CHAPTER 6

SECURITY ANALYSIS OF PROPOSED HASH FUNCTIONS AGAINST GENERIC ATTACKS

As we know cryptographic hash functions are important primitives in cryptology. They are used in a wide range of security applications. Collision resistance, preimage resistance and second reimage resistance are the important properties of cryptographic hash function. If the size of the hash value is $n$-bit, then it is well known that collisions can be found in $2^{n/2}$ efforts and preimages (first or second) can be found in $2^n$ efforts. There are several generic attacks are developed to find collisions and preimages on hash functions with less efforts. Generic attacks can be applied to any hash function directly. Generic attacks make few or no assumption about the compression function. Examples of generic attacks are multicollision, long message second preimage and herding. Some recent results have shown weaknesses in popular hash algorithms such as MD5 and SHA-1. For these hash algorithms it was shown that how to find collisions faster than $2^{n/2}$ and preimages faster than $2^n$. These results demonstrate some weaknesses of the underlying compression function. Most of the popular hash functions are built on Merkle-Damgård construction. Recent results have suggested that this construction is not a good choice and inspired interest in new constructions for hash functions, which prevent the generic attacks as well as other cryptanalytic attacks. In this chapter we have analyzed the security of proposed hash functions against different generic attacks.

6.1 Resistance against Birthday Attack

The birthday paradox is a well known problem in probability theory that refers to the probability that in a room of $k$ randomly chosen people, at least two of them share the same birthday. It is called a paradox because this probability is much higher than most people expect when first confronted with the problem. In fact, with only 23 people, the probability is over 50%.

This is easy to show by calculating $p_k$, the probability that no two people in a room of $n$ people share the same birthday, and that gives us the probability that at least two people do share the
same birthday, $1 - p_k$, where $k \geq 1$. If $k \geq 366$ then $p_k = 0$ because then there are more people than unique birthdays which means that the probability that two people do not share a birthday is 0.

\[
p_k = \frac{365 \cdot 364 \cdot 363 \cdot \ldots \cdot 365 - (k - 1)}{365 \cdot 365 \cdot 365 \cdot \ldots \cdot 365 - k}
\]

...(6.1)

\[
p_k = \frac{365 \cdot 364 \cdot 363 \cdot \ldots \cdot (365 - (k - 1))}{365^k} = \frac{365!}{365^k (365 - k)!}
\]

...(6.2)

The smallest $k$ where $p_k$ is less than 50% is $k = 23$ when $p_k \approx 0.493$ which implies that we have 50.7% probability of two people sharing a birthday. The birthday paradox can be generalized to sets of any size, and it is not hard to prove that for a set of size $N$, the number of randomly chosen elements from that set that needs to be included to get a probability of at least 50% that two of them are the same is about $\sqrt{2N \ln 2} \approx \sqrt{N}$. This can be shown by generalizing the expression above to $p_{k,N}$ where $N$ is the number of available values. We want to find $k$ given $N$ such that $p_{k,N} \approx 50%$,

\[
p_{k,N} = \frac{N}{N} \left( 1 - \frac{1}{N} \right) \left( 1 - \frac{2}{N} \right) \ldots \left( 1 - \frac{k - 1}{N} \right) \approx \frac{1}{2}
\]

...(6.3)

\[
1 - p_{k,N} = 1 - 1 \cdot \left( 1 - \frac{1}{N} \right) \cdot \left( 1 - \frac{2}{N} \right) \ldots \left( 1 - \frac{k - 1}{N} \right) \approx \frac{1}{2}.
\]

By the Taylor’s expansion:

\[
e^x = 1 + \frac{x^1}{1!} + \frac{x^2}{2!} \ldots \approx 1 + x
\]

\[
1 - p_{k,N} \approx e^{-\frac{1}{N}} \cdot e^{-\frac{2}{N}} \ldots e^{-\frac{k - 1}{N}} = e^{-\sum_{i=1}^{k-1} \frac{i}{N}} = e^{-\frac{k(k - 1)}{2N}}
\]

\[
- \frac{k(k - 1)}{2N} = \ln \left( \frac{1}{2} \right) \Rightarrow k \approx \sqrt{2N \ln 2} \approx \sqrt{N}
\]

...(6.4)

From the above expression it is clear that for a cryptographic hash function with a digest size of $n$ bits, i.e. there are $N = 2^n$ different digests, we have a probability of $2^{n/2}$ for getting the collision attack. Since MDA-192 has a hash length 192 bits, i.e., there are $2^{192}$ possible outputs, and if we can generate $2^{96}$ different hash digests, the probability that there is at least one collision is around

92
50%. Similarly $2^{80}$ and $2^{128}$ efforts are required to produce collision for DSHA-1 and MNF-256 respectively.

