CHAPTER 4

DSHA-1: A 160-BIT SERIAL STRUCTURED HASH FUNCTION
BASED ON DITHER CONSTRUCTION

We have observed that strong message expansion and improved step function reduces the effect of differential collision but still the design of hash function discussed in previous chapter is vulnerable to several generic attacks because of weak MD construction method. Recently some serious flaws [45, 46, 47] have been presented against general Merkle-Damgård iterative mode of operations. As a result of these attacks new domain extension algorithm is proposed by Rivest in [34] to tweak the Merkle-Damgård construction using a third input called dither input to avoid the attacks based on fixed points that can be iterated an arbitrary number of times.

In this chapter, we describe the design of dithering based SHA, DSHA-1 hash function. The design goal of DSHA-1 is strengthening SHA-1 by rebuilding it on dither construction. The DSHA-1 hash function based on the design principle of SHA-1[98]. The DSHA-1 hash function produces a message digest of 160-bit length from a message of length less than $2^{64}$ bits. Dither construction based compression function of DSHA-1 takes three inputs: 160 bit intermediate state, a message block and dither input to generate 160-bit hash value. There are two alternatives have been suggested to insert the dither inputs. One of them is to generate dither inputs through a pseudorandom number generator and another one is to modify the preprocessing of hash function. In the later, instead of dividing the message into 512 bit message blocks we divide it into 496 bit blocks. Remaining 16 bits are reserved for dither input. These 16 bits are generated either as any pseudorandom sequence or by inserting a sequence of alternating 0’s and 1’s depending on the position of bit in the dither sequence or number of message sub block. The use of short dither increases the efficiency of hash function. The compression function processes one 512-bit message block per iteration. Message expansion is applied to each 512-bit message block, where 16 32-bit message words are expanded to 80 32-bit message words using the message expansion suggested in [149]. Rest of the chapter is formed as follows: Section 4.1 describes the weaknesses of Merkle-Damgård construction, its remedy. Rationale of the design choices of DSHA-1 is mentioned in section 4.2. Section 4.3 gives the formal algorithmic description of proposed hash
function and its robustness against collision attack is analyzed in section 4.4. Section 4.5, conclude the chapter.

4.1 Weaknesses of MD Construction and Its Remedy

Since 2004, some of the existing hash functions MD5 [96], SHA-0 [97], SHA-1 [98] are broken. The existing hash functions are built by a same structure called Merkle-Damgård structure, which is made of iterating compression functions. These type of results are important indicatives, because collision resistance is a classical property and if collisions were easy to construct, it would become easier to mount stronger types of attacks which rely on the possibility to find collisions, such as Joux’s [45] multicollision attack. Following this result, Kelsey and Schneier [47] extended a previous result of Dean [46], and showed that Joux’s idea can be used to mount an efficient second preimage attack. It seems that iterating hash functions based on MD construction can hardly be secure anymore. Here we present how the MD construction is vulnerable to these specified attacks.

Multicollision attack introduced by Joux [45], found that when iterative hash functions are used, finding a set of $2^k$ messages all colliding on the same hash value is as easy as finding $k$ single collision for the hash function. Finding a collision in the compression function, i.e., a single block collision one can find $k$ of such collisions each starting from the chaining value produced by the previous one block collision. In other words, one has to find two message blocks $M_i$ and $M_i'$ where $M_i \neq M_i'$ with $f(H_{i-1}.M_i) = f(H_{i-1}.M_i')$ where $f$ represents the compression function and $H_i$ the chaining value. Then it is possible to construct $2^k$ messages with the same hash value by choosing for block $i$ either the message block $M_i$ or $M_i'$. Since multicollisions rely on cheap, fast birthday attacks, a simple defense is to use a larger intermediate hash value. Classical Merkle-Damgård transform does not provide large internal state, so multicollision attack can be mounted on it. Joux also showed that the concatenation of two different hash functions is not more secure against collision attacks than the strongest one.

The security goal for any $n$-bit hash function is that collisions require about $2^{n/2}$ work, while preimages and second preimages require about $2^n$ work. Dean [46], demonstrated that this goal
could not be accomplished by Merkle-Damgård construction based hash functions whose compression functions allowed the easy finding of fixed points. In a hash function, fixed points occur when the intermediate hash value does not change after hashing a given message block. Essentially, the hash value hashes onto itself. Dean showed that if fixed points can be easily found, the security of second preimage resistance breaks down, especially when the message being hashed is very long. This attack is particularly worrisome because it effectively bypass security measures of MD construction, and can extend a message by one block.

