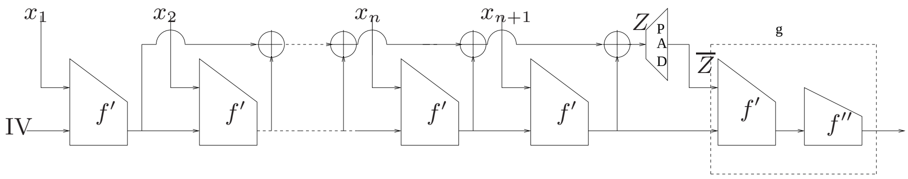

Fig. 5. The **3C** wide-pipe hash hybrid construction

result of XOR function in the *accumulation chain* is negligible. Unlike the double hashing scheme, **3C** is a single hashing scheme.

7 The 3C+ construction: An improved 3C construction

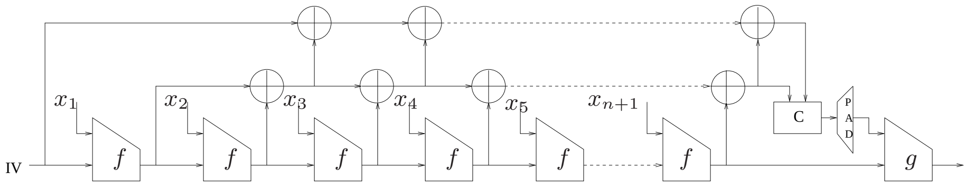

Fig. 6. The **3C+** hash construction

Figure 6 shows the **3C+** construction. Clearly it contains within it both the **MD** and the **3C** structures. A third internal chain has been added after the processing of the second message block. This extra chain is the XOR sum of the accumulation chain from the internal **3C** structure. The final compression function takes as "message" the concatenation of the data in the accumulation and the extra chains, padded with 0 bits to make a block.

Much of the security analysis we have given for **3C** also applies directly to **3C+**. However there is an increased resistance to multi-block collision attacks.

Theorem 3. *The 3C+ construction has ideal security against any MBCA that operates on the underlying **MD**-style structure. Every MBCA on t-bit **3C+** requires effort at least $2^{\frac{t}{2}}$, even if the MBCA on **MD**-style is free.*

Proof: Similar to the proof of Theorem 1. However, the required difference-set collision attack is now conducted upon *two* chains of t-bits each and thus the attacker must repeat the MBCA at least $2^{\frac{2t-Q}{2}}$ times. Let the MBCA attack on the **MD**-style structure have complexity 2^C, then the attacker's total effort is $2^Q + 2^{C+\frac{2t-Q}{2}}$. This is already more than $2^{\frac{t}{2}}$ whenever $Q \geq \frac{t}{2}$. In the case where $Q < \frac{t}{2}$ the attacker's effort is greater than $2^{C+\frac{t}{2}}$ which is greater than $2^{\frac{t}{2}}$ whenever $C \geq 0$. Thus the **3C+** structure has ideal security versus MBCA. □

It is clear from the proof that no further improvement can be made. **3C+** is the simplest structure based on **MD**-style that offers total immunity to MBCA.

8 Conclusion

The recent cryptanalysis of hash functions such as MD5 and SHA-1 exploited the **MD** iterative structure of these hash functions using multi-block collision search techniques. In this paper, we proposed a variant to the **MD** hash construction called **3C** construction which offers more resistance to multi-block collision attacks. We have proved that a multi-block collision attack on **3C** based on f succeeds with a complexity less than $2^{t/2}$ *only if* the multi-block collision attack works on an **MD** hash function with the same f with a complexity less than $2^{t/4}$. Moreover, the **3C+** construction provides provable immunity to all MBCA. While the wide-pipe and double-pipe hash functions proposed by Lucks [15] work as failure tolerant hash functions offering resistance against generic attacks on hash functions, **3C** (resp. **3C+**) is another failure tolerant hash function, this time offering resistance (resp. immunity) against multi-block collision attacks.

The proposed constructions can be implemented by simple adjustments to the existing **MD**-style implementations.

