The Data Integrity Problem
and
Multi-layered Document Integrity

Ben Moss, B.Sc.

Thesis submitted to the University of Nottingham
for the degree of Doctor of Philosophy

July 2007

To Katie and William

Abstract

Data integrity is a fundamental aspect of computer security that has attracted much interest in recent decades. Despite a general consensus for the meaning of the problem, the lack of a formal definition has led to spurious claims such as "tamper proof", "prevent tampering", and "tamper protection", which are all misleading in the absence of a formal definition.

Ashman recently proposed a new approach for protecting the integrity of a document that claims the ability to detect, locate, and correct tampering. If determining integrity is only part of the problem, then a more general notion of data integrity is needed. Furthermore, in the presence of a persistent tamperer, the problem is more concerned with maintaining and proving the integrity of data, rather than determining it.

This thesis introduces a formal model for the more general notion of data integrity by providing a formal problem semantics for its sub-problems: detection, location, correction, and prevention. The model is used to reason about the structure of the data integrity problem and to prove some fundamental results concerning the security and existence of schemes that attempt to solve these sub-problems.

Ashman's original *multi-layered document integrity* (MLDI) paper [1] is critically evaluated, and several issues are highlighted. These issues are investigated in detail, and a series of algorithms are developed to present the MLDI schemes. Several factors that determine the feasibility of Ashman's approach are identified in order to prove certain theoretical results concerning the efficacy of MLDI schemes.

Acknowledgements

I would like to express my most sincere thanks to Nottingham University's School of Computer Science and Information Technology for all its support. The School has provided various funding throughout this research — both directly and indirectly — for which I am extremely grateful. Alongside the School, I wish to thank BT Exact for jointly funding the initial three years of this research, and for allowing me the freedom to investigate a broad range of security problems.

Furthermore, I would like to thank many members of the School for their help and support during the course of this work. First and foremost I would like to thank my supervisors, Dave Elliman and Graham Kendall, for reviewing my drafts and generally "fostering" much of the writing-up phase of this thesis. Their encouragement, advice, and wisdom has been invaluable. Thanks also go to my examiners, David Hilton and David Brailsford, for their useful comments and suggestions that helped to mature this thesis to completion.

I would also like to acknowledge and thank: Helen Ashman for introducing me to the idea of multi-layered document integrity, which formed the initial investigation area of this research; Leon Harrison for his advice on supervisory matters; Dario Landa-Silva for his support and help on problem semantics; Roland Backhouse for his excellent introduction to rigorous proof styles; Graham Hutton for his long-standing support and encouragement; Colin Higgins for his help and support with funding in recent months; and Edmund Burke for his support and wisdom when it was really needed.

Many thanks go to my friends and family for their support and encouragement over the past seven years — particularly since the arrival of Katie and William, which provided a most-pleasant interlude in this research. A special thanks go to my good friend Jonathan Morley for providing numerous helpful suggestions concerning the programming aspects of this research, and for ensuring I retained some form of social life. For their continual support and encouragement throughout my education, I would like to thank my parents, and to extend thanks for their support when Katie and William were born. I would also like to thank Holly's parents for their support and encouragement, which has meant a great deal to me.

I am clearly indebted to many for their help in the completion of this thesis, but none more so than Holly. Her constant support and encouragement have been invaluable over the past six years, and I am extremely grateful for her patience and understanding.

Thank you all — for your support, patience, encouragement, and understanding.

Ben Moss
University of Nottingham
July, 2007

Contents

List of Figures

List of Tables

List of Propositions

List of Definitions

List of Algorithms

List of Examples

List of Notations

Abbreviations

CHF Cryptographic hash function, page 13

DIP Data integrity problem, page 34

ECC Error-correcting code, page 12

IDS Integrity detection scheme, page 17

IP Invulnerable preservative, page 30

MLDI Multi-layered Document Integrity, page 3

PGP *Pretty Good Privacy*, page 19

PHF Perfect hash function, page 57

POHF Perfect one-way hash function, page 57

RCS Revision control system, page 10

SIDS Signature-based integrity detection scheme, page 17

VP Vulnerable preservative, page 28

Data

b Bit length of a practically bounded data block, page 59

c data block (candidate), page 72

d Data block (original), page 27

e Data block (exposed), page 35

f data block (tampered), page 28

g Guess made by a party involved in some protocol, page 13

h Hash or digest, page 13

i Integer index or counter, page 27

k Branching factor of a tree, page 60

$k_{\overline{\mathcal{A}}}$ Private key of Alice, page 16

$k_{\mathcal{A}}$ Public key of Alice, page 16

$k_{\overline{\mathcal{AB}}}$ Key shared by Alice and Bob, page 15

l Depth of a tree, page 60

m Message, page 15

n Bit length or input size, page 27

p Preservative data block (original), page 28

q Preservative data block (exposed), page 29

r Preservative data block (tampered), page 29

s Signature, page 16

u Hash collision probability, page 73

v Validation, page 16

w Information concealed by a party involved in some protocol, page 14

x Generic data or value, page 27

y Generic data or value, page 53

Existing Functions

C Correction algorithm, page 40

D Detection algorithm, page 39

E Secret-key/Public-key encryption/decryption function, page 15

F Generic function, page 107

G Generic algorithm, page 35

$\overleftrightarrow{H}$ Hash function (perfect), page 57

$\vec{H}$ Hash function (perfect one-way), page 57

H Hash function (cryptographic), page 13

$\dashrightarrow{H}$ Hash function (bounded-preimage), page 77

$I(x)$ Shortest description of x, page 30

$K(x)$ Kolmogorov complexity of x, page 30

L Location algorithm, page 39

$O(x)$ Complexity order of x ("big-O notation"), page 59

P Prevention algorithm, page 40

$\Pr(Z)$ Conditional probability of event Z, page 73

R Resolution algorithm, page 92

$S_{\overline{\mathcal{A}}}$ Sign function belonging to Alice, page 16

$V_{\mathcal{A}}$ Validate function belonging to Alice, page 16

Original Functions

ε Bit exposable function, page 53

β Candidate quantification function, page 87

χ Correction behaviour function, page 37

X Correction solvability function, page 40

δ Detection behaviour function, page 35

Δ Detection solvability function, page 39

ϵ Exposable function, page 53

γ General behaviour function, page 41

Γ General solvability function, page 41

Λ Location solvability function, page 39

λ Location behaviour function, page 37

α Possibility quantification function, page 87

ϖ Preserving function, page 28

π Prevention behaviour function, page 38

Π Prevention solvability function, page 40

τ Tamper function, page 28

Operators

$\approx$ Approximate equality, page 73

$=$ Equality, page 13

$\neq$ Non-equality, page 29

$:=$ Assignment, page 14

$|x|$ Number of bits in x, page 27

$\#(x)$ Number of blocks in x, page 27

$\lceil x \rceil$ Ceiling of x, page 61

$\|$ Concatenation, page 14

$::=$ Definition, page 27

Participants

$\mathcal{A}$ Alice, page 15

$*$ Everyone (Universal party of all participants), page 25

$\mathcal{B}$ Bob, page 15

$\mathcal{J}$ Jude (Mutually-trusted third party), page 25

$\mathcal{T}$ Tom (Tamperer), page 25

Spaces

$\mathbb{B}^b$ Leaf data block search space, page 65

$\mathbb{B}^n$ Root data block search space, page 65

$\mathbb{C}$ Candidate data block search space, page 72

Part I

Background

If a man will begin with certainties, he shall end in doubts; but if he will be content to begin with doubts, he shall end in certainties.

— Francis Bacon, *The Advancement of Learning* (1605)

Chapter 1

Introduction

No man can purchase his virtue too dear, for it is the only thing whose value must ever increase with the price it has cost us. Our integrity is never worth so much as when we have parted with our all to keep it.

— Ovid [*attributed*]

Integrity is not only a fundamental aspect of virtue, it is also a fundamental problem in computer security. In both contexts it has a similar meaning; *the state of being true, honest, pure or whole.* In the field of security, the integrity problem concerns the unauthorized modification of data — in other words, tampering.