Even worse, in 2005 [47], Kelsey and Schneier, extending a recent result of Joux [45], gave a generic second preimage attack against Merkle-Damgård construction. They show how to use fixed points to create a powerful new second preimage attack called expandable messages. They showed that finding second preimages for $n$-bit iterative hash functions can be done with much less work than the expected $2^n$. Their attack generalized Dean’s fixed point attack into expandable messages that do not assume an easy way to find fixed points. Their attack is thus applicable to any Merkle-Damgård hash function. Their technique is related to the multicollision attack of Joux [45].

In light of these attacks, it is better to design existing hash functions on strong MD variants to increase their security level and to resist generic attacks. Dither construction is one of the good alternatives to MD construction. The dither construction is given by Rivest in [34] considered as a strong variant of MD construction which includes an additional counter like input. The design intension behind the dither construction is to add an iteration dependent input to the compression function in order to defeat above mentioned generic attacks.

### 4.2 Design Choices for DSHA-1Hash Function

The compression function of DSHA-1 has iterated through dither construction. A counter like input dither is added to the compression function as an additional input. The dither construction provides strong resistance against the generic and differential attack. Dither inputs are generated through a pseudorandom number generator. The initial working variables are specified constants, and the final chaining value is used as the output. The compression function processes one 512-bit message block per iteration. Message expansion is applied to each 512-bit message block, where 16 32-bit message words are expanded to 80 32-bit message words. The compression
function consist of four rounds, each round is made up of a sequence of 20 steps. DSHA-1 uses five 32-bit chaining variables to which new values are assigned in each round. The construction along with random step operations and message expansion help to generate higher minimum hamming weight between similar words and ensures random output. The detail description of design choices made is as follows:

(i) **Mode of Operation**

Mode of operation for DSHA1 is Dither construction. Dither construction is a Merkle-Damgård construction where a bit counter called as dither input is added to the input of the compression function, amongst other features. In the dither hash function, every call to the compression function $f$ has the three inputs: the dithering sequence $D = D_1, D_2, \ldots, D_t$, which depends on the iteration, the chaining value, and the next message block from $M = M_1, M_2, \ldots, M_t$. The padding is done by appending a single 1 bit followed by as many 0 bits as needed to complete an $m$-bit block after the message length is appended. Figure 4.1 shows the Dither construction.

![Figure 4.1: Dither construction.](image)

The dither value can be selected in many ways: one of the ways by following the suggestion of Kelsey and Schneier the dither value can be selected as the index, $D_i = i$. This approach is called the dithering by counter but this approach requires that compression function accept an arbitrary large input. Another suggestion for selecting the dither value can be a sequence of alternative 0’s and 1’s. A pseudorandom sequence can also be used as a dither value. This provides protection
against message block repetition. In his proposal Rivest suggested the use the infinite abelian square-free sequence. The abelian square-free sequence is an aperiodic sequence over a finite alphabet with the property that no sub-word is repeated. However, the method proposed for integrating the dither value into concrete hash functions is inefficient, in the sense that it increases the number of calls to the compression function. No indifferentiability result is known for the dither construction.

(ii) Design principle

DSHA-1 consists of the iterative application of a compression function, which transforms a 160-bit chaining variable $H_{i-1}$ into $H_i$, based on a 512-bit message block $M_i$. The proposed hash function DSHA-1 is based on design principles of SHA-1 and it includes two main stages for the computation of 160-bit hash value: first is preprocessing stage and second is computation stage. Preprocessing stage contains three steps: message padding, message parsing and initialization of five chaining variables. The computation involves message expansion and application of compression function. At the core of the compression function lies a block cipher used in Davies-Meyer mode. The block cipher itself consists of two parts: a message expansion and a state update transformation. The purpose of the message expansion is to expand a single 512-bit input message block into eighty 32-bit words $(W_0, \ldots, W_{79})$. The state update transformation takes as input a 160-bit chaining variable $H_{i-1}$ which is used to initialize five 32-bit registers $(A, B, C, D, E)$. These registers, referred to as chaining variables or state variables, are then iteratively updated in 80 steps. The output of round fourth of compression function is added to the chaining variable of the previous round to produce the next chaining variable for subsequent message block. The addition is modulo $2^{32}$ and it is performed separately on each of the five words in the buffer. The output from the very last round is the 160-bit hash result which we obtain after incrementally processing of all $N$ 512-bit blocks of the message. Upon the completion of the compression function the output obtained according to feed forward mode. The final message digest is stored in the $(A, B, C, D, E)$ buffer.