6.2 Preimage Resistances

A preimage is a message that hashes to a given value. In a preimage attack, it is usually assumed that at least one message that hashes to the given value, exists. Therefore, one often says that the adversary (also called the attacker) is given $y = h(M)$ for some (randomly chosen) message $M$, which the attacker does not know. One method of finding preimages that works for any hash function is the brute force attack: hash random messages until the given hash value is reached. Since the hash value is $n$ bits in size, the number of random messages that must be tried is expected to be $2^n$. This is under the assumption that the hash function is balanced, which means that the preimage sets of all $2^n$ elements of the co-domain of the hash function are about the same size. We note that on average, one preimage for each element of the set $\{0,1\}^n$ is found in the above brute force attack. This means that each preimage has an average cost of 1. However, this is not the complexity that we are interested in. The setting is that the attacker is given an image, and must produce a preimage of that image. Of course, the attacker is free to keep a record of every trial hashing he has made. This may be seen as a pre-computation, having time complexity $2^n$, and it requires storing about $2^n$ hash/message pairs. After that, a preimage can be looked up in a constant time. Hence for MDA-192, DSHA-1 and MNF-256 hash functions the complexity of preimage attack is $2^{192}$, $2^{160}$ and $2^{256}$ respectively.

6.3 Second Preimage Resistance

A second preimage is a message that hashes to the same value as a given (randomly chosen) message, called the first preimage. Obviously, the second preimage must be different from the first. Here, we assume that the attacker is also given the hash value of the first preimage. If not, then the attacker can compute it himself. In the latter case the cost of hashing the first preimage is placed on the attacker, which we do not assume here. A brute force preimage attack can also be used to find a second preimage. One simply ignores the first preimage, except that one may take care not to try a message that is identical to the first preimage. By selecting messages at random, assuming that the domain of the hash function is much larger than the co-domain, the probability
of the second preimage being equal to the first is negligible, and therefore we usually ignore this possibility. Due to the above attack, finding a second preimage seems to never be harder than finding a (first) preimage. However, there are artificial constructions that allow preimages to be found in constant time, but which are collision and second preimage resistant.

Let, \( g \) be a secure \( n \)-bit hash function, and define the \((n+1)\) bit hash function \( h \) as follows:

\[
h(M) = \begin{cases} 
1 || M & \text{if } |M| = n \\
0 || g(M) & \text{otherwise.}
\end{cases}
\]

...(6.5)

Here ‘\( || \)’ denotes concatenation, and \(|M|\) means the bit length of \( M \). The numbers ‘0’ and ‘1’ are to be interpreted as 1-bit strings. \( h \) inherits the collision and second preimage resistance of \( g \). However, preimages can clearly be found in constant time if the first bit of the image is ‘1’. On the other hand, if a message \( M \) is chosen uniformly at random from the domain of the hash function, which is expected to be much larger than the image, and \( y = h(M) \) is given to an attacker that must find a preimage, then with overwhelming probability, the first bit of \( y \) will be a ‘0’, and in this case, it seems that the attacker is no better off than he would be in a second preimage attack.

To generalise, let \( h \) be an arbitrary \( n \)-bit hash function, and let \( \tilde{A} \) be an algorithm that is able to find preimages for \( h \). To find a collision, one chooses \( M \) at random, and gives \( y = h(M) \) to \( \tilde{A} \), which returns \( M' \) such that \( h(M') = y \). Now, if \( M \neq M' \), then \( (M,M') \) is a collision for \( h \). The probability that \( M \neq M' \) increases with the amount of compression that \( h \) is capable of providing. If \( h \) accepts inputs of size up to \( N \) bits, then one may choose \( M \) randomly from \( \{0,1\}^N \), resulting in the probability that \( M \neq M' \), being at the most \( 2^{n-N} \). Here, \( n \) is much smaller than \( N \).