Many consider the integrity problem to have been solved with the advent of public-key cryptography, hash functions and digital signatures, which provide a robust means of determining data integrity. This view is reflected by the fact that relatively little active research exists in this area compared to other security problems such as privacy. However, determining integrity is only part of the overall problem, and does not really *solve* anything when the tamperer is persistent. In the presence of a persistent tamperer, the problem is concerned with maintaining and proving the integrity of data, rather than just determining it.

Consider the hypothetical situation described in Example 1.1. It highlights some of the current problems with the all-or-nothing solution to the integrity problem. If the data is original then it has some value, otherwise it has been tampered with and is completely worthless.

Example 1.1 (Alice and Bob's Integrity Problem)

Alice wishes to send a message to her fiancé Bob, but Bob is away on a remote expedition in the Amazon. Being such a remote location, their only chance of communication is by email, and Internet connectivity in the middle of the jungle is extremely poor. Unfortunately, every email that Bob receives from Alice is corrupted. At least, his "secure" and "user-friendly" email software tells Bob the emails are corrupted. Bob blames this constant corruption on the poor Internet connectivity, but the real cause of the problem is Tom, another member of the expedition. Tom is Alice's secret admirer, and frequently borrows Bob's laptop to "write-up his notes". In reality Tom has been tampering with the emails in Bob's inbox to suggest Alice wants to break-off their engagement. Of course, Bob's email client is secure enough to prevent these scandalous emails from ever being seen, but Bob still has the problem of not receiving anything from Alice. If only he could identify some parts of Alice's emails that had not been corrupted or, better still, recover the original emails.

In 2000, Ashman proposed a new approach to protecting a document from tampering, known as multi-layered document integrity (MLDI) [1]. The approach involves checking a document's integrity at different levels, rather than the integrity of the document as a whole. The paper briefly describes a method, which is claimed to have substantial benefits over normal hashing, providing the ability to detect, locate, and correct tampering within a document. Tampering is detected by hashing the entire document using a standard cryptographic hash function, in the same way as current schemes. Subsequent tamper location is achieved by hashing each subsection of the document, which is then iterated through further subsections. Tamper correction can be achieved by using small preimage hashing of the subsections, which allows an exhaustive search through all possible subsections to determine the original. There are several problems with MLDI as it was originally stated, but despite these problems the idea postulated by Ashman is sound and warrants further development. A critical evaluation of the original MLDI paper is given in Chapter 2.

The idea of tamper detection is synonymous with that of *data integrity* and, despite the lack of a formal definition, there is a general consensus for the meaning of the problem. However, the ideas of locating and correcting tampering, as described in Ashman's paper, suggest a more general notion of data integrity. Even the definition of tampering has two similar – but fundamentally different – definitions in common use. The common definition is that tampering is the act of unauthorized modification of data, irrespective of whether the receiver accepts it as valid or not. The resulting problem of tamper detection is that of determining whether the data is original. An alternative definition of tampering is associated with the act of the receiver accepting an illicitly modified (or counterfeit) message as a valid message from the sender. This blurred definition can be used to justify claims such as "tamper proof", "prevent tampering" and "tamper protection", which are misleading in the absence of a formal definition [4, 29, 36, 42].

1.1 Aims and Objectives

The initial aim of this thesis was to investigate the idea of multi-layered document integrity, and to implement Ashman's algorithm in a proof-of-concept software tool that can protect a document from tampering. The tool was to provide a testbed for analysis of the algorithm, and a basis for further research. However, the process of designing and developing the software, and the subsequent preliminary results, highlighted several issues with MLDI. In particular, simulating the end-to-end process in software, highlighted issues with the process, initially raising concerns with the claims made in Ashman's original paper. Furthermore, preliminary results from the software tool suggested that correcting small amounts of data in a reasonable time, required a disproportionately large amount of redundant hash data.

To enable further research on MLDI, a more thorough definition of the problem was required, which prompted the main stem of theoretic research documented in this thesis. Moreover, the literature review identified a lack of research covering the broader, and potentially more significant, area concerning the problem of data integrity in general. Consequently, the main aim of this thesis is to provide a better understanding of the general data integrity problem. More specifically, this thesis aims to differentiate the problem of data integrity from other problems concerning modification of data; and provide a means to effectively describe and reason about the problem and its potential solutions.

This thesis also aims to address the issues with MLDI as it was originally presented, providing an in-depth understanding of the underlying principles involved with the idea. By resolving these issues, this thesis aims to develop the idea of multi-layered document integrity into a concrete scheme, in order to verify Ashman's claims. The final aims are to determine the efficacy of MLDI schemes, and evaluate Ashman's idea for solving the data integrity problem.

The objectives of this thesis are to conduct an overview of the general literature concerned with the modification of data, in which the discussion should be focussed towards deliberate and unauthorized modification. The review should highlight any of the problem's perceived differences that occur within the literature, providing a basis for a formally defined problem model. The model should consider the semantics of data integrity beyond that of the tamper detection problem, to encapsulate a broader notion of the problem.

This thesis will critically review the idea of MLDI as it was originally postulated, to determine if such an idea can be realized. The idea will be presented in depth, and with sufficient justification; introducing all the relevant concepts, principles, and algorithms. A theoretical model for MLDI will be developed in order to determine whether the idea can provide an effective solution to the data integrity problem.

1.2 Scope

This thesis is concerned with the preservation of data integrity where the data is subject to modification from a persistent adversary. The deliberate and unauthorized nature of the modification considered in this thesis differentiates this work from the subjects of revision control systems (see §2.1) and error-correcting codes (see §2.2), which otherwise have commonalities with aspects of this work.

Within the context of computer security, this thesis broadens the commonly perceived notion of data integrity to include the problems of tamper prevention, correction, and location (see Figure 1.1). This generalized notion of data integrity is discussed in the context of a malicious adversary that attempts to undermine all aspects of data integrity (i.e. more than tamper detection).

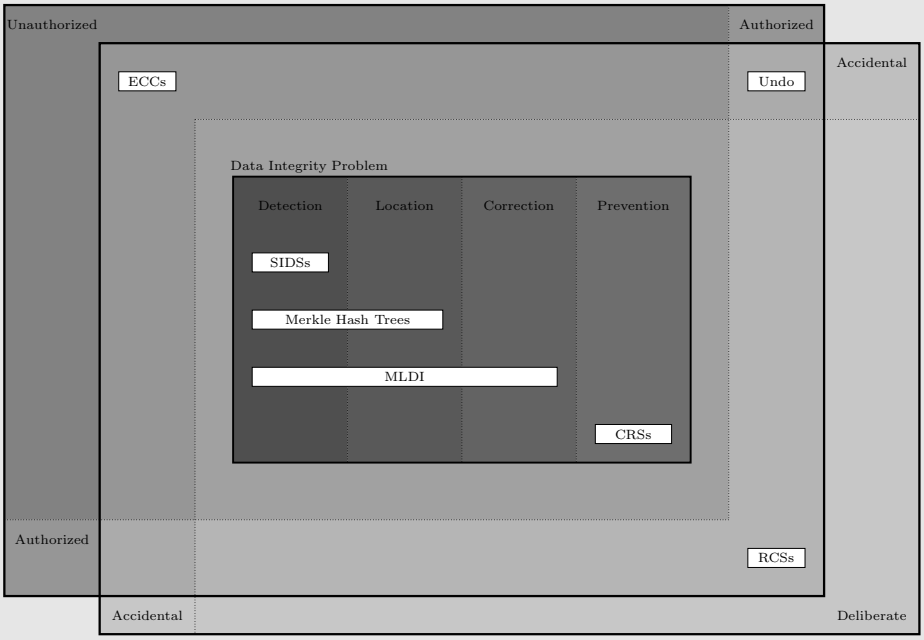

Figure 1.1: Thesis Scope in the Context of Modification Types

This work specifically focusses on the problem of a single atomic copy of the data, which is stored on a single device or transmitted over a single channel. The possibility of distributing — either entire copies or parts of — the data is not considered, and therefore the sole copy of the data is susceptible to tampering. Data privacy is not considered as a co-requisite for integrity, and hence the sole data copy is assumed publicly readable. These limitations model the notion of data transmission as originally described in MLDI, and are reflected in the adversary and process models described in Chapter 3.