(iii) Message Padding and Parsing

The padding procedure follows same principle as of MD4-based hash function padding rule. The
message $M$ is padded before hash computation begins. The purpose of this padding is to ensure that the padded message is a multiple of 512. The message is always padded although the message already has the desired length. The input message is processed by 512-bit block. The hash function pads a message by appending a single bit 1 next to the end of a message, followed by zero or more bit 0’s until the length of the message and finally 64-bit message length. Suppose that the length of the message $M$ is $l$ bits, append the bit 1 to the end of the message, followed by $m$ zero bits, where $m$ is the smallest, non-negative solution to the equation $l + 1 + m \equiv 448 \mod 512$. Then represent the length of $M$ (before padding) by a 64-bit block and append it to the padded message. The entire message length is now exact multiple of 512. After a message has been padded, it is parsed into $N$ 512-bit blocks, $M = M_1, M_2, \ldots, M_N$ before the hash computation can begin.

(iv) Initial Chaining Variables

The five registers each of 32 bits, referred to as chaining variables or state variables are required to hold intermediate and final results of the hash function. The initial values of variables $(A, B, C, D, E)$ are exactly the same as the initial variables used in SHA-1. These, chaining variables are iteratively updated in 80 steps of the compression functions to produce message digest.

(v) Message Expansion

The message word used in each step is derived through a message expansion. The compression function processes one 512-bit message block per iteration. The 512-bit message block $M_i, i = 0, \ldots, N$ is divided into sixteen 32-bit words $(W_0, W_1, \ldots, W_{15})$. These sixteen 32-bit words are expanded to 80 32-bit words with the help of following relation, for $t = 16, \ldots, 79$.

\[
\begin{align*}
(W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16}) \oplus ((W_{t-1} \oplus W_{t-2} \oplus W_{t-15}) << 1), & \quad \text{for } t = 16, \ldots, 35 \\
(W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16}) \oplus ((W_{t-1} \oplus W_{t-2} \oplus W_{t-15} \oplus W_{t-20}) << 1), & \quad \text{for } t = 36, \ldots, 79
\end{align*}
\]

The advantages of using modified message expansion have been discussed in detail in previous chapter. The message expansion of DSHA-1 is exactly the same as that of suggested in [149]. For faster execution of step operations we considered here left cyclic rotation by 1 instead of 13.
(vi) **Dither Input**

The general idea of dithered hashing [34] is to perturb the hashing process by using an additional input to the compression function, formed by the consecutive elements of a fixed dithering sequence. This gives the attacker less control over the input of the compression function. There are many ways to select dither inputs. We have suggested two alternatives. One of them is to generate dither sequence from pseudo-random function. In particular, its goal is to prevent attacks based on expandable messages. In [190], pseudo-random function is used as a replacement for Boolean functions of step operations in SHA-1. For analyzing and evaluating the power of this algorithm authors generated 1000,000 numbers. According to their findings, all generated 1000,000 numbers are regularly distributed, unique and totally different from each other.

So, to generate unique random dither value we choose a pseudo-random function which generates a unique pseudo-random number according to its input message block. Total 80 32-bit pseudo-random numbers are generated. These random numbers are fed to compression function as a third input. Each step operation makes the use of exact one 32-bit dither input. The algorithm details are:

\[
P(t) = \{x_t \times \sqrt{2}\} \quad \ldots (4.2)
\]

where \(\{.\}\) is number’s fractional part and \(x_t\) is the value defined by users. In this algorithm \(x_t\) is a 32-bit message word. The pseudo-code of this algorithm is given below:

```plaintext
temp = Math.sqrt(2) * x_t;
for t = 1 to k {
    a[t] = toString(temp - Math.floor(temp));
    a[t] = substring(a[t], 2, 11);
}
```

Different dither inputs for compression function are computed as:

\[
d_t = P_t\{W_t \times \sqrt{2}\}, \text{ for } t = 0, \ldots, 79. \quad \ldots (4.3)
\]

For each 512-bit message block \(M_i\) there is a different set of 80 32-bit dither values \(d_t\). The benefits of this algorithm are high speed of random number generation with the longest repeating period. We have opted this method for generating dither inputs.
In second option we can deliver dither sequence directly into the message blocks. The message is divided into 496 bit blocks. Each block consist of sixteen 31-bit sub-blocks. After this 16 bits are generated either as any pseudorandom sequence or by alternating a sequence of 0’s and 1’s depending on the position of bit in the dither sequence or number of message sub block.