In a second preimage attack, one is given a (randomly selected) first preimage \( M \). The preimage attack algorithm \( \tilde{A} \) can be used to find a second preimage as follows. One computes \( y = h(M) \), and passes \( y \) to \( \tilde{A} \), which returns the preimage \( M' \). Again, \( M' \) is not equal to \( M \) with a probability that increases with the amount of compression taking place in \( h \). When \( M \neq M' \), \( M' \) is a second preimage. By transposition, the above methods show that second preimage resistance independently imply preimage resistance, so finding a second preimage in MDA-192, DSHA-1...
and MNF-256 require $2^{192}, 2^{160},$ and $2^{256}$ operations respectively, since the amount of compression taking place is to 192, 160 and 256 bits.

6.4 Resistance against Length Extension Attack

For a given hash function $h$, if one can find a collision for two messages $M$ and $M'$ such that $h(M)$ and $h(M')$ collide, then one can apply a length extension attack. For any message $m$ one can easily produce a collision for $(M \overline{m})$ and $(M' \overline{m})$ as $h(M \overline{m}) = h(M' \overline{m})$. Padding rule of the algorithm avoids such type of attacks since we concatenate the length of the message to the message itself.

The compression function $f$ of MDA-192 compresses the message block and chaining inputs as follows:

$$\{0,1\}^n \times \{0,1\}^m \rightarrow \{0,1\}^n$$  \hspace{1cm} \text{... (6.6)}

As we have seen, hash functions DSHA-1 and MNF-256 also accept messages of arbitrary length, and produce a fixed-length output. The compression function $f$ of DSHA-1 and MNF-256 takes three inputs:

$$\{0,1\}^n \times \{0,1\}^m \times \{0,1\}^d \rightarrow \{0,1\}^n$$  \hspace{1cm} \text{... (6.7)}

Where, input chaining variables are of $n, n \in \{192,160,256\}$ bits for different proposals, input message block size is $m$ bits (512 bits) and dither input size is $d$ bits (32 bits). It is obvious that $m > n$ for all hash functions. To hash a message $M$ of size $N$ bits, we split $M$ into a number of chunks, or message blocks, of size $m$ bits each, and then process each block one at a time using $f$.

The first input to $f$ is a state, that is updated either by message block only or by two inputs message block and dither input. The output is the new, updated state.

To be more precise, MDA-192 iterate a compression function $f$ as follows. Let $M$ be the message to be hashed, and assume that it is split into $t$ message blocks $M = M_1, M_2, \ldots, M_t$. Let an initial state $IV$ which is fixed for the hash function. Let $H_0 = IV$, then for each $i$ from 1 to $t$:

$$H_i = f(H_{i-1}, M_i) \hspace{1cm} i = 1, 2, \ldots, t, \hspace{1cm} \text{... (6.8)}$$

DSHA-1 and MNF-256 iterate a compression function $f$ as follows.
\[ H_i = f(H_{i-1}, M_i, D_i) \quad i = 1, 2, \ldots, t, \quad \ldots(6.9) \]

The last state \( H_t \) is the output of the hash function, i.e., \( h(M) = H_t \). In proposed hash functions we use a padding call \( \text{Padding}_m \) that enlarges \( M \) so that its length is a multiple of \( m \). Hence, instead of feeding \( M \) to \( h \) directly, we feed \( \text{Padding}_m(M) \) to \( h \). This means, of course, that \( \text{Padding}_m(M) \) must not produce collisions, because if it did, then it would mean a collision for \( h \) as well. This also means that every message must be padded, even if its length is already a multiple of \( m \). If it was not, then it would be easy to find a collision: Let us consider an arbitrary message \( M \) of length not a multiple of \( m \). Pad \( M \) to \( M' \) i.e., \( M' = \text{padding}_m(M) \). Since \( M' \) considered as a message in its own right, would not be padded, \( M \) and \( M' \) will collide.

