CHAPTER 1: INTRODUCTION

1.1 INFORMATION SECURITY

The exchange of information is an important social, economic and political activity. The importance of the exchange of information regarding all aspects of life has increased dramatically over the last 50 years. This period has seen the emergence of massive communication systems which have the sole purpose of facilitating the exchange of information. The proliferation of telephone and computer networks (both local and wide area networks) are manifestations of this phenomenon. As both the amount of information being exchanged and the significance attached to that information have increased, the need for ensuring the privacy and authenticity of the information have increased. This is especially true when considering communication systems where no human interaction is required. A computer network is a typical example of such a communication system. The need for privacy and authenticity is especially acute if a breech in either secrecy or authenticity may result in losses (financial or otherwise) for the participating parties in a communication system.

Organisations and people that use communication systems often express their needs for information security and trust in terms of three distinct requirements, frequently referred to as the *CIA-Model* of security:

- *Confidentiality* – ensuring the non-disclosure of sensitive information
- *Integrity* – ensuring that data and information is only changed and modified in a specified and authorised manner
- *Availability* – ensuring that systems work promptly and correctly, and that service is not denied to authorised users.

Over the past three decades numerous techniques have been invented and developed to ensure both privacy, authenticity and availability of information in communication systems. These techniques are often constructed from a number of basic cryptographic primitives. One of the primitives most widely used is the cryptographic hash function.

1.2 HASH FUNCTIONS AND SECURITY

Informally, a hash function is defined as a function that compresses an input string of arbitrary length to an output string of fixed length (see Figure 1.1).

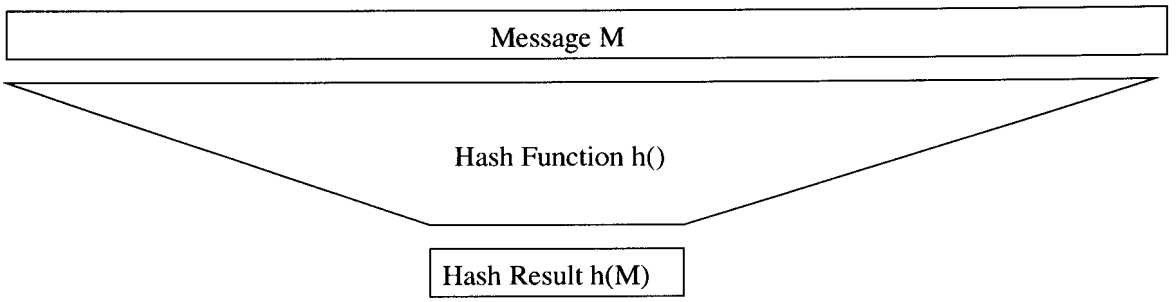

Figure 1.1: Hash Function

More formally the following definition applies to a hash function [1]:

Definition 1.1 (Hash Function) *A hash function is defined as an easily computable function, $h()$, that maps every binary sequence of length μ or greater to a binary sequence of length m, where μ and m are specified parameters.*

A wide range of terminology exists for hash functions used in security applications [2],[3]. These terms include message integrity codes, message authentication codes, manipulation detection codes, cryptographic hash functions or simply hash functions. In this document hash functions used for security applications are referred to as cryptographic hash functions or hash functions for conciseness.

1.2.1 Applications of Hash Functions

Hash functions have their origin in the field of computer science and were originally used for data storage and retrieval [4]. For data storage applications a hash function is used to compute an abbreviated representation of a filename. This abbreviation is then used to index and store the file. When the file is retrieved, the hash value for the given filename is used to retrieve the data. Using a hash function reduces the storage requirements for a data retrieval system, since only the hash value has to be stored, instead of the entire filename.

A number of other uses have been found for hash functions. Amongst others, hash functions are used in compiler symbol tables, graph theory problems, transposition tables in computer games, spell checkers, tests for set equality and security applications [2], [3]. It is the application of hash functions in security solutions that constitutes the topic for this dissertation. A short list of security applications that rely on the use of cryptographic hash functions is shown below:

- Authentication protocols [5].
- Digital signatures [6].
- Electronic commerce [5].
- Encryption schemes [7] [8] [9].

Because of the wide ranging applications of hash functions in security solutions they are considered important cryptographic primitives. The above list of applications can roughly be split into two categories:

- Authentication services.
- Encryption services.

Authentication Applications

The list of security applications primarily represents authentication and non-repudiation services. When using a cryptographic hash function in an authentication scheme the authenticity of the message is transferred to the hash value. It is then necessary to provide protection and authentication for the hash value only, instead of the entire message. The hash value serves as an authentication tag for the message, and can be appended to a message. A number of cryptographic protocols and electronic commerce implementations rely on cryptographic hash functions to provide these services. These protocols include S/MIME, SSL, TLS, WTLS, SET as well as the EMV specifications.

Encryption Applications

The non-linear and one-way property of cryptographic hash functions are exploited when used in encryption schemes. Cryptographic hash functions can be used as non-linear elements in block ciphers based on the Feistel structure. Four block ciphers, LION, BEAR, LIONESS and AARDVARK that are based on the existence of secure cryptographic hash functions were recently proposed in [7], [8] and [9]. It is also possible to exploit the non-linear and one-way properties of a cryptographic hash functions in the construction of stream ciphers.

1.2.2 Properties of Hash Functions

Earlier in this section a hash function is defined as a function that compresses an input string of arbitrary length to an output string of fixed length. The inherent weakness of all hash functions is contained within this definition. An intuitive explanation of the inherent weakness of hash functions is presented in this section.

Consider the projection of an M dimensional space onto an N dimensional space with $M > N$, f is the mapping (hash) function (see Figure 1.2).

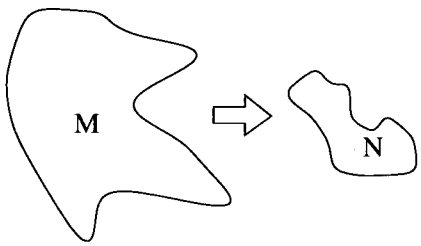

Figure 1.2: Hashing: A Spatial View $f : M \mapsto N$

From Figure 1.2 it is clear that the projection of all possible representations in M onto N is not unique if $M > N$. The lack of uniqueness of the mapping function $f()$ implies that more than one vector in M is mapped to the same vector in N. This is known as a collision. A graphical representation of the above statement is given for $M = 3$ and $N = 2$ in Figure 1.3.

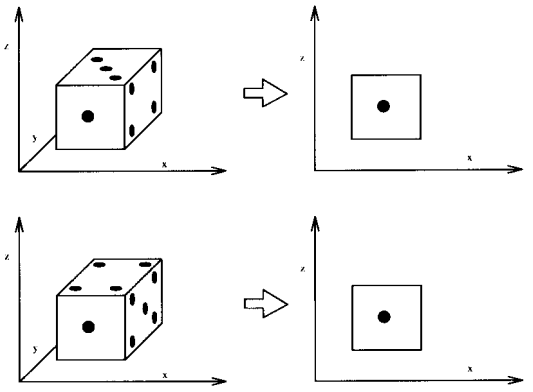

Figure 1.3: Graphical example of a collision

A more formal representation of the above is given below:

$$f() : m_1 \mapsto n_1$$
$$f() : m_2 \mapsto n_1$$

with:

$$\begin{aligned} n_1 &\in N \\ m_1, m_2 &\in M \\ m_1 &\neq m_2. \end{aligned}$$

From the above it follows intuitively that the number of collisions increases as the ratio $\frac{M}{N}$ increases.

This argument leads to the conclusion that collisions exist for all hash functions as previously defined. The existence of collisions for all possible hash functions is an inherent weakness. This weakness is, by definition, also present in cryptographic hash functions. Consequently it is required that cryptographic hash functions exhibit the properties of one-wayness and collision resistance. These properties make it computationally intractable to find collisions.

The properties of one-wayness and collision resistance allow hash functions to be used in order to provide the services of integrity, digital signatures and non-repudiation.

One-way Property

From a cryptographic point of view one-way hash functions are of particular interest. The one-way property is defined in [1] as:

Definition 1.2 (One-Way Hash Functions) *A one-way hash function is defined as a hash function such that, for virtually every binary string y of length m it is infeasible to find a binary string x of length μ or greater such that $y = h(x)$.*

Collision Resistant Property

A further desirable property of cryptographic hash functions is that of collision resistance. The concept of a collision was introduced earlier in this section. It is appropriate to introduce a more formal definition of a collision before defining the concept of collision resistance [59].

Definition 1.3 (Collisions) *A collision is obtained for a given hash function $h()$ if two distinct messages, M and $\tilde{M}$, are found, such that for a specific initial value (denoted by IV):*

$$h(IV, M) = h(IV, \tilde{M}).$$

Consequently a collision resistant hash function is defined as [59]:

Definition 1.4 (Collision Resistant Hash Function) *A collision resistant hash function is defined as a hash function for which it is computationally intractable to find collisions.*

Generally when referring to cryptographic hash functions it is expected that they exhibit the properties of one-wayness and collision resistance.

The One-way Property, Collisions and Information Theory

It can be shown that the presence of collisions is a pre-requisite for one-wayness by applying the principles of information theory and source coding to the hashing problem.In this model

the message to be hashed represents the source, the hash result corresponds to the encoded symbols or messages, and the hash algorithm represents the source encoding algorithm. Let $H(X)$ be the entropy of the message source (X is a random variable). Let each symbol x_i represent a message for $i = 1, 2, 3, \ldots, M$. It is then known that $H(X) \leq \log_2 M$. It is also known that for each symbol to be encoded uniquely with N bits, N is chosen such that $N = \log_2 M$. This implies that M should not exceed 2^N if no collisions are required. However, remember that for cryptographic hash functions:

1. The requirement of one-wayness has to be fullfilled.
2. The input alphabet should have an arbitrary size.

Hash functions used in cryptographic applications have to be one-way. If the hash result corresponds to one, and only one input, the property of one-wayness is violated.

The requirement that the input alphabet should have an arbitrary size implies that $M \geq 2^N$ or $M < 2^N$. If $M < 2^N$ the source encoding algorithm may become inefficient. From a cryptographic point of view this is a minor problem. If $M \geq 2^N$ the probability of an encoding error, P_e, becomes non-zero. If $P_e \neq 0$, collisions exist. If P_e approach 1, the function becomes one-way. In order for P_e to approach 1, $N \leq H(X) - \epsilon$, for any $\epsilon > 0$. By setting N to a fixed size and choosing $M \ggg 2^N$, the above condition is satisfied. Thus the source coding algorithm becomes one-way but produces collisions. Thus collisions exist, not only as a result of the requirement of encoding messages of arbitrary length but also as a result of the requirement for one-wayness. As long as it remains difficult to obtain messages that have the same hash result, the function is called collision resistant.