The first option will require extra 2560-bit input to the compression function which causes an obvious degradation in the overall performance of hash function but in second option we can generate dither sequence in preprocessing step without sacrificing much on efficiency.

(vii) Compression Function
At the core of the compression function lies a block cipher used in Davies-Meyer mode. The block cipher itself consists of two parts: a message expansion and a state update transformation. The state update transformation takes as input a 160-bit chaining variable \(H_{i-1}\) which is used to initialize five 32-bit registers \((A, B, C, D, E)\).

The execution of the compression function involves 80 steps. These eighty steps are divided into four rounds of processing as shown in Figure 4.2. The four rounds are almost identical, with the main difference being that each round uses a different primitive Boolean function, denoted by \(f_i\) in the specification. Each of them takes three 32-bit words as input and yields one 32-bit word as output. Boolean functions have SAC (Strict Avalanche Criterion) property which means that for any 1 bit input difference the output difference becomes zero with probability 1/2. Boolean functions and additive constants used in different rounds are same as the constants and Boolean functions used in SHA-1 algorithm. The Boolean function \(f_i\) defined as:

\[
\begin{align*}
    f_i &= (B \land C) \lor (\neg B \land D) & \text{for } t = 0, \ldots, 19 \\
    f_i &= (B \oplus C \oplus D) & \text{for } t = 20, \ldots, 39 \\
    f_i &= (B \land C) \lor (B \land D) \lor (C \land D) & \text{for } t = 40, \ldots, 59 \\
    f_i &= (B \oplus C \lor D) & \text{for } t = 60, \ldots, 79 
\end{align*}
\]

Each round takes as input the current 512-bit message block \(M_i\) and the 160-bit buffer value \((A, B, C, D, E)\) and produces as output an updated value of the buffer also referred to as the chaining variables. In addition to it, each round also uses dithering sequence and one of the four
constants which are 32 bits of fractional parts of square root of first four prime numbers. The values of 32-bit constants are given below:

\[ k_t = \begin{cases} 
0x5A827999 & \text{for } t = 0, \ldots, 19 \\
0x6ED9EBA1 & \text{for } t = 20, \ldots, 39 \\
0x8F1BBCDC & \text{for } t = 40, \ldots, 59 \\
0xCA62C1D6 & \text{for } t = 60, \ldots, 79 
\end{cases} \]

The processing a step of a compression function is defined in equation 4.4. The step function is non-linear so that the compression function cannot be simply described as a linear system of equations that can easily be solved and thus inverted. Moreover, it tries to create very complex dependencies of intermediate hash values on all the message block bits. It does so by mixing different mathematical operations (like modular addition, bit-wise functions and bit-wise shift rotations same as that of SHA-1) for which there is no unified ‘calculus’ that allows simplifying the mathematical expression defining the compression function. This effect is amplified by using each message bit multiple times (here, 5 times) over various spread-out steps. As a result of the lack of such a calculus, solving a set of equations over the compression function that results in a collision or pre-image attack appears to be very difficult.

\[ T = (A << 5) + f_i(B, C, D) + E + W_i + k_i + d_i \]
\[ E = D \]
\[ D = C \]
\[ C = B << 30 \]
\[ B = A \]
\[ A = T \]  

\[ \text{...}(4.4) \]

4.3 Description DSHA-1 Hash Function

DSHA-1 generates hash by iterating a compression function \( f : \{0,1\}^n \times \{0,1\}^m \times \{0,1\}^d \rightarrow \{0,1\}^n \) according to following algorithmic steps:

1. Pad and split a message \( M \) into \( t \) blocks \( (M_1, \ldots, M_t) \) of \( m \) bits each.
2. Set \( H_0 \) to the initialization value \( IV \).
3. For each message block \( M_i \) compute \( H_i = f(H_{i-1}, M_i, D_i) \).
4. Output $h_f(M) = H_1$.