The strength of the any construction lies in the fact that there is a reduction proof that reduces the collision resistance of the hash function to the collision resistance of the compression function. This security proof requires a slightly more complicated padding function. Consider the constructions (6.8) and (6.9), with the padding rule that does not includes length of the message. A collision may be obtained by the use of a so-called length extension attack. To overcome this problem, all the proposed hash designs are strengthened by encoding the length into the last block. Hence due to this property all three proposals are resistant to length extension attack.

### 6.5 Resistance against Meet in the Middle Attack

This attack is a variation of birthday attack and is applicable to the hash functions that use a round function. Instead of message digest, intermediate chaining variables are compared. This attack enables a cryptanalyst to construct a message with a pre-specified message digest, which is not possible in case of a simple birthday attack. The attacker generates \( s_1 \) samples for the first part and \( s_2 \) samples for the last part of a forged message. The attacker then goes forwards from initial value and goes backwards from the hash value and expects that the two intermediate values collide to the same value. Onewayness of the compression functions of DSHA-1 and MNF-256 restrict an attacker to construct a message with a pre-specified hash value. MDA-192 is vulnerable to this attack because it is easy to find preimage and second preimage against it using generic attacks faster than respective brute force search.
6.6 Resistance against Fixed Point Attack

A fixed point of a compression function is a pair \((H_{i-1}, M_i)\) that satisfies \(H_i = H_{i-1} = f(H_{i-1}, M_i)\). Finding a fixed point of a compression function that is constructed in accordance with Davies-Meyer mode, is easy. A decryption of the value 0 with an arbitrary message \((M_i)\) results with a fixed point, i.e., \(H_{i-1} = D_{M_i}(0)\), with no control over the value of \(H_{i-1}\). In order to make such a fixed point suitable for an attack, \(H_{i-1}\) should be a chaining value that is received by hashing message blocks with the standard initial values. In order to find such a chaining value, an attacker executes a meet-in-the-middle attack [44] that aims at finding a chaining value that equals the fixed point. The attack proceeds by finding \(2^n/2\) fixed points and storing their values. Then, message blocks are hashed with the standard initial value \(H_0\) and a match with the stored fixed points is searched. A match is expected after hashing about \(2^n/2\) different message blocks. Thus, finding a fixed point requires about \(2^{n/2+1}\) executions of the compression function. With this match, messages of arbitrary length and with the hash value of the fixed point may be constructed. These messages are formed by the first block just found, and a concatenation of as many fixed points the attacker likes (up to the maximum message length the hash function allows).

![Fixed point attack](image)

Figure 6.1: Fixed point attack.

Merkle-Damgård construction based MDA-192 is not able to foil this attack. For this case the attacker starts with finding a fixed point. Consequently, a match with one of the chaining values of the message \(M\) length of the message to the last block, the hash results of the two messages equal only before the last block is searched as earlier. However, the \(2^{n-L}\) single-blocks that the attacker hashes, are hashed with the hash result of the fix-point instead of the standard initial values. Hence, once a match is found the attacker adopts the length by adding as many blocks of
fixed point as required. The second preimage is then \( M_1 | M_2 | \ldots | M_{2^L} \). Thus, the attacker succeeds in finding a second preimage for \( n \)-bit hash function with a complexity of \( 2^{n-L} \) rather than \( 2^n \). This attack is particularly worrisome because it effectively bypasses length padding, and can extend a message by one block.

The complexity of the attack is determined by the complexity of finding expandable messages. Starting from an arbitrary initialization vector, expandable messages are groups of messages of varying lengths whose hash values collide just prior to entering the finalization function i.e. just before the message length is appended. These are messages of varying sizes such that all these messages collide internally for a given initial values. These expandable messages can be quite long, and can be used to generate second preimages for a lot less than \( 2^n \) work. Fixed point attacks in this form cannot be applied to the DSHA-1 and MNF-256 because we include the random dither values in each iteration of compression function which does not allow to find expandable messages. Dither value at each step avoids the existence of trivial fixed point for both dither based designs.

