Iterated hash functions have been the most successful method for constructing fast and secure hash functions. Usually, hash functions are built upon two components: a compression function and a domain extension algorithm. The compression function has the same security requirements that a hash function but takes fixed length inputs. The domain extension algorithm defines how to use the compression function in order to handle arbitrary length inputs. Almost all hash functions are iterative processes which hash inputs of arbitrary length by processing successive fixed-size blocks of input. A number of recent attacks on hash functions have highlighted weaknesses of both specific to hash functions, and the underlying construction method. In this chapter, we discuss some popular known iterative hashing constructions, compression methods and different attack methods on hash functions. Common iterative structure is shown in the Figure 2.1.
2.1 Construction Methods

Iterated construction method is the core component in the design of hash functions. The construction method iterates compression function to map arbitrary length input to fixed size output. Merkle-Damgård construction is a commonly used iterative method for this purpose. Several other variants of Merkle-Damgård construction have been proposed to overcome the weaknesses of Merkle-Damgård construction. In this section we give different construction methods.

2.1.1 Merkle-Damgård Construction

From the early beginning of hash functions in cryptography, designers relied on the Merkle-Damgård (abbreviated to MD) construction. The MD construction was discovered by Merkle [14] and Damgård [15] in 1989 independently. Majority of famous hash functions such as MD4 [95], MD5 [96], SHA-0 [97], SHA-1 [98], RIPEMD-160 [101] etc. follow the iterative MD method. A compression function which takes a fixed input length value and outputs a fixed length hash value is core component of this construction. A compression function accepts two inputs: a chaining variable and a block of message. Recent results, however, highlight some intrinsic limitations of the MD approach. This includes being vulnerable to multicollision attacks [45], long second preimages attacks [47], and herding attack [48]. In general, and due to certain structural weaknesses, MD-based hash functions do not behave like random oracles. Various proposals have been suggested to overcome the problems of MD construction such as, sponge [21], wide-pipe and double-pipe [23], PFMD [28], LH [19], EMD [29], MDP[30], zipper [32], RMX [33], dithering [34], HAIFA [35], NI [37], SH [38], CS [39]. Few of them are discussed in subsequent sections.

2.1.2 Tree Construction

This is the most parallelizable class of constructions and is mainly suited for multi core platforms where multiple processors can independently operate on different parts of the message simultaneously. Damgård [15] briefly described a tree hashing approach for extending collision resistance hash functions. It allows parallel processing of the message. Figure 2.2 illustrates a typical tree based hashing construction. Damgård tree construction was later optimised by Sarkar and Scellenberg [16]. The main difference between SS construction and previous constructions is
that authors consider the number of available processors to be fixed while the length of the message can be arbitrarily long.

Figure 2.2: Tree construction.

Similarly, Carter and Wegman [17] used tree hashing techniques to build universal hash functions. This was followed up by Naor and Yung [18] and Bellare and Rogaway [19] in the context of UOWHFs (Universal One Way Hash Functions). In [20] Bellare and Micciancio proposed the randomize-then-combine paradigm, where the message is split into blocks, each block is processed via randomizing function (derived from some standard hash function) and finally combined by an operation such as XOR.

2.1.3 Sponge Construction

Sponge construction [21] is an iterative hash function construction, builds upon a fixed length transformation or permutation instead of a compression function and can generate output strings of infinite length. Sponge hashing proceeds in two phases, the absorbing phase and the squeezing phase. Since the sponge construction supports variable length output, the user chooses the length of the final hash value which determines how many of the returned blocks in the squeezing phase need to be returned. Sponge construction can be used to build both hash functions and stream ciphers. The sponge construction has influenced hash functions such as Keccak [136] and
PHOTON [22]. Keccak has recently been selected as the winner of SHA-3 competition. Figure 2.3 illustrates the sponge construction.

2.1.4 Wide and Double Pipe Constructions
Lucks has proposed a wide pipe and double pipe hash function constructions [23] which provides an enhancement of the Merkle-Damgård construction. The wide pipe construction shown in the Figure 2.4 intended to increase the size of the internal state of $n$-bit hash function and $w$-bit compression function, where $w > n$. 

Figure 2.3: Sponge construction.

Figure 2.4: Wide-pipe construction.
This means that the wide pipe design obtains a greater internal state than message digest length by using a larger compression function. Constructing a collision-resistant compression function with \( w > n \) output bits may be simpler than constructing an \( n \)-bit compression function with the same level of collision resistance. On the other hand, the double pipe design shown in the Figure 2.5 maintains twice the hash size using the \( w = 2n \) compression function in parallel to process each message block.

From these designs Lucks showed that increasing the size of the internal state (i.e. the chaining variable) to become larger than the size of the final hash value, would significantly improve the security of the hash function. An obvious drawback of the wide and double pipe construction, however, is a degraded efficiency as the compression function now has larger input and output while keeping the hashing rate constant (the size of the compression function input corresponding to a message block is fixed) since the chaining variable input is increased. Also, adapting the wide and double pipe construction for existing hash functions may be difficult since it might be the only reasonable way to increase the internal state by using multiple compression function
calls in parallel, for every iteration. Recently, Yasuda [24] adopted a slightly modified variant of the double pipe construction and proved its unforgeability beyond the birthday barrier.

2.1.5 The 3C Construction

The 3C construction is the simplest variant of the MD construction that one can obtain to improve its security against multi block collision attack [25]. The 3C hash function processes the intermediate chaining values of the MD construction by maintaining a second internal chaining variable containing a value produced by repeatedly XORing the chaining variables while hashing a message; this variable is then processed in an extra finalisation call to the compression function. To increase the security level of 3C, 3C+ design has been proposed. In the 3C+ hash construction, there is an additional chain called the final chain. The final chain is added to the cascade and accumulation chains of the 3C hash construction. However, in [26, 27], it was shown that both 3C and 3C+ are indeed susceptible for multi-block attack, second preimage and herding attacks.

2.1.6 MDP Construction

The Merkle-Damgård with permutation (MDP), according to Hirose et al. is a simple variant of the original Merkle-Damgård design [30]. The only difference with the Merkle-Damgård construction is that a permutation is applied before the processing of the last message block. MDP construction is shown in the Figure 2.6. Also, recently it was shown that MDP is neither pre-image nor second preimage resistant [31].

![Figure 2.6: Merkle-Damgård with permutation (MDP) construction.](image)
2.1.7 Dither Construction

The dither construction by Rivest is another variant of MD construction which includes an additional counter-like input [34]. The design intention behind the dither construction is to add an iteration-dependent input to the compression function in order to defeat certain generic attacks. The additional input, called the “dithering” input, to the compression function is formed by the consecutive elements of a fixed sequence. This gives the attacker less control over the input of the compression function, and makes the hash of a message block dependent on its position in the whole message. In particular, its goal is to prevent attacks based on expandable messages.

