CHAPTER 3
MDA-192: A 192-BIT SERIAL STRUCTURED HASH FUNCTION
BASED ON MD CONSTRUCTION

Several attacks have been reported against the SHA-family based hash function due to the weaknesses in the expansion mechanism [129, 137, 138, 140, 144, 146]. The design goal of the new hash function is to improve the algorithmic process of SHA-1 such that the resultant hash function can resists against attacks and provide high level of security. In this chapter we propose a new serial structured dedicated hash function MDA-192 that encodes a message of arbitrary length into a hash value with a fixed 192-bit length. MDA-192 is based on the design principle of SHA-1 [98]. It applies the Merkle-Damgård (MD) paradigm [14, 15] to a dedicated compression function. MDA-192 takes a 512-bit message block divided into 16 32-bit words and expands them into 96 32-bit words by the means of modified message expansion function of SHA-1. The compression function has four rounds and each round of it consists of 24 steps. New message expansion, data dependent rotations and the increased use of input message words in the step operations are the outstanding features of the proposed hash function.

The rest of the chapter is organized as follows: In Section 3.1 we give the short description of SHA-1 algorithm and attacks on it. In Section 3.2, we present design choices made for MDA-192 hash function. In Section 3.3 formal design description of proposed hash function is provided. In Section 3.4 we discuss about differential collision characteristics of modified message expansion. In Section 3.5 indifferentiality of compression function of MDA-192 is presented. Finally, in Section 3.6, we conclude the chapter.

3.1 A Short Description of SHA-1 Algorithm

SHA family is the most famous design principle for dedicated hash functions. The SHA-1 hash function was specified in 1995 by the U.S. National Security Agency as a successor of SHA-0 [97]. Since its publication, SHA-1 has been adopted by many government and industry security standards, in particular, standards on digital signatures for which a collision resistant hash function is required. In addition to its usage in digital signatures, SHA-1 has also been deployed
as an important component in various cryptographic schemes and protocols, such as user authentication, key agreement, and pseudorandom number generation. Consequently, SHA-1 has been widely implemented in almost all commercial security systems and products.

SHA-1 computes a 160-bit hash of messages up to $2^{64}$ bits in size. Each message needs to be pre-processed by padding the message, appending the message length and splitting it into blocks. So padded message $M$ is divided into several messages $M_k$ each 512-bit long ($M = M_1 \parallel M_2 \parallel \ldots \parallel M_n$). The message is padded to realize a multiple of 512 bits. These divided messages are input to the compression function. Then the compression function processes each input block and computes intermediate hash values by iterating over compression function 80 times. It has two types of inputs: a chaining input of 160-bit and a message input of 512-bit. The initial chaining value is a set of fixed constants and the result of the last call to the compression function is the hash of the message. The structure of the compression function of SHA-1 is given in following steps:

(i) Divide the input message block into 32-bit message sub-blocks ($W_0, W_1, \ldots, W_{15}$).

(ii) Calculate $W_{16}$ to $W_{79}$ by a relation

$$W = (W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16}) \ll 1, \text{for } t = 16 \text{ to } 79.$$  

(iii) Calculate chaining variables $(A_t, B_t, C_t, D_t, E_t)$ in step $t$ by the following procedures.

$$A_t = (A_{t-1} \ll 5) + f(B_{t-1}, C_{t-1}, D_{t-1}) + W_{t-1} + k_{t-1},$$

$$B_t = A_{t-1}, C_t = B_{t-1} \ll 30, D_t = C_{t-1}, E_t = D_{t-1}.$$  

(iv) $(A_0 + A_{80}, B_0 + B_{80}, C_0 + C_{80}, D_0 + D_{80}, E_0 + E_{80})$ is the output of the compression function.

Symbol “$\ll l$” denotes left cyclic shift by $l$ bits. The above process of (ii) and (iii) is repeated 80 times. Initial values $(A_0, B_0, C_0, D_0, E_0)$ for the compression function of the first block are the initial values of SHA-1. $(A_0, B_0, C_0, D_0, E_0)$ for the compression function of the second block are the output values of the previous block. Steps 0-19 are called the Round 1. Steps 20-39, 40-59, and 60-79 are Round 2, Round 3 and Round 4, respectively. $k_i$ is a constant defined in each round.