6.7 Resistance against Multicollision Attack

Joux described that finding multiple collisions in a Merkle-Damgård construction based hash function is not much harder than finding single collisions in [45]. A multicollision consists of \( t \) messages \( (M_1, \ldots, M_t) \) in which \( h(M_1) = h(M_2) = \ldots = h(M_t) \). In his multicollision attack, Joux assumed an algorithm \( \tilde{A} \) that given an initial state, returns two colliding messages. For this \( \tilde{A} \) may use the birthday attack or any other attack exploiting weaknesses in the corresponding hash function. A multicollision is expected to be found with a complexity of \( 2^{(n-1)t/6} \). Using the iterative structure of the hash function it is shown that the complexity of finding it, is about \( t \cdot 2^{n/2} \) executions of the compression function. The Figure 6.2 illustrates concatenation of \( t \) pairs of colliding messages, in which each pair starts with the chaining value of the previous pair. The complexity of finding such a pair is \( 2^{n/2} \), thus the complexity of finding the whole chain is about \( t \cdot 2^{n/2} \). From these \( t \) pairs, \( 2^r \) pairs that produce the same hash result can be constructed, thus the complexity of finding the \( 2^r \) collision is much lower than expected. This attack can also be applied
to a cascading hash function where, \( h(M) = h'(M) \parallel h''(M) \) and \( h' \) and \( h'' \) are independent \( n \)-bit hash functions.

Figure 6.2: Multicollision attack.

If either \( h' \) and \( h'' \) is a function built using the Merkle-Damgård construction, then the expected number of computations of the underlying compression function is only \( n/2 \times 2^{n/2} \). 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 \( 2^t \) messages with the same hash value by choosing for block \( i \) either the message block \( M_i \) or \( M'_i \). Joux showed that the concatenation of two different hash functions is not more secure against collision attacks than the strongest one. The complexity of the attack depends on the size of internal state of compression function since DSHA-1 and MNF-256 do not use large internal states but they are complex enough and it require at least brute force search to find collision.

Assume that message \( M \) contains \( t \) blocks \( M = M_1, M_2, \ldots, M_t \) and \( M' = M'_1, M'_2, \ldots, M'_t \) are corresponding collision blocks, i.e., then, each block is freely replaced by \( M'_i \) and \( 2^t \) collision message can be found.

In dither iterating mode we assume \( f \) is an ideal compression function,

\[
f : H_i = f(H_{i-1}, M_i, D_i)
\]

Besides \( H_{i-1} \), each input contains the two neighboring inputs message block and dither input. We assume the two message blocks \( M_i \) and \( M'_i \) yield a collision in the \( i^{th} \) iteration (Although the complexity of finding a pair of collisions is \( 2^{n/2} \)), such that
\[ f(H_{i-1}, M_i, D_i) = f(H_{i-1}, M'_i, D'_i) \neq H_i, \quad \ldots \text{(6.11)} \]

For this iteration we will not be able to find same hash results because a tiny difference in message will generate completely different dither input \( D_i \neq D'_i \). Here dither input depends upon message blocks. In dither based designs even the corresponding collision blocks can be caught by brute force with the complexity of \( 2^{n/2} \), they can't be freely substituted each other to build a new collision message due to highly random dither input.

MDA-192 uses small internal steps in compression process of message, so it is easy to find multicollisions in MDA-192. Its weak construction is not able to foil this attack.

Multicollisions exploit several properties of iterated hash functions built upon compression functions. First, the complexity of birthday attack is \( 2^{n/2} \), and therefore the fastest kind of generic hash function attack. Second, the compression function calculates an intermediate hash value after each message block, so any collisions within a single block can immediately be exploited. These attacks must be done in order i.e., from first to last. Since multicollisions rely on cheap, fast birthday attacks, a simple defense is to use a larger intermediate hash value. For instance, if the algorithm generates hash length of 256 bits, then the intermediate hash value should be 512 bits or more. We use the classical Merkle-Damgård transform in MDA-192 and the internal state of proposed hash function is not large, so attack can be mounted on it. Although we have made algorithmic changes, it does not help to increase the security against this attack. Finding multicollisions for MDA-192 is not much harder than finding ordinary collisions.

The complexity of the attack depends on the size of internal state of compression function since DSHA-1 and MNF-256 do not use large internal states but they are complex enough and it require at least brute force search to find collision.