Furthermore, it is assumed that: the adversary knows the system being adopted for security (Kerchoffs' Principle); the defending and attacking parties have the same resources and capabilities; and no physical measures are used to compromise the schemes.

Due to the vastness of the subject and the theoretical results which limit the use of MLDI, only algorithms that fulfil the properties described in Ashman's original paper [1] are discussed in detail. Further optimization of the basic algorithms is discussed in §9.3.

Despite its theoretical nature and heavy reliance on Mathematics, the research presented in this thesis falls firmly within the discipline of Computer Science. Extensive use of mathematics can be found in a wide range of fields within Computer Science, including many areas of Computer Security.

1.3 Contributions

This thesis contributes knowledge that is distinct from any other research, being (as far as the author is aware) the first to consider the wider data integrity problem. The thesis also constitutes the first in-depth investigation of Ashman's multi-layered document integrity idea.

The following is a list of the most significant contributions of this research, which are discussed in more detail in §9.2. The main areas of contribution concern models for the generalized data integrity problem and Ashman's proposed solution.

- A general review of the literature concerning the modification of data;
- A formal hierarchical definition of the data integrity problem;
- Establishing that detection is the only solvable problem in the vulnerable preservative model;
- Establishing the non-existence of data-only integrity detection;
- Establishing an underlying model for multi-layered document integrity;
- Describing a well-defined set of algorithms for preservation and resolution in MLDI;
- Outlining several feasible attacks against MLDI algorithms;
- Establishing the efficacy of multi-layered document integrity;
- Establishing that detection is the only solvable problem in the invulnerable preservative model.

Aside from the literature review, the contributions from the two models are defined in terms of their propositions, corresponding proofs, and related original definitions.

1.4 Structure

This thesis is divided into four distinct parts, which are then divided into one or more chapters.

- Part I discusses the background to this work; giving an overview of prior knowledge in this research field.
  - Chapter 1 provides the overall motivation for this work, and also includes an overview of the notational style adopted throughout the remainder of the thesis.
  - Chapter 2 surveys related work, covering research that may be confused with this work, research that approaches it, and research that solves the problem using alternative methods. It provides a discussion and review of related and similar work, including an in-depth description and critical discussion of Ashman's original paper on multi-layered document integrity.
- Part II addresses matters relating to the general problem of data integrity.
  - Chapter 3 defines the fundamental aspects of the problem in terms of users, information and processes.

  - Chapter 4 presents the problem model as a formal problem semantics, defining each sub-problem in terms of how a solution should behave and whether a given solution can solve an instance of the problem. These formal definitions are then used as the basis for the security model.
  - Chapter 5 reasons about the data integrity problem with a discussion and series of propositions directed towards establishing whether each of the sub-problems is solvable in the predefined models.

- Part III develops and evaluates a postulated solution.
  - Chapter 6 introduces the fundamental concepts of MLDI within the context of a simplified deterministic model. The model is based around the assumption that collision-free hashing is used. The notion of bounding and the divide-and-conquer approach for reducing the search space are discussed in this chapter, and the first preservation and resolution algorithms are given.
  - Chapter 7 gradually introduces the more-realistic probabilistic model, in a series of increasingly complex collision models. Each model is described in terms of its increasingly realistic assumptions, and the required algorithm modifications are described.
  - Chapter 8 presents results concerning MLDI; defining the main constraints of MLDI, and determining whether these constraints can be satisfied.

- Part IV concludes with a summary and discussion of future work.
  - Chapter 9 gives a summary of the entire thesis and its findings. The contributions of this research are given in full, and some possible areas of future work are described. The thesis concludes with a short discussion regarding the findings of this research.

This thesis has two appendices.

- Appendix A is a number of short definitions for pre-existing function properties referred to in the main text. Compiling them in an appendix prevents repetition and allows for brevity in some of the more complex definitions within the main text.
- Appendix B is aimed at readers who wish to implement any of the algorithms. It outlines some of the preliminaries used in the algorithms, and presents all of the MLDI algorithms in full.

1.5 Notational Style

The convention adopted for this thesis is to denote data (values, types, data types, objects) with single lower-case Latin characters. Processes (functions, algorithms) are denoted in two different ways, depending on their originality. If a process was documented before the work of this thesis, then it is denoted by a single upper-case Latin character; If the process is original to the work of this thesis, then it is denoted by a single Greek character of either case.

To denote several instances of a data item, the common notation of a numerical right subscript is used. Non-numerical right subscripts denote ownership of data or a process, and this is extended to private ownership using both an over bar and under bar on the subscript. An exception to this notation occurs when a counter variable is used to denote a generic instance of a data item (e.g. d_i).

Persons or parties (groups of persons) are denoted with single upper-case script characters when used in definitions, propositions, examples, etc., and named fully in standard text when discussed in the main text body.

Universal sets are denoted with single upper-case blackboard-bold characters, which may be embellished with a right superscript to add a maximal length for elements of the set.

	Examples	UPPERCASE	lowercase
Roman	H, h	Existing Process	Data
Greek	Π, π	Original Process	Original Process
Script	$\mathcal{A}, \mathcal{T}$	Party or Person	-
Blackboard Bold	$\mathbb{B}, \mathbb{N}$	Universal Set	-

Table 1.1: Summary of Notation

To be consistent throughout, and in order to formally prove some results, a notational style for function application is used that may be unfamiliar to some readers. Rather than the conventional style of using the function name followed by parentheses which contain the function's arguments, a more formal approach is adopted. Functions can only have a single argument, which may be a tuple containing multiple data items. This approach also allows the composition of two or more functions into one, when the relevant input and output data types correspond accordingly. In the case where functions are abstract and the notation is well-established, this alternative notation is deviated from in favour of the more common notation. For example, $\log(\ldots)$ is used for the logarithm function, $O(\ldots)$ is used for computational complexity, and $K(\ldots)$ for the Kolmogorov complexity.

When deriving one statement from another in a proof or example, the derivations are written in the following form:

$$
\begin{array}{rl}
 & \textit{statement } A \\
= & \quad \{\text{explanation of why } \textit{statement } A = \textit{statement } B\} \\
 & \textit{statement } B \\
= & \quad \{\text{explanation of why } \textit{statement } B = \textit{statement } C\} \\
 & \textit{statement } C \\
\Rightarrow & \quad \{\text{explanation of why } \textit{statement } C \Rightarrow \textit{statement } D\} \\
 & \textit{statement } D
\end{array}
$$

etc...

This style of derivation is adapted from that of Gries and Schneider's book, *A logical approach to discrete math* [24], which has been an extremely useful tool in producing some of the results of this thesis. The style used for linear quantification has also been adapted from the same text, following a similar style to set and list comprehension, where braces and square brackets are used respectively. In the case of quantification, angled brackets are used to delimit the scope

of dummy variables, with the *quantifier*, *range*, and *term*, written as follows:

$$\langle\, quantifier \mid range\colon\ \ term \,\rangle.$$

A single equals symbol (i.e. $=$) is used to denote equality between its left and right operands. The statement $x = y$ should be read as a proposition that x is equal to y. Assignment is denoted with a single colon followed by an equals symbol (i.e. $:=$), so the statement $x := y$ is used to denote that x takes the current value of y. Finally, definition is denoted with a double colon followed by an equals symbol (i.e. $::=$), so the statement $x \ ::= \ y$ is used to denote that x is defined in the same manner as y, such that it is not possible to fully-evaluate y (e.g. if y is a function with an undefined argument).

Notations used exclusively in the algorithms, are discussed in Appendix B.

Chapter Summary

The common consensus that data integrity concerns only the problem of tamper detection has led to spurious claims of systems that can guarantee tamper-free data. Furthermore, this view of data integrity does not encapsulate the true meaning of integrity, where maintaining the truth is equally as important as determining it. The idea of multi-layered document integrity postulated by Ashman aims to provide a level of data integrity beyond that of detection, thus furthering the need for a more general notion of the problem.

This thesis aims to conceptualize the general data integrity problem, and determine whether multi-layered document integrity can provide a feasible solution. The aims and objectives have been discussed in detail, and the scope of this thesis has been defined; differentiating the focus from a variety of schemes involved in the processes of data modification. The main contributions of this thesis have been outlined and its significance has been discussed. A description of the various notational styles used throughout this thesis has been provided for the less-obvious conventions used to present this research.

