A New Scheme for Hash Function Construction

Chi Sing Chum\textsuperscript{1} and Xiaowen Zhang\textsuperscript{2}
\textsuperscript{1} Computer Science Dept., Graduate Center, CUNY
365 Fifth Ave., New York, NY 10016, U.S.A.
Email: cchum@gc.cuny.edu
\textsuperscript{2} Computer Science Dept., College of Staten Island, CUNY
2800 Victory Blvd, Staten Island, NY 10314, U.S.A.
Email: xiaowen.zhang@csi.cuny.edu

Abstract—Most cryptographic hash functions used today, such as MD and SHA families, are iterative hash functions. Although there are many suggestions to improve the security of the iterative hash functions, the general idea of processing the message block by block still enables many attacks, which take advantage of the intermediate hash values. This paper proposes a new scheme for hash function construction that applies randomize-then-combine technique used in the incremental hash functions to the iterative hash function to prevent those attacks.

Keywords: Hash function, incremental hash function, herding and Nostradamus attack, random-then-combine, pair block chaining.

1. Introduction

A cryptographic hash function \cite{17} takes an input string of arbitrary length and generates an output string of fixed length, called message digest, or hash value, or simply hash. Most currently used hash functions are based on Merkle-Damgård construction \cite{6}, \cite{13}. Hash functions have many information security applications, such as digital signatures, message authentication codes, and authentication protocols.

As we know, the applications of hash functions depends on certain properties: fast, preimage, second preimage, collision resistance. However, recent weakness of SHA-1 has been founded. In 2005, Wang, Yin and Yu \cite{19} showed that an attack on full SHA-1 can be done in \(2^{63}\) operations which is less than the ideal situation \(2^{80}\) by birthday attack. Because of this, two cryptographic hash workshops \cite{5}, \cite{14} were held to discuss this attack, improvements of existing SHA-1, other short and long term options. The workshops also encouraged more hash function researches. Many suggestions were proposed, such as replacing SHA-1 by SHA-2, using double pipelining, randomizing hashing, and improving Merkle-Damgård construction by adding two parameters, i.e., the number of bits hashed so far and a salt. However, the general idea of processing the message block by block still leaves chances for many attacks, which make use of the intermediate hash values. In this paper we propose the consideration of using randomize-then-combine technique as that used in the incremental hash functions to prevent those attacks. We combine an incremental hash function with an iterative hash function to prevent different attacks, which are based on the intermediate hash values.

The rest of paper is organized as follows. In Section 2, we discuss iterative hash function and incremental hash function. In Section 3, we introduce our new scheme for constructing hash function, followed by the analysis of the complexity and advantages of the scheme. Section 4 explains the reasons why the new scheme makes various attacks, which are based on intermediate hash values, much more difficult, if not impossible. Section 5 concludes the paper.

2. Related Work

2.1 Iterative Hash Function

Most of the cryptographic hash functions used today are iterative. The design of these hash functions basically follows the Merkle-Damgård construction \cite{6}, which is a common method for constructing a hash function by repeating the underlying compression function.

Let \(H\) be an iterative hash function and \(C\) be its underlying compression function. The compression function \(C\) maps a longer fixed-length input to a shorter fixed-length output, defined as: \(C : \{0,1\}^s \rightarrow \{0,1\}^t\), where \(s > t > 0\).

After the necessary padding, any message \(M\) of arbitrary length will be made as a multiple of a pre-determined fixed block size \(s\). The message \(M\) will then be divided into blocks and fed to the compression function one block at a time until all the blocks are processed.
Let \( M = m_1 \| m_2 \| \ldots \| m_n \), and \( h_i = C(h_{i-1}, m_i) \), where \( i = 1 \) to \( n \). Let \( h_0 = IV \), the initial vector; \( h_n = h \), the final hash. All the \( m_i \) are of length \( s \), and all \( h_i \) are of length \( t \). \( C(h_{i-1}, m_i) = h_i \) means that the compression function \( C \) takes a message block \( m_i \) and an intermediate hash \( h_{i-1} \) as input and generates a new intermediate hash \( h_i \) as output. \( H(IV, M) = h \) means that the hash function \( H \) takes a message \( M \) as input and after a series of applications of its underlying compression function \( C \), outputs a final hash \( h \). For simplicity, we write it as \( H(M) = h \), instead.