In order to demonstrate the need for cryptographically secure hash functions consider the following example:

Example

Consider a typical electronic transaction. Two parties agree to the sale of a specified item for R10000,00. An (electronic) contract is drafted. The seller computes the hash value of the contract and applies his digital signature to the hash value. The buyer does the same and the sale is agreed upon. However the parties involved did not utilise a collision resistant hash

function. Consequently the seller was able to draft an alternative contract which has the same hash result as the original, except that the agreed upon price is changed from R10000,00 to R20000,00. The buyer now finds that he is committed to purchase the item in question at twice the agreed upon price.

In the above example the participating parties relied upon the hash function to provide assurance of the data integrity. In effect the message integrity was transferred to the integrity of the hash function. It is shown that the use of a weak hash function compromises the security objective of data integrity. Similar examples pertaining to authentication protocols and encryption schemes may be listed where the failure of the hash function undermines the security objective. For this reason efficient and strong cryptographic hash functions are required.

1.2.3 Hash Functions Today

Hash functions are widely used in cryptographic applications. As demonstrated in the previous section the properties of one-wayness and collision resistance are of particular importance in security applications. During the last decade numerous proposals were made to construct dedicated hash functions that are both one-way and collision resistant. These proposals include MD4, MD5, SHA, SHA-1, HAVAL, RIPEMD-128 and RIPEMD-160. MD4 was published in 1990 by Rivest [10]. By the end of 1991 it was demonstrated that neither the first two rounds (Merkle) nor the last two rounds of MD4 (Bosselaers and den Boer [11]) are collision resistant. The lessons learned from these attacks led to the design of MD5 [12] and SHA [13]. In 1996 Dobbertin showed that MD4 is not a collision resistant hash function by demonstrating a technique which allowed the construction of collisions for all three rounds of MD4 [14]. Within six months Dobbertin demonstrated that collisions may be found for the compress function of MD5 [12]. Although details of this attack have not been published, it is believed to be based on similar techniques as described in [14]. Dobbertin has also shown that these attacks are applicable to RIPEMD-128. The speed with which these attacks could be adapted to different hash functions derived from the same basic construction is a cause for concern since it may be indicative of a fundamental flaw in the design of the basic construction. This concern has led to the design of RIPEMD-160 to replace RIPEMD-128 [15]. In 1998 Dobbertin showed that the first two rounds of MD4 are not one-way. The attacks formulated by Dobbertin utilises techniques borrowed from a wide range of disciplines ranging from genetic algorithms to Boolean algebra. These hash functions are all based on

the same design principles and criteria. The weaknesses found in these hash functions may be indicative of a common design weakness.

As shown above a number of the popular dedicated hash function constructions were found to be cryptographically inadequate. In particular it was found that the requirement for collision resistance is hard to satisfy. One of the reasons for this is the threat model used when considering the property of collision resistance. In this threat model the cryptanalyst not only has full knowledge of the algorithm used (Kerckhoff's principle [59]) but also has control over all aspects of the input to the hash function. The attacker is often a legitimate participant in the system and is trusted to a certain extent. Given the above threat model it should remain computationally difficult to construct collisions or find a specified hash results.

1.3 PROBLEM STATEMENT

Cryptographic hash functions are important cryptographic primitives and are widely used in security applications where message integrity is required. The design of cryptographic hash functions have proved to be a difficult task. A recent spate of attacks showed that a number of commonly used hash functions exhibit cryptographic weaknesses. The absence of secure cryptographic hash functions will make dependable message integrity, non-repudiation and message authenticity impractical. It is therefore important to understand the basis of the attacks, determine if they share common elements and establish design criteria to foil these attacks.

1.4 HYPOTHESIS

It is the hypothesis that the recent spate of attacks formulated by Dobbertin has a common underlying structure and that these attacks exploit certain architectural properties of the MD4 family of hash functions.

1.5 SCOPE

In this dissertation only dedicated, iterated cryptographic hash functions are studied. In particular the MD4 family of hash functions are considered. Although a general review of

generic attacks are included in this dissertation only the attacks formulated by Dobbertin are considered in depth.

1.6 DISSERTATION OBJECTIVES AND METHODOLOGY

This dissertation has the following objectives:

1. Lay a foundation for the analysis and design of cryptographic hash functions.
2. Reconstruct the attacks on MD4 and MD5 as formulated by Dobbertin.
3. Generalise the analysis of MD4 and MD5 to create a framework for the analysis of iterated dedicated hash functions.
4. Apply the generalised analysis framework to practical cryptographic hash functions.
5. Formulate design criteria to prevent the successful application of the generalised analysis framework.

In order to lay a foundation for the analysis and design of cryptographic hash functions we present an in-depth study of the current state of cryptographic hash functions. Included in this study are the definitions (Chapter 1), taxonomy (Chapter 2), generic threats (Chapter 3), common requirements (Chapter 4) and general designs of cryptographic hash functions (Chapter 5). Once a general foundation is laid for the understanding of cryptographic hash functions the focus is shifted to practical dedicated cryptographic hash functions.

As part of the focus on dedicated cryptographic hash functions the attacks on MD4 and MD5 are reconstructed (Chapters 6 and 7). The C-programs used to reconstruct these attacks are attached as Appendix B, C, D and E. This is one of the main objectives of the dissertation. A novel approach is derived that allows the attack on MD4 to be optimised to obtain a reduction in computation time for a collision by a factor 64.

Based on the reconstruction of these attacks a generalised attack is formulated (Chapter 8). The generalised attack provides a framework for the analysis of the collision resistant property of any cryptographic hash function.

The newly derived framework for analysing a cryptographic hash functions is applied to reduced versions of SHA and HAVAL (Chapters 9 and 10). Extensive simulations were performed using the C programming language. A sample of the resulting source code is included as Appendices F and G. To the best of our knowledge this is the first cryptanalytical result that has been published on the HAVAL hash function. The result shows that a collision can be established for a reduced version of HAVAL in less than a minute on a 200 MHz Pentium Pro. This result suggests that three and even four round HAVAL should not be used for security applications where message integrity and non-repudiation is required.

The dissertation is concluded by presenting design criteria for dedicated cryptographic hash functions (Chapter 11). The design criteria is based on the common weaknesses identified in the analysis of MD4, MD5, SHA, SHA-1 and HAVAL. It is the intention that the application of these design criteria will defeat the generalised attack on iterated cryptographic hash functions presented earlier in the dissertation.

1.7 RESULTS

The following are the main results of this dissertation:

1. Successful reconstruction of the attacks on MD4 and MD5 as formulated by Dobbertin.
2. Speedup the attack on MD4.
3. Create a generalised framework for the analysis of iterated dedicated hash functions based on the MD4 family.
4. Apply the generalised analysis framework to reduced versions of HAVAL and SHA.
5. Formulate design criteria to prevent the successful application of the generalised analysis framework.

CHAPTER 2: TAXONOMY OF CRYPTOGRAPHIC HASH FUNCTIONS

2.1 INTRODUCTION

In Chapter 1 the relevant definitions and properties related to hash functions were defined. In this chapter a taxonomy of practical cryptographic hash functions is presented, along with the common approaches to the design and analysis of cryptographic hash functions.

The taxonomy is based on the terminology that exists in the banking community and is taken from [3]. Cryptographic hash functions are divided into the following categories:

1. Message Authentication Codes (MAC).
2. Manipulation Detection Codes (MDC).
   (a) One Way Hash Function (OWHF).
   (b) Collision Resistant Hash Function (CRHF).

A graphical summary of the above taxonomy is shown in Figure 2.1.

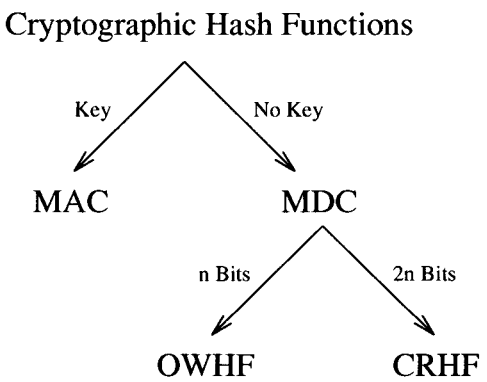

Figure 2.1: A Taxonomy of Cryptographic Hash Functions

Informal definitions for the categories of hash functions are suggested in [16] and refined in [3]. The distinction made between the different cryptographic hash functions is based quantitatively on the following definitions.

2.1.1 MAC

A MAC is a hash function for which a secret key is required. This adds to the security of the hash scheme, since the attacker's abilities decrease as his knowledge decreases. However the requirement for a secret key does not protect the users against an attack by an insider. The addition of a secret key leads to the additional problem of key management. It does however have the advantage that a secure channel[1] is no longer required for the hash value, since the secret key protects the hash value. It is however necessary to provide a secure channel for the key used in the MAC. More formally:

Definition 2.1 *A* **MAC** *is a function h() satisfying the following conditions:*

1. *The description of h() must be publicly known and the only secret information lies in the key, K, (extension of Kerkhoff's principle).*
2. *The argument X can be of arbitrary length and the result h(K,X) has a fixed length of n bits ($n \leq 32 \ldots 64$).*
3. *Given h(), K and X, the computation of h(K,X) must be easy.*
4. *Given h() and X, it is hard to determine h(K,X) with a probability of success significantly higher than 2^{-n}. Even where a large set of pairs $\{X_i, h(X_i, K)\}$ is known, where X_i have been selected by the opponent, it is "hard" to determine the key K or to compute h(K,X') for any $X_i \neq X'$.*

2.1.2 MDC

A MDC is a hash function that is computed without knowledge of a secret key. These functions are known publicly. For these hash functions, no key management is required, but an authentic channel needs to be provided for the hash value.

Two variants of MDCs are identified in [16] and [3]. The following definitions are used to distinguish between one way hash functions (OWHF) and collision resistant hash functions (CRHF).

[1] An authentic or secure channel could be provided through encryption of the hash value, a separate channel or a courier.