**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 block of 512-bit each, $M = M_1, M_2, ..., M_N$.

**Step 2: Setting the Initial Chaining Variables**
The chaining variables are initialized to the following 32-bit values in hexadecimal:

- $A = 0x67452301$
- $B = 0xEFCDAB89$
- $C = 0x98BADCFE$
- $D = 0x10325476$
- $E = 0xC3D2E1F0$

**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 $\overline{E}$,

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

(b) for $t = 0, ..., 79$

\[
T = (A \lll 5) + f_t(B, C, D) + E + W_t + k_t + d_t
\]

- $E = D$
- $D = C$
- $C = B \lll 30$
- $B = A$
- $A = T$
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:

\[ A = \bar{A} + A \]
\[ B = \bar{B} + B \]
\[ C = \bar{C} + C \]
\[ D = \bar{D} + D \]
\[ E = \bar{E} + E \]

After processing the last 512-bit message block, we get final 160-bit hash value.

![Diagram of DSHA-1 step operation](image)

**Figure 4.2:** Step operation of DSHA-1.
4.4 Indifferentiability of DSHA-1 Compression Function

In near-collision attack one tries to find two message pairs whose hash values are not same but difference between them is as small as possible. Moreover, message pairs must be found less than $2^{n/2}$ trials. Chosen-prefix collision method is a good way to produce near-collision attack against SHA-1. In this attack the intruder uses two different prefixes and message blocks but identical suffixes to produce the near-collisions. According to the attack method for a hash function $h$ based on classical iterative mode (MD mode) with compression function $f : \{0,1\}^n \times \{0,1\}^m \rightarrow \{0,1\}^n$ that uses an intermediate hash values IMD and message block of bit size $n$ and $m$ respectively. There exists a chosen-prefix collision attack against $h$ that is faster than a brute force collision attack against $h$ if $h$ satisfies the following criteria:

(i) There is a set $I$ of at least four IMD differences $d$ for which there exists near-collision attacks against compression function that for fixed $IMD_{in}$ and $IMD'_{in}$ searches for message blocks $b$ and $b'$ such that:

\[ IMD'_{out} - IMD_{out} = IMD'_{in} - IMD_{in} - d \], where

\[ IMD_{out} = f(IMD_{in}, b), IMD'_{out} = f(IMD'_{in}, b'). \]

By inserting small dither sequence we will always obtain different message blocks, so this criteria is not fulfilled for DSHA-1.

(ii) Each of the above near-collision attacks can be modified to start with any given $IMD_{in}$ and $IMD'_{in}$ such that the total average complexity in calls to compression function, consisting of the average cost of modifying the near-collision attack together with the average runtime complexity of the modified near-collision attack, is at most $\sqrt{\Pi} \cdot 2^{n-1} / 4$.

By inspecting nature of compression function it is not possible to find near-collisions in specified efforts for DSHA-1.

(iii) Given any $IMD_{in}, IMD'_{in} \in \{0,1\}^n$ the probability that $f(IMD_{in}, b) = f(IMD'_{in}, b')$ for uniformly-randomly chosen $b \neq b' \in \{0,1\}^m$ is $2^{-n}$.

This criteria is valid for compression function of DSHA-1 as well.
There exists a function $\phi : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ such that the probability $P_{\text{useful}}$ that $y - x \in I$ for $\text{IMD}$-values $x$ and $y$ with $\phi(x,0) = \phi(y,1)$ is at least $\max(8 \cdot 2^{K-n}, \Pi \cdot 2^{-K})$.

Again due to dither insertion this criteria is not fulfilled for DSHA-1 compression function.

Chosen-prefix collision method allocates near-collision attack on hash function. Given chosen prefixes $P$ and $P'$, append padding bit strings $S$ and $S'$ such that the bit lengths of $P \| S$ and $P' \| S'$ are both equal to $m \cdot 512 - K$, where $m, K \in \mathbb{N}^+$ and $K$ is a constant value. Let $\text{IMD}_{m-1}$ and $\text{IMD}'_{m-1}$ be the intermediate hash values after processing the first $(m-1) \cdot 512$ bits of $P \| S$ and $P' \| S'$, respectively. Furthermore, let $b$ and $b'$ be the last $512 - K$ bits of $P \| S$ and $P' \| S'$, respectively.