Chapter 2

Related Work

It is a capital mistake to theorize before one has data. Insensibly one begins to twist facts to suit theories, instead of theories to suit facts.

— Sir Arthur Conan Doyle, *A Scandal in Bohemia* (1891)

Many research problems are concerned with the modification of data, and commonalities exist between different solutions. This chapter discusses some of the well-established research problems (and solutions) that relate to data modification. In particular, the focus is on methods for detecting the presence of modifications, locating where the modifications have occurred, and correcting or restoring the modified data to its original state.

When discussing this work within the research community, the similarities with revision control or error-correcting codes are often noted. Therefore, these are briefly discussed in the first two sections of this chapter, and the fundamental differences are highlighted.

The first section deals with modifications that were made legitimately, but require monitoring; and the second with accidental modification, such as that experienced in transmission over a noisy channel. The final sections discuss the problems surrounding malicious modification, with which the problem of tampering is associated.

2.1 The Revision Control Problem

When data is modified by someone who is legitimately allowed to make changes to the data — the modification is *authorized*. Authorized modification is implicitly deliberate, as permission is given assuming modifications are intended. When modification of the data is expected, the changes can be recorded to allow later reversal of the modifications, if desired. This is the case for the undo feature in many software applications, and is also the basis behind version control or *revision control systems*.

The problem of revision control is that of how to keep multiple file versions well organized. One solution to revision control in text files is the software tool RCS (Revision Control System) [41]. RCS manages multiple revisions of a file for storage and retrieval, but also provides features for logging, identification, merging revisions, and access control, which facilitates work in group-based environments. It is most useful for frequently revised data, such as source code and documentation.

RCS manages revisions in an ancestral tree, which has a root revision and successive revisions for each version. Revisions are stored in the form of separate deltas, which are essentially the

differences between successive revisions. A delta is the sequence of modifications to transform one string into another. RCS deltas are line-based, so insertion and deletion of lines are the only modifications permitted. If just one character in a line is modified, the entire line is considered to have been changed. In RCS, the most recent revision on the trunk is stored intact. All other revisions on the trunk are stored as reverse deltas, which describe how to reverse a modification.

The ancestral tree in RCS holds information about the changes which have already occurred, allowing them to be reversed afterwards. As such, updates to the ancestral tree must also be made from an authorized party. Therefore this method of restoring modified data is not suited to situations where the modifications are unauthorized.

2.2 The Fault Problem

In many situations, whether data is stored or transmitted it may be subject to faults or errors. These errors are usually due to fault-prone storage or a noisy transmission channel. Television and radio broadcasts, static and mobile telephone networks, and particularly computer networks, all rely upon the detection (and sometimes correction) of errors. Since the errors are of a non-deliberate nature, they occur randomly. Such changes are generally detected more easily than non-random changes, because any structural patterns that exist within the data are disturbed. For example, if the data consists of a string of bytes in which every first bit is redundant and is always unset (of value 0), then the detection of a byte with its first bit set (of value 1) would indicate an error. This simple example only allows the detection of errors (and only if the byte's first bit was affected), but there are many complex schemes that are able to detect, locate, and correct these accidental errors.

2.2.1 Error Detection

The most widely used and well-known scheme for error detection is parity checking, which uses parity bits to check that data has been transmitted correctly. The transmitted information has a parity bit appended to each chunk of data, usually 7-bit chunks with a parity bit (a byte in total). Each chunk has either an odd or an even number of set bits. Assume, for example, that communication is being done with even parity (the most common form of parity checking). As data is transmitted, the number of set bits in each chunk of seven bits is counted. If the number of set bits is even, then the parity bit is unset, but if the number of set bits is odd, then the parity bit is set. In this way, every byte has an even number of set bits. As the data is received, each byte is checked to make sure that it has an even number of set bits. If a byte contains an odd number of set bits, then there was an error during its transmission. This is a basic form of error detection used in communications: Some errors are detected, but it cannot detect situations in which an even number of bits in the same data chunk have been changed; and, the parity bit is itself is also susceptible to error. Detection of an error might prompt a request that the data is retransmitted. However, in many situations this is not possible, and so there is a need for methods that correct errors detected in the data.

2.2.2 Error Correction

As discussed, an error correcting mechanism that can cope with multiple errors is crucial for data to be transmitted safely. For example, information transmitted from satellites exploring outer space is, not only expensive, but also highly subject to error. It is therefore increasingly expensive if the data is made useless due to non-correctable faults. Just as parity checking embeds redundant information to detect faults, an error-correcting code (ECC) embeds information to correct them.

A repetition ECC can correct multiple errors by repeating each bit of the original data as it is being sent [33]. The code uses two codewords with length equal to the number of repetitions. For example, a $(5, 1)$-repetition ECC uses the codewords `11111` and `00000` to represent set and unset bits respectively. Receiving the word `10111` could have resulted from two situations; sending the codeword `11111` with a single error or sending the codeword `00000` with 4 errors; the former being the most probable would suggest the word is corrected to `11111`. This ECC can correct up to two errors, but if three errors were to occur then the ECC cannot determine the original word. Obviously, the greater the codeword length the greater the number of errors that can be corrected, but this is also directly proportional to the volume of data to be transmitted. Generally, a $(k, 1)$-repetition ECC can correct up to $(k - 1)/2$ errors, and is not very efficient. However there are many other ECCs that are more efficient, but at the cost of using a more complex techniques (e.g. Hamming [26] or Reed-Solomon [34] codes). ECCs are an extremely useful tool, but fall outside the scope of this work. Whilst ECCs can be very effective at detecting, locating and correcting accidental modifications (errors), they cannot be applied to situations where modifications are deliberate (tampering). It is necessary to store or transmit this redundant data separately from the original data in a multi-layered document integrity scheme, so that an adversary cannot tamper with the original data and then change the redundant data to hide the tampering.

This work is more concerned with the problem of unauthorized and deliberate changes in data as opposed to authorized or accidental changes. This problem is fundamentally more difficult to solve, since the modifier can make subtle changes to the data that might imitate the original data. The modifier may also make changes to redundant bits used to track changes or detect and correct errors so as to reflect their subtle changes, thereby hiding any evidence of their unauthorized modifications.

2.3 The Commitment Problem

The commitment problem is one that occurs frequently in games played with two or more players. One of the most simple ways to illustrate the problem is described in Blum's coin flipping by telephone [28], and is shown in the following example:

Example 2.1 (Unfair Coin Flipping by Telephone)

Alice and Bob have just divorced and want to decide who gets the car by flipping a coin. As they live in different cities, they must make the decision over the telephone. They decide that Alice will flip the coin and that the Boolean values *true* and *false* denote heads and tails respectively.

Alice flips the coin to determine the Boolean flip value x, she keeps this value to herself. Then Bob makes his guess g of the flip value, which he tells Alice over the telephone. If $x = g$ then Alice should tell Bob that he has won. However, Alice can ensure she always wins by changing the flip value whenever Bob guesses correctly to indicate that Bob guessed incorrectly.

The problem can be solved with the use of a primitive known as a *cryptographic hash function* (CHF), for which there is no obvious relation between its input and output (see Definition 2.1). Moreover, one cannot predict the change in hash for a given change in the input and vice versa.

Definition 2.1 (Cryptographic Hash Function)

A *cryptographic hash function* H is a function that maps an arbitrary finite-length input string to a fixed-length output string h, known as the hash, hash value or digest [23, 29].

Properties:

(i) Easy to compute;

(ii) Compressive;

(iii) Preimage resistant;

(iv) Second preimage resistant;

(v) Collision resistant.

Cryptographic hash functions are most commonly used to check a file or message's integrity. Properties (i) and (ii) ensure that the function is practical to use; the former in terms of time and the latter in terms of space. Property (iii) allows a message's hash to be made public without revealing any information about the contents of the message from which it was derived. This is particularly important, so hash functions can be used to time-stamp a message without revealing its contents to the time-stamping service. Properties (iv) and (v) are similar, and are justified in the situation of hashes used in digitally signing messages. Property (iv) prevents the situation whereby an adversary can find a counterfeit message with matching signature to a signed message they have been given, then later switching the messages. Property (v) prevents the situation whereby an adversary can find two different messages (one true and one counterfeit) which will have the same signature, presenting the true message to be signed and then later switching the signature to the counterfeit message.