The iterative hash function will process messages of arbitrary length up to a limit (a very large number like \( 2^{64} \)) rather than messages of fixed length as in the case of a compression function.

### 2.2 Incremental Hash Function

One drawback for an iterative hash function is that whenever there is a change in the message, no matter how small it is, we have to re-calculate the new hash for the updated message starting from the scratch. Certainly, this is not efficient to hash a lot of messages having similar contents.

Bellare, Goldreich, and Goldwasser [1] introduced incremental hash function. An incremental hash function is designed in such a way that the time to re-calculate the new hash of an updated message is directly proportional to the number of changes and independent of the message length. So, an incremental hash function is suitable for situations where the next hash message is slightly different from the previous one. We calculate the new hash based on the old hash and the number of changes. There is no need to go through the whole message again.

**a) Randomize-then-combine** was proposed by Bellare and Micciancio [4]. Each message block will be input into a random function \( R \), and all the outputs from \( R \) will be combined together by a combination operation \( \ast \) to get the hash \( h \).

If \( M \) is the message and let \( M = m_1 \| m_2 \| \ldots \| m_n \). Then we have

\[
\begin{align*}
    h_1 &= R(<1> \| m_1), \\
    \ldots \ldots \ldots \ldots , \\
    h_i &= R(<i> \| m_i), \\
    \ldots \ldots \ldots \ldots , \\
    h_n &= R(<n> \| m_n),
\end{align*}
\]

where \(<i>\) is the binary representation of the message block index and \(\|\) is the concatenation.

The concatenation of binary representation of the message block index with message block prevents creation of collision messages from rearranging the message blocks.

After the randomize-step as above, all the elements \( h_1, h_2, \ldots \) will be combined by the combination operation \( \ast \).

\[
h = h_1 \ast h_2 \ast \ldots \ast h_n
\]

**b) Pair block chaining** was introduced in the recent designs of incremental hash functions. Bellare, Goldrick, and Goldwasser [2] introduced the incremental XOR scheme. Goi, Siddiqi, and Chuah [8] proposed incremental hash function based on pair chaining and modular arithmetic computing, and did an analysis on the complexity and implementation aspects in [9]. Phan and Wagner [15] discussed security considerations for incremental hash functions based on pair block chaining. Two subsequent blocks are concatenated together and input into the random function \( R \), which outputs an randomized string of the hash size. All the outputs from \( R \) will be combined together by a combination operation \( \ast \) to get the hash \( h \). Let

\[
\begin{align*}
    h_1 &= R(m_1 \| m_2), \\
    \ldots \ldots \ldots \ldots , \\
    h_i &= R(m_{i-1} \| m_i), \\
    \ldots \ldots \ldots \ldots , \\
    h_{n-1} &= R(m_{n-1} \| m_n),
\end{align*}
\]

then \( h = h_1 \ast h_2 \ast \ldots \ast h_{n-1} \).

If the message blocks are cyclically chained under the scheme, an extra step for calculation of \( h_n = R(m_n \| m_1) \) is needed. In this case \( h = h_1 \ast h_2 \ast \ldots \ast h_n \). In either case, \( a) \) or \( b) \), \( h_1, h_2, \ldots \) are elements of a group \( G \) with the group operation \( \ast \), which is assumed to be hard to solve.

### 3. A New Scheme for Hash Function Construction

#### 3.1 Scheme Description

Many attacks on an iterative hash functions are based on the series of intermediate hash values. The main reason is that the processing of blocks is always forward, block by block until the end. Here, we suggest a new scheme which uses the random-then-combine as that in the incremental hash functions, to improve the security of an iterative hash
Let the message $M = m_1||m_2||\ldots||m_n$. Instead of the usual iteration as $h_i = C(h_{i-1}, m_i)$, we proceed as follows:

$$p_i = R(m_{i-1} \ast R(m_i)), \text{ and } h_i = C(h_{i-1}, p_i),$$

where $i = 1$ to $n$, $h_0 = IV$ the initial vector, $h_n = h$ the final hash, $m_0 = m_n$ and $m_i, p_i$ are of the same block size.