Function $f_i$ is a Boolean function defined in each round as shown in Table 3.1.
Table 3.1: Boolean functions and constants used in SHA-1.

<table>
<thead>
<tr>
<th>step number $t$</th>
<th>$f_t$</th>
<th>$k_t$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-19 (Round -1)</td>
<td>$(B \land C) \lor (\neg B \land D)$</td>
<td>0x5A827999</td>
</tr>
<tr>
<td>20-39 (Round -2)</td>
<td>$(B \oplus C \oplus D)$</td>
<td>0x6ED9EBA1</td>
</tr>
<tr>
<td>40-59 (Round -3)</td>
<td>$(B \land C) \lor (B \land D) \lor (C \land D)$</td>
<td>0x8F1BBCDC</td>
</tr>
<tr>
<td>60-79 (Round -4)</td>
<td>$(B \oplus C \oplus D)$</td>
<td>0xCA62C1D6</td>
</tr>
</tbody>
</table>

In the past few years, there have been significant research advances in the analysis of hash functions. No weakness has been found in the SHA-1 until 2005, when Wang, Yin, and Yu presented astonishing collision attacks [144]. This function is still in use. Chabaud and Joux [137] published the first theoretical collision attack against first member of this family, SHA-0 and Biham and Chen [138] introduced the idea of neutral bits, which led to the computation of a real collision with four blocks of message [139]. Then, a novel framework of collision attack, using modular difference and message modification techniques, surprised the cryptography community [120, 129, 140, 144]. This necessitates the need for design of more powerful hash function.

In this proposal the size of intermediate state and final hash value increased by 32-bits, which results into more computational efforts for brute force attack. In addition to this uses of modified message expansion, increased non linear step operations and data-dependent rotations decrease collision probability by a factor of $2^{-280}$.

### 3.2 Design Choices for MDA-192 Hash Function

Most modern hash functions are the product of the combination of a compression function, hashing a small number of bits into a smaller number, and of a mode of operation, describing how the compression function should be used to process arbitrarily big messages. The MDA-192 algorithm accepts as input a message with a maximum length of $2^{64}$-1 and produces a 192-bit message digest as output. It follows Merkle-Damgård iterative structure as a mode of operation. The compression function takes as inputs one message block and a 192-bit chaining variable. The output is the 192-bit updated value of chaining variable. MDA-192 uses six 32-bit chaining variables to which new values are assigned in each round. The compression function processes one 512-bit message block per iteration. The compression function consist of four rounds, each
round is made up of a sequence of 24 steps. The message word used in each step is derived through a message expansion. Message expansion is applied to each 512-bit message block, where 16 32-bit message words are expanded to 96 32-bit message words.

(i) **Mode of Operation**

The MDA-192 uses most popular and well known mode of operation the Merkle-Damgård construction, introduced in 1989 and named after its two independent inventors [14, 15]. One of its distinctive features is that it promotes the collision resistance and preimage resistance of the compression function to the full hash function. Formally it can be defined as follows:

Let \( f : \{0,1\}^b \times \{0,1\}^n \rightarrow \{0,1\}^n \) be a compression function which takes a \( b \)-bit message block and an \( n \)-bit chaining value. Let \( h : \{0,1\}^n \rightarrow \{0,1\}^n \) be a MD construction built by iterating the compression function \( f \) in order to process a message of arbitrary length. A message \( M \) to be processed using \( h \) is always padded in a manner such that the length of the padded message is a multiple of the block length \( b \) of \( f \). Bit-length \( b \) corresponds to input length of desired compression function \( f \). The padding is done by adding after the last bit of the last message block a single 1-bit followed by the necessary number of 0-bits. Let \( |M| \) be a binary representation of the length of the message \( M \). The binary encoding of the message length is also be added to complete the padding. This is called a Merkle-Damgård strengthening. This inclusion of the length at the end of the message is important for preventing a number of attacks, including long message attacks. Merkle-Damgård construction proves that the security of hash function relies on the security of the compression function. Thus, in order to build a collision resistant hash function, it is sufficient to design a collision resistant compression function. Then input \( M \) subsequently divided into \( t \) blocks \( M_1, M_2, \ldots, M_t \) each of bit-length \( b \). The hash function \( h \) can then be described as follows:

\[
H_0 = IV,
\]

\[
H_i = f(H_{i-1}, M_i) \; \text{for} \; i = 1,2,\ldots,t, \text{and} \; h(M) = H_t.
\]  \( \ldots(3.3) \)
Where $f$ is the compression function of $h$, $H_i$ is the intermediate chaining variable between stage $i-1$ and stage $i$, and $H_0$ is a pre-defined starting value or the initial value $IV$.

Figure 3.1: Merkle-Damgård construction.

The compression function is iterated using Merkle-Damgård construction shown in the Figure 3.1.

Figure 3.2: Detailed view of Merkle-Damgård construction.
The computation of the hash value is dependent on the chaining variable. At the start of hashing, this chaining variable has a fixed initial value (IV) which is specified as part of the algorithm. This process continues recursively, with the chaining variable being updated under the action of different part of the message, until the entire message has been processed. The final value of the chaining variable is then output as the hash value corresponding to that message. Figure 3.2 shows detailed view of MD construction.

(ii) Compression Function

The compression function takes 512-bit blocks for processing, so it is necessary to pad the variable length message. The padding procedure follows same principle as of MD4-based hash function padding rule. The padded message is divided into block of 512-bit each, $M = M_1, M_2, \ldots, M_N$.

The compression function of MDA-192 is composed of four round processing of 96 steps. Each round consists of 24 steps. The four rounds are structurally similar to one another with the only difference that each round uses a different Boolean function $f_t (0 \leq t \leq 95)$, and one of four different additive constants $k_t (0 \leq t \leq 95)$ which depends on the step under consideration. Each Boolean function is operating on three words and producing one word as output. Boolean functions and additive constants are exactly the same as the Boolean functions and constants used in SHA-1 algorithm as shown in Table 3.2.

<table>
<thead>
<tr>
<th>step number $t$</th>
<th>$f_t$</th>
<th>$k_t$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-23 (Round -1)</td>
<td>$(B \land C) \lor (\neg B \land D)$</td>
<td>0x5A827999</td>
</tr>
<tr>
<td>24-47 (Round -2)</td>
<td>$(B \oplus C \oplus D)$</td>
<td>0x6ED9EBA1</td>
</tr>
<tr>
<td>48-71 (Round -3)</td>
<td>$(B \land C) \lor (B \land D) \lor (C \land D)$</td>
<td>0x8F1BDC</td>
</tr>
<tr>
<td>72-95 (Round -4)</td>
<td>$(B \oplus C \oplus D)$</td>
<td>0xCA62C1D6</td>
</tr>
</tbody>
</table>

A 192-bit ($6 \times 32$-bit) buffer consists of six variables $(A, B, C, D, E, F)$ known as chaining variables is used to hold intermediate results of the compression function and then final result of the MDA-192 hash algorithm. The initial values of chaining variables $(A, B, C, D, E)$ are exactly
the same as the five initial chaining variables used in SHA-1. The added sixth new chaining variable used here is initialized by first 32 bits of fractional parts of square root of sixth prime number. These values of chaining variables along with first 512-bit message block are used to input the compression function for starting the process. The chaining variables are updated in a single step operation is as shown in equation 3.4.

\[
T = ((A << 5) + f_i(B, C, D) + F + W_t + k_t) << (W_t \mod 32) \\
F = E \oplus D \\
E = D \oplus C \\
D = C \oplus B \\
C = B << 30 \\
B = A \oplus (W_t << 15) \\
A = T
\] ...(3.4)