6.8 Resistance against Long Second Preimage Attack

Kelsey and Schneier gave a generic second preimage attack. This attack can be considered as an attack against the mode of operation itself. This attack finds a second preimage of any message of length about \( l \) (number of blocks in a message) by evaluating the compression function only \( 2^n/l \) times. DSHA-1 and MNF-256 have optimal resistance against generic second preimage
attacks where as Merkle-Damgård construction based MDA-192 is susceptible to generic second preimage attack. The following explanation proves the intuitive idea that the known generic second preimage attacks does not work against dither based designs.

Amongst the numerous notions of second preimage resistance, we consider the one defined by the following game: a second preimage adversary $A$ has oracle access to a compression function $f$. It receives a randomly generated challenge $M$ of length $l$, and succeeds if it outputs a second message $M'$ such that $M \neq M'$ and $h_f(M) = h_f(M')$ where $h$ is an iteration mode for $f$ (dither).

Such an adversary $(q,l,\epsilon)$ – breaks $h_f$ if after at most $q$ queries to $f$ its success probability is lower bound by $\epsilon$. A hash function $h_f$ is $(q,l,\epsilon)$ – second preimage resistant if the advantage for the message of length $l$ of any attacker asking at most $q$ queries is upper-bounded by $\epsilon$.

Here, $f$ be a public random function and $h_f$ be the dither-iteration of $f$, and $A$ a second preimage adversary that $(q,l,\epsilon)$ – break $h_f$. Where $q$ denotes the number of queries sent to the compression function $f$ by adversary $A$. Then: $\epsilon \leq q/2^{n-1}$.

In this explanation it is assumed that the compression function is an ideal primitive i.e. computationally unbounded adversaries have oracle access. If execution of the adversary $A$, bookmark the queries sent by $A$ to $f$ and if it is supposed that $A$ evaluates $h_f(M)$, then $A$ sends the corresponding queries to the oracles at some point. These particular queries are denoted by

$$y = f(H_{i-1}, m_i, d_i, H_i), 1 \leq i \leq l. \text{ and } h_f(M) = H_i.$$ 

If $|M| \neq |M'|$, $|M|$ represents the length of $M$ in blocks then the values of the dither input entering the compression function in its last invocation are different. Therefore, $A$ has found a second preimage on $f$. Each query has a probability $2^{-n}$ to give this preimage, because $f$ is a random function.

Here, if $A$ wins and $|M| = |M'|$ then event $E$ follows it realized, because it is known that there is a collision on the compression function where one of the colliding hash value is one of the $H_i$. Let us note $M = m_1, \ldots, m_l, M' = m'_1, \ldots, m'_l, H_0 = H_0' = IV, H_i = f(H_{i-1}, m_i, d_i)$ and $H'_i = f(H'_{i-1}, m'_i, d_i)$. 

101
Since \( H_t = H'_t \), either there is a collision on \( f \) or \((m_i, H_{t-i}) = (m'_i, H'_{t-i})\). This argument repeats. Since \(|M| = |M'|\), then either there is a collision for \( f \) at some point or \( m_i = m'_i \), for all \( i, 1 \leq i \leq t \).

In the later case, \( M = M' \), which is not possible. When \( A \) submits its \( i^{th} \) query, a random value is chosen and returned to \( A \). The event \( E \) is realized if and only if this value is \( H_i \), and this happens with probability \( 2^n \). This query may also allow \( A \) to invert \( H \) with probability \( 2^n \). Each query allows \( A \) to win with probability \( 2^{-(n-1)} \), and there are such \( q \) queries.

From the above explanation it is clear that there is no generic second preimage attack faster than exhaustive search on DSHA-1 and MNF-256.

Again consider adversary \( A \) that \((q, l, \varepsilon)\) breaks the second preimage resistance of \( h_f \). Hashing procedure involved in MDA-192 considers compression of message \( M \) along with chaining variables. If \( A \) succeeds in finding a second preimage, then in particular \( A \) has found a collision.

As we know about the nature of iterating procedure of Merkle-Damgård construction, in the presence of the strengthening, this implies a collision on the compression function \( f \). Because compression function \( f \) is random, all the intermediate chaining variables are random values. Every time \( A \) submits a new query to the oracle, it receives a uniformly distributed random value.