$R$ is a random function as the one in an incremental hash function. It takes two blocks of messages as input and returns a randomized string of the same block size (propagation may be necessary), which will then input into the compression function $C$. We consider the case that both $R$ and $C$ take the blocks of the same size here.

Two message blocks will be processed in either one of the following two ways depending on the incremental hash function we choose:

$$p_i = R(m_{i-1} \ast R(m_i)), \text{ or } p_i = R(m_{i-1}||m_i).$$

1) We consider $m_0 = m_n$, the last block of the message.
2) In our suggested new scheme, binary representation of the message block index is not necessary. The same applies to other settings which prevent message substitution and make the incremental hash function collision free.
3) If we use pair block chaining, combination operation is not necessary.
4) When we mention randomize-then-combine, it might imply pair block chaining as well.

### 3.2 Comparison of the Traditional Iterative Hash Function and the New Scheme

Let $M = m_1||\ldots||m_{i-1}||m_i||\ldots||m_n$; $M' = m_1'||\ldots||m_{j-2}'$. $H$ is the iterative hash function, $C$ the compression function of $H$, and $R$ the random function in the new scheme.

#### 3.2.1 Traditional Iterative Hash Function:

If we can find a message block $m^*$, such that $C(h'_{j-2}, m^*) = h_{i-1}$ (one direction, forward), then $M$ and $M^* = M'||m^*||m_i||\ldots||m_n$ are colliding pairs of messages, see Fig. 1. In other words, if the hash of any message equals to an intermediate hash of an other message, then we can construct a pair of colliding messages. This is due to the fact that each message block is processed only once and always forward.

![Figure 1: Construction of colliding messages - $M$ and $M^*$](image)

#### 3.2.2 The New Scheme:

Unlike the traditional case, it is not enough to find $m^*$ such that $C(h'_{j-2}, R(m'_{j-2}, m^*)) = h_{i-1}$, because $C(h_{i-1}, R(m_*), m_i))$ is not necessary equal to $h_i$. Instead, we need to find a message block $m^*$, such that

$$C(h'_{j-2}, R(m'_{j-2}, m^*)) = h'_i \text{ (forward, } m^* \text{ before } m'_{j-2}),$$

$$C(h'_i, R(m^*, m_i)) = h_i \text{ (backward, } m^* \text{ after } m_i)$$

then $M$ and $M^* = M'||m^*||m_i||\ldots||m_n$ are colliding pair of messages. See Fig. 1. Here we need to find such a message block $m^*$ that satisfies both conditions, one forward and one backward, since each message block is processed twice. Note that $C(IV, R(m'_{0}, m'_1)) = h'_1$, and $m'_0 = m_n$ as required under the new scheme.

### 3.3 Complexity Analysis

Suppose we have the following intermediate hashes for message $M = m_1||m_2||\ldots||m_n$.

$$h_i = C(h_{i-1}, R(m_{i-1}, m_i))$$

and

$$h_{i+1} = C(h_i, R(m_i, m_{i+1}))$$

We say an attacker breaks the scheme if he can find a message $M'$ with $m'$ and $h'$ as the last message block and hash such that $h_{i+1} = C(h', R(m', m_{i+1})$. Here, the first iteration of $M'$ is using $m'_0 = m_n$. Based on this, we estimate the complexity and hence the security level of the proposed scheme.
Let us consider the complexity, in general, for finding such \( m' \) under the assumption that the output of \( R \) is random, and \( C \) is a compression function of an iterative hash function with hash size \( n \) and message block size \( b \).

Let \( R(m', m_{i+1}) = p_{i+1} \). The attacker takes the following steps.