One Way Hash Function (OWHF)

Definition 2.2 *A* **One Way Hash Function** *is a function h() satisfying the following conditions:*

1. *The description of h() must be publicly known and should not require any secret information for its operation (extension of Kerkhoff's principle).*
2. *The argument X can be of arbitrary length and the result h(X) has a fixed length of n bits ($n \leq 64$).*
3. *Given h() and X, the computation of h(X) must be easy.*
4. *The hash function must be one way in the sense that:*
   (a) *given a Y in the image of h(), it is "hard" to find a message X such that h(X) = Y.*
   (b) *given X and H(X) it is "hard" to find a message $X' \neq X$ such that $h(X) = h(X')$.*

Collision Resistant Hash Function (CRHF)

Definition 2.3 *A* **Collision Resistant Hash Function** *is a function h() satisfying the following conditions:*

1. *The description of h() must be publicly known and should not require any secret information for its operation (extension of Kerkhoff's principle).*
2. *The argument X can be of arbitrary length and the result h(X) has a fixed length of n bits ($n \leq 128$).*
3. *Given h() and X, the computation of h(X) must be easy.*
4. *The hash function must be one way in the sense that:*
   (a) *given a Y in the image of h(), it is "hard" to find a message X such that h(X) = Y.*
   (b) *given X and H(X) it is "hard" to find a message $X' \neq X$ such that $h(X) = h(X')$.*

5. *The hash function must be collision resistant: This means that it is hard to find two distinct messages that hash to the same result.*

The nature of the differences between OWHF and CRHF is discussed in [17]. The underlying difference between OWHF and CRHF is related to the the type of attack the respective hash functions are required to withstand. For cryptographic purposes a CRHF is of greater value than an OWHF.

Implicit to the above definitions are the requirements for one-wayness, computational intractability, collision resistance and simplicity. These requirements are related to both the functional and security properties of cryptographic hash functions.

A new hash function should therefore be designed to adhere to the above definitions and implied requirements. The definitions and requirements can be made more formal by specifying quantitative criteria for the terms *hard* and *easy*.

2.2 APPROACHES TO THE DESIGN AND ANALYSIS OF CRYPTOGRAPHIC HASH FUNCTIONS

Two approaches could be considered for the analysis, design and classification of cryptographic hash functions. Since hash functions are used extensively in authentication applications and protocols [17], hash functions could be classified along the same lines as authentication codes. In [18] the following classification is given for authentication schemes:

1. Computationally secure.
2. Provably secure.
3. Unconditionally secure.

The above classification is not satisfactory when dealing with hash functions. As explained in Chapter 1 collisions exist for all hash functions. This property of hash functions leaves only the computationally secure classification as a viable option. The above classifications does not contribute a great deal to design criteria for cryptographic hash functions.

In [3] Preneel suggests that the same classification scheme be used as that proposed by Rueppel for stream ciphers. Accordingly one of three approaches are available:

1. Information theoretic approach.
2. Complexity theoretic approach.
3. System based or practical approach.

The information theoretic approach and complexity theoretic approach yields interesting constructions of variable security. In general the constructions based on these two approaches are impractical. This leaves the system based or practical approach. In the system based approach, practical schemes with fixed parameters and dimensions are studied.

There has been numerous proposals for the design of cryptographic hash functions based on the system based approach to hash functions. Many of these designs are based on existing cryptographic primitives such as block and stream ciphers. Other proposals utilise modular arithmetic and the hardness of number theoretical problems as a basis for design. However the hash functions which have found the widest acceptance in industry are dedicated hash functions. The following definition of a *d*edicated cryptographic hash function is presented:

Definition 2.4 (Dedicated Cryptographic Hash Functions) *A dedicated cryptographic hash function is a hash function which has been designed to meet the requirements set for cryptographic applications and is to be used explicitly for hashing purposes.*

One family of dedicated hash functions, known as the MD4-family of hash functions, has found widespread acceptance in industry. Members of this family of dedicated hash functions are used by the Secure Electronic Transaction (SET) protocol specified by Mastercard and Visa, Secure Socket Layer (SSL) protocol commonly used for securing Internet commerce as well as the Secure MIME (S/MIME) protocol used to secure electronic mail to name a few of the more popular protocols.

In this dissertation the practical approach is used to analyse dedicated cryptographic hash functions and establish suitable design criteria for dedicated cryptographic hash functions.

CHAPTER 3: THREATS AGAINST HASH FUNCTIONS

3.1 INTRODUCTION

Before proceeding to establish requirements for hash functions, it is appropriate to consider the threats against hash functions. In this chapter both the attackers as well as the attacks they are capable of are considered.

Attackers are classified with regard to their capabilities and their position regarding the system under attack. As the wealth and resources of an opponent increases, the difficulty of designing a secure hash function increases. For this reason it is important to be aware of the capabilities of various classes of attackers. When designing a hash function it should be decided which class of attacker is to be denied a successful attack.

In addition to the attackers and their capabilities, the attacks they are capable of are considered. A taxonomy of these attacks are presented in this chapter. For the attacks described in this chapter, the computational power and storage capabilities required for the execution of these attacks are emphasised. These requirements are stated as a function of the number of bits, n, contained in the hash length. In this report the attacks specific to MDCs, MACs and hash algorithms based on block ciphers are considered.

3.2 TAXONOMY OF ATTACKERS

A distinction is made between the capabilities of attackers and their position with regard to the hash function they seek to attack.

3.2.1 Capabilities

The capability of an attacker is measured in terms of the resources available to him. In [19] a taxonomy of attackers on tamper resistant devices is presented. This classification is based on the resources available to the attackers. This taxonomy can be extended to security mechanisms in general, including hash algorithms. Attackers are categorised as follows:

Class I (clever outsiders): They are often very intelligent but may have insufficient knowledge of the system. They may have access to only moderately sophisticated equipment.

They often try to take advantage of an existing weakness in the system, rather than try to create one.

Class II (knowledgeable insiders): They have substantial specialised technical education and experience. They have varying degrees of understanding of parts of the system, but potential access to most of it. They often have access to highly sophisticated tools and instruments for analysis.

Class III (funded organisation): They are able to assemble teams of specialists with related and complementary skills backed by great funding resources. The are capable of in-depth analysis of the system, designing sophisticated attacks, and using the most advanced analysis tools. They may use Class II adversaries as part of the attack.

The threat from Class I and Class II attackers can be dealt with by placing a hash algorithm in the public domain and allowing experts in the field to analyse and review the algorithm before widespread implementation. This approach will also ensure that the threat from Class III attackers are minimised. When designing a hash function it is advised that the hash function should be able to withstand attacks from a Class III opponent. This is difficult since it is not always known what a Class III opponent's capabilities are.

3.2.2 Position

In addition to the taxonomy of attackers based on their capabilities, a taxonomy of attackers is presented with regard to their position concerning the system they seek to attack. Regarding cryptographic hash functions the following attackers are identified:

Legitimate Participants: These are participants who rightfully share in a communication process. They are allowed to generate, sign and transmit messages. In the case of MACs they have access to the shared secret key. These attackers can generate two messages that yield the same hash value and substitute the one message for another when convenient.

Active Eavesdroppers: These attackers are not allowed to generate, sign and transmit messages. They are hostile eavesdroppers who seek to intercept and modify messages without detection. This imply that they would attempt to construct a false message

that has a specific hash value and replace a valid message when intercepted. They are not expected to have access to shared secret keys for MACs.

These attackers can belong to Class I, Class II or Class III attackers, depending on their capabilities. The taxonomy of attackers is summarised in Figure 3.1.

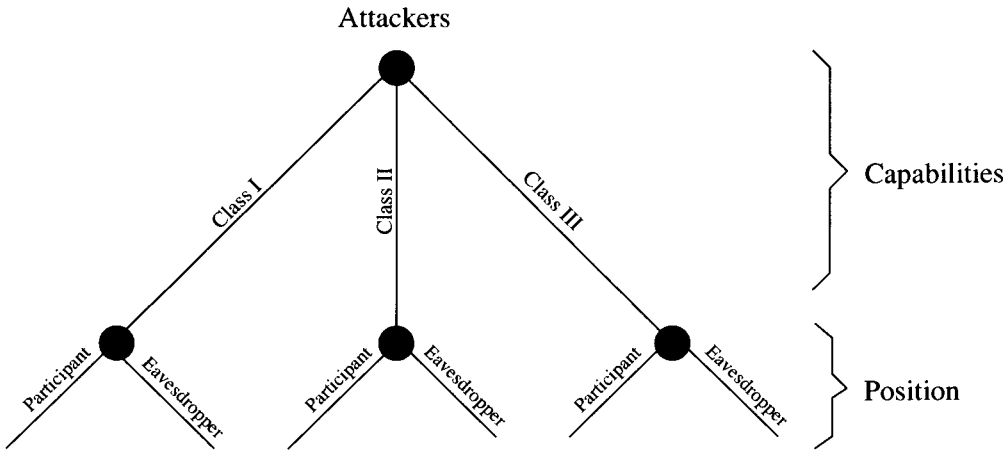

Figure 3.1: Taxonomy of Attackers

3.3 TERMINOLOGY

In this chapter the following attacks are considered:

1. Attacks on MDCs.
2. Attacks on MACs.
3. Attacks on underlying block ciphers.
4. High level attacks.
5. Attacks dependent on the algorithm.

Before proceeding with a description of the generic attacks on hash Functions, it is useful to consider the terminology used in describing the generic attacks. The terminology introduced in this section serves as an indication of what an attacker could hope to achieve when attacking a hash function.

3.3.1 MDC Terminology

When attacking a MDC, an attacker could hope to construct a:

Pre-image: Establishing a pre-image is equivalent to finding a message that results in a specified hash value.

Second pre-image: A second pre-image requires the attacker to find two messages that results in a specific hash value.

Pseudo-pre-image: A pseudo-pre-image requires that two messages, X and X', with two different initial values, IV and IV' should be found such that the $h(IV, X)$ and $h(IV', X')$ result in the same specified hash value.

Collision: A collision is established if an attacker can find two messages X and X' such that $h(IV, X)$ and $h(IV, X')$ result in the same unspecified hash value.

Collision for different IV's: A collision for different IV's is established if two messages and two IV's can be found such that $h(IV, X)$ and $h(IV', X')$ hash to the same hash value.

Pseudo-collision: A pseudo collision is established if an attacker can find two messages X and X' such that $h(IV, X)$ and $h(IV', X')$ yield the same hash value for two specified IV's.

Constructing collisions, collisions for different IV and pseudo-collisions are easier than constructing a second pre-image or pseudo pre-image. Attacks specific to MDCs are considered in Section 3.4.