Chosen-prefix collision method spreads over eighty steps, out of which first 20 steps are dedicated for deriving a differential path and last 60 steps for finding good disturbance vectors and joining the local collisions.

Choose the birthday search space $V$ and the birthday step function $g : V \to V$ as follows:

\[ V = \mathbb{Z}_{2^{31}} \times \mathbb{Z}_{2^{31}} \times \mathbb{Z}_{2^{31}} \times \mathbb{Z}_{2^{25}} \times \mathbb{Z}_{2^{25}}; \]

\[ f(v) = \begin{cases} 
\phi(f(\text{IMD}_{m-1}, b \| v)) & \text{if } \tau(v) = 0; \\
\phi(f(\text{IMD}'_{m-1}, b' \| v)) - (0, 0, 0, 2^{31}) & \text{if } \tau(v) = 1,
\end{cases} \]

Where, $\phi : \mathbb{Z}_{2^{31}} \to V$ and $\tau : V \to \{0,1\}$ are defined as:

\[ \phi(A,B,C,D,E) = ((A[i]_{i=19}^{31}, B[i]_{i=14}^{31}, C[i]_{i=0}^{30}, D[i]_{i=7}^{31}, E)); \]

\[ \tau(A,B,C,D,E) = w(A) \mod 2. \]

These choices were made with the following considerations:

(i) The target difference values are all of the form $(A,B,\mu \cdot 2^{31}, \nu \cdot 2^1, 2^{31})$, where $\mu \in \{0,1\}, \nu \in \{-1,1\},$ and $A,B \in \mathbb{Z}_{2^{31}}$.

(ii) Adding $A$ to a randomly chosen $x \in \mathbb{Z}_{2^{31}}$ affects bit position 19 and higher.

(iii) Adding $B$ to a randomly chosen $x \in \mathbb{Z}_{2^{31}}$ affects bit position 14 and higher.

(iv) Adding $D$ to a randomly chosen $x \in \mathbb{Z}_{2^{31}}$ affects bit position 7 and higher.

For a birthday search collision $g(v) = g(w)$ with $\tau(v) \neq \tau(w)$, let $(x, y) = (v, w)$ if $\tau(v) = 1$ and $(x, y) = (w, v)$ otherwise. Then,
IMD'_m = (A', B', C', D', E') = f(IMD'_{m-1}, b' \parallel x),
IMD_m = (A, B, C, D, E) = f(IMD_{m-1}, b \parallel y).

For each of the target difference determine the probability $P_{\delta IMD_{\text{diff}}}$ that $\delta IMD_m = \delta IMD_{\text{diff}}$. A birthday search collision pair $v, w$ with $f(v) = f(w)$ has a probability of $q = 2^{P_{\delta IMD_{\text{diff}}}}$ that $\tau(v) \neq \tau(w)$, and $\delta IMD_m$ is one of the target difference values.

But when we consider an additional input in the form of dither to the compression function and improved message expansion then it increases the expected birthday search complexity because many conditions are required to build the differential path for DSHA-1. Also disturbance vectors do not minimize the number of conditions in a path, so it is hard to construct local collisions. The proposed hash function does not satisfy any criteria necessary for finding collisions using chosen-prefix collision method and combined complexity for finding near-collision attack and birthday attack is much higher than $2^{80}$. For DSHA-1 it is estimated to $\sqrt{\frac{\Pi \cdot |V|}{2 \cdot q}} > 2^{80}$. So DSHA-1 is not susceptible to near-collision attack due to chosen prefix collision method.

4.5 Conclusion

In this proposal we have suggested a 160-bit hash function which is based on strong MD variant dither construction. It accepts three inputs; message block, chaining variables and dither input and generates a 160-bit hash value. The compression function shows indifferentiality against collision attack. In chapter 6 security analysis shows DSHA-1 preserves all the essential security properties and provides good resistance against generic attacks. Performance of DSHA-1 is analyzed in Chapter 7. Since single block compression function of DSHA-1 takes extra dither input of 2560-bit for all 80 steps, it results in excess time to compute the final hash value. This could be optimized by inserting dither input bits (either 1-bit or 2-bit) to the different message sub blocks itself and values of these input bits may depends upon index value of particular message sub block but this could lead to diluting the desired level of dithering. Other option includes use of the parallel iterative structure for design of hash function as discussed in the next chapter.