1. Just like finding a second pre-image, it takes \( 2^n \) steps to find \( p_{i+1} \) so that \( C(h', p_{i+1}) = h_{i+1} \).
2. Then he needs to find the inverses of \( p_{i+1} \). Suppose the inverses of \( p_{i+1} \) are \( m'' \) and \( m''' \). If \( R \) is one-way function, it will be difficult to get back the inverses. The complexity of this steps depends on \( R \).
3. He needs to check if \( m'' = m' \) and \( m''' = m_{i+1} \). If not, he has to go back to step 2 to look for another set of inverses. If he can not find such a set of inverses that satisfies the above conditions, he needs to go back to step 1 to try again for another \( m' \).

The security level of the scheme increases and is more difficult to break.

### 3.4 Advantages

The following are advantages of the proposed new scheme:

1. This applies to any iterative hash function.
2. We can choose any secure hash function \( R \) for the implementation. We can replace it if another new and better one becomes available.
3. The operations can be overlapped. We can make use of the parallelizability of an incremental hash function so not to slow down the processing time of the iterative hash function very much.

### 4. Attacks Prevented under the New Scheme

In this section, we discuss how this new scheme can prevent different attacks that make use of intermediate hashes.

#### 4.1 Multi-collisions Attack

Joux [10] showed that it would be much easier to find multicollisions in an iterative hash function based on intermediate hash values. Trappe and Washington [18] has detailed descriptions on this. Suppose we have an iterative function \( H \) and its associated compression function \( C \). The output of \( H \), and hence \( C \), is \( n \) bits. If we apply birthday attack to the first blocks of messages, in approximately \( 2^{n/2} \) steps we can find two blocks \( m_0 \) and \( m'_0 \) such that \( C(h_0, m_0) = C(h_0, m'_0) \), where \( h_0 = IV \) the initial value of \( H \). Let \( h_1 = C(h_0, m_0) \). Similarly, in approximately \( 2^{n/2} \) steps we can find two blocks \( m_1 \) and \( m'_1 \) such that \( C(h_1, m_1) = C(h_1, m'_1) \). We repeat this process until we get \( t \) pairs of blocks \( (m_0, m'_0), (m_1, m'_1), \ldots, (m_{t-1}, m'_{t-1}) \) such that \( h_0 = IV \) initial value of \( H \), and

\[
h_i = C(h_{i-1}, m_{i-1}) = C(h_{i-1}, m'_{i-1}), \quad 1 \leq i \leq t.
\]

By all possible combinations of these \( t \)-pair blocks, two choices for each pair, we can build up \( 2^t \) messages as follows (see Fig. 2):

\[
\begin{align*}
&\overline{m_0} \overline{m_1} \overline{m_2} \ldots \overline{m_{t-1}}, \\
&\overline{m'_0} \overline{m'_1} \overline{m'_2} \ldots \overline{m'_{t-1}}, \\
&\ldots \ldots \ldots \ldots
\end{align*}
\]

If these \( 2^t \) messages are input into \( H \), it is not difficult to see they have the same hash. That means we have a \( 2^t \)-collision. This process takes approximately \( t \times 2^{n/2} \) operations. So, it is much easier to find multi-collisions in an iterative hash function.

First, it would be difficult to find pairs of collisions under the new scheme. Second, by mimicking the above even the attacker can find the pairs of collisions as follows:

\[
\begin{align*}
C(IV, R(m_n, m_1)) &= h_1 \\
C(h_1, R(m_1, m_2)) &= h_2 \\
C(h_2, R(m_2, m_3)) &= h_3 \\
\ldots \ldots \ldots & \ldots \ldots \ldots
\end{align*}
\]

But it is not necessary the following is true:

\[
C(h_i, R(m_i, p_{i+1})) = C(h_i, R(p_i, m_{i+1})).
\]

Even he repeats the above process \( k \) times to get \( k \) pairs of colliding blocks, this scheme prevents him from building a set of \( 2^k \) messages.
4.2 Long Message Attack

Let \( H \) be an iterative hash function and \( C \) be the corresponding compression function. Suppose that \( M \) is a message, and it is broken into blocks as \( m_1 || m_2 || \ldots || m_n \). Then we have

\[
\begin{align*}
h_1 &= C(h_0, m_1), \\
h_2 &= C(h_1, m_2), \\
& \quad \ldots \ldots \\
h_n &= C(h_{n-1}, m_n).
\end{align*}
\]

Again here \( h_0 \) is the initial vector and \( h_n \) is the final hash.