The complete step operation of compression function is represented in Figure 3.3, where \( S1, S2, S3 \) and \( S4 \) shows the left shift of a 32-bit word by \((W_t \mod 32), 5, 15 \) and 30 respectively.

\[i\]

Figure 3.3: Step operation of MDA-192.
A distinguishing feature of the proposed compression function is data-dependent rotations [189]. While the existing MD family hash functions have fixed values in the rotation of step operation, our scheme uses variable number of times bit rotations. Bit rotations frequency is dependent on the input messages. Thus, hash results are more strongly dependent on the input messages. Due to these strong data dependent rotations the attack by den Boer, Bosselaers [116] and Dobbertin [118] is not directly applicable to this hash algorithm. Also this feature may help differential cryptanalysis to be weak, since bit rotations are processed randomly. It is conjectured that the best way to find collision pairs is by using the birthday attack.

(iii) Message Expansion

The SHA-1 hash function has been undermined by its poor message expansion. It is observed independently in [143, 146] that the interleaving process in SHA-1 is not quite good. To understand this let us rewrite the equation 3.1 of SHA-1 as:

\[ W_i = W_{r+2} \oplus W_{r+8} \oplus W_{r+13} \oplus (W_{i+16} >> 1), t = 0 \ldots 63 \]  

\[ \text{(3.5)} \]

It shows that a difference created in the last 16 words propagates to only up to 4 different bit positions. It is shown in [149] that the modified expansion code of SHA-1 as shown in equation 3.6 is the remedy of this problem. In this expansion code the spread of differences to the neighboring bits is more frequent.

\[ W_i = (W_{r-3} \oplus W_{r-8} \oplus W_{r-16}) <<< 1 \oplus W_{r-14}, t = 16 \ldots 79. \]  

\[ \text{(3.6)} \]

This indicates that, meaningful selection of coefficients of the parity check equations can generate the minimum distance large between message words (keys). A useful heuristic that is often used in the analysis of SHA-0 and SHA-1 is that each bit difference in the key lowers the probability of success on an average by a factor of \(2^{-2.5}\). Hence according to this, new modified message expansion mechanism could produce good differential collision characteristics and could leads to a compression function which is virtually unbreakable in time better than that required by a birthday attack.

For MDA-192 the message keys \((W_{16}, W_{17}, \ldots, W_{95})\), are expanded from the relation shown in equation 3.7. The relation is divided in two parts for better control on bit difference propagation in forward direction of message expansion. An extra term \(W_{r-20}\) is included in the second relation.
for the purpose. The minimum distance in the last eighty words is estimated to at least 112 with
the proposed modified message expansion. Thus, we expect our proposed compression function
to have a differential collision characteristic of probability close to $2^{-2.5\times 112}$.

$$
\begin{align*}
W_{r-3} \oplus W_{r-8} \oplus W_{r-14} \oplus W_{r-16} \oplus (W_{r-1} \oplus W_{r-2} \oplus W_{r-15}) &\lll 13, \\
W_{r-3} \oplus W_{r-8} \oplus W_{r-14} \oplus W_{r-16} \oplus (W_{r-1} \oplus W_{r-2} \oplus W_{r-15} \oplus W_{r-20}) &\lll 13,
\end{align*}
$$

for $t = 16, \ldots, 51$

$$
\begin{align*}
W_{r-3} \oplus W_{r-8} \oplus W_{r-14} \oplus W_{r-16} \oplus (W_{r-1} \oplus W_{r-2} \oplus W_{r-15} \oplus W_{r-20}) &\lll 13, \\
W_{r-3} \oplus W_{r-8} \oplus W_{r-14} \oplus W_{r-16} \oplus (W_{r-1} \oplus W_{r-2} \oplus W_{r-15} \oplus W_{r-20}) &\lll 13,
\end{align*}
$$

for $t = 52, \ldots, 95$ \quad (3.7)

The message expansion is linear in nature with respect to XOR operation, for example
$(W_i \oplus V_j)_{i=0}^{95}$ is an expanded message if $(W_i)_{i=0}^{95}$ and $(V_j)_{j=0}^{95}$ are expanded messages. For any 16
consecutive words $(W_i, \ldots, W_{i+15})$ we can uniquely determine the entire expanded message
$(W_0, \ldots, W_{95})$ since message expansion relation is reversible. Also any left rotation, forward or
backward shift in any expanded message will generate another expanded message.