The probability that \( A \) wins is upper bounded by the probability that this random value is one of the intermediate value. This reduces the problem of finding a second preimage for hash function to the problem of finding a second preimage of one out of many random chaining values for \( f \). So a second preimage of a message of size \( l \) on Merkle-Damgård construction based MDA-192 in less than \( 2^n/l \) operations.

### 6.9 Resistance against Klima and Gligoroski Attack

Klima and Gligoroski [187] showed collision attack against classic Merkle-Damgård hash designs when hashing long messages of \( 2^k \) message blocks. For either based designs DSHA-1 and MNF-256 their attack does not reduce the collision search, from the generic bound of \( 2^n/2 \) to \( 2^{n/2-k/2} \) number of hash calls, where hashing is done over messages of length \( 2^k \) blocks. For the definition of hash function, let us denote:
\( f(H, M, D) \) - a compression function \( f \) with chaining variable \( H \), message block variable \( M \), and dither input \( D \).

\( hl \) - the length of the chaining variable, i.e. the length of compression function output.

\( ml \) - the length of the message block.

\( hashlen \) - the length of the hash function output.

\( dlen \) - the length of the dither input.

If the compression function has the property, that for every pair \((M, D)\) the function \( f(H, M, D) \equiv f_{(M, D)}(H) \) is an ideal random function of the variable \( H \), we denote it as \( IRF(H) \).

If the compression function has the property, that for every value \( H \) the function \( f(H, M, D) \equiv f_{H}(M, D) \) is an ideal random function of the variables \((M, D)\), we denote it as \( IRF(M, D) \). The hash function is \( h: \{0, 1\}^* \rightarrow \{0, 1\}^n \) defined by a compression function \( f: \{0, 1\}^n \times \{0, 1\}^{ml} \times \{0, 1\}^{dlen} \rightarrow \{0, 1\}^n \). Our proposed compression functions in DSHA-1 and MNF-256 also include a unique 32-bit dither input in each step operation which is represented as \( dlen \).

Here, \( n = hashlen = (160, 256) \). The general case where the hashed message is padded and divided into 512-bit blocks. Let us suppose that a message \( M \) is divided into two parts \( A \) and \( B \), i.e. \( M = A \parallel B \), where the part \( A \) consist of just one message block of 512 bits, and rest of the number of 512-bit blocks in the part \( B \) (let \( N = 2^{35} \)). Let us denote by \( H^A \) the intermediate chaining value, obtained after hashing the part \( A \) of the message \( M \) and let us suppose that the content of the part \( B \) is never changing- so it consists of constant message blocks \( C_1, C_2, \ldots, C_N \) and corresponding dither values \( D_1, D_2, \ldots, D_N \). If padding is a part of the definition, then it is also a constant block. We compute the final hash with the following iterative procedure:
\[
H_1 = f(H^A, C_1, D_1) \\
H_2 = f(H_1, C_2, D_2) \\
H_3 = f(H_2, C_3, D_3) \\
\vdots \\
H_N = f(H_{N-1}, C_N, D_N) \\
h(M) = H_N
\] ...(6.12)

If the compression function \( f \) is \( IRF(M, D) \), then the chaining values are not loosing the entropy in above \( N \) steps. We obtain that the entropy of the final hash \( H_N \) is equal to

\[
E(hash) = hashlen + 1 - \log_2(N) + dlen, \quad \ldots (6.13)
\]

It computes \( 2^{E(hash)/2} \), \( H_N \) for different parts \( A \) (whereas the part \( B \) remains unchanged), which is much higher than brute force searches \( (2^{80}, 2^{256}) \). Cryptographically strong hash function \( H \) should require at least \( (2^{80}, 2^{256}) \) hash computations for finding a collision. According to the birthday paradox it is not sufficient for finding a collision in the set of these values with probability around \( 1/2 \). So this generic attack is not applicable on DSHA-1 and MNF-256.