3.3.2 MAC Terminology

A MAC makes use of a secret key to compute a hash value. Thus for a MAC the collision is dependent on both public knowledge (the message) and secret knowledge (the key). The attacker is therefore faced with two problems. The first deals with the key, the second deals with the construction of collisions. When dealing with a MAC, an attacker could hope to achieve one of the following objectives [3]:

Key recovery: The attacker can determine the secret key K.

Universal forgery: The attacker constructs an alternative algorithm that mimics the MAC algorithm.

Selective forgery: For a message chosen by the attacker the correct MAC can be determined.

Existential forgery: An attacker can determine a correct MAC for at least one plaintext. The resulting plaintext may be random or non-sensical.

Once the secret key is known to an attacker, he can determine the MAC for any message. With the MAC algorithm and the secret key known, an attacker can proceed to construct a collision. Techniques describing key retrieval and the construction of forgeries are considered in Section 3.5. The objectives for generating a collision for a MAC when the secret key is known are similar to those for a MDC.

3.4 ATTACKS ON MDCS

In this section a number of generic attacks on MDCs are considered. These attacks can be classified as belonging to one of two categories. They are:

1. Attacks independent of the algorithm.
2. Attacks dependent on the chaining.

These attacks are generic and could be used against any hash function. In this section, these attacks are summarised and evaluated according to the computational power required to execute them successfully.

3.4.1 Attacks Independent of the Algorithm

For MDCs two attacks are considered to be independent of the algorithm. This implies that these attacks can be carried out against the ideal cryptographic hash function described in Chapter 1. These attacks are known as the random attack and the birthday attack.

Random Attack

In this attack it is assumed that the attacker is given a message X and requires a message X' such that $X' \neq X$ and $h(IV, X') = h(IV, X)$ (i.e. the attacker has to find a second pre-image). This can be accomplished by randomly selecting X' from all possible admissible messages. The probability of success is 2^{-n} with n the length in bits of the hash value. If an attacker performs T trials, the probability of finding a valid value for X' so that $X' \neq X$ and $h(X') = h(X)$ becomes $T \cdot 2^{-n}$. Thus, the larger n the larger number of trials T are required. According to [1] approximately $0.7 \cdot 2^{-n}$ trials are required to find a collision using this technique. Thus for a n-bit hash value the expected workload to find a second pre-image is in the order of $O(2^n)$.

Birthday Attack

This attack is based on the the birthday paradox from probability theory. According to this paradox, it can be shown that the probability that two individuals in a group of 23 people share a birthday, is approximately 52%. The number of people in the group is much smaller than expected. A related problem states that for two groups of 17 people, the probability that two people have a common birthday, is larger than 50%. This property can be exploited to attack hash functions as explained below.

For two sets of messages r_1 and r_2 it is shown in [3] that:

$$P_r(h(X) = h(X')) = 1 - e^{-\frac{r_1 \cdot r_2}{n}}$$

for:

$$r_1 = r_2 = O(\sqrt{n})$$

the probability

$$\begin{aligned} P_r(h(X) = h(X')) &= 1 - e^{-1} \\ &\approx 63\%. \end{aligned}$$

An example of an algorithm that makes use of this result is presented as algorithm 3.1.

Algorithm 3.1 *Birthday attack for hash functions*

1. *For a n bit hash value let $r_1 = r_2 \approx O(2^{\frac{n}{2}})$*
2. *Generate r_1 variations on the valid message X.*
3. *Generate r_2 variations on the forged message X'.*
4. *Compare the hash values for the r_1 variations of X with the r_2 variations of X'. When a message X and X' is found for which $h(IV, X) = h(IV, X')$, a collision is established.*

The attacker can now generate a message that contains X and then later replace X with X' and claim that he originally generated X', since the hash values for both messages are the same.

In [20] and [21] alternative algorithms for efficient collision search is proposed. These techniques are based on Pollard's ρ method for finding cycles in periodic functions in a finite domain. These techniques were used in the analysis of DES.

The significance of this attack is that the number of operations required to find a collision is $O(2^{\frac{n}{2}})$ instead of $O(2^n)$ for the random attack. A similar order of magnitude is required in storage capabilities. Thus a birthday attack requires less operations than a random attack. The only way to defend against birthday attacks is by increasing the number of bits n in order to make it computationally infeasible to launch a birthday attack.

3.4.2 Attacks Dependant on the Chaining

If a message is longer than the maximum block length of the hash algorithm, the message is segmented. The segments are then processed iteratively (Chapter 5 Section 5.3). This is known as the Damgård-Merkle scheme [22], [23]. A number of attacks have been derived which are only applicable if an iterated structure is used. The attacks are summarised below.

Meet in The Middle Attack

This is a variation of the birthday attack. This attack allows the attacker to construct two messages, X and X', for which $h(X) = h(X')$. The messages X and X' should be at least twice as long as the elementary block length of the hash function. The following algorithm describes the meet in the middle attack.

Algorithm 3.2 *Meet in the middle attack for hash functions*
Consider Figure 3.2.

1. *For a n bit hash value let $r_1 = r_2 \approx O(2^{\frac{n}{2}})$*
2. *Generate r_1 variations on X_1'.*
3. *Generate r_2 variations on X_2'.*
4. *Work forward from the IV and compute r_1 variations of the intermediate values $IM1$ with $IM1 = h(IV, X_1')$ and save this in a buffer of intermediate values IM_1.*
5. *Work backward from $h(X)$ and compute r_2 variations on $h(X) = h(IM2, X_2')$ and save the intermediate values in a buffer of intermediate values IM_2.*
6. *Use a search algorithm and search for two intermediate values that are equal in $IM1$ and $IM2$ respectively. If two equal values are found, a collision is established.*

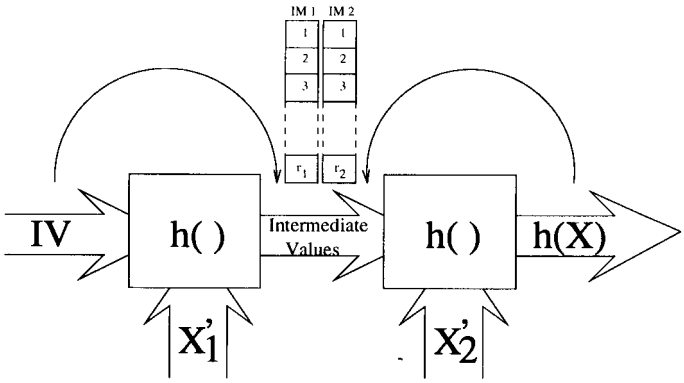

Figure 3.2: Meet in the middle attack

As in the case of the birthday attack the number of operations required to establish a collision are in the order of $O(2^{\frac{n}{2}})$. The advantage of this attack is that it allows an attacker to hit a

specific hash value. This attack is only possible if the message is longer than an elementary message block.

It is possible to defend against these attacks by increasing the number of bits in a hash value to such an extent that the meet in the middle attack is computationally infeasible. Another defence against this attack is to constrain the message lengths to less than the elementary block length.

When imposing constraints on the solutions obtained with the meet in the middle attack, the attack is called the constrained meet in the middle attack.

Generalised Meet in The Middle Attack

To avoid the meet in the middle attack, two-fold iterated schemes were suggested in [6]. These schemes include computing two hash values for a given message using two different IV's ($h(IV, X)$ and $h(IV', X)$) or computing the hash value on the concatenation of the message to itself ($h(IV, X||X)$). These schemes can be extended to so-called p-fold schemes where p hash values are computed for the same message using p initial values, or by concatenating the message p times to itself and then computing a hash value ($h(IV, X||X||\ldots X)$). A graphical representation of these p-fold schemes are shown in Figure 3.3.

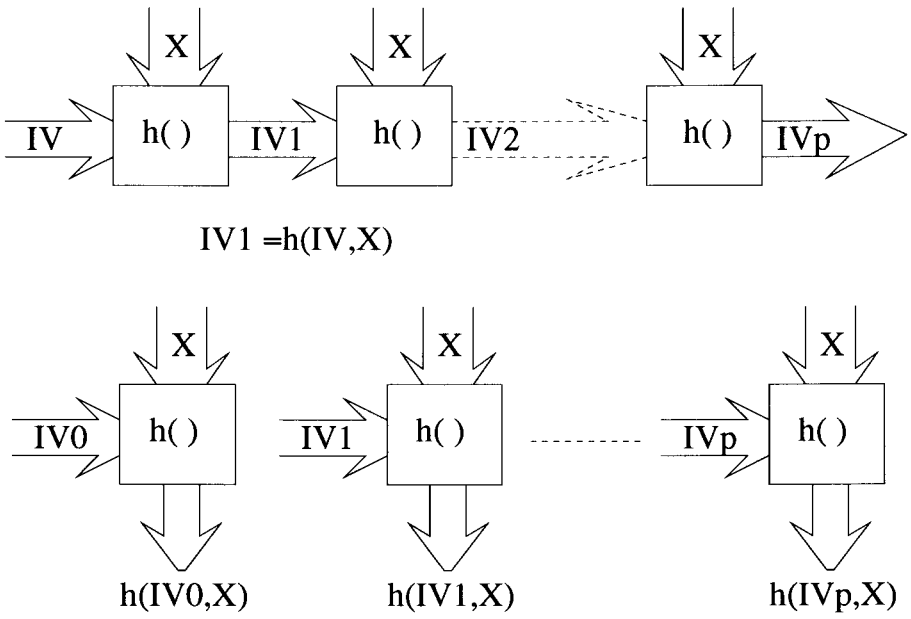

Figure 3.3: Two P-Fold Hashing Schemes

It has been shown in [24] and [25] that the meet in the middle attack can be extended to break these schemes. The extension of the meet in the middle attack to attack p-fold schemes is called the generalised meet in the middle attack. For this attack only $O(10^p \cdot 2^{\frac{n}{2}})$ operations are required instead of $O(2^{\frac{pn}{2}})$ [24], [25].

This attack can be foiled by choosing the number of bits, n, large enough in order to make the attack computationally infeasible.

Correcting Block Attack

Several variants of the correcting block attack exists. The first variant assumes that an attacker has a message X for which a forgery, X', has to be constructed. All the blocks in X' are then changed so that they differ from X. One message block in X', X'_i is then constructed so that $h(X) = h(X')$. The block X'_i is then designated as the correcting block. The correcting block is usually inserted as the last block in the message, but may be inserted at the beginning of a message or in the middle of a message. For this variant of the correcting block attack, the construction of the correcting block X'_i may be accomplished with the random attack. Since a specific hash value has to be generated, the birthday attack cannot be used. If two correcting blocks are allowed, it is possible to use the meet in the middle attack to generate two blocks, X'_i and X'_{i+1}, that together cancel the effect that message blocks X'_0 to X'_{i-1} have on the chaining variable. Another alternative in constructing the correcting block requires the attacker to have knowledge of the algorithm. By manipulating the algorithm a correcting block X'_i can be constructed. Note that the construction of a message block by manipulating the algorithm depends on the algorithm. If the construction of X'_i is independent of the algorithm, the amount of work required is $O(2^{\frac{n}{2}})$.