In terms of the commitment problem, property (iii) *conceals* Alice's flip so she can send it to Bob before his guess, whilst property (v) *binds* Alice to the flip, preventing her from changing after Bob has guessed. The following steps show how Alice and Bob can "flip a coin" fairly by telephone [28]:

1. Alice flips the coin by randomly selecting a bit value x (i.e. `1` and `0` might denote heads and tails respectively);
2. Alice picks a large random number x_1;
3. Alice conceals her flipped value as w using a cryptographic hash function H, such that $w := H \cdot (x \| x_1)$;
4. Alice sends her concealed value w to Bob;
5. Bob makes a guess g and sends it to Alice;
6. Alice reveals both the flipped value x and random number x_1 to Bob;
7. Bob verifies that the flipped value is legitimate by computing $w = H \cdot (x \| x_1)$.

Bob will now win whenever his guess g is equal to the flipped value x, since for Alice to convince Bob she has won in this case she must determine a value x_2 such that $w = H \cdot (\bar{g} \| x_2)$. The *preimage resistance* (see Definition A.5) of the CHF, prevents Bob from determining the true value of the flip from the concealed value before making his guess. The second preimage resistance and collision resistance of the CHF prevent Alice from determining the counterfeit involving x_2.

Cryptographic hash functions are documented extensively in literature [3], and their design constitutes its own field of active research. Three well-known and widely used cryptographic hash functions are MD5 and SHA-1 [35, 19]. In 1996, the use of MD5 has been discouraged in favour of SHA-1 [16], and more recently, successful attacks against the MD5 algorithm were confirmed in 2004 [45]. In 2005, attacks against the full SHA-1 algorithm were found that require less than 2^{69} hash operations to determine a collision [46] — significantly less than an exhaustive brute-force search. Currently, hash functions in the SHA-2 family [20] are recommended.

2.4 The Privacy Problem

The privacy problem is probably the oldest and most widely discussed problem in cryptography. Essentially the problem involves three parties: Alice and Bob, who wish to communicate a secret over an insecure channel; and Eve who wishes to discover the secret by eavesdropping on their communications.

If the channel is insecure, then anything transmitted on it is effectively public information, and hence the adversary has access to it. The problem can be solved by the use of a symmetric encryption scheme, also known as a shared- or *secret-key cryptosystem*.

Definition 2.2 (Secret-key Cryptosystem)

A secret-key cryptosystem is comprised of a public encryption/decryption function E and a private key $k_{\overline{\mathcal{AB}}}$ shared between communicating parties $\mathcal{A}$ and $\mathcal{B}$. For simplicity, secret-key encryption/decryption using a key shared by Alice and Bob is denoted $E_{\overline{\mathcal{AB}}}$.

Properties:

(i) $E_{\overline{\mathcal{AB}}}$ is self-invertible;

(ii) $E_{\overline{\mathcal{AB}}}$ is preimage resistant.

In terms of the privacy problem, property (ii) ensures that only Alice and Bob can determine a message m from $E_{\overline{\mathcal{AB}}} \cdot m$, since only they know $k_{\overline{\mathcal{AB}}}$. If Alice wishes to send a message m to Bob using this cryptosystem:

1. Alice conceals the message as w, such that $w := E_{k_{\overline{\mathcal{AB}}}} \cdot m$;
2. Alice sends the concealed message w to Bob;
3. Bob determines the message using $m := E_{k_{\overline{\mathcal{AB}}}} \cdot w$.

Although this solves the privacy problem, there is a further problem in the initialization of this protocol: It requires Alice and Bob to have prior knowledge of a shared secret — the key $k_{\overline{\mathcal{AB}}}$. This problem is commonly referred to as the *key agreement* or *key exchange problem*.

2.5 The Key Exchange Problem

The key exchange problem [15] is essentially the problem of establishing a secure transmission channel via the secure exchange of a key, which itself requires a secure transmission channel. This problem occurs when there is a need to transmit data securely, but is not such an issue when storing data securely. This is due to both encryption and decryption occurring in the same physical location; hence the key is not exchanged and can be secured more easily. For example, the key could be stored on a physically secured mobile device (e.g. a smart card) or even memorized by its owner as a passphrase [18].

The problem is solved with an asymmetric encryption scheme; commonly known as a *public-key cryptosystem* (see Definition 2.3). In such a scheme it is computationally difficult to find the decryption function from the encryption function, allowing the encryption function to be made public without compromising security.

Definition 2.3 (Public-key Cryptosystem)

A *public-key cryptosystem* is comprised of a public encryption/decryption function E and two keys: one public encryption key $k_{\mathcal{A}}$ and one private decryption key $k_{\underline{\mathcal{A}}}$.

Properties:

(i) $E_{\mathcal{A}}$ is invertible by $E_{\underline{\mathcal{A}}}$;

(ii) $E_{\mathcal{A}}$ is preimage resistant;

(iii) Given E and $k_{\mathcal{A}}$, $k_{\underline{\mathcal{A}}}$ is hard to compute.

For a public-key cryptosystem, properties (ii) and (iii) ensure that only Alice can determine m from $E_{\mathcal{A}} \cdot m$. The idea of a function that is hard to compute unless some secret information is used, is known as a *trapdoor function*. This idea is used, not only solve the key exchange problem (and therefore the privacy problem), but also in solving many other computer security problems such as authentication, zero-knowledge proofs [21] and verifiable secret sharing [10].

2.6 The Authentication Problem

The authentication problem is that of allowing the receiver of a message to authenticate its origin. For example, if Alice receives a message claiming to be from her business partner Bob asking her to send back the combination of their safe, how does she know the message really came from Bob? Alice has no way to authenticate the origin of the message.

The problem occurs because Tom knows everything that Bob knows (i.e. nothing) and so Tom has the same abilities as Bob. Therefore, if Bob can use some protocol to convince Alice that he is Bob, Tom can use this same protocol to convince Alice that he is Bob.

Definition 2.4 (Digital-signature Scheme)

A *digital-signature scheme* is comprised of two functions: one private function $S_{\underline{\mathcal{A}}}$ that allows its owner $\mathcal{A}$ to *sign* messages and produce a *signature* s; and one public function $V_{\mathcal{A}}$ that allows anyone to *validate* a the owner's signature to determine its authenticity v.

Properties:

(i) Universal validity;

(ii) Forgery resistant.

Various levels of forgery have been defined [22], which essentially correspond to the different resistance properties of a cryptographic hash function (see Definition 2.1). In the context of this thesis, if an adversary is given a message and its signature and is able to construct a different message with the same signature, then this constitutes forgery.

2.7 The Integrity Problem

The integrity problem is closely related to that of authentication, in that unauthorized modifications are considered to be a compromise of integrity. The majority of literature discussing data integrity focusses on the problem of detecting any modification, whilst relatively little attention has been given to the problems of locating and correcting modifications [42, 38, 36, 14, 37].

2.7.1 Tamper Detection

The tamper detection problem is concerned with determining unauthorized modification in data. If Alice sends Bob a message, then Bob needs to determine whether the message he receives is the same message that Alice sent. The solution is an integrity detection scheme (IDS), which is commonly implemented using digital signatures. A signature-based integrity detection scheme is essentially the same as a digital-signature scheme, but the primary goal of signature-based integrity detection is to protect the document's integrity rather than its authenticity.

Definition 2.5 (Signature-based IDS)

A signature-based integrity detection scheme (SIDS) is comprised of the same two functions as a digital signature scheme (see Definition 2.4).

The sign function computes the encrypted hash of a given message using the private encryption function, therefore $S_{\underline{A}}$ must remain private. The validation function evaluates to *true* only when given a document with the same hash (using $H{\cdot}m$) *and* when corresponding public and private keys are used to construct the functions. Otherwise, it evaluates to *false*. Since both H and $E_{\mathcal{A}}$ are public, $V_{\mathcal{A}}$ is also publicly available.