References

1. 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"`.
2. 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.
3. Jean-Sebastien Coron, Yevgeniy Dodis, Cecile Malinaud, and Prashant Puniya. Merkle-damgard revisited: How to construct a Hash Function. In Victor Shoup, editor, *Advances in Cryptology—CRYPTO '05*, volume 3621 of *Lecture Notes in Computer Science*, pages 430–448. Springer, 2005, 14–18 August 2005.
4. Ivan Damgard. 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.
5. Richard Drews Dean. *Formal Aspects of Mobile Code Security*. PhD thesis, Princeton University, 1999.
6. Bert denBoer and Antoon Bosselaers. Collisions for the compression function of MD5. In T. Helleseth, editor, *Advances in Cryptology — Eurocrypt '93*, volume 765 of *Lecture Notes in Computer Science*, pages 293–304, Berlin, 1994. Springer-Verlag.
7. Hans Dobbertin. Cryptanalysis of MD5 compress. Presented at the rump session of Euro Crypto'96 Rump Session, 1996.
8. Niels Ferguson and Bruce Schneier. *Practical Cryptography*, chapter Hash Functions, pages 83–96. John Wiley & Sons, 2003.
9. 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/`.
10. 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.
11. Charanjit S. Jutla and Anindya C. Patthak. Is SHA-1 conceptually sound? Cryptology ePrint Archive, Report 2005/350, 2005. `http://eprint.iacr.org/`.
12. 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/`.
13. John Kelsey and Bruce Schneier. Second Preimages on n-bit Hash Functions for Much Less than 2n̂ Work. In Ronald Cramer, editor, *Advances in Cryptology - EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 474–490. Springer, 2005.
14. Donald E. Knuth. *The Art of Computer Programming*. Addison-Wesley in Computer Science and Information Processing. Addison-Wesley, 1973.

15. 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.
16. Alfred J. Menezes, Paul C. Van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*, chapter Hash Functions and Data Integrity, pages 321–383. The CRC Press series on discrete mathematics and its applications. CRC Press, 1997.
17. 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.
18. Bart Preneel. *Analysis and design of Cryptographic Hash Functions*. PhD thesis, Katholieke Universiteit Leuven, 1993.
19. 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.
20. Michael Szydlo and Yiqun Lisa Yin. Collision-Resistant usage of MD5 and SHA-1 via Message Preprocessing. Cryptology ePrint Archive, Report 2005/248, 2005. `http://eprint.iacr.org/`.
21. Paul C. van Oorschot and Michael J. Wiener. Parallel collision search with cryptanalytic applications. *J. Cryptology*, 12(1):1–28, 1999.
22. 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/`.
23. Xiaoyun Wang, Xuejia Lai, Dengguo Feng, Hui Chen, and Xiuyuan Yu. Cryptanalysis of the Hash Functions MD4 and RIPEMD. In Ronald Cramer, editor, *Advances in Cryptology - EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2005.
24. 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.
25. 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.
26. 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.

A Another way of generating multi-block collisions (instead of using near-collisions)

A special case is the near-collision after first block, but in principle any two different chaining variables could be used to assist in creating a full hash collision after some subsequent message blocks are processed.

So one attack is: Begin with an off-line computation and do steps (1) and (2). They could be done in parallel or in any order, or partly interleaved, etc.... 1) with the known fixed IV as chaining input, collect many pairs of message block input and chaining value output. Sort this list by chaining output and call it "A". 2) collect many triples (input chaining value, message block, output chaining value), sort this list by chaining input and call it "B". Then search for a match in these lists: find "a" from A and "b" from B such that output of element "a" is exactly the input in element "b". Sort the lists by these important data, so the list searching can be done in logarithmic time using well-known list search algorithms (see [14]). If a match is found then (by taking note of the message blocks used in the matching elements) we can instantly construct a 2-(message)-block collision for the two iterated compression functions and by extension attack many other real hash collisions. Given that the compression functions have n-bit chaining variables, then if both list A and list B have size $2^{n/2}$, by Birthday Paradox we should expect to find a match. By using clever algorithms for generating list B (for example distinguished points method [21]) we can have a virtual list size much larger than we can store, at the expense of extra operations to search the list.