Another variant of the correcting block attack is described next. An attacker generates two messages X and X' and then generates two correcting blocks Y and Y'. The correcting blocks are then concatenated to X and X'. The correcting blocks Y and Y' should be chosen such that $h(IV, X||Y) = h(IV, X'||Y')$. If the final hash value is specified, the correcting blocks Y and Y' can be constructed using the random attack. If the final hash value is not specified, the birthday attack can be used to generate a collision. If more than one block is allowed as a correcting block, the meet in the middle attack can be used. Note that with the meet in the middle attack the attacker can generate a specific hash value. It is also possible to construct the correcting blocks Y and Y' by manipulating the algorithm. The attack then

depends on an analytical weakness in the algorithm.

It is possible to defend against block correcting attacks by adding redundancy to the message before hashing. Redundancy includes padding rules and attaching the number of blocks or bits in a message as the last block. If the attacker is the originator of the message these measures are not sufficient, since the attacker can control the number of blocks or the length of the message under consideration. Choosing the value of n large enough, makes the block correcting attack computationally infeasible. If the hash function itself can be manipulated to produce a correcting block, the attack becomes dependent on the algorithm used. The algorithm should be replaced if that is the case.

Fixed Point Attack

A fixed point may be defined as a hash value for which $h(X_i, H_{i-1}) = H_{i-1}$. This property allows an attacker to insert an arbitrary number of blocks corresponding to X_i after the first occurrence of X_i (see Figure 3.4).

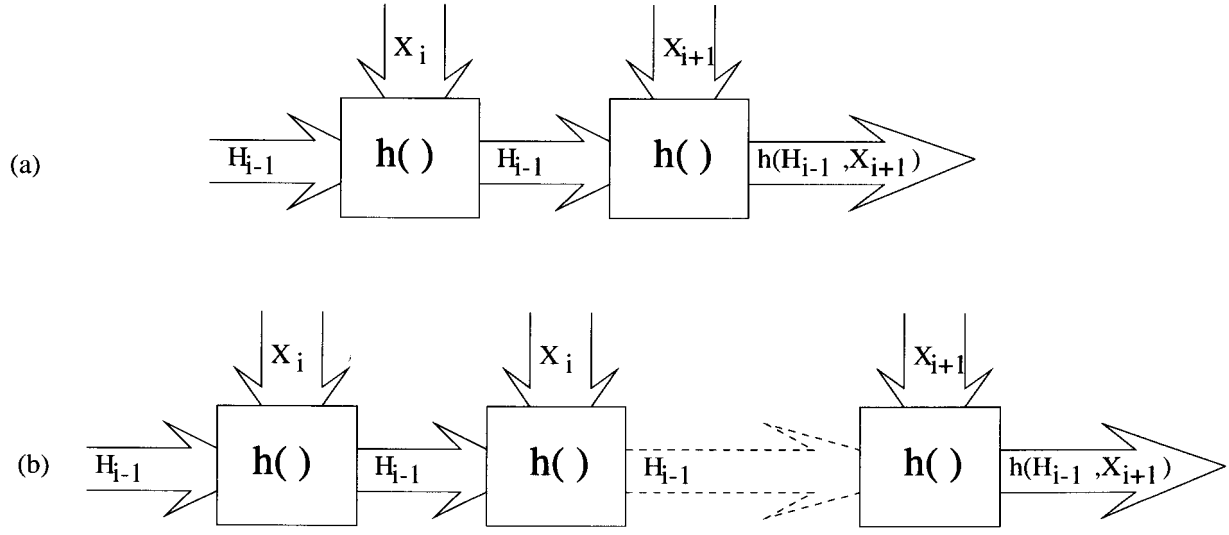

Figure 3.4: The fixed point attack

A collision can be established if the chaining variables can be set equal to H_{i-1}. This can be accomplished by using the random attack to find a suitable value for X_{i-1} or the meet in the middle attack which would allow the attacker to specify X_{i-1} and X_{i-2} so that H_{i-1} can be established. It might also be possible to manipulate the hash algorithm to find a suitable

value for X_{i-1} that will produce H_{i-1}. It is possible to foil this attack by adding redundancy to the message. The redundancy should contain the number blocks in the message. The fixed point can be found by the random attack, or by the meet in the middle attack. This implies that the work factor is approximately $O(2^{\frac{n}{2}})$. By choosing n sufficiently large, it becomes computationally infeasible to perform the random attack or the meet in the middle attack. The value of an attack in which one block is repeated a number of times and still yields the same hash code is debatable.

Differential Attacks

Differential cryptanalysis is based on the study of the relationship between input and output differences in iterated cryptographic algorithms. Since hash functions are usually based on iterated algorithms, differential cryptanalysis is applicable to cryptographic hash functions. The differential attack against hash functions is a probabilistic attack. An attacker searches for input differences that will result in specific output differences. If an attacker intends to create a collision the output difference should be zero.

The differences can be found with a probabilistic search (random attack). For a random attack, IV, $h(IV, X)$ and IV' are specified. The attacker has to find X' so that $h(IV, X) = h(IV', X')$. Another technique that an attacker could use is the birthday attack. For the birthday attack it is assumed that IV and IV' are specified. It now remains to find values for X and X' such that $h(IV, X) = h(IV', X')$ with the birthday attack as described in Section 3.4.1. It is also possible to use the meet in the middle attack to establish a collision. This requires that the attacker should be able to choose two message blocks in both the original and the forged message. This places the magnitude of the work factor at $O(2^{\frac{n}{2}})$. It is also possible to execute the differential attack by manipulating the hash algorithm. The differential attack then becomes dependent on the algorithm. When block ciphers are used, the differences should be chosen in such a way as to exploit the chaining.

Differential attacks can be launched against the chaining of an iterated hash Function, or against the hash algorithm itself. In Chapters 6, 7, 9 and 10 it is shown how differential analysis is employed against the compress function of dedicated hash functions such as MD4, MD5, SHA and HAVAL.

3.5 ATTACKS ON MACS

All of the attacks against MDCs are applicable to MACs if the key for the MAC is known to the attacker. In addition to the attacks on MDCs, the following attacks are applicable to MACs.

3.5.1 Key Collisions

A key collision occurs when, for two distinct keys, K_1 and K_2, $h(X, K_1) = h(X, K_2)$. Due to the presence of a key, this attack is applicable to MACs. MACs based on block ciphers are especially vulnerable to this attack, since this phenomenon has been observed in block ciphers [20], [21]. It was shown in [20] that this attack can be implemented against the Data Encryption Standard (DES) using the meet in the middle attack. In [21] a refined technique based on the theory of distinguished points are proposed. Both attacks resulted in the discovery of key collisions for DES. This attack allows an attacker to construct a message that yields the same hash value for different keys used. For DES, 21 collisions were found in [21] and 48 collisions are described in [20]. When the meet in the middle or birthday attacks are used, approximately $O(2^{\frac{n}{2}})$ operations are required (n is the number of bits in the hash value). For these attacks a large storage space and an efficient sorting algorithm is required. In [21] a technique is suggested that reduces the storage requirements and eliminates the necessity of an efficient sorting algorithm.

Another block cipher with a large number of known key collisions is LOKI. It is known that 15 key collisions exist for every key in LOKI [26]. For this reason it is advised that LOKI is not used as a round function for the construction of a MDC or MAC [27]. For a MDC the birthday attack or the meet in the middle attack can be used. For a MAC a key collision can be established.

3.5.2 Exhaustive Key Search

Exhaustive key search is intended to recover the key for a MAC. For a given MAC both the hash value, $h(K, X)$, and the message, X, are known. The key is recovered by exhaustively trying all possible keys, K_i, until a key, K, is found that results in $h(K, X)$. If key collisions exist for the hash function, $h()$, several messages X_i and their corresponding hash values,

$h(K, X_i)$, are required to confirm that K is a valid key. Therefore this attack is effectively a known plaintext-MAC attack. The effort required to find a k-bit binary key is on the order of $O(2^k)$. In [3] it is stated that for a k-bit key and a n-bit hash value the number of plaintext-MAC pairs, M, required to determine the key uniquely is slightly larger than $\frac{k}{n}$, provided that no key collisions occur.

3.5.3 Chosen Text Attacks

This attack allows the generation of a forgery and, in general, does not allow key recovery. The attack is described [28] and [29]. The attack requires that the MAC is based on an iterative structure (see Chapter 5 Section 5.3). The basic idea is that for two messages, X_1 and X_2, with:

$$\begin{aligned} X_1 &= a_1||b. \\ X_2 &= a_2||b. \\ a_1 &\neq a_2. \end{aligned}$$

The MAC results, $h(K, X_1) = h(K, X_2)$, are likely to occur given $O(2^{\frac{n}{2}})$ chosen messages, with n the hash length. This is reminiscent of the birthday attack. Given $h(K, X_1) = h(K, X_2)$ it is expected that $h(K, a_1) = h(K, a_2)$. This implies that for an arbitrary string c, the MAC values $h(K, a_1||c)$ and $h(K, a_2||c)$ are equal. Thus, a forgery can be obtained by requesting $h(K, a_1||c)$, and in effect obtaining the MAC for $h(K, a_2||c)$.

This attack imposes requirements on the storage space available to an attacker and also assumes that the secret key, K, is not changed before $O(2^{\frac{n}{2}})$ chosen messages can be requested. Note that a forgery is obtained without retrieving the secret key K.

3.5.4 Known and Chosen Text Attack

This attack is presented by Preneel and van Oorschot in [29]. It is viewed as an extension of the chosen text attack described above. The attack is outlined as a proof in [29]. Note that this attack, as is the case for the chosen text attack, generally enables forgery attacks, not key retrieval.