If an attacker should modify the message or its signature in any way, then $V_{\mathcal{A}}$ will evaluate to *false*, since the adversary cannot compute a new corresponding pair (m_2, s_2) such that $V_{\mathcal{A}}{\cdot}(m_2, s_2)$ evaluates to *true*, since $S_{\underline{A}}$ is private.

2.7.2 Tamper Location

The tamper location problem concerns determining which parts or blocks of data have been modified. One possible solution is to split the data into a series of blocks and apply an IDS to each block. In practice this method is not used due to its large computational and memory overheads. A more practical solution is the *Merkle hash tree* [30], which is commonly used to prevent damaged and counterfeit data from propagating through peer-to-peer networks.

A Merkle hash tree (originally known as an authentication tree) is a binary tree in which the leaf nodes are hashes of the data blocks of a file (or set of files) and internal nodes are the hashes of their concatenated children's hashes. Figure 2.1 shows an example Merkle hash tree where h_i denotes $H{\cdot}m_i$, and $h_{i\|j}$ denotes $H{\cdot}(h_i\|h_j)$.

The master hash (also known as the root or top hash) is obtained by the client from a trusted source (e.g. a trusted web site), which is used to verify the integrity of the hash tree

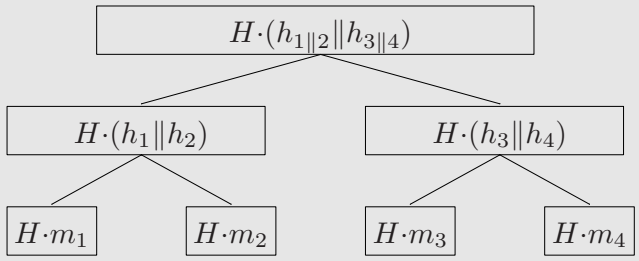

Figure 2.1: A Simple Merkle Hash Tree

from any non-trusted source (i.e. a peer on the network). Since large files shared on peer-to-peer networks are shared more efficiently when split into smaller data blocks, the use of hash trees allows the integrity of these blocks to be verified individually. The larger the file, the larger its hash tree becomes. However, only the branches which correspond to the blocks being downloaded are required for verifying those blocks.

For example, suppose a document is split into four blocks (m_1, m_2, m_3, and m_4) as shown in Figure 2.1. The client Alice has determined the master hash from a trusted source, such as a well-known Web page. Alice can verify the integrity of m_1 downloaded from an untrusted server (Bob or Tom), by also requesting the hashes h_2 and $h_{3\|4}$ from the server. By hashing m_1 to determine h_1, Alice can then determine $h_{1\|2}$ and therefore $H{\cdot}(h_{1\|2}\|h_{3\|4})$, which can be verified against the master hash.

Assuming the downloaded block and hashes verify correctly, Alice can then repeat the process to download another block from the same source, utilizing the hashes already verified to reduce the overheads. If, at any point, the current block and hashes do not verify correctly, then the latest unverified information can be disregarded and the process can continue with an alternative source.

Supposing Alice chooses Bob as the untrusted server. Since Bob is honest, this should correctly verify the integrity of m_1, allowing her to repeat the protocol for m_2. However, if she chooses Tom as the untrusted server, the integrity verification of the block will fail, since Tom is dishonest. Alice would then repeat the protocol for m_1, but selecting another untrusted server (e.g. Bob), since she now knows not to trust Tom.

Given that Merkle hash trees can verify integrity block by block, there is a relatively small increase in the amount of transmitted data for all untrusted servers in comparison to verifying the data as a whole. However, there is a relatively large decrease in the amount of transmitted data when one or more dishonest servers are selected.

The Merkle hash tree method exploits network diversity, and relies on the fact that trusted servers exist. If a similar method is adopted with a single dishonest server (or alternatively all servers on the entire network are dishonest), then the scheme will only provide integrity location and some blocks will remain unverified. Example 2.2 highlights why it could be dangerous to utilize partly-verified data.

Example 2.2 (Party-verified Data)

Supposing one intends to utilize the verified blocks of the following partly-verified data in which the first two data blocks were tampered with (denoted with "?"s):

"?????? PAY THE COMPANY ONE THOUSAND POUNDS"

The original data could have been either of the following possibilities that have contrary meanings:

"PLEASE PAY THE COMPANY ONE THOUSAND POUNDS"

"DO NOT PAY THE COMPANY ONE THOUSAND POUNDS"

2.7.3 Tamper Correction

The tamper correction problem involves both determining which blocks of data have been modified and reverting them to their original state. It is this problem which constitutes the main focus of this thesis, and therefore a detailed discussion is left for the following chapters. In this section Ashman's idea of *multi-layered document integrity* (MLDI) [1] is discussed, which the author believes to be the only work prior to this that tackles the problem.

Ashman tackled the problem of tampering with the approach that hash functions are non-deterministic, and hence the only way to determine the possible inputs for a corresponding output is to conduct an exhaustive search through all possible inputs. If, however, the hash function's input data size (preimage size) is small enough then the search space is reduced. So by sectioning the data into smaller chunks and hashing each chunk, the search space for determining the input for a given chunk's hash is reduced. The effect of this also allows an adversary to determine multiple chunks of data with identical hashes, and hence weakens the integrity provided by hashing. However, by hashing a document at many levels (hashing the whole document followed by hashing its subsections), the integrity can be preserved. This also allows for location of tampering, since checking the hash of a subsection determines whether or not that subsection has been changed.

This scheme can easily be applied to documents with a tree structure, where hashing occurs at each node and encompasses the data of all its sub-nodes. Therefore a change in any leaf node results in a mismatch of all hashes along the path connecting itself to the root (inclusive). To section the document at many levels, Ashman states that "*the document must be decomposed in such a way that every lower-level element belongs to some higher-level element*". By doing this, it is effectively superimposing a tree structure onto the document. This is done until the lowest level of the document, which may be decided by the size of the search space at that level and the importance of its recovery. The process of decomposing the document forms a type of *document tree*, which can then be hashed by hashing each level below the root in turn, until the lowest level is reached. The hashed data can either be sent separately to the document or it can be encrypted using a signature-based integrity detection scheme (similar to the way that *Pretty Good Privacy* (PGP) secures its integrity data [47]). When the document is received at its destination, its integrity as a whole is determined by comparing the top-level hash to the

hash of the received document. If there is a mismatch, then changes are further located by computing and comparing all hashes at the next level down. This is repeated until the changes have been located to the lowest level. At this stage the original leaf node can be determined by a brute-force search through all possible leaf nodes. It is highly probable that multiple leaf nodes have matching hashes, but identifying which of these multiple possibilities is correct, can be determined by checking them at higher levels.

An *Ashman hash tree* (originally known as a *tree of hashes*) is an arbitrary tree in which the leaf nodes and internal nodes are weakened hashes of the data blocks of a file (or concatenated data blocks for internal nodes). The top level or root hash is a cryptographic hash of all data blocks concatenated together (i.e. a hash of the entire file) (see Figure 2.2).

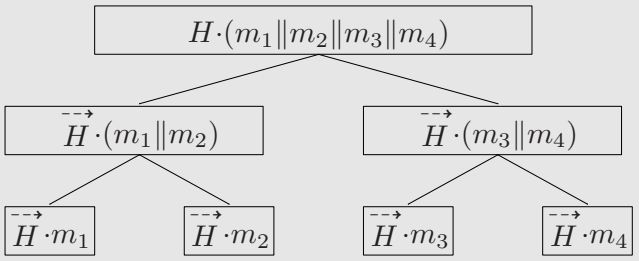

Figure 2.2: A Simple Ashman Hash Tree

There are two main differences between Merkle and Ashman hash trees. The first difference concerns their respective internal nodes, which are are direct hashes of data in an Ashman hash tree, rather than hashes of other hashes as in Merkle hash trees. The second difference concerns non-root nodes, since an Ashman hash tree uses weakened hashing as opposed to the cryptographic hashing used in Merkle hash trees. The reasoning behind these differences stems from Ashman hash trees aiming to provide the ability for correction, which requires weak hashes. If a Merkle-like "hash of child hashes" approach is used, then a counterfeit occurring at one node would not be recognized at its parent nodes, and the root hash (used for detection) would match, irrespective of any tampering.