2.1.8 HAIFA Construction

HAsh Iterative FrAmework (HAIFA) is a modified Merkle-Damgård construction proposed by Dunkelman and Biham [35]. It preserved all the good properties of Merkle-Damgård construction. HAIFA modifies Merkle-Damgård by introducing extra input parameters to the compression function. These are: a salt value and the number of bits hashed so far, which thwarts many of the generic attacks against the plain Merkle-Damgård construction. An obvious drawback of HAIFA is efficiency degradation since the compression function now has more input parameters to process. Furthermore, HAIFA cannot be (easily) used to patch existing Merkle-Damgård based hash functions because a compression function designed for the Merkle-Damgård construction would not naturally accommodate the extra HAIFA parameter inputs. The idea is incorporated also in few SHA-3 candidates: BLAKE, ECHO and SHAvite-3.

2.2 Design Methods of Compression Functions

The compression function is the core part of iterative hash function designs. Based on the structure and operational features iterative hash functions are classified into hash function based on block cipher, dedicated hash functions, and hash functions based on modular arithmetic. The block cipher based method is widely preferred due to its efficiency and minimum requirement in hardware and software. In the process of designing hash functions it is desirable to reuse cryptographic components that are already reviewed and established but efficient to implement. Block ciphers could take this place since they provide some appropriate properties and they are well examined. Hash functions are also built from scratch; these are known as dedicated hash functions or customized hash functions. There are number of hash functions built on existing
mathematical components and stream ciphers as well. Cellular automata, Knapsack problem, and Chaos theory are other approaches used in hash functions for compression. These compression methods make hash functions resistant against preimage, second preimage and collision attacks.

2.2.1 Block Cipher Based Hashing

Block ciphers are used as compression functions because they are highly trusted functions that provide practical security and are easy to evaluate. The main idea of using a block cipher as a compression function is the minimization of designing efforts and the implementation costs. Preneel, Govaerts and Vandewalle introduced “PGV-style construction” defining the 64 possible ways of constructing hash functions from a block cipher [55]. It was then reported that 12 out of the 64 PGV constructions are collision resistant, but later Black et al. [56] showed that another 8 PGV constructions are also collision resistant if they were properly iterated, even if their underlying compression functions are not collision resistant.

The main problem of constructing a hash function from a block cipher is the lack of onewayness of the block cipher. Thus, adopting the block ciphers to hash functions some extra operations are needed before and/or after encryption operations. On the other hand, they must execute the key setup function many times, so the performance is very slow. Another approach for constructing the compression function is to use fixed-key block cipher [60, 61] but it has weak collision resistance.

For hash functions based on block ciphers, the amount of data compressed for each application of the block cipher is measured by hash rate. For an iterated hash function based on block cipher, it is defined as number of message blocks processed by one decryption or encryption of the block cipher.

Hash functions based on block ciphers are classified into two categories: single block length hash functions and double block length hash functions. A single block length hash function is a hash function the length of whose output is equal to that of the block cipher. The length of the output of a widely used block cipher is 64 or 128. All the designs with single block length have rate 1. The most widely adopted construction Davies-Meyer [1], Matyas-Meyer-Oseas [57] and Miyaguchi-Preneel [55, 58] are single block length construction have rate 1.
The length of the output of a double block length hash function is twice larger than that of the block cipher. The compression functions of double block length hash functions are classified by the number of encryptions and the key length of the block cipher. Most known examples are MDC-2 (manipulation detection codes with 2 block cipher) [59] and MDC-4 [59] (manipulation detection codes with 4 block cipher). Another compression function, Hirose is described in [62] consists of a block cipher along with a permutation function.

Hohl, Lai, Meier and Waldvogel [63] presented the security of rate $1/2$ construction. In [64], Knudsen et al. studied the security of double block length hash function with rate 1 and found that double block length hash function is not sufficiently resistant against preimage and collision finding attacks. Satoh et al. [65] analyzed the double block length hash functions with the compression functions with one encryption and two encryptions of block cipher. They stated that no effective attacks were found for the double block length hash functions with the compression functions with two encryptions of block cipher.

### 2.2.2 Stream Cipher Based Hashing

The general construction of stream cipher based hash function consist of a stream cipher function and an additional function that inputs a message into the internal state of the stream cipher function. Therefore, a model of stream cipher based hash function consists of a pre-computation function and a stream cipher function. Stream cipher function is the core component of the model and appropriate algorithm is selected from among existing stream cipher algorithms. The pre-computation function is used to absorb the message into the internal state of stream cipher function. Although the stream-cipher based approach is less popular than the block-cipher based approach, in the recent many constructions for new hash functions have been presented as alternatives to the traditional algorithms. The main differences between block cipher based and stream cipher based hash functions are the size of the block and the number of rounds. In block cipher based, the message blocks are usually large, and iterated a small number of rounds, while in stream cipher based, the block size is small, with more rounds. Thus, in block cipher based, a good compression function is necessary but in stream cipher based, even a weak compression function may provide sufficient security.
Panama [178] is the first hash function based on a stream cipher. However, an attack against Panama is proposed, thereafter RadioGatun [180], an improved version of Panama, is proposed and claimed to offer better security than MD4 primitives. In this approach, iterative use of a simple round function and inclusion of input blocks are proposed. After inclusion of all input blocks, the state is updated a number of times without producing any output. Another hash function RC4-Hash [66] is based on the very popular stream cipher RC4. Some candidates of NIST SHA-3 competition such as CubeHash [67] hash functions are also based on stream ciphers.

2.2.3 Modular Arithmetic Based Hashing

The design of these types of hash functions is based on hard mathematical problems. A cryptographic hash function can use modular arithmetic as the basis of its compression function. This allows the reuse of existing implementation of modular arithmetic. Number theory problems such as discrete logarithm problem, factorization problem, expander graphs are used to design these hash functions. Security of such hash function is directly proportional to the hardness of these problems. The purpose of employing modular arithmetic is to save on implementation costs. Hash functions that are based on modular arithmetic can have variable digest length, depending on the size of modulus. An advantage of these schemes is that it is easy to scale the security level by choosing a modulus of appropriate length. A significant disadvantage is that hash functions based on modular arithmetic are very slow, even when compared to other security primitives such as block cipher and stream cipher based constructions. These types of designs are also vulnerable to fixed points and multiplicative attacks. The two most important cryptosystems, based on modular arithmetic are RSA public key cryptosystem [4] and ElGamal cryptosystem [183]. One of the first modular arithmetic based hash function that used discrete logarithm problem as core component was designed by Gibson in [68]. Other examples of hash functions based on discrete logarithm problem are given in [69, 70, 71]. Example of hash function based on the factorization problem is VSH (Very Smooth Hash) presented by Contini et al. in [72]. There also exists some hash functions based on expander graphs [73]. Recent proposal syndrome based hash is another example of provably secure hash functions which is based on decoding problems in the theory of error-correcting codes [74]. Finiasz et al. presented new versions of syndrome
based hash with the same compression function, but different security parameters and an additional final transformation at Ecrypt’07 [75].

2.2.4 Successive Iterative Hashing
Successive iterative hashing known as dedicated hash functions are specially designed from the scratch for the purpose of hashing a plain text with optimized performance and without being constrained to reusing existing system components such as block ciphers and modular arithmetic. These hash functions are not based on hard problems such as factorization and discrete logarithms. Dedicated hash functions have optimal combination on the desired output size. Dedicated hash functions are more efficient than hash functions based on block cipher or stream cipher. The most popular method of designing compression functions of dedicated hash functions is a serial successive iteration of a small step function. Almost all the dedicated hash functions are based on the iterative Merkle-Damgård construction. To reduce the construction efforts, hash functions use block ciphers as their compression function. These designs use Davies-Meyer mode with modular addition as the feed forward. The block cipher involves the iteration of a step function: a oneway transformation of the internal state. The step function is an unbalanced Feistel design that is sourced by an input message word. MD5 [96], SHA-1 [98] and RIPEMD-160 [101] are some examples of dedicated hash functions. More details of such designs are given in section 2.4.

2.2.5 Nonconventional Hashing Approaches
(i) Construction of Hash Functions from Chaos Theory
The recently proposed chaos based hash functions exhibit an attractive design direction. These hash functions are based on chaos theory which is the mathematical representation of dynamic systems. Chaos systems have features of initial value sensitivity, pseudorandom, onewayness and unpredictable orbit, which is also required to build hash function. Hash functions based on chaos theory use chaotic maps. Chaotic maps such as logistic map, tent map, are functions show particular chaotic behaviors. Wong developed a hashing scheme built on the number of iterations of one dimensional logistic map [76]. Xiao et al. presented a oneway hash function based on the chaotic map with changeable parameter [77]. Yi proposed a hash function based on chaotic tent maps [78]. Wang et al. [79] gave a oneway hash function construction based on two dimensional
coupled map lattices. Yang et al. [80] proposed a oneway hash function construction based on chaotic map network. Almost all chaos based hash functions are less efficient than other hash functions due to their inherent complex structure.

(ii) Construction of Hash Functions from Knapsack Problem

Another important class of problems in cryptography is knapsack problem. The knapsack problem is NP-complete problem. Several cryptographic primitives including hash functions have been designed on the Knapsack problem. This can be defined as:

Given \( n \) values \( x_1, \ldots, x_n \) and \( n \) weights, a desired value \( X \) and a maximal weight \( Y \), decide whether there exists a subset \( J \subset \{1, \ldots, n\} \) such that

\[
\sum_{j \in J} x_j \geq X \quad \text{and} \quad \sum_{j \in J} y_j \leq Y
\]

Knapsack problem based hash functions are classified as: additive knapsack based hash functions and multiplicative knapsack based hash functions. Although knapsack based schemes lead to very efficient and parallelizable schemes but their security is questionable despite of the NP-completeness of the knapsack problem. Most of the schemes have been attacked [82] which made knapsack based design approach less attractive.

A notable example of such hash functions is the one proposed by Damgård [15] based on additive knapsack. In the proposal \( n = 256 \) random numbers \( r_j \) of \( l = 120 \) bits are chosen and each message \( M = m_1 \parallel \ldots \parallel m_n \) is compressed to a value \( f(r_1) \parallel \ldots \parallel r_n, M) = \sum_{j=1}^{n} r_j m_j \), where addition is modulo \( 2^n \).

Damgård’s proposal was attacked by Camion and Patarin in [82] and later it was cryptanalysed by Patarin in [83]. In [84] Ajtai proposed a significant improvement to knapsack based hash functions. Ajtai showed that random instances of some lattice problems are as hard as the hardest instances of some other problems. The multiplicative knapsack based hash function was broken in [85].

(iii) Construction of Hash Functions from Cellular Automata

Cellular automata (CA) are dynamical systems in which space and time are discrete. A CA consists of an array of cells organized in a grid, each of which can be in one of a finite number of
possible states, updated synchronously in discrete time steps, according to a local and identical interaction rule. Each cell can have a value 0 or 1. The state of a cell at the next time step is determined by the current states of a surrounding neighborhood of cells. Cellular automata were originally conceived by Ulam and von Neumann to provide a formal framework for investigating the behavior of complex, extended systems, that is, its main goal was to design self replicant artificial systems that are also computationally universal [86, 87]. Wolfram’s work played a significant role in this area [88]. Several proposals of hash functions based on cellular automata have been appeared in [15, 89, 90, 91, 92, 93].

2.3 Attack Methods against Hash Functions

Three classical security requirements for a hash function are collision resistance, preimage resistance and second preimage resistance. An attack is an attempt to violate one or more of core security properties. If any of the three security properties, preimage, second preimage or collision, can be found with an effort less than $2^n$, $2^n$ and $2^{n/2}$ (for $n$-bit hash function) then hash function can be considered to be weak. Attacks on hash functions can be classified into two types. First of them is generic attacks which are attacks on construction methods, where the underlying primitive is assumed to be secure in all respects. It mainly exploits the weaknesses of the hash functions in a general way. Generic attacks are applicable to almost every hash function. Another attack is an attack that exploits the structural weakness of underlying primitive is called shortcut or specific attack. Shortcut attacks are cryptanalysis methods for specific hash function. In this section, we describe different types of attacks.

2.3.1 Attacks Independent of Hash Functions

(i) Brute force attack

A brute force attack method can be used to calculate preimages and collision for a hash function. Brute force attack works on all hash functions independent of their structure and any other working details. In brute force preimage attack, for a regular $n$-bit hash function $h$, the attacker evaluates $h$ with every possible input message until he obtains the given value $h(M)$, where $M$ is a message. In brute force second preimage attack, for a given message $M$ and the hash function $h$, the adversary evaluates $h$ with every possible input message $M' \neq M$ until he obtains the value
In brute force collision attack for a given hash function \( h \), the attacker tries to find two messages \((M, M')\) such that \( M \neq M' \) and \( h(M) = h(M') \). For a hash function, it requires for attacker about \( 2^n \) computations to find preimage and second preimage and \( 2^{n/2} \) for collision due to Birthday paradox discussed next.

(ii) **Birthday Attack**

The birthday attack is the most widespread attack on cryptographic hash functions. This attack describes the expected number of random messages that must be tested before a preimage or collision is discovered with a probability greater than 50%. This attack is based on birthday paradox.

According to the birthday paradox, for a group of 23 people, the probability that at least two persons were born on the same day is larger than 1/2. The birthday attack was first given, in relation to hash function by Yuval [40]. Quisquater and Delescaillie [41] converted the collision problem to the problem of detecting cycles in an iterative mapping. In [42] van Oorschot and Wiener introduced a parallel version. Moreover, a more generalized version of the birthday attack has been given by Mckinney in [43] which provides collision in more than two messages.

2.3.2 **Attacks Dependent to Hash Functions**

(i) **Meet In the Middle Attack**

Meet in the middle attack is a variant of the birthday attack. It breaks the preimage resistance of hash functions. It enables to construct a message whose hash is same with a given one. A meet in the middle attack searches for collisions of intermediate data. In this attack, chaining values are compared rather than hash values. If the compression function is invertible, then meet in the middle attack is feasible. It applies the compression function to \( 2^{n/2} \) time random messages and apply it backward to \( 2^{n/2} \) time other random messages. By the birthday paradox, there is a large probability that the attacker finds a common value in the middle i.e. preimages can be computed by extending the birthday attack. The memory free version of this attack has been given in [44].

(ii) **Differential Attack**

Differential attack is perhaps the most famous and seminal of all attacks. The premise is simple:
the attacker chooses two plaintexts with a known and fixed difference between them, and sends both plaintexts through the cipher to obtain two ciphertext outputs. Attacker then compares the two ciphertexts to find their difference, and keeps track of the result. After doing this for many plaintexts, subtle non random patterns will emerge and the attacker will be able to gradually recover bits of the key. It is also applicable on hash functions. Most collision attacks on customized hash functions are based on differential cryptanalysis, which originates from block ciphers. The idea is to consider a differential characteristic, a sequence of differences in the internal state throughout the iteration of the compression function. If the trail ends with zero difference then any message pair, such that the two iterations provide the desired difference during the iteration, gives a collision.

### 2.3.3 Attacks on Iterative Construction

(i) **Fixed Point Attack**

It is stated by Dean [46] that for an iterative hash function, if the fixed points of compression function can be calculated easily then finding second preimage is easier than expected. In a hash function, fixed points occur when the intermediate hash value does not change after digesting a given message block. Essentially, the hash value hashes onto itself. Dean’s attack accurately suits for designs based on Davies-Meyer block cipher construction, because it is easy to find fixed points in this type of block cipher construction.

(ii) **Length Extension Attack**

Length extension attack is one of the simplest attacks on iterative Merkle-Damgård construction. The motive of length extension attack is to produce a hash value which is related to or contains a part of a message $M$ without fully knowing it. In this attack an adversary is able to extend the length of an unknown message by computing the hash of the concatenation of the unknown message and a suffix.

(iii) **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]. Multicollision set consists of messages that all hash to the same value. The idea is to build collisions one after another, which
leads to a set of $2^k$ colliding messages after only $k$ trials of the collision search. If a hash function has an iterative structure, the attack can be always maintained. The actual complexity, however, depends on the size of the internal state.

(iv) **Long Message Second Preimage Attack**
In [47] Kelsey and Schneier generalized the Dean’s fixed point attack. They showed that there exists a generic second preimage attack on an $n$-bit iterated hash functions with the Merkle-Damgård construction, regardless of the compression function used. The key idea is to use so-called expandable messages. An $(a,b)$-expandable messages is a multicollision between messages of lengths $a$ and $b$ blocks. As soon as the attacker finds two collided expandable messages of different length, collisions with messages of the same size can be derived fast.

(v) **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.

Nostradamus attack, a non-standard type of attack [48] is a chosen target forced prefix attack (CTFP) based on herding attack. It is the use of herding to commit to the hash of a message that the attacker doesn’t even know. This destroys the ability to use hashes, for which collisions can be found, to prove prior knowledge of any information. Merkle-Damgård approach is also vulnerable to the Nostradamus attack

(vi) **Slide Attack**
Slide attacks have mostly been used for block cipher cryptanalysis. The slide attacks also form a potential threat for a certain class of hash functions, e.g., sponge-function like structures. In
Biryukov and Wagner presented slide attacks that exploit highly repetitive and cyclical round functions. The attack is particularly devastating against ciphers that use the same subkey in every round. Even if a small set of subkeys is used in a cyclical way, the slide attack can break the cipher faster than brute force. But if the cipher had other variables that changed several times, each change would help to thwart a slide attack. For instance, the XTEA cipher [51] was designed to be resistant to slide attacks.

(vii) Rebound Attack
The rebound attack was proposed by Mendel et al. in [52] for the cryptanalysis of AES [181, 182] based hash functions. The rebound attack uses truncated differences and is related to the attack by Peyrin [53] on the hash function Grindahl [54]. It is a differential attack, using several new techniques to improve upon existing results. The advantage of rebound attack is that it can maximize the effect of message modification. However, for most of the hash functions that are not using Matyas-Meyer-Oseas-like compression function structure, it would be difficult to develop a rebound attack into a collision attack.

2.4 Dedicated Hash Functions
In current security applications, there are two widely used cryptographic hash functions: MD5 [96] and SHA-1 [98]. These are examples of dedicated hash functions, that is, hash functions that were especially designed for hashing purpose only. Other examples of dedicated hash functions are MD2 [94], MD4 [95] and MD5 [96] (the MD-family), SHA-0 [97], SHA-1 [98], SHA-2 [99] and SHA-3 [100] (the SHA-family), RIPEMD [165], RIPEMD-128, RIPEMD-160 [101], RIPEMD-256 and RIPEMD-320 (the RIPEMD-family), FORK-256 [102] and NewFORK-256 [103] (the FORK-family), TIGER [104], HAVAL [105] and Whirlpool [106].

Most of dedicated hash functions are designed by using 32-bit operations for high speed software implementations on 32-bit processors, which makes them very popular, even though their security is only based on heuristic arguments. They are expected to be practically impossible to invert and there must be virtually a unique correspondence between a given message and its hash. None of the desired properties of cryptographic hash functions can actually be proven for them. Surprisingly, recent results [107, 144, 147] have shown that some sophisticated and computation
intensive techniques can actually lead to find preimages and collisions for the hash functions used in current applications and standards. The studies on the vulnerabilities of the SHA-2 standard, in particular, are taking on a decisive role in security research. In fact, due to the numerous breaches discovered in MD5 [96] and SHA-1 [98], SHA-2 is currently the only alternative to these weak hash functions in many standards. A deeper understanding of the robustness and weaknesses of SHA-2 is thus of paramount importance. Although between 2003 to 2008, many attacks [108, 109, 110, 111, 112] on the reduced round of SHA-2 are published, but no result gives any threat to the security of SHA-2. In the mean time NIST announced SHA-3 [100] competition in 2007. All hash functions submitted for the SHA-3 competition are divided on the following broad category: balanced feistel network, unbalanced feistel network, wide pipe design, key schedule, MDS matrix, output transformation, S-box and feedback register. NIST announced Keccak [161] as the winner of the competition. But it is still an important issue to analyze the hash function based on the weak design principle of MD4.

In this section we discuss the most widely used dedicated hash function families. It covers the short descriptions of hash functions and survey of attacks on these functions.

2.4.1 MD Family
In MD-family is series of cryptographic hash functions: MD2, MD4 and MD5 where MD stands for “message digest”. MD1 and MD3 have never been published; MD5 is merely an extension of MD4. MD-family was developed by Rivest. All three functions generate 128-bit hash values.

MD2 was designed for security of RSA in 1988. It has been published in RFC 1319 [94]. It produce message digest of 128-bit length. MD2 was non-conventional byte oriented design. The function was designed to hash byte streams, and optimized for 8-bit machines. MD2 uses a checksum of 128-bit computed from the whole message and appended as the last input block of the compression function. Hence it does not follow Merkle-Damgård design. Its block size is 16 bytes. The first attack against the full MD2 hash function was a preimage attack published by Muller [114] in 2004. The attack is divided into two parts: in the first part one finds many preimages of the compression function and in the second part one finds those preimages which conform with the checksum function. The complexity of the attack is about $2^{73.6}$ evaluations of
the compression function. This attack was improved by Knudsen and Mathiassen in [115]. They have found a pseudo collision along with a preimage in MD2. Furthermore, they have shown multicollisions for the compression function and a pseudo preimage attack. The multicollision attack is expected to generate eight messages and has a complexity of $2^{72}$. Pseudo collisions can be found with a complexity of $2^{16}$, but it is important to note that the checksums for both messages is equal only because the messages were both fixed to 0 and two different intermediate hash values $H$ and $H'$ were calculated, resulting in the pseudo collision. The efforts are $2^{95}$ for the pseudo preimage attack, and $2^{97}$ for the preimage attack depending on the desired message length. Moreover they were always able to find many preimages for one given hash value. As a consequence of these attacks, MD2 can no longer be considered a secure one way and collision resistant hash function.

MD4 [95] was published in 1990 which takes at most $2^{64}$ bits of input, as its padding rule permits, and produces 128-bit fingerprint of it. It is built on the Merkle-Damgård principle. MD4 takes message blocks of 512 bits and produces 128-bit output. Soon after the publication of MD4, it was realized that the design of MD4 represents an uncomfortable compromise between security and speed. In [116] Boer and Bosselaers described an attack against the last two rounds of MD4. Merkle described an attack against the first two rounds but the work was not published. Vaudenay [117] described another attack against the first two rounds of MD4. In 1996, Dobbertin gave a collision attack on MD4 which finds a collision with probability $2^{22}$ [118]. Dobbertin also showed that the first two rounds of MD4 are not one-way [119]. He found preimages in the hash function with a complexity of $2^{32}$ compression function calls. More recently, Wang et al. found a very efficient collision attack [120] on MD4, which was improved by Sasaki et al. [121]. Yu et al. presented a second preimage attack on MD4 [122]. However this kind of attack is not what we usually call a second preimage attack because it only works for a small subset of the message space. This attack has a complexity of one compression function, but it works only with probability $2^{56}$ and cannot be repeated when it fails. Yu and Wang presented a new type multicollision attack on the compression function of MD4 [123]. Due to all these attacks MD4 is no longer used as a collision resistant hash function.
After attacks on MD4, Rivest realized that design of MD4 is sacrificing security over speed. He proposed a slower but more strengthened extension of MD4 namely MD5 in 1992 [96]. It is specified in RFC 1321. It also takes a message of arbitrary length as input and produces a 128-bit hash value. MD5 is very similar to MD4 in many aspects. It uses the same message padding rule. Block size and initialization of chaining variables are also same. MD5 has an additional fourth round. In the second round the majority function \((B \land C) \lor (B \land D) \lor (C \land D)\) has been replaced by the multiplexer function \((B \land D) \lor (C \land \lnot D)\). Now it has a multiplexer function in the first and second round. A new function \((B \lor \lnot D) \oplus C\) is introduced for the fourth round. The order in which the message words are used has been changed, and 16 different rotation amounts were introduced to avoid certain collision and preimage attacks and to improve the diffusion quality. Each of the 64 steps (16 steps per round) has a unique additive constant. Each step adds in the result of the previous step so that changes made to one chaining variable will propagate quickly to later calculations. Several attacks have been discovered on MD5. In 1993 Boer and Bosselaers found pseudo-collision for MD5 with two different \(IVs\) [124]. In 1996, Dobbertin published an attack that found a collision in MD5 with a chosen \(IV\) that is different from \(IV\) of MD5 [125]. Since proposed attack does not use original \(IV\), it is not considered as real attack for MD5. However he was unable to extend the collision to the full MD5. In 2004, Wang et al. announced first MD5 collision as well as collisions in other hash functions [126], but they have not discussed attack strategy in detail. Several researchers have tried to explain the method used by Wang and team to break the collision resistance of these hash functions [127, 128]. Finally in 2005, Wang et al. described their attack on MD5 in detail [129]. They presented their differential path and a set of sufficient conditions for this differential path to happen. They also introduced message modification techniques to efficiently find message blocks for which the conditions hold. Their original attack can find collisions for MD5 in about fifteen minutes up to an hour with a computational efforts equivalent to about \(2^{39}\).

In [130, 131] better message modification techniques have been published than Wang’s method. An improved attack is presented by Klima in [132]. This attack could find collisions on MD5 in less than a minute on a regular notebook. These faster attacks use techniques based on tunnels. A more powerful collision attack called a chosen-prefix collision attack was introduced in 2007.
Using a single-block message identical-prefix collisions have been reported by Xie and Feng in [134].

NIST had launched a public competition with the aim of identifying a new standard for cryptographic hashing (SHA-3). Besides a high security level, candidate algorithms should show good performance on various platforms. MD6 is one of the earliest announced SHA-3 candidates, designed by Rivest and team [135]. It accepts input messages of any length up to $2^{64} - 1$ bits, and produces message digests of any desired size from 1 to 512 bits, inclusive, including the SHA-3 required sizes of 224, 256, 384, and 512 bits.

### 2.4.2 SHA Family

Two years after the publication of MD5, NIST published SHA-0 (SHA-0 was originally simply SHA) in 1993 [97]. The first revision to this algorithm was published in 1995 due to an unpublished flaw found, and was called SHA-1 [98]. In 2001, SHA-2 [99] was produced by the NSA and standardized by NIST in order to anticpate the potential cryptanalysis results and also to increase its security with regard to the fast growth of the computation power. All standards from SHA-0 to SHA-2 follow similar design principles as Rivest’s algorithms. Several cryptanalytic results against MD-family and SHA-0/1/2 had led to the initiation of the NIST hash function competition in 2007 to develop a new hash standard, called SHA-3 [100]. NIST announced Keccak [161] as the winner of the competition.

SHA-0 (Secure Hash Algorithm) is published as a Federal Information Processing Standard (FIPS) by U.S. National Security Agency in 1993. SHA-0 inspired from MD Family. Like MD-family hash functions SHA-0 takes 512-bit message blocks (at most $2^{64}$ bits of message) as input and outputs however 160 bits hash value instead. Output is the concatenation of 5 words of 32 bits. This raises the complexity for a brute-force collision search from about $2^{64}$ hash function computations to about $2^{80}$. SHA-0 function uses the Merkle-Damgård principle for extension domain and their compression function, even if considered conceived from scratch, is built upon a dedicated block cipher in Davies-Meyer mode: the output of the compression function is the output of the block cipher with a feed-forward of the chaining variable. The first published attack on SHA-0 has been proposed by Chabaud and Joux in 1998 [137]. It focused on finding linear differential paths composed of interleaved 6-step local collisions, which have probability 1 to
hold in a linearized version of SHA-0. However, in the standard version of SHA-0, a local collision only has a certain probability to hold. The overall probability of success of the attack is the product of the holding probability of each local collision.

It is a differential attack that uses a weakness of the expansion algorithm of SHA-0. It had a complexity of about $2^{61}$ hash function computations, which is better than the brute force method of about $2^{80}$. In 2004, Biham and Chen found two near-collisions of the full compression function of SHA-0 [138]. They showed that in SHA-0 near collisions are easy to find than full collisions. The hashes differ by only 18 bits; 142 bits out of 160 bits are equal. In [139] a collision for the full SHA-0 algorithm with complexity of $2^{51}$ was announced by Biham, Chen, Joux, Carribault, Lemu et and Jalby. This was done by using a generalization of the Chabaud and Joux attack. A collision attack by Wang et al. in [140] could find a collision with complexity $2^{39}$, which is within practical reach. In 2006, Naito et al. [141] improved the attack by reducing the complexity to an estimated $2^{36}$ hashes by using a technique submarine modification. A boomerang attack was introduced by Manuel and Peyrin in 2007 that can find a collision on SHA-0 in about one hour [142]. It took only five years for the initial SHA function to be broken.

SHA-1 was issued by the NIST in 1995 as a Federal Information Processing Standard [98] as a new and more reliable function to be used in cryptographic applications. It is also based on similar design principles as MD4/5 and overcomes the vulnerabilities then identified for its short-lived predecessor SHA-0. Since its publication, SHA-1 has been adopted by many governments and industry security standards, in particular standards on digital signatures for which a collision resistant hash function is strictly required. SHA-1 is very similar to SHA-0. SHA-0 and SHA-1 only differ in the way the 80 words of the expanded message are generated. Message expansion in SHA-1 just makes use of a simple rotation.

Researchers also found weaknesses in SHA-1. In early 2005, Rijmen and Oswald published an attack on a reduced version of SHA-1, 53 out of 80 rounds, which find collisions with a complexity of fewer than $2^{80}$ operations [143]. After different cryptanalysts found several attacks on reduced versions of SHA-1, Xiaoyun Wang and her colleagues presented a collision finding method [144] for the standard SHA-1 80-step version with a complexity $2^{69}$ lower than the theoretical bound of $2^{80}$, using the nonlinear characteristics. Although this complexity is still out
of reach but these results significantly influenced subsequent attacks against SHA-1. Soon, improvements to these attacks were announced in [145] where the attack complexity has been reduced to $2^{63}$. Matusiewicz and Pieprzyk presented attack on SHA-1 in [146]. Canniere and Rechberger [147] described a way to automatically find complex nonlinear characteristics and used it to determine a two block colliding message pair for a weakened 64-step version of SHA-1. Canniere et al. then presented a collision for a 70-step version in [148], while an equivalent result was obtained by Manuel and Peyrin in [142]. In 2008, Canniere and Rechberger also showed preimages for reduced SHA-0 and SHA-1 [150].

In addition to the SHA-0/1 hash functions, the NSA and NIST also published a set of more complex hash functions to incorporate the need for hashes that offer more security, especially longer hash values ranges from 224-bit to 512-bit. These hash algorithms, called SHA-224, SHA-256, SHA-384 and SHA-512 (referred to as SHA-2) [99] are more complex because of the added nonlinear functions to the compression function. Their suffix originates from the bit length of the hash value they produce. The versions SHA-224 and SHA-384 are obtained by truncating the result from SHA-256 and SHA-512 respectively. Fortunately, the SHA-2 hash functions produce longer hashes, making a feasible attack more difficult. Consider for example the SHA-512 hash function, producing 512 bit hashes and thus having an approximate complexity against collision attacks of 2256. Even if the logarithmic complexity would be halved (from 256 to 128), this would still be out of reach for practical purposes for the coming decade or so. SHA-256 uses a block size of 512-bit, and iterates 64 steps, while SHA-512 uses a 1024-bit block size and has 80 steps. Furthermore, SHA-512 uses an internal word size of 64-bit instead of the 32 bit used by all other SHA variants. The SHA-2 algorithms follow the same structure of message expansion and iterated state update transformation as SHA-1, but both message expansion and state update transformation are much more complex. The first known cryptanalysis of the SHA-2 family was published by Gilbert and Handschuh [108]. Analysis of message schedule determines limits on the probability of collision for SHA-2. They have shown 9-step local collisions which hold with a probability of $2^{-66}$. Hawkes et al. [151] have improved these results to get local collisions with a probability of $2^{-39}$ by considering modular differences. In [109], Mendel et al. have analyzed how collision attacks can be applied to step reduced SHA-256. They have shown that the properties of
the message expansion of SHA-256 prevent an efficient extension of the techniques of Chabaud and Joux [137] and Wang et al. [144]. Nevertheless, they presented a collision for 18 steps of SHA-256. In [152], Sanadhya and Sarkar have revisited the problem of obtaining a local collision for the SHA-2 family, and in [153] they have shown how to use one of these local collisions to construct another 18-step collision for SHA-256. Nikolic and Biryukov [110] found a 9-step differential using modular differences which can be used to construct a practical collision for 21 steps and a semi-free-start collision for 23 steps of SHA-256. This was later extended to 22, 23 and 24 steps by Sanadhya and Sarkar in a series of papers [154, 155, 112]. In [156] 27 steps collision for SHA-256 found.

The recent results on MD4/5 and SHA-0/1 along with the fact that the SHA-2 family of hash functions was designed with a similar structure have led to the initiation of the NIST SHA-3 competition. The SHA-3 hash function must allow for message digests of length 224; 256; 384 and 512 bits, it should be efficient, and most importantly it should provide an adequate level of security. NIST received 64 new hash function proposals, of which 51 were accepted as meeting the submission criteria for the first round. In the second round, 14 candidate hash functions were in the race. Five candidates were selected among the 14 competitors to advance into the third round. In final round there were only 5 candidates, Blake [157], Groestl [159], JH [160], Keccak [161], and Skein [162]. Finally, Keccak was crowned with the title SHA-3.

2.4.3 RIPEMD Family
As a new version of MD4 the first RIPEMD hash function was designed in 1992 under the European RIPE (RACE Integrity Primitives Evaluation) project [163]. It is a function that produced a 128-bit hash value and had its design based on the MD4 algorithm. The rotations and the order of the message words are modified to decrease vulnerability against previous attacks. In distinction to MD4, two instances of the algorithm, that only differ in the constants, are run in parallel, but with the same input. After processing a 512-bit block, both chaining variables are combined together with the initial chaining variables. Later two strengthen versions of RIPEMD are released, RIPEMD-128 and RIPEMD-160. RIPEMD-128 also produces 128-bit message digest as its predecessor. Both RIPEMD-128 and RIPEMD-160 are extended to RIPEMD-256 and RIPEMD-320 respectively. RIPEMD-160 was designed by Dobbertin, Bosselaers and
Preneel in 1996 as a replacement for RIPEMD [101] and is part of the international standard ISO/IEC10118-3:2004 on dedicated hash functions. It is an iterative hash functions based on the Merkle-Damgård design principle and produces a 160-bit hash value by processing message blocks of 512 bits. Like its predecessor RIPEMD, the compression function of RIPEMD-160 consists of two parallel streams. The two streams of RIPEMD-160 are designed more differently than those of RIPEMD. In each stream the expanded message block is used to update the state variables. After the computations the results of both streams are combined with the chaining input.

Dobbertin found a collision attack on two rounds of RIPEMD with complexity about $2^{31}$ hash computations [164]. The basic idea of the attack is to find an inner collision for the compression function using a very simple input differential pattern. In 2004, Wang et al. presented collision attacks on MD4 and RIPEMD. The attack on RIPEMD has a complexity of about $2^{18}$ hash computations [120]. The basic idea of all attacks is to use differences in more than one message word to find an inner collision within a few steps in the last round and then find a suitable characteristic for the remaining steps.

Even though RIPEMD-160 relies on the same design principles as MD5 and SHA-1, the dual-stream structure makes RIPEMD-160 more secure against recent attacks on other members of the MD4 family. For this reason, no serious result on RIPEMD-160 has been published to date. The work regarding the collision resistance of RIPEMD-160 has been published by Mendel et al. in [165]. In this work, the application of the differential attacks on RIPEMD has been studied. However, due to the increased number of steps and the two streams are more different than in RIPEMD, they concluded that RIPEMD-160 might be secure against these types of attacks. The best currently known attack on the hash function RIPEMD-160 is a preimage attack for 31 out of 80 steps by Ohtahara et al. [166]. However, the complexity of the attack is very close to the generic complexity of $2^{160}$. Recently, Sasaki and Wang [167] have shown non random properties for up to 51 steps when starting from round 2. However, the complexity of the attack is very high $2^{158}$ and the attack setting is much weaker than in a collision attack. In [168], Mendel et al. provided the first analysis of unmodified RIPEMD-160 against collision attacks. They applied collision attacks of Wang et al. on up to 3 rounds of the RIPEMD-160 compression function.
They presented semi-free-start collisions for 36 steps and semi-free-start near-collisions for 48 steps out of 80 steps when starting at round 2. In summary, due to the dual-stream parallel structure of RIPEMD-160 the collision or preimage finding attacks are difficult.

2.4.4 FORK Family

The hash function FORK-256 [102] was introduced at the first NIST hash workshop and at FSE 2006. Later, in 2007 the same team of researchers has published its improved version NewFORK-256 [103]. In this new version they modified step operations, removed some additions and XORs and changed non-linear operations of FORK-256. The compression function of FORK-256 and NewFORK-256 consists of four independent branches. Each one of these branches takes in the 256-bit chaining value and a 512-bit message block to produce a 256-bit hash result. These four branch results are combined with the chaining value to produce the final compression function result. Both algorithms are entirely built on shift, XOR, and addition operations on 32-bit words. The four branches are structurally equivalent, but differ in scheduling of the message words and round constants.

Matusiewicz, Contini, and Pieprzyk attacked FORK-256 by using the fact that the functions \( f \) and \( g \) in the step operation were not bijective. They used microcollisions to find collisions of 2-branch FORK-256 and collisions of full FORK-256 with complexity of \( 2^{126.6} \) [169]. Independently, Mendel, Lano, and Preneel [170] published the collision-finding attack on 2-branch FORK-256 using microcollisions and raised possibility of its expansion. At FSE 2007 [188], Matusiewicz, Peyrin, Billet, Contini, and Pieprzyk presented another attack which finds a collision with complexity of \( 2^{108} \).

NewFORK-256 hash function was introduced in 2007. It includes bijective function in step operation. Markku-Juhani O. Saarinen presented collision attack against NewFORK-256 using meet-in-the-middle technique [171]. For this researcher used a method for finding messages that hash into a significantly smaller subset of possible hash values. The complexity of this collision attack is \( 2^{112.9} \). This attack is also applicable for FORK-256.
2.4.5 Tiger

Tiger is a cryptographic hash function proposed by Anderson and Biham in 1996 [104], and was specially designed to fit 64-bit processors. It is an iterative hash function that processes 512-bit input message blocks and produces a 192-bit hash value. Tiger uses a block cipher based compression function and the Davies-Mayer type feed-forward structure, which is different from the most popular architecture. The 8-bit input and 64-bit output S-boxes of Tiger for the compression function provide faster diffusion in comparison with integer arithmetic, such as addition and subtraction, where a bit flip only propagates to an adjacent bit. Anderson and Biham also demonstrated that software implementation of Tiger provides high-speed performance on 32-bit processors, not only on 64-bit processors. Tiger can be modified to the Tiger-160 or Tiger-128 forms with a 160-bit or 128-bit message digest, respectively.

Different cryptanalytical results of Tiger show weaknesses in round-reduced variants of the hash function. At FSE 2006, Kelsey and Lucks presented a collision attack on 16 and 17 (out of 24) rounds of Tiger [184]. The attack has a complexity of about $2^{44}$ evaluations of the compression function. Furthermore, they present a pseudo near-collision for a variant of Tiger reduced to 20 rounds with a complexity of about $2^{48}$. At Indocrypt, these results were improved by Mendel et al. in [185]. They show that a collision can be found for Tiger reduced to 19 rounds with a complexity of about $2^{62}$ evaluations of the compression function. Furthermore, they present a pseudo-near-collision for Tiger reduced to 22 rounds with a complexity of about $2^{44}$. In [186] Mendel and Rijmen show a pseudo near-collision for the full Tiger hash function with a complexity of about $2^{47}$ hash computations and a pseudo collision (free-start-collision) for Tiger reduced to 23 rounds with the same complexity.

2.4.6 HAVAL

HAVAL was proposed by Zheng, Pieprzyk, and Seberry at Auscrypt’92 [105]. HAVAL is an extension of MD5. HAVAL was proposed with 3, 4, or 5 passes, i.e. 96, 128, or 160 steps. HAVAL is an iterated hash function based on the Merkle-Damgård design principle. The first modification in HAVAL is that the size of both message block and chaining variable is doubled to respectively 32 and 8 words. It has message blocks and hash values twice as large as MD5, i.e. 1024 bits (32 words) and 256 bits (8 words) respectively. The number of rounds can be 3, 4, or 5.
and each round consists of 32 steps. The simple nonlinear functions are replaced by highly nonlinear functions of 7 variables that satisfy some specific properties like the Strict Avalanche Criterion or SAC. Moreover a single function is used in every round, but in every step a different permutation is applied to the inputs. Again a new message order has been introduced, and every step (except for those in the first round) uses a different additive constant. Two rotations over 7 and 11 positions have been introduced. At the end of the algorithm, one can apply a folding operation to reduce the size of the hash value to 16, 20, 24, or 28 bytes. The choice in the number of rounds and in the size of the output yields 15 different versions of the algorithm.

In [172], Rompay, Biryukov, Preneel, and Vandewalle presented collisions for three rounds at Asiacrypt 2003. The attack has a complexity of $2^{29}$ calls to compression function. Because of the structure of HAVAL, the attack produces colliding messages for all hash lengths. A team of Chinese researchers Wang, Feng, Lai, Yu found collisions for three-round HAVAL-128 with with $2^6$ hash computations [126]. They also found collisions for HAVAL-160 with a probability of $2^{-32}$. Then after, security of HAVAL underwent critical analysis [173, 174]. Wang et al. presented a two-block collision computation for 4-pass HAVAL in [175]. The complexity is about $2^{32}$ for the first, and $2^{29}$ for the second block. At FSE 2006, Yu et al. have published two different two block collision attacks on 4-pass HAVAL in [176]. This attack presents two methods, the first on finds the collision in a pair of messages differing in only one message word with a complexity of $2^{43}$ and another finds collision differing in two message words with $2^{36}$ efforts. They have also shown a theoretical collision attack on 5-pass HAVAL with a probability of $2^{-123}$. Aumasson, Meier, and Mendel discovered preimage attack [177] on 3-Pass HAVAL with a complexity of $2^{224}$.

### 2.4.7 PANAMA

Panama is a cryptographic module that was presented at the FSE Workshop in '98 by Daemen and Clapp [178]. Panama can be used as both a stream cipher [158] and a hash function. The Panama hash function maps messages of arbitrary length to a hash result of 256 bits. It is designed for 32-bit architectures; it is as fast as MD4 which is the fastest hash function in the MD family. However, the performance of Panama hash function is not very efficient for the short messages due to large number of iterations in initialization. The security of Panama is analyzed in
and an attack with complexity $2^{82}$ and negligible amount of memory to find collisions faster than birthday attack is proposed.

2.4.8 RadioGatun

RadioGatun family was presented at the NIST Hash Workshop in 2006 [180]. There are 64 hash functions in this family. Its design is not similar to traditional hash functions. It is not a blockcipher based hash function such as the Davies-Meyer construction of compression function and it does not use the Merkle-Damgård design scheme to transform a compression function into a hash function. RadioGatun is built on the sponge paradigm. The RadioGatun hash function is based on the Panama hash function [178]. The main differences between Panama and RadioGatun are; addition of feedforward mechanism from Mill to Belt, smaller input block size and belt width and enlargement of the Mill from 17 to 19 words. Additionally, instead of a compression function, RadioGatun uses an iterative mangling function. RadioGatun has an internal state of 58 words; the size of those words, from 1 to the recommended 64 bits, defines the actual size of the internal state.

2.5 Conclusion

In this chapter, three important aspects of design principles of hash functions; construction methods, attack methods and compression methods are reviewed critically. Iterative methods are an important part of all hash construction designs. Most popular hash function construction method is Merkle-Damgård (MD) iterative method. Generic attacks presented against MD paradigm demonstrated intrinsic structural weaknesses in it. As a result several alternative designs have been derived from Merkle-Damgård construction method such as dither, HAIFA, sponge and MDP. There are several types of hash functions that use different methods to compress the data. Among all successive iterative hashing is widely used. A categorical discussion about the popular attack methods, generic attacks and dedicated attacks is given in section 2.3. Generic attacks are independent from the specification of the hash function and they mainly exploit the weaknesses of the hash functions in a general way. Dedicated attacks focus on structural weaknesses and exploit the weaknesses of the hash algorithm. In section 2.4 we have surveyed most popular hash functions from different families and attacks found on them. Since
introduction of MD4 the design strategy of MD4 has been most popular for designing dedicated hash functions. MD5, RIPEMD, RIPEMD-128/160, HAVAL, SHA-0/1, SHA-256/224, SHA-512/384, FORK-256 are well-known hash functions which follow MD4 design principles. All of them except RIPEMD-160 had been broken. NIST suggested that most widely deployed hash function SHA-1 must be ruled out from security applications and protocols. All these attacks led the NIST to organize a competition to make a new hash standard. Recently Keccak is selected as the SHA-3 standard. Comparison of popular hash functions is given in Table 2.1.

Table 2.1: Comparison of popular dedicated hash functions.

<table>
<thead>
<tr>
<th>Hash function</th>
<th>Hash size</th>
<th>State size</th>
<th>Block size</th>
<th>Message size</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD2</td>
<td>128</td>
<td>384</td>
<td>128</td>
<td>-</td>
</tr>
<tr>
<td>MD4</td>
<td>128</td>
<td>128</td>
<td>512</td>
<td>$2^{64}-1$</td>
</tr>
<tr>
<td>MD5</td>
<td>128</td>
<td>128</td>
<td>512</td>
<td>$2^{64}-1$</td>
</tr>
<tr>
<td>SHA-0/1</td>
<td>160</td>
<td>160</td>
<td>512</td>
<td>$2^{64}-1$</td>
</tr>
<tr>
<td>SHA-256/224</td>
<td>256/224</td>
<td>256</td>
<td>512</td>
<td>$2^{64}-1$</td>
</tr>
<tr>
<td>SHA-512/384</td>
<td>512/384</td>
<td>512</td>
<td>1024</td>
<td>$2^{128}-1$</td>
</tr>
<tr>
<td>RIPEMD</td>
<td>128</td>
<td>128</td>
<td>512</td>
<td>$2^{64}-1$</td>
</tr>
<tr>
<td>RIPEMD-128/256</td>
<td>128/256</td>
<td>128/256</td>
<td>512</td>
<td>$2^{64}-1$</td>
</tr>
<tr>
<td>RIPEMD-160/320</td>
<td>160/320</td>
<td>160/320</td>
<td>512</td>
<td>$2^{64}-1$</td>
</tr>
<tr>
<td>FORK-256</td>
<td>256</td>
<td>256</td>
<td>512</td>
<td>$2^{64}-1$</td>
</tr>
<tr>
<td>Tiger</td>
<td>192/160/128</td>
<td>192</td>
<td>512</td>
<td>$2^{64}-1$</td>
</tr>
<tr>
<td>HAVAL</td>
<td>256/224/192/160/128</td>
<td>256</td>
<td>1024</td>
<td>$2^{64}-1$</td>
</tr>
<tr>
<td>PANAMA</td>
<td>256</td>
<td>8736</td>
<td>256</td>
<td>-</td>
</tr>
<tr>
<td>RadioGatun</td>
<td>Arbitrary</td>
<td>58 words</td>
<td>3 words</td>
<td>-</td>
</tr>
</tbody>
</table>