### 3.3 Description of MDA-192 Hash Function

The hash function $h_f : \{0,1\}^* \rightarrow \{0,1\}^n$ is built by iterating a compression function
$f : \{0,1\}^n \times \{0,1\}^m \rightarrow \{0,1\}^n$. The hash process works as follows:

(i) Pad and split a message $M$ into $t$ blocks $(M_1, \ldots, M_t)$ of $m$ bits each.

(ii) Set $H_0$ to the initialization value $IV$ .

(iii) For each message block $M_i$ compute $H_i = f(H_{i-1}, M_i)$.

(iv) Output $h_f(M) = H_t$.

The complete description of algorithm is defined in following steps.

**Step 1: Message Padding and Parsing**

The original message is padded to make its length congruent to 448 modulo 512. The entire
message's length is now a multiple of 512. Then padded message is divided into $N$ blocks of 512-
bit each, $M = M_1, M_2, \ldots, M_N$. 

49
**Step 2: Setting the Initial Chaining Variables**

The chaining variables are initialized to the following 32-bit values in hexadecimal:

\[ A = \text{0x67452301} \]
\[ B = \text{0xEFCDAB89} \]
\[ C = \text{0x98BADCFE} \]
\[ D = \text{0x10325476} \]
\[ E = \text{0xC3D2E1F0} \]
\[ F = \text{0x50A28BE6} \]

**Step 3: Processing**

The processing of a compression function is defined as follows:

(a) Set \( A \) as \( \overline{A} \), \( B \) as \( \overline{B} \), \( C \) as \( \overline{C} \), \( D \) as \( \overline{D} \), \( E \) as \( \overline{E} \), \( F \) as \( \overline{F} \).

\[
\overline{A} = A \\
\overline{B} = B \\
\overline{C} = C \\
\overline{D} = D \\
\overline{E} = E \\
\overline{F} = F 
\]

(b) for \( t = 0, \ldots, 95 \)

\[
T = ((A \ll 5) + f_t(B, C, D) + F + W_t + k_t) \ll (W_i \mod 32) \\
F = E \oplus D \\
E = D \oplus C \\
D = C \oplus B \\
C = B \ll 30 \\
B = A \oplus (W_t \ll 15) \\
A = T 
\]

**Step 4: Output**

Upon the completion of the compress function the output is obtained according to Davies-Meyer feed forward mode and that is as follows:
\[ A = \overline{A} + A \]
\[ B = \overline{B} + B \]
\[ C = \overline{C} + C \]
\[ D = \overline{D} + D \]
\[ E = \overline{E} + E \]
\[ F = \overline{F} + F \]

After processing the last 512-bit message block, we get final 192-bit hash value.

### 3.4 Differential Collision Characteristics of Modified Message Expansion

Recent attacks on MD-5, SHA-0 and SHA-1 [137, 138, 140, 144] have targeted on the poor message expansion of these compression functions. Essentially, all these hash functions follow the same underlying design principle: the 512-bit message is first expanded linearly into \( N \) words, and then the \( N \) words are used as step keys (sometimes known as round keys) in \( N \) steps of a (non-linear) block cipher invoked on an initial vector. The output of the block cipher is the output of the compression function. The most effective attack against such compression functions is to launch a differential attack, where a difference in the messages leads to a zero difference in the output of the block cipher, thus leading to a collision.