Let \( M' \) be a message with the hash equals to one of the intermediate hash, say \( h_i \). Then an attacker can construct a new message \( M'' = M'||m_{i+1} \ldots m_n \) and he has a collision pair of messages \( M \) and \( M'' \) with the same hash value. Under long message attack \([11]\), there are many intermediate hashes available and the chance of looking for a match is greater and hence the time for the attack is shorter.

As shown in 3.2 and 3.3, the new scheme makes the long message attack more difficult.

4.3 Expandable Message with Fixed Point

Long message attack can be prevented by Merkle-Damgård strengthening, i.e., adding the length of the original message to the last block of the padded message. However, Dean \([7]\) pointed out that if it is easy to find fixed points in the underlying compression function of the hash function, we can use fixed point to build an expandable message to bypass the Merkle-Damgård strengthening.

An expandable message is group of colliding messages with different length. An \((a, b)\) expandable message is a group of colliding messages which cover all messages of the length between \(a\) and \(b\) message blocks. And a fixed point \((h, m)\) is defined as: \( C(h, m) = h \). That means we can repeatedly process the block \(m\) without changing the hash \(h\).

The idea to build an expandable message using fixed points is as follows \([11]\).

1) Find a list of \(2^{n/2}\) fixed points: \((h_1, p_1), (h_2, p_2), (h_3, p_3), \ldots \) such that \( C(h_i, p_i) = h_i, \ i = 1, \ldots, n/2\).

2) Build a list of \(2^{n/2}\) hash values which can be reached from the initial vector: \( H(IV, m_1) = h'_1, H(IV, m_2) = h'_2, \ldots \) such that \( H(IV, m_i) = h'_i, \ i = 1, \ldots, n/2\).

3) \(2^{n/2}\) different hash values set up in 1. \(2^{n/2}\) different hash values generated in 2. We expect \(2^{n/2}(2^{n/2}/2^n) = 1\) collision.

4) Look for a fixed point \((h_i, p_i)\) such that its hash equals to one of the hash values in step 2, say \(h'_j\).

5) We can then expand the message \(m_j\) to a desired length by repeating the compression function to the fixed point \((h_i, p_i)\). And the expanded message will be \(m_j||p_i||p_i|| \ldots ||p_i\).

Under the new scheme, let \((h, R(p_1, p_2))\) be the fixed point and \(h\) be a hash value that can be reached from the initial vector. Then we have \(C(h, R(p_1, p_2)) = h\) and \(H(IV, m) = h\).

Let the last two blocks of \(m\) be \(m_1\) and \(m_2\). If we want to append to the message \(m\) with the fixed point, the combination scheme will take \(m_2\) and \(p_1\) as input. But \(C(h, R(m_2, p_1))\) does not necessarily equals to \(h\) again. In this case, an attack that uses an expandable message with fixed point can be prevented.

4.4 Expandable Message with Multi-collisions

If it is not easy to find fixed points, Kelsey and Schneier \([11]\) extended Joux’s idea to generate a full set of colliding messages to cover a range of length. Suppose we want to build a full \((k, k+2^k-1)\) expandable message. We proceed as follows:

1) We find a colliding pair of messages, one consists of one block and the other consists of \(2^{k-1} + 1\) blocks, using the initial hash.

2) We find a collision pair of messages, one consists of one block and the other consists of \(2^{k-2} + 1\) blocks, using the hash from the previous step.

3) We continue this process until we get a pair of messages of one block and the other consists of two blocks respectively.

Now, we have \(k\) pairs of message components which can be used to generate all the messages with number of blocks from \(k\) to \(k+2^k-1\).