The attack states that r known plaintexts are required with $r = \sqrt{2} \cdot 2^{\frac{n}{2}}$. For the previous

attack it was assumed that the MAC was given by a permutation $g()$ of the hash result, $h()$ (e.g. the unity mapping). For this attack it is assumed that a random mapping is used with the resulting MAC having m bits instead of n bits with $m < n$. When considering this attack, it is necessary to differentiate between a collision before the random mapping is applied, (internal collision) and a collision after the random mapping is applied (external collision). With $g()$ being a random mapping function, 2^{n-m} external collision are expected. Additional computations are now required to generate a verifiable forgery. An internal collision can be identified by attaching a known string y to each collision pair and checking whether the corresponding MACs are equal. This requires $2 \cdot (1 + 2^{n-m})$ chosen text-MAC requests. For internal collisions the resulting MACs are always equal. Retain only the text-MAC pairs resulting in internal collisions. At this stage 2^{n-2m} external collisions and a single internal collision should be available. If more than one external collision remains, these external collisions have to be eliminated by choosing a different value for y and proceed as before, until a single external collision remains. It is believed to be highly probable that the remaining external collision is the result of an internal collision. It is expected that the number of chosen and known texts required to find an internal collision are:

$$\frac{2 \cdot 2^{n-m} \cdot 2^m}{2^m - 1} + 2\left\lceil \frac{n}{m} \right\rceil \approx 2 \cdot 2^{n-m} + 2\left\lceil \frac{n}{m} \right\rceil .$$

Finding a single internal collision that allows the generation of a MAC forgery requires:

$$\begin{aligned} \text{Known text-MAC pairs} &= \sqrt{2} \cdot 2^{\frac{n}{2}} \\ \text{Chosen text-MAC pairs} &= 2 \cdot 2^{n-m} + 2\left\lceil \frac{n}{m} \right\rceil . \end{aligned}$$

This attack is extended to cover the case where all the known texts have a common sequence of s trailing blocks. It is stated that if that is the case, fewer known and chosen texts are required. In particular it is shown that if:

$$r^2 \cdot s = O(2^n)$$

the probability that the set of known messages, r, contains two messages that collide under the hash function, $h()$, is approximately:

$$1 - e^{\left(-\frac{r^2 \cdot (s+1)}{2^{n+1}}\right)}.$$

Given this probability, an internal collision for $h()$ can be found if the known text-MAC pairs have s identical trailing blocks. If $g()$ is a random mapping, this attack requires:

$$\text{Known text-MAC pairs} = \sqrt{\frac{2}{s+1}} \cdot 2^{\frac{n}{2}}$$

$$\text{Chosen text-MAC pairs} = 2 \cdot \frac{2^{n-m}}{s+1} + 2 \left\lceil \frac{n - \log_2(s+1)}{m} \right\rceil .$$

In [29] both these attacks are applied to MAA [30] and CBC-MAC (see Chapter 5 Section 5.4). The refined version of this attack with s-blocks is shown to be effective against MAA. The refined attack cannot be applied to CBC-MAC with maximal feedback, since the round function is bijective. The unrefined attack described initially can however be employed against CBC-MAC.

In [31] these attacks are extended to and tailored for forgery and key recovery for MAA and the envelope MAC constructions (see Chapter 5 Section A.5).

It is noted that a large number of chosen texts and known texts along with their MAC values are required. If it is assumed that key collisions does not occur for the chosen MAC, the secret key, K, may not change while collecting the known and chosen texts. It is therefore suggested that the chosen text attacks, and known and chosen text attacks, described in this and the previous section can both be foiled by effective key management. By changing the secret key at a regular interval, it becomes impossible to collect enough chosen and known texts to execute these forgery attacks.

3.6 ATTACKS ON UNDERLYING BLOCK CIPHERS

It has been suggested that block ciphers could be adapted for use as building blocks for cryptographic hash functions [23], [32], [3]. The motivations for these suggestions are presented in Chapter 5. It should be noted that when using block ciphers in a hash function configuration, additional attacks based on the underlying block cipher are possible. The attacks discussed in Sections 3.4 and 3.5 are applicable to hash functions derived from block ciphers. Specifically the construction of fixed points is considered easy if the underlying block cipher is either DES or LOKI [3].

3.6.1 Complementation Property

Symmetry under complementation was one of the first properties discovered for DES [33]. Let $E(K, X)$ denote the encryption of X with the key K. Then for a message, X, and a key, K, the complementation property is stated as follows:

$$\forall K, X : C = E(K, X) \Leftrightarrow \overline{C} = E(\overline{K}, \overline{X})$$

with C the ciphertext and $\overline{C}$ the complement of C. This property is known to exist for DES and LOKI91 [34]. When used in a hash function construction as a MAC, this reduces the effort required for exhaustive search by a factor of two. In addition, this property allows the construction of trivial collisions. It is known that LOKI89 has a large set of keys for which this property holds [34].

3.6.2 Weak Keys

A weak key is a key for which the following property holds:

$$\begin{aligned} E(K, X) &= C \\ E(K, C) &= X, \quad \forall X. \end{aligned}$$

Thus the encryption, E, and decryption, D, operations are equivalent for a given key K. Thus for certain keys, some block ciphers are involutions. This property holds for certain keys of DES, LOKI89, LOKI91 [34] [35] and IDEA [36].

For a semi-weak key the following property holds:

$$\begin{aligned} E(K_1, X) &= C \\ E(K_2, C) &= X, \quad \forall X. \\ K_1 &\neq K_2. \end{aligned}$$

These properties of block ciphers can be exploited in certain hash functions based on blocks ciphers to yield fixed points (see Section 3.4.2).

3.7 HIGH LEVEL ATTACKS

In addition to the above attacks, the following attacks are considered feasible. These attacks are not so much an attack on the hash algorithm, than an attack on the interaction of the hash algorithm with the environment in which it is used.

3.7.1 Differential Fault Analysis

In [37] an attack on public key systems implemented in hardware is described by Boneh, DeMillo and Lipton from the Math and Cryptography Research Group, Bellcore. Based on the attack presented in [37], Biham and Shamir describes an attack that retrieves the key for a hardware implementation of DES [38]. This attack is termed differential fault analysis (DFA). Both of these attacks are specifically applicable to algorithms implemented in hardware, and for this reason are considered as high level attacks.

The attacks described in [38] exploit the effect of a transient error in a hardware device. The resulting erroneous output is then analysed to determine the secret key. In [38] it is claimed that less than 200 ciphertexts are required to find the last sub-key in a DES implementation. The remaining eight key bits can be found by exhaustive search.

In [39] Quisquater claims that this technique is applicable to MACs implemented in hardware. According to [37] attacks based on differential fault analysis can be countered by verifying results before output and protecting the registers used to store values using error correcting codes. Thus differential fault analysis imposes conditions on the implementation of a hash algorithm, rather than on the design of a hash algorithm.

3.7.2 Differential Power Analysis

Differential power analysis was first proposed by Kocher []. This attack allows an attacker to derive the secret key used by an algorithm. This is accomplished by observing the fluctuations in power consumption of the device, while performing cryptographic operations. It is a non-destructive attack. This type of attack is especially efficient against smart card implementations. It has been demonstrated that a secret key can be obtained using this approach. This attack can be used against MACs implemented in hardware.

3.7.3 Attack on the Interaction with the Signature Scheme

As remarked in Chapter 1 hash functions are often used in digital signature schemes. It has been shown in [40], that even if the hash function is a collision resistant hash function, the signature scheme can be attacked successfully. The success of this attack is due to the underlying multiplicative structure of both the hash function and the signature scheme [3].

3.7.4 Attacks on the Protocol

These attacks are concerned with attacks such as replay of messages and the construction of valid messages from previously intercepted messages. It is thus in effect an attack on the protocol. These attacks can be thwarted by the use of a nonce or a timestamp. It is thus necessary to protect a system in which messages and hash values are vulnerable to interception with a suitable protocol.

Equivalent Systems

The calculation of the equivalent shift register length for a given stream Cipher, allows an attacker to construct a linear feedback shift register that mimics the operation of the stream cipher. This calculation is based on the Massey-Berlekamp algorithm. Similarly an attacker could attempt to construct an equivalent system for a cryptographic hash function. This attack is applicable to MACs in particular [3]. The attacker attempts to find a system that produces the same MAC for a given message, without knowledge of the secret key [3]. No standard technique is known to exist for mimicking hash functions.

3.7.5 Attacks Dependant on the Algorithm

These attacks exploit a weakness in the algorithm. These attacks are usually discovered only after the publication of a hash algorithm. These attacks are only a threat if the work factor for finding a collision is substantially less than $O(2^{\frac{n}{2}})$. This dissertation investigates attacks of this nature.

3.8 ATTACKERS AND ATTACKS

This section relates the attacks the various classes of attackers are capable of.

The distinction between classes of attackers are based on their capabilities, the information at their disposal and their position with regard to the system under consideration. The difference with regard to Class I and Class II attackers lie, to a large degree, within the information at their disposal. The advantage that Class II attackers have over Class I attackers can be eradicated by publishing the hash algorithm and its design criteria.

All three classes of attackers are capable of executing any of the attacks presented in this chapter. The probability of success however, increases as the attacker's capabilities increases. The feasibility of the attacks presented in this chapter are measured in terms of effort required to establish a collision for a hash function. In general Class III attackers have the largest resources in terms of computing power, followed by Class II and then Class I attackers.

A further point of interest is the position of the class of attacker with respect to the system under consideration. A legitimate participant can construct two messages that result in the same hash value. One message can be signed and transmitted and at a later stage the messages can be swapped. An active eavesdropper is not allowed to construct and sign messages. Active eavesdroppers are expected to intercept and modify messages. The eavesdropper is restricted to finding messages that hash to specific values. As seen in Section 3.3 the computational effort required to construct two messages that result in the same hash value is considerably less than that required to hit a specific hash value. For example, even though a Class II attacker might not be capable to construct a message that results in a specific hash value, it may be possible for the Class II attacker to construct two messages that yield the same hash value.

Thus although all three classes of attackers are capable of all possible attacks, the probability of success differs substantially as the attackers' knowledge and capabilities differs. In addition it appears that legitimate participants require less effort to construct messages that result in collisions, than active eavesdroppers.

3.9 FEASIBILITY

All of the above attacks pose requirements in terms of processing power and storage space. When designing a hash function the parameters that contribute the security of the function, such as the hash length, should be chosen in such a manner as to render any of the above attacks infeasible. In order to make a sufficiently informed choice for these parameters the capabilities of an opponent has to be known or estimated. Estimating the computational power of an opponent is a complicated process. The following aspects should be considered when estimating computational capabilities.

1. Processor speed.
2. Volatile memory access time.
3. Amount of volatile memory available.
4. Storage space access time.
5. Storage space available.
6. System bandwidth.