In 1998, Chabaud and Joux presented an attack on SHA-0 [137]. They used a linearized variant of SHA-0 to find a characteristic. In the linearized variant SHA-0 all modular additions are replaced by XOR and the Boolean functions of Round 1 and Round 3 are replaced by Round 4 Boolean function. The basic idea of the attack is to derive conditions such that the probability of the linearized characteristic in the original hash function is as high as possible. Chabaud and Joux observed that the probability is related to the hamming weight of the characteristic: in general, the lower the weight of the characteristic, the higher its probability. But detailed analysis reveals that there is no fixed relation between the hamming weight of the characteristic and its probability. Not every ‘1’ bit in the characteristic results in the same decrease in probability of the characteristic. This is due to the following effects:
(a) Because of carry overflow effects, ‘1’ bits in certain positions of a 32-bit word, causes a lower decrease in probability than bits at other positions.

(b) The probability may depend on the round, the ‘1’ bit occurs in, due to the different non-linear functions used.

(c) Bits in neighboring positions can be grouped. As a group, they result in a lower decrease of probability than would be the case if the bits could not be grouped.

On average it is concluded that each bit difference in the key lowers the probability of success on an average by a factor of \(2^{-2.5}\). So the Hamming weight of a characteristic can be used to get a good idea about its probability.

Unfortunately, in MD-5, SHA-0 and SHA-1, it is possible to start with a message difference which leads to a small difference in the \(N\) expanded keys.

This in turn allows for a manageable overall differential characteristic of the above kind, hence leading to a collision attack. In particular, in MD-5 a 3 bit difference in the 512-bit message leads to a difference of only 12 bits in the expanded \((N = 64)\) keys. In SHA-0, there exists a message difference which leads to a 28 bit difference in the expanded \((N = 80)\) keys. It turns out that the differential characteristic corresponding to the first 16 steps can be assured with probability 1. Thus effectively, only the differences in latter steps contribute to lowering the probability of the differential characteristic holding. In SHA-0, the difference in the last 60 keys can be as low as 17 bits. Similarly, in SHA-1, there exists a message difference which leads to only a 27 bit difference in the last 60 keys.

The MDA-192 message expansion code: 512 information bits are packed into 16 32-bit words \((W_0,W_1,\ldots,W_{15})\), and 80 additional words are generated by the following recurrence as mentioned in the equation 3.8. The 96 words \((W_0,\ldots,W_{95})\), can visualize as a linear code over \(\mathbb{F}_2\) with the defined parity check equations over the words. Moreover if \((W_0,\ldots,W_{95})\), is a code word, then so is \((W_0 \ll j,\ldots,W_{95} \ll j)\), for all \(j = 1\ldots31\). This can further be interpreted as the code-word \((W_0^0,W_1^0,\ldots,W_{95}^0,W_0^1,\ldots,W_{95}^1,\ldots,W_{95}^{32})\), where \(W_i^j\) denotes the \(j^{th}\) bit of \(W_i\). Then it is clear that
this code is invariant under a rotation of 96 bits. We show here that it has much better minimum distance in comparison to SHA-1.

The code in MDA192 equation 3.7 uses a left rotation by 13 bit. Since, as long as the amount of rotation is relatively prime to 32, the code remains the same up to a permutation of its columns. In particular, its minimum weight does not change if left rotate by 13 is replaced by a left rotate by 1. Therefore instead of \( \text{invariant} \), we consider the following code \( \text{invariant}' \) which is equivalent up to a permutation in the codeword position. The following explicit permutation applied to the columns in \( \text{invariant} \) yields \( \text{invariant}' \):

\[
\Pi : \{0,1,\ldots,31\} \rightarrow \{0,1,\ldots,31\} \quad \text{where } j \mapsto (5 \cdot j) \mod 32, \text{ since } 5 \text{ is the inverse of } 13 \text{ modulo } 32.
\]

Let \( m_0,\ldots,m_{15} \) be the message blocks. Then for,

\[
i = 0,\ldots,15, W_i = m_i
\]

\[
W_t = \begin{cases} 
W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16} \oplus (W_{t-1} \oplus W_{t-2} \oplus W_{t-15}) \ll 1, & \text{for } t = 16,\ldots,51 \\
W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16} \oplus (W_{t-1} \oplus W_{t-2} \oplus W_{t-15} \oplus W_{t-20}) \ll 1), & \text{for } t = 52,\ldots,95
\end{cases} \quad \text{(3.8)}
\]