In MDA-192, we have no third input. The compression function is dependent on message blocks and chaining variables. In above computation it looses entropy in all steps, so in final hash value as well. For MDA-192

### 6.10 Resistance against Correcting Block Attack

Collisions and second preimage can easily be found by correcting block attack. In this attack, the cryptanalyst uses an existing message and its hash and tries to change one or more message blocks such that the resulting hash remains unchanged. Starting with two arbitrary messages \( M = (M_1, \ldots, M_t) \) consisting of \( t \) blocks which produce the hash value \( h(M) \), and another message \( N \) of \( k \) blocks which is shorter than \( M \) and searching for set of message blocks \( Y \) such that \( h(M) = h(N||Y) \). In general, the attack is applied by correcting the last message block before padding and called correcting last block attack. In this case, attacker produces a message \( N \) of length \( t - 1 \) blocks. Let \( H_i \) and \( H'_i \) be, respectively, the final chaining values of messages \( M \) and
before processing padding blocks. Then the attacker searches for a message block $M_L$ such that $f(H_{i-1}', M_L) = H_i$, where $f$ is the compression function of the hash algorithm. This way $h(N\|M_L)$ will be equal to $h(M)$ and attacker will get a second preimage for $M$. The choice of message block to be corrected has no restriction and can be any message block but the first. To find a collision, attacker randomly chooses two messages of the same length $M$ and $N$. Then, attacker searches for message blocks $M_L$ and $M'_L$ such that $h(M\|M_L) = h(N\|M'_L)$ and gets a collision pair. The complexity of finding such a correcting block is equal to exhaustive collision search for most of the modern hash functions. However, some hash functions are easy to manipulate and this process can be less complicated. Preprocessing of all the proposed hash functions foil this attack, since it includes message length in final block. Computation steps of all proposals are random enough which show sufficiency to resist the correcting block attack.

6.11 Resistance against Herding Attack

Kelsey and Kohno presented herding attack in [48]. This attack is closely related to the multi-collision and second preimage attacks discussed above. In this attack, the attacker presents the hash value of a message without knowing the beginning of the message. An attacker using this attack can commit to a value available publicly, which corresponds to a meaningful message. After the announcement of the result, the attacker publishes a message that has the pre published value, and the correct information, along with a suffix. The main idea behind this attack is to start with a possible number of chaining values and selects the value, which helps the attacker to perform a preimage attack on the actual result obtained. Attacker chooses a target hash $T$, then a prefix $P$ for a message is given to the attacker and the attacker must create a suffix $S$ such that $h(P\|S) = T$. In this case $T$ is not random. The main idea of the attack is building a so called diamond structure: a binary tree of collisions. Similarly to the previous multicollision attack, in order to build the diamond structure, internal collisions have to be built. The attack proceeds in two steps:

Step-1: construct a diamond structure and calculate the value $T$.

Step-2: given a prefix, find a suitable suffix and herd it to $T$ through the diamond.
In step 1, the attacker constructs a diamond structure, where the vertices are hash values and the edges are messages. If two messages meet in a vertex, they collide at that vertex. Initially, the attacker randomly generates an arbitrary large number of initial messages, \((M_1, \ldots, M_Y)\), hashes them and tries to find collisions, then repeat until reaching the root of the diamond, \(T\). Once the diamond is constructed, any path from the initial messages to \(T\) will hash to \(T\). In step 2, the attacker herds a given prefix \(P\) to hash to \(T\) as follows: first the attacker searches for a suitable 1-block suffix \(S\) that if concatenated with \(P\), it will produce a hash colliding with one of the hash values of the initial messages \(h(M_i)\), where \(i \in \{1, 2, \ldots, Y\}\); for \(2^k\) initial messages, \(2^{n-k}\) trials are required to find such a suffix, where \(n\) is the length of the final hash. Once a match is found, \(P, S\) and the sequence of messages from the matching \(h(M_i)\) to \(T\) are concatenated, and this whole string will eventually hash to \(T\).

The compression function of MDA-192 follows MD construction. It has shown weaknesses against the multicollision and long second preimage attacks. Also it is easy to find fixed point for MDA-192. Since herding attack is closely related to these generic attacks, it is easy to mount herding attack on MDA-192.

For dither based designs it is difficult to generate internal collision using a diamond structure with fixed initial values, random message and a random dither sequence. Therefore both dither based designs are resistant to the herding attack.

6.12 Conclusion
In this chapter we have analyzed the security of all proposed hash functions against the major generic attacks. It is found that MDA-192 is vulnerable against most of them due to weak MD construction and both dither based designs DSHA-1 and MNF-256 are showing good resistance against generic attacks.