So, expanding the message by length \(L\) such that \(k <= L <= k+2^k-1\) is just by writing \(L\) in binary and depending on the bits on and off, we choose the appropriate message component. Starting from the most significant bit, for example, if the first bit is on, we choose the component with \(2^{k-1} + 1\) blocks; otherwise we choose the component with 1 block. If the second bit is on, we choose the component with \(2^{k-2} + 1\) blocks; otherwise we choose the component with 1 block, and so on.

The proposed new scheme can prevent this or at least make it more difficult by the same arguments as in the multi-collisions attack in 4.1.
4.5 Herding Hash Functions and Nostradamus Attack

This method also makes use of the intermediate hash values which all can go to the same final hash, say $h$. Kelsey and Kohno [12] have a detail analysis of this attack. Stevens, Lenstra and Weger [16] applied this technique to predict the winner of the 2008 US Presidential Elections using a Sony PlayStation 3 in November 2007. They claimed that they have correctly predicted the next US president, and committed the hash of the result to the public. And the correct prediction and the matching hash will be revealed after the election.

For launching the Nostradamus attack [12], a diamond structure has to be built. The first step is to build a large set of intermediate hashes at the first level: $h_{11}, h_{12}, \ldots, h_{1n}$. The second step is to build a set of intermediate hashes at the second level: $h_{21}, h_{22}, \ldots, h_{2n/2}$ so that the followings are satisfied:

- There exists a message block $m_{11}$ such that $C(h_{11}, m_{11}) = h_{21}$.
- There exists a message block $m_{12}$ such that $C(h_{12}, m_{12}) = h_{21}$.
- There exists a message block $m_{13}$ such that $C(h_{13}, m_{13}) = h_{22}$.
- There exists a message block $m_{14}$ such that $C(h_{14}, m_{14}) = h_{22}$.
- \ldots \ldots \ldots \ldots \ldots

By repeating this process, message blocks are linked so that each intermediate hash at any level can reach the final hash, say $h$. Now we create the diamond structure (see Fig. 3). We claim we can predict something, like US Presidential Election result, happens in the future by announcing the hash of the event result to the public. When the result is available, we construct a message as follow:

\[
M = M_{\text{Prefix}} || m_{\text{Link}} || M_{\text{Suffix}}
\]

$M_{\text{Prefix}}$ contains the result that we claimed we knew beforehand. $m_{\text{Link}}$ is a message block which can link the $M_{\text{Prefix}}$ to one of the intermediate hash at level 1. $M_{\text{Suffix}}$ is the rest of message blocks which linked the $m_{\text{Link}}$ to the final hash.

Under the new scheme, we need to find a link block $m_{\text{Link}}$ such that

\[
C(h_P, R(m_P, m_{\text{Link}})) = h_P and
C(h_P, R(m_{\text{Link}}, m_S)) = h_{11},
\]

where $h_P$ is the hash of $M_{\text{Prefix}}$, $m_P$ is the last block of $M_{\text{Prefix}}$, $m_S$ is the first block of $M_{\text{Suffix}}$, and $h_{11}$ is one of the intermediate hashes at level 1 in the diamond structure.

This will be more difficult to look for such $m_{\text{Link}}$ compared to the original iterative hash function situation as in the case of long message attack in 4.2.

5. Conclusions

Iterative hash function generates an intermediate hash after each application of the underlying compression function. Many attacks take advantage of this property as we have seen. Once a match to any intermediate hash is found, we can set up a starting point to link to the rest of the original message to set up another modified message with the same final hash. Even we append the message length at the end of the original message, as Merkle-Damgård strengthening, an expandable message can easily bypass this enhancement.

The proposed scheme can prevent these attacks by applying a randomize-then-combine technique as in an incremental hash function to an iterative hash function. An iterative hash function is always processing forward, and this proposed scheme can fix this weakness. It can thus prevent those attacks as mentioned above, and increases the overall security of the hash function.

Acknowledgments

Authors would like to express their gratitude to Professor Michael Anshel for his valuable suggestions. Xiaowen Zhang was supported in part by a PSC-CUNY Research Award.
References