If we denote \( A = \langle X_0,\ldots,X_{31} \rangle, B = \langle X_{32},\ldots,X_{47} \rangle, C = \langle X_{48},\ldots,X_{95} \rangle \) then for any nonzero codeword in \( \text{invariant}' \), say \( X = \langle X_0,\ldots,X_{95} \rangle \) is equivalent to \( \langle A,B,C \rangle \). Let us consider the following code \( C \),

Let \( m_0,\ldots,m_{15} \) be the message blocks. Then for \( t = 0,\ldots,15, W_t = m_t \) and

\[
W_t = \begin{cases} 
W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16} \oplus (W_{t-1} \oplus W_{t-2} \oplus W_{t-15}) \ll 1, & \text{for } t = 16,\ldots,19 \\
W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16} \oplus (W_{t-1} \oplus W_{t-2} \oplus W_{t-15} \oplus W_{t-20}) \ll 1), & \text{for } t = 20,\ldots,63
\end{cases} \quad \text{(3.9)}
\]

From equation 3.8 it is observed that the code \( \text{invariant}' \) is completely determined by specifying any consecutive 16 words block provided the block starts anywhere in 0 to 36, since the rest can then be obtained by solving the recurrence relation. If \( B \) is taken as the message symbols then any non-zero codeword \( \langle B,C \rangle \) becomes codeword in \( \text{invariant}' \).

Finding lower bounds for minimum distance of such codewords is the NP-hard problem. Jutla
and Patthak [149] shown using computer assisted restricted search technique that code C has minimum distance at least 82 and 52 in the last 64 and 48 words respectively. They have divided the estimation into two main cases. First case, there is no zero columns in a codeword with every column on the average at the most 3 bits ON and Second case, there is all zero column in the codeword with the first neighboring non-zero column and so on.

(i) All Columns Non Zero Case
It is claimed by Jutla and Patthak that for any such codeword with any non-zero column, $C^0$ and columns, to the left of it by $C^1, C^2, \ldots, C^3$ with all $C^i$ are non-zero the combined weight is at least $3 \cdot (k + 1)$ where $0 \leq k \leq 7$. They have observed using restricted search that $k \leq 7$ column does not assured an average 3 bits are ON in each column, with column length 64 bits. According to considered method, they have created a partition of the 32 columns into good and bad groups of columns. A good column is the column which satisfies the condition of minimum number of ON bits present where as a bad one does not. There could be $(32 - e)$ good columns, where $e$ is the size of bad column group. The bad group has expected weight at least 1 and good group has expected weight at least 3. They have found that $e$ could be at most 7. Therefore the total weight of the codeword is at least $3 \cdot (32 - e) + e = 96 - 2 \cdot e \geq 82$.

(ii) At Least One Column Zero Case
It is considered that at least $C^0$ be a zero column then since the codeword is non-zero then certainly there are few left and right neighbor columns of it are non-zero. The non zero columns to the left of $C^0$ as $C^1, C^2, \ldots$ and non-zero columns towards right of it as $E^1, E^2, \ldots$. If with column $j$ it happens that $C^j = E^j$, i.e., $C^{j+1}$ is all zero. Then a homogeneous system of linear equations over $\mathbb{F}_2$ can be set up. There are $64 \times j$ variables in column $C^i$ through $C^j$. It is well known that such a system can have non-trivial solutions $2^{2^{j-1}}$ if and only if the rank of the coefficient matrix is strictly smaller than the number of variables and it is verified by them that $j = 1, 2, 3$ the rank is exactly the same. This inferred to them that there are 4 consecutive non-zero columns. They have performed an exhaustive search for $j = 4$ and $j = 5$ where the rank is smaller and estimated minimum weight for the latter case is 90. They have considered all cases
for the free variables and concluded that the minimum weight for last 64 words of such codeword is 82.