There are several problems with Ashman's scheme as it is described, mostly due to explanations being too brief and, at times, ambiguous, but also due to some oversights. First the ambiguities are addressed, to fully understand the idea. Ashman describes the idea in terms of sending a document, for example, over the Internet, but this is not necessarily the only application. The idea can equally apply to the storage of a document on some tamper-vulnerable storage device. A second ambiguity is the use of the term "small hashes", which could mean that the hash input (preimage) is small in size, but equally could mean that the hash output (image) is small in size. From the context of Ashman's argument, the author assumes that "small hashes" actually refers to small preimage hashes, which make a brute-force search of the preimage space more realistic. However, small image hashes would also be beneficial to such a scheme, as it reduces the total size of the hash data. This poses a problem with Ashman's algorithm, in that, if standard cryptographic hashing was applied throughout the document tree

then the hash data would almost certainly be greater in size than the original data. Therefore the original data could be stored or transmitted in the same (tamper-invulnerable) manner as the hash data. Ashman does not comment on this scenario, but it is discussed in more detail in Chapter 3.

A further ambiguity is in the description of when to stop decomposing the document. In the first description, it is stated that the size of the search space at that level and the importance of its recovery determine the lowest level of document decomposition required. However, in the later example describing the algorithm in terms of decomposing image data, it is stated that decomposition occurs until the document section is homogeneous. The first description is correct: The size of the search space must be a factor in the decision (see Definition 6.4), but this statement is too vague to implement the algorithm. The problem of data (document) decomposition is discussed more thoroughly in Chapter 3.

2.8 The Censorship Problem

The censorship problem is fundamentally similar to that of integrity, with the common goal of data preservation in the presence of a malicious adversary. The problem concerns three participants: a *publisher*, a *viewer* and a *censor*. The publisher publishes documents for the viewer to view. However, the censor is attempting to suppress some or all of the published document via whatever means is necessary. Typically this situation might apply on the Internet, where a censor might pressure a publisher through legal action to remove content they have published on a Web page. Whilst there are many justifiable reasons for censoring information, in many cases censorship can also be unjustified. *Censorship resistance* is the term used to describe how well a publishing method can overcome the problem of censorship.

Example 2.3 (Alice and Bob's Censorship Problem)

Whilst working on a remote expedition in the Amazon, Bob's only means of communication is through the Internet, and his email has been unreliable since the start of the expedition. All the emails he's received from his fiancée, Alice, have been corrupted, and none of his replies to her appear to have been received. Wishing to allay any fears Alice might have for his safety, Bob opts to publicly announce his well-being (and email problem) by posting a message to a public newsgroup that he knows Alice (the viewer) reads frequently.

Initially, Bob experiences connectivity problems with the newsgroup server but, after several hours of persistence, finally manages to post his message. When Bob checks the newsgroup for a reply the following morning, he finds his message has been removed. Despite further attempts, none of Bob's messages appear on the newsgroup, and he is out of ideas.

The cause of Bob's problem is Tom, Alice's secret admirer, who does everything in his power to prevent them from communicating. Tom (the censor) initially tried to suppress Bob's message using a denial of service attack against Bob's Internet service provider, but Bob's persistence defeated the attack. Tom then successfully managed to bribe the newsgroup's editor (the publisher) to suppress all posts from Bob. If only Bob had access to some anonymous or unsuppressible publishing method, he would be able to communicate with Alice.

The main approaches to censorship resistance have been categorized as schemes for data replication, anonymous communication, server deniability and data entanglement [32]. *Data replication schemes* typically involve the creation and distribution of multiple copies of the data under the assumption that censorship of all copies is unlikely. *Anonymous communication schemes* attempt to provide censorship resistance through the use of encryption to protect the origins of the data. The assumption is that if the source of data cannot be determined, then it cannot be censored. *Server deniability schemes* such as Publius [44] rely on the idea of distributing encrypted copies of the data amongst many servers with each server holding only part of the decryption key. The assumption is that any given server cannot be held responsible for the content of the data as it does not posses the ability to determine its content. *Data entanglement schemes* such as Tangler [43] and Dagster [40] split data into blocks in such a way that a single block becomes part of several documents. The assumption is that there exist documents which are both desirable and undesirable for the censor, and hence removal of any undesirable document will also remove data blocks (and hence documents) which were desirable.

All *censorship resistance schemes* (see Definition 2.6) work on the principle of data distribution, although not necessarily in terms of replicated distribution. Each type of scheme has various problems, but these are beyond the scope of this discussion. Instead, the reason why censorship-resistance schemes do not provide a solution to the data integrity problem is highlighted.

Definition 2.6 (Censorship Resistance Scheme)

A censorship resistance scheme is any document publishing scheme that provides *non-revocation* and *accessibility*.

Properties:

(i) Non-revocation: It should be computationally infeasible for a censor to be able to force a publisher to revoke a published document.

(ii) Accessibility: It should be computationally infeasible for a censor to be able to prevent a viewer from accessing a published document.

There are two common approaches to achieving censorship resistance by satisfying properties (i) and (ii). The first approach involves the use of an anonymous publishing method, which prevents the censor identifying the publisher and therefore satisfies (i). This method may also helps towards satisfying (ii), since it may be difficult to block access to an unknown source, but may also introduce problems for the viewer obtaining the document from an unknown source. The second approach involves data distribution, which makes the blocking of all sources more difficult and therefore satisfies (ii). This method also helps towards satisfying (i), since the censor must determine multiple sources and then force them all to revoke the published document. (All sources need to be forced if all sources are publishing the entire document.)

In practice, all the schemes encountered use data distribution; probably due to it being both simpler and more robust. However, the use of distribution makes it equally difficult for authorized modifications to be made as for unauthorized censorship; therefore nobody is authorized to remove old documents or modify documents that may contain mistakes. This characteristic of censorship resistance schemes differentiates them from integrity correction schemes, and make them unsuitable for solving the data integrity problem.

Chapter Summary

Various methods of addressing modification in data have been discussed through a series of problems, describing the general or widely-accepted solution of each. Whilst revision control and error detection/correction schemes provide useful methods of monitoring changes, neither is suitable in the presence of a malicious adversary. The problems that followed, led to the common cryptographic primitives used for (and related to) solving parts of the integrity problem.

Ashman's idea of multi-layered document integrity has been discussed [1] as a solution to the tamper correction problem (as well as detection and location). However, several problems have been noted within MLDI [31] that need to be resolved to develop it into a concrete algorithm that can be implemented. The author believes that the most fundamental problem is the lack of a solid theoretical foundation, preventing the idea and its algorithm being described clearly. Despite the problems with MLDI, it seems to be a reasonable approach to solving the tamper correction problem, and therefore deserves further development.

Part II

The Data Integrity Problem

It isn't that they can't see the solution. It's that they can't see the problem.

— Gilbert Keith Chesterton, *The Point of a Pin* (1935)

Chapter 3

Fundamentals

Everything should be made as simple as possible, but not simpler.

— Albert Einstein [*attributed in Reader's Digest* (October 1977)]

This chapter discusses the fundamental problem of digital tampering. The aim is to highlight both the obvious and the not-so-obvious concepts, in order to define the scope of later chapters. To begin, some of the fundamental terms and concepts associated with the problem are outlined.

3.1 Participant Model

A *party* is considered to be a group of *participants* with a common *goal*. However, the *role* that each participant plays to achieve the goal may differ.

Essentially, the problem concerns two parties with adverse goals: The honest *defending party* attempts to protect data against tampering; whereas the dishonest *attacking party* attempts to modify the defending party's data. This model can be applied when the data is stored by Alice ($\mathcal{A}$) on a device and retrieved from that device by Alice at some later point in time. Therefore Tom ($\mathcal{T}$) might have an opportunity to tamper with the data before it has been retrieved.

A slightly different model is required when the data is transmitted by Alice over a channel and received by Bob ($\mathcal{B}$) somewhere else. Here the defending party consists of two different participants: a source and a destination. In this situation the process can be described using three participants. Alice is the honest defending source of the data, Tom is the dishonest attacker or *adversary*, and Bob is the honest defending destination of the data.

In order to reason about the problem in a simplified manner, these two models are generalized into one, where Alice is known as the *preserver*, Tom is known as the *tamperer* and Bob is known as the *resolver*.