In addition to these aspects, additional factors, such as the opponent's ability to construct dedicated hardware or hardware subsystems and the interconnection of these systems to realise the above attacks, should be taken into account. The feasibility of an attack also depends on the class of attacker dealt with (see Section 3.2).

The generic attacks described in this chapter are not considered feasible for hash lengths of 128-bits or more. It is already considered that 128 bits will not provide sufficient protection within the next few years, due to the increase in available computational power [41]. The development of new fields in computing such as quantum computing may also change the estimate of computationally secure hash lengths [42].

3.10 CONCLUSION

A taxonomy of possible attackers were presented. The taxonomy is based on the attackers' capabilities and their position with regard to the system they seek to attack.

A review of the general attacks against MDCs were presented in Section 3.4. These attacks are discussed in [17]. It was shown that these attacks require a maximum of $O(2^n)$ operations and a minimum of $O(2^{\frac{n}{2}})$. Due to the definition of hash functions, these attacks cannot be avoided. When designing a hash function, the relevant parameters should be chosen to minimise the effect of these attacks. The feasibility of a specific attack is measured by the workload associated with each of these attacks. The workload is expressed in terms of the number of bits, n, contained in the hash values. This implies that an attack can be made infeasible by choosing n sufficiently large.

In addition to the general attacks on MDCs, the general attacks on MACs were presented in Section 3.5. These attacks are aimed at either retrieving the secret key, or obtaining a forgery for the MAC. The probability of success for these techniques are proportional to the size of the secret key and the number of bits, n, in the resulting hash function.

When constructing cryptographic hash functions based on block ciphers, additional attacks are possible. These additional attacks were presented in Section 3.6. These attacks exploit certain properties of block ciphers and allows the establishment of collisions. As before, a legitimate participant has a larger probability of success than an active eavesdropper.

Several high level attacks were presented. These attacks concentrate on the environment in which a hash function is used. Hash functions and the systems within which they are used, should be implemented in a secure manner to avoid these attacks.

The relationship between attackers and the type of attacks they are capable of were investigated in Section 3.8. It is shown that an attacker operating as a legitimate participant has a larger probability of success than an attacker operating as an active eavesdropper. Therefore legitimate participants poses a more significant threat than active eavesdroppers. Likewise, it is more likely that a Class III attacker will execute an attack successfully than a Class II or Class I attacker.

When designing a cryptographic hash function, both the attackers and the attacks they are capable of should be considered. Cryptographic hash functions should be designed to withstand the attacks described in this chapter.

CHAPTER 4: REQUIREMENTS FOR CRYPTOGRAPHIC HASH FUNCTIONS

4.1 INTRODUCTION

This chapter contains a description of the requirements for cryptographic hash functions. These requirements are based on:

1. The definition of a cryptographic hash function.
2. Known attacks on cryptographic hash functions.

The requirements are divided into two classes, namely functional requirements and security requirements [43]. These requirements are often contradictory. The contradictions are not restricted to the security requirements and the functional requirements, but are sometimes found in the functional or security requirements themselves. The conflict between security and functional requirements are treated in a separate section in this chapter.

4.2 FUNCTIONAL REQUIREMENTS

The functional requirements deals with the practical implementation of a hash function. The requirements presented in this section are intended as goals to be met by practical hash algorithms.

4.2.1 Message Reduction

According to the definition of a hash function a message of arbitrary length is compressed to a string of fixed length. Thus a fundamental functional requirement for a cryptographic hash function is the reduction of a message of arbitrary length to a string of fixed length.

Concerning the input length of the message, several solutions are possible. The designer could specify different hash algorithms for different message lengths. This is an impractical solution. Alternatively a scalable hash algorithm can be used. The third possibility is the use

of padding to extend the message to a specific length, or a multiple of a specific length. This is the most often used technique.

If the message is longer than the elementary block length of the hash algorithm, the message is padded to be a multiple of the elementary block length. The padded message is then segmented and processed iteratively by the hash algorithm. This process is known as chaining and is commonly used in hash functions (see Chapter 5 Section 5.3).

4.2.2 Repeatability

Repeatability is important since it should be possible to produce the same hash value for the same message.

In Chapter 5 an ideal construction is presented for a cryptographic hash function. In this construction the hash value is generated independent from the message. The construction requires the use of a database to produce the same hash value for the same message. The construction is impractical due to the storage requirements, accessibility of such a database and the difficulty of constructing binary symmetric sources. Attaining repeatability through the use of a database is therefore infeasible.

Repeatability can be achieved in a practical hash function by making the hash value dependent on the message. Thus possession of the message implies possession of the hash value.

4.2.3 Data Type Independence

The hash function should not be dependent on the type of data to be processed. In other words, a specific file type should be processed in the same manner by the given hash function as any other file (e.g. a binary executable should be processed in the same manner as an ASCII file). A hash function should therefore not be designed to process a specific data type, especially if the hash function is intended for widespread use.

4.2.4 Fast Calculation

The fast calculation of hash values for messages is an important requirement. When using a digital signature scheme the hash value instead of the message is signed. For this reason it should be faster to compute and sign the hash value than to sign the entire message. The reduction in time and bandwidth requirements are important motivations for using hash functions.

The speed of a hash function can be increased by efficient implementation or efficient design. A fast hash function can be designed by either optimising the algorithm for a specific computer architecture or by simplifying the design to reduce the number of operations required by the hash algorithm.

4.2.5 One Pass per Message

This requirement specifies that the message should be processed only once. This effectively rules out all of the so-called p-fold schemes. Thus a message is loaded into memory only once. When computing online this has the advantage that no permanent storage is required. When computing the hash value for a file on a computer disk drive, the time required to access the file and load it into memory more than once is eliminated. This requirement augments the requirement for fast calculation.

4.2.6 Minimum of Secret Information

This requirement is the main reason why MDCs are preferred over MACs. A MAC requires that a secret key is shared between two or more users. This introduces the problem of key management. It also serves as an argument against the use of MACs in an environment where the issue of key management poses a problem.

4.2.7 Modular Design

The hash function should be designed to be modular. This allows the hash algorithm to be replaced within a system if a weakness or deficiency in the algorithm is detected after implementation in a system. Implicit in this requirement is the need for a well defined

interface between the hash function and the system it is used in. The interface components of importance are:

1. Input block size.
2. Output hash value.
3. Key size (MACs only).

4.2.8 Ease of Implementation

If the hash algorithm is intended for widespread use, ease of implementation is important. This requirement ensures the proliferation of the algorithm, since many unskilled users can implement and use the algorithm independently. It is also important to supply sufficient test data for an independent user to verify the operation of the algorithm.

4.2.9 Machine Independence

An algorithm optimised for a specific architecture is advantageous in terms of speed and ease of implementation for the architecture concerned [10], [44], [45]. It is, however, possible that a penalty is paid when transferring the algorithm to another architecture. The penalty may be paid in both a loss of speed and increased complexity of implementation.

4.2.10 Distribution and Obtainability

It is important that a hash function's algorithm, test data, documentation and implementation should be easily obtainable if intended for use in the public domain.

4.3 SECURITY REQUIREMENTS

The security requirements for cryptographic hash functions deals with those properties of a hash function that influences the security of the hash function.

4.3.1 Confusion and Diffusion

The concepts of confusion and diffusion in cryptography were first introduced by Shannon [46].

Confusion is described as: *"The method of confusion is to make the relation between the simple statistics of the ciphertext and the simple description of the key a very complex and involved one"*.

Diffusion is described as: *"In the method of diffusion the statistical structure of the plaintext which leads to its redundancy is dissipated into long range statistics."* These concepts are interpreted in [1] as follows:

Confusion: The ciphertext statistics should depend on the plaintext statistics in a manner too complicated to be exploited by the cryptanalyst.

Diffusion: Each digit of the plaintext and each digit of the secret key should influence many digits of the ciphertext.

The concepts of confusion and diffusion can be applied to hash functions. For MDCs a secret key is not required. Thus the security of the MDC depends solely on the concept of diffusion. In other words, there should be no apparent relationship between the input to the MDC and the resulting output. For MACs a secret key is required, thus in addition to the requirement of diffusion the concept of confusion is considered relevant. Thus, for a MAC, a person in possession of the MAC algorithm, the message, X, and the MAC result, it should be difficult to determine the key K, used to determine the MAC result.

4.3.2 Message and Hash Value Independence

For the ideal hash function construction presented in Chapter 5 a binary symmetric source is used to generate the hash function. Thus there is no dependence between the hash value and the message. In terms of hash functions this is the most secure a hash function could be, since it is impossible to manipulate a message to yield a specific hash value. The ideal hash function construction presented in Chapter 5 is impractical due to the requirement for repeatability. In practical hash functions the hash value depends on the message. For a secure

hash function it is required that there is no apparent or predictable relationship between the hash value and the message. This requirement is related to the concept of diffusion.

4.3.3 Computational Feasibility

When designing a cryptographic hash function the computational feasibility of the known generic attacks should be considered. It should be computationally infeasible to employ any of the generic attacks described in Chapter 3 to construct a collision. It should not be possible to construct a collision in less than $O(2^{\frac{n}{2}})$ operations (n being the hash length). Several degrees of computational infeasibility is defined.

Collision Resistance: This requirement states that it should be computationally infeasible to construct two arbitrary messages, X and X', such that $h(X) = h(X')$. Hash functions that meet this criterion are called collision resistant hash functions (CRHF).

Finding a Specific Hash Value: This requirement states that, given a message, X, and the corresponding hash value $h(X)$ it is computationally infeasible to find a second message, X', such that $h(X) = h(X')$. A hash function that satisfies this criterion is said to be a one way hash function (OWHF). The condition for a OWHF is weaker than the condition for a CRHF.

Sensical Collisions: A sensical collision is a collision for which X and X' can be constructed such that $h(X) = h(X')$, and X and X' are meaningful in the context used. This is the weakest condition imposed on cryptographic hash functions.

It is recommended that a cryptographic hash function is designed to meet the requirements set for a collision resistant hash function.

4.3.4 Interaction with other Algorithms

Supplementary to the requirements of computational infeasibility of finding collisions for a hash function is the requirement for the computational infeasibility of finding a combination of the hash function and the signature scheme that results in a forged signature.

4.3.5 MDC Hash Space

Central to the computational feasibility of finding a collision for a hash function lies the hash space. The generic attacks described in Chapter 3 are evaluated in terms of the order of the number of operations required to establish a collision. None of the generic techniques requires less than $O(2^{\frac{n}{2}})$ operations. The hash length n should be chosen large enough to render the generic attacks harmless, given limited time and resources. Current estimates recommend that $n \geq 128$ bits, and preferably $n = 160$ bits. The length of the hash value should be updated every 3-5 years to accommodate the increase in computational power and advances in computing technology.