The similar analysis for estimating the minimum weight in the last 48 words also performed by them and they have shown it is 52. Since codeword C is the valid codeword in $\square'$ so it can be inferred that this codeword also having atleast minimum weight in the last 64 and 48 words are 82 and 52 respectively. Further from this we can deduce that minimum weight of last 16 words of codeword $\langle B \rangle$ is atleast 30. While expanding the message, previous 16 words take responsibility to propagate further bit difference in the next newly expanded words which is certainly gets further weak in the subsequent words expansion. It indicates that the first 16 words of $\langle B \rangle$ is also having minimum weight at least 30.

Hence the value of minimum weight in the last eighty words of the expanded message shown in equation 3.7 can be concluded to 112 and it further deduces to the good amount of difference among message words also reflects in chaining variable uniformly and finally in the computed hash value.

3.5 Indifferentiability of MDA-192 Compression Function

For the collision attack on hash function, Wang et al. [144] use the following strategy. The total number of steps of the hash function is split into two groups, denoted here by $S1$ and $S2$. $S1$ consists of the steps at the start of the hash function, for which it is possible to efficiently solve the equations imposed by the characteristic, by using methods like for instance so-called basic message modification and advanced message modification. $S2$ consists of the remaining steps. Consequently, the conditions imposed by the characteristic through $S1$ have no impact on the complexity of the attack. Therefore, the characteristic through $S2$ determines the attack complexity. Finding a good characteristic is then divided into two phases. First, a low weight linearized characteristic is determined, that leads to a pseudo collision in $S2$. Then by using a nonlinear characteristic in $S1$, the zero difference in the state variables at the start of $S1$ is transformed into the desired difference in the state variables at the start of $S2$. Hence, it becomes possible to turn the pseudo collision for $S2$ into a collision.
MDA-192 resists such attacks. For MDA-192 in $S1$ it is difficult to impose message modification techniques because first few steps use the original message words in which provide good mixing of words. A small change in input produces a large diffusion in subsequent words. It is also not easy to obtain low weight linearized characteristics in $S2$ for MDA-192 because by using message words heavily in variable bit rotations and computation of step operations, we introduce the redundancy in the round functions of MDA-192. Hash results are more strongly dependent on the input message. Also working variables are XORed to update values of chaining variables in step operations to provide good diffusion. The compression function of MDA-192 uses more number of step operations (24 step operations) than SHA-1 in each round. After the execution of few step operations the difference between message words becomes more complex and hard to control. Thus the compression function of the proposed hash algorithm improves the bit interdependency; it is therefore no longer possible to find a full differential path resulting in a collision for the compression function. The message expansion between steps 16 to step 95 also provides high randomness, good mixing of bits and lesser control over the propagation of difference in the words. For MDA-192 the minimum Hamming distance between two set of message words is far better than SHA-1. Message expansion of MDA-192 provides higher minimum distance between the similar message words as we have observed in previous section. For the steps of MDA-192, if the 1st bits of two adjacent words are changed, then the probability that the correction will succeed is zero. Clearly this method provides an equal distribution of 0s and 1s. This equal distribution in turn guarantees the desirable high probability of equal distribution of 1s and 0s in the output of the compression function.

Since it provides higher minimum distance and equal distribution of 0s and 1s, no two colliding messages using the method described in [137, 144] can be found.

Birthday Attack will succeed against all hash algorithms, including this one. Apart from the Birthday attack, there is no systematic way for generating messages with the same hash value. There is no good starting point for an attack.

### 3.6 Conclusion

In this chapter we have proposed SHA-1 design based 192-bit hash function which is built on Merkle-Damgård (MD) construction. It generates 192-bit hash value. It includes an extra 32-bit
chaining variable than SHA-1 which provides higher security against brute force attack. Also we have shown how the modified message expansion and improved step operation provide good resistance against differential attacks. Security analysis of MDA-192 against generic attacks and its performance evaluation are discussed in chapter 6 and chapter 7 respectively. In the next chapter we have shown how dither construction can make hash function more secure against fixed point and multicollision attack with high complexity.