An additional party is the *neutral party*, which can be considered impartial to both the attacking and defending parties. The goal of the neutral party is to ensure fairness between the other parties. Jude ($\mathcal{J}$) acts as an impartial judge on transactions between participants from the attacking and defending parties. Table 3.1 summarizes the roles of these participants within the proposed model.

Finally, the universal party Everyone ($*$) is added, which includes all participants.

Participant	Symbol	Party	Trust	Role
Alice	$\mathcal{A}$	Defending	Honest	Preserver
Bob	$\mathcal{B}$			Resolver
Tom	$\mathcal{T}$	Attacking	Dishonest	Tamperer
Jude	$\mathcal{J}$	Trusted Third Party	Mutual	Judge
Everyone	$*$	Universal party of all participants	N/A	N/A

Table 3.1: Contrasting Roles of the Parties

3.2 Transaction Model

When data is in the possession of a participant it is assumed that only that participant can modify it, and any modifications are authorized with respect to all participants in their party. So when Alice has some data, only she can modify it, and this modification is authorized with respect to Bob too (since they belong to the same party). A *transaction* is considered to be any process in which data is obtained or released by a participant; it signifies the point at which a participant gains or loses their possession of the data.

When data is not in the possession of a participant, it is assumed to be in storage on a *device* or in transmission over a *channel*; it cannot be passed directly from one participant to another. The terms *device* and *channel* are generalized into the term *medium* (see Definition 3.1). When data is *stored on* a device or *transmitted over* a channel, the term *expose via* a medium is used. When data is *retrieved from* a device or *received from* a channel, the term *obtain via* a medium is used.

The terms *vulnerable* and *invulnerable* are used to describe whether a medium is vulnerable or invulnerable to tampering, respectively. For example, "obtained via a vulnerable medium" would mean the data has been vulnerable to tampering, whereas "obtained via an invulnerable medium" means the data will be tamper free. Note that an invulnerable medium is similar to, but not the same as, an *authentic channel*, which is typically simulated using a digital signature scheme to provide authentication [29]. A simulated authentic channel provides a means for integrity detection, but does not prevent tampering from occurring. However, authentication must be a prerequisite for an invulnerable medium, otherwise the data may have an unauthorized origin.

Definition 3.1 (Medium)

The generic term used in place of *storage device* or *transmission channel* when no distinction should be made between the processes of storage and transmission. If the medium is vulnerable to tampering then it said to be a *vulnerable medium*, whereas an *invulnerable medium* is invulnerable to tampering.

3.3 Data Model

A partitionable data model is required for discussion of the tamper location problem (see Definition 3.2).

Definition 3.2 (Data Block)

Data is considered to be a *data block* d, which is a list of size $\#(d)$ that consists of one or more discrete sub-blocks d_i, such that

$$d \ ::= \ [d_0, d_1, \ldots, d_{\#(d)-1}]$$

As $\#(d)$ denotes the number of sub-blocks in d, the length of d in bits is denoted $|d|$ or simply n. Therefore d is one of 2^n possible data blocks in the set $\mathbb{B}^n$. Example 3.1 shows how this notion of a data block might be applied.

Example 3.1 (Data Block)

Consider a 32-bit encryption key k, which is a string of binary digits

$$k := 10011000101010111010000100101110.$$

It might be represented as a single data block with $n = 32$ and $\#(d) = 1$

$$d := \texttt{[10011000101010111010000100101110]}$$

or split into several data blocks with $n = 32$ and $\#(d) = 4$

$$d := \texttt{[[10011000]}, \texttt{[10101011]}, \texttt{[10100001]}, \texttt{[00101110]]}.$$

3.4 Adversary Model

By nature, tamperers are *active* adversaries (as opposed to *passive*), since they are able to modify the data during exposure. The modification can be categorized into three types of transformation: addition, deletion and replacement.

A *destructive* adversary has the ability to modify data with these transformations, but without being able to selectively modify the data. More formally, they apply a finite, arbitrary number of random transformations, provided that this has the cumulative effect of reducing the data's entropy [2]. Alternatively, an *arbitrary* adversary can select whichever transformations they wish to apply, allowing them unrestricted write-access to the data, which can range from deleting the data entirely to replacing it with any other data. In either case the cumulative transformation can be represented as a single transformation (see Definition 3.3).

Definition 3.3 (Tampering Function)

The deliberate modification of data is modelled by the *tampering function* τ, where tampering can only occur when the data is vulnerable (i.e. exposed via a tamper-vulnerable medium). For simplicity, original data d that has been tampered with is denoted f, such that

$$f \ ::= \ \tau \cdot d.$$

A *malicious* adversary is aware that the defending party will attempt to protect their data's integrity by any means available to them, and therefore the adversary uses their abilities to overcome any protection schemes adopted. A malicious adversary can be either destructive or arbitrary.

Finally, a *persistent* adversary will continually tamper with the defending party's data, rendering the *try and try again* method useless for the defending party. A persistent adversary may be malicious and/or arbitrary (or neither).

The adversary assumed in this thesis (Tom) is arbitrary, malicious and persistent.

3.5 Preservative Model

The defending party is attempting to protect the integrity of their data. Irrespective of the exposure method, they must adopt some kind of scheme to do this, which will require some information that is redundant with respect to the data itself. This redundant information is defined as the *preservative*, since its purpose is to preserve integrity. The preservative is computed before the data is exposed and acts as a guard or monitor for the data during exposure. This can then be used, when the data has been obtained, to provide information regarding its integrity.

Definition 3.4 (Preserving Function)

The *preserving function* ϖ is the initialization process applied to original data d as a preemptive method to preserve integrity. The process produces redundant data, the *preservative* p, such that

$$p \ ::= \ \varpi \cdot d.$$

This work describes two different preservative models — vulnerable and invulnerable. In the *vulnerable preservative model* (VP model), the preservative is exposed along with the data, allowing the adversary access to modify the data and its preservative (see Definition 3.5). This paradigm can be used to model an unrestricted medium which (intentionally or unintentionally) permits unauthorized write access. Many real-world systems fit this model: For example, transmission over the Internet using an signature-based integrity detection scheme (see Definition 2.5).

Definition 3.5 (Vulnerable Preservative Model)

In the vulnerable preservative model the original data d and its original preservative p are exposed together via a vulnerable medium. When the information is obtained, the exposed data e is either the original data d or the tampered data f, such that

$$(e = d) \neq (e = f)$$

$$\equiv$$

$$\neg(d = f). \tag{3.1}$$

Similarly, the exposed preservative q is either the original preservative p or the tampered preservative r, such that

$$(q = p) \neq (q = r)$$

$$\equiv$$

$$\neg(p = r). \tag{3.2}$$

The alternative paradigm would be a restricted medium which permitted only authorized write access. This is modelled with the *invulnerable preservative model* (IP model) (see Definition 3.8), where the preservative is exposed via an invulnerable medium, protecting its integrity. However, if such a medium existed, then it should be possible to expose the data via this invulnerable medium, therefore achieving tamper-free data without the need for a preservative. This situation is described in an analogous manner to the key exchange problem (see §2.5). In the key exchange problem the *key* had to be *exchanged* via a *private channel* to establish a *private channel*, whereas in the *preservative exposure problem* the *preservative* has to be *exposed* via an *invulnerable medium* to establish an *invulnerable medium* (see Definition 3.6).

Definition 3.6 (Preservative Exposure Problem)

The problem of using a preservative to achieve an invulnerable medium for data exposure, which is only possible if the preservative itself is exposed via an invulnerable medium.

An important distinction between keys and preservatives is that a key is generally unrelated to the data it is used to encrypt. As such, a key can generally be used to encrypt multiple data blocks. However, a preservative is generated for specific data and is of no use to other data. For similar reasons encryption keys are of fixed-size and can be kept relatively small in comparison to the data they are used to encrypt, whereas a preservative will be of variable-size relative to the size of its original data.

Solving the key exchange problem via whatever means, whether it is an expensive secure courier or a computationally-heavy public-key cryptosystem, becomes worthwhile when considering the ratio of key size to encryptable data size is high (for a single key). Therefore this work could employ