4.3.6 MAC Key and Hash Space

The matter of key space is applicable to MACs. A MAC key should be chosen long enough in order to prohibit exhaustive key search. For a k-bit key approximately $O(2^k)$ operations are required to recover a key through exhaustive search. It is proposed that the length of the key should be at least 64 bits.

When considering the hash length of a MAC two factors should be kept in mind. First is the use of a secret key. The secret key adds to the security of the MAC, and consequently the requirements imposed on the hash length is reduced to half of that specified for a MDC with a similar level of security (see Chapter 2). The reduction in the hash space reduces the effort required by an opponent who is in possession of the secret key, K, to find a collision or establish a forgery. Thus for a MAC the security is derived from the difficulty of retrieving the key rather than the difficulty of finding a collision. Thus a MDC with hash length n would afford the same level of security as a MAC with a key of length $\frac{n}{2}$. The second fact which should influence the choice of a MAC length is the results obtained from the attack in [29]. In [29] it is shown that the workload for generating a collision doubles if the length of the MAC value, m is less than the length of the chaining variable, n. The mapping from n to m should appear random. It is proposed that the chaining variables should be at least 128-bits with the final MAC value chosen as 64 bits.

The importance of key management should be kept in mind when designing a system that makes use of a MAC.

4.3.7 Message Dependence

Due to the functional requirement for repeatability, the hash value depends on the message. If the hash value depends on the message, it is important that the hash value depends on every bit in the message. If this is not the case an attacker could easily manipulate the bits not used in the computation of the hash value to produce another message that yields the same hash value.

4.3.8 One-Wayness

It should be computationally infeasible to reconstruct a message from its hash value. This is both a functional and a security requirement. From the point of view of security requirements, the construction of a message from its hash value allows an attacker to construct the message even if the message is encrypted. The requirement for one-wayness can be wavered if both the hash value and the message is encrypted. This solution incurs a time penalty, due to the additional encryption required.

4.3.9 Error Extension

A cryptographic hash algorithm should exhibit maximum error extension. This implies that if one bit is changed in the message, approximately half the bits in the hash value should change. This ensures that an attacker can not expect a collision close to a specific message with a high probability. This implies that the attacker has to search the entire hash space. This is similar to the requirements set for a block cipher.

4.3.10 Distribution of Preimages

The output values and preimages of a hash function should be distributed smoothly. This condition is required to prevent an attacker from searching for those preimages which are known to occur more frequently, thus reducing the effort required for finding a collision. When a hash function based on the Damgård-Merkle scheme is used, the preimages and output values of every block should be distributed smoothly to minimise the possibility of a successful attack on the chaining (see Chapter 3 Section 3.4.2).

4.3.11 Decomposable Algorithms

In [43] it is proposed that a hash algorithm should not be a decomposable algorithm. This would prevent an attacker from analysing individual building blocks to construct a collision. This requirement is intended to prevent attacks dependent on the algorithm.

4.3.12 Conditions on Chaining

If the Damgård-Merkle scheme is used every bit in the chaining variables should be used in the next iteration of the hash algorithm. This ensures that the applicability of the meet in the middle attack is minimised. An additional condition on the chaining variables is imposed by the meet in the middle attack. The number of operations required for the meet in the middle attack are $O(2^{\frac{n}{2}})$, with n the number of bits in the chaining variable. Consequently the chaining variable should have at least the same length as the hash value. For ease of implementation the chaining variable is often chosen the same as the hash value. The number of bits, n, should be at least 128 bits.

4.3.13 Redundancy

Redundancy could be added to the message to prevent certain variants of block correcting and fixed point attacks. Redundancy is especially useful for detecting the addition of message blocks. Redundancy appended to messages include:

1. Time-stamps.
2. Message length.
3. Intermediate value or chaining variable.

Time-Stamps

The use of a time-stamp has the advantage that it prevents replay attacks. The addition of time-stamps introduces several problems and potential weaknesses which could result in a successful high level attack. The use of time-stamps require a synchronised clock. Synchronised clocks are expensive to implement over distributed networks. The use of time-stamps

require a timing window to allow for transition times, especially when used in distributed systems. As the timing window increases, the possibility of a successful replay attack increases. If the timing window becomes too small, synchronisation becomes a problem. An additional risk is introduced when utilising a small timing window. A small timing window implies that the grain of the time-stamp becomes small. A time-stamp with a fine grain increases the probability that an attacker could find two messages with a given time stamp that results in a collision. Once such a pair is generated, the attacker waits for the time specified by the time-stamp before sending the messages.

Message Length

The addition of the message length to the message has the advantage that the message length can not be increased or decreased. When using the Damgård-Merkle construction, padding is required to extend the length of the message to a multiple of the elementary block length. The padding scheme should be designed to contain sufficient redundancy to prevent the addition of blocks.

Intermediate Value

The addition of the message length to the message before hashing makes it easy to detect if a segment of the message was added or deleted. It does not allow the user to detect if a segment of a message was replaced with a different segment. Substitution of message segments can be detected by adding an intermediate result to the message. When using the Damgård-Merkle construction, a chaining variable could be appended to the message before hashing.

4.4 FUNCTIONAL VS. SECURITY REQUIREMENTS

Several of the above requirements for cryptographical hash functions are contradictory. In this section a short review is given of the contradictions. Where applicable resolution strategies for these conflicting requirements are suggested.

4.4.1 Repeatability and Security

This contradiction stems from the definition of an ideal cryptographic hash function. For practical purposes the repeatability in a hash function can not be achieved through the use of a database as is the case with the ideal cryptographic hash function. Instead, repeatability is obtained by deriving the hash value from the message. This implies that the hash value depends on the message and not on a binary symmetric source. It is therefore possible to manipulate the message to yield a specific hash value. This is not possible when using a binary symmetric source to generate the random numbers. Thus by making the hash value dependent on the message, repeatability is obtained at the cost of reduced security. This problem can be overcome by using the message as a seed to a good pseudo random number generator. Therefore it is sufficient to state that there should be no apparent relationship between the message and the hash function.

4.4.2 Chaining and Security

In Section 4.2.1 the need for chaining is introduced as a functional requirement. Unfortunately the use of chaining structures make the hash function vulnerable to a variety of attacks, designed specifically to exploit the chaining mechanism (Chapter 3 Section 3.4.2). The attacks on the chaining require at least $O(2^{\frac{n}{2}})$ operations. The alternatives to chaining are considered impractical. If possible the message length should be limited to a single block to avoid chaining and the consequent attacks. If chaining can not be avoided it is advised that the number of bits, n, be chosen such that the most powerful attack on chaining is computationally infeasible. Suitable values for n exceeds 128 bits.

4.4.3 Speed and Security

The requirement for speed is rooted in limited computing resources and CPU power. Hash operations are specifically intended to speed-up digital signature schemes. When designing a hash algorithm for fast execution the designer is often faced with a trade-off between speed of execution and security [3]. The designer should be careful not to increase the security and thereby sacrificing an unacceptable amount of processing time. Likewise the designer should not increase the speed of an algorithm at an excessive cost to the security of the hash function. The designer should decide what level of security and what hashing times are

tolerable. Note that a slow algorithm is not necessarily more secure than a fast algorithm (a badly designed algorithm can be both slow and insecure).

4.4.4 Speed and Machine Independence

A contradiction between the two functional requirements specified in Section 4.2.4 and Section 4.2.9 exist. In order to design a fast hash function without paying severe penalties in terms of security, hash algorithms are optimised with a specific computer architecture in mind [10], [44] [45] and [47]. This is acceptable as long as the majority of the intended users share this architecture. When the specified architecture is not common to the majority of users, penalties are paid in terms of speed and a lack of ease of implementation. If a hash function is designed with a specific architecture in mind, the design should be optimised to make use of the general architecture of the processors or systems concerned (e.g. 32-bit Intel architecture). The design should not require specific instructions available only to certain processors if diverse processors architectures are used (e.g. multiply and accumulate instruction in a DSP is not available in the 80x86 family).

Note that when implementing an algorithm the implementation should be optimised for the architecture used. However, when defining an algorithm it should not be overly optimised for a specific architecture.

4.4.5 Decomposability and Ease of Implementation

In Section 4.3.11 it is stated that a hash function should not be decomposable. This implies that the hash function does not have individual building blocks. This presents a problem in terms of the functional requirement for ease of implementation. If a hash algorithm is not composed from individual building blocks, it becomes difficult to implement the algorithm step-wise and test the functionality of each block. It is proposed that a hash function is constructed from individual building blocks, but that analysis of individual blocks and the interaction of these blocks does not allow the hash function to be manipulated in a deterministic way. This approach allows a user to implement and test each block and its interaction with other building blocks before constructing the entire hash algorithm.

4.4.6 Security and Bandwidth

In Sections 4.3.5, 4.3.6 and 4.3.3 security requirements are set which influences the length of a message and the consequent hash value. These security requirements are formulated to foil the known generic attacks presented in Chapter 3. Most of these attacks can be made computationally infeasible by increasing the number of bits n. The disadvantage of this defence mechanism is that as the computational power of computers increase the number of bits n has to be increased. When the hash values are transmitted over a channel, an increase in n results in an increase in bandwidth required. Thus, unless bandwidth availability grows at the rate of computational power, poorer throughput and reduced performance of communication systems that make use of cryptographically secure hash functions will be the result.

Another defence technique is to add redundancy to the message and then compute the hash value. The addition of redundancy once again implies reduced performance.

If bandwidth is constrained, a designer could decrease the security to improve the throughput of a system. If security is deemed to be of greater concern than bandwidth requirements, the number of bits, n, should be chosen to be secure in terms of computational feasibility.

4.5 CONCLUSION

This chapter contains a description of both functional and security requirements for cryptographic hash functions. The functional requirements deals with the successful implementation and proliferation of cryptographic hash functions. The security requirements stated in this chapter are intended to render the known generic attacks on hash functions harmless.

Many of the requirements mentioned in this chapter are conflicting. Some of these contradictions are discussed in Section 4.4. It is the intention of this discussion to make a designer aware of these contradictions and to suggest strategies to find optimal trade-offs for these contradictions.

Due to the conflicting nature of the requirements for cryptographic hash functions, a designer should consider these requirements as a guideline. The designer should be influenced by the application for which the cryptographic hash function is to be designed and should accordingly make appropriate trade-offs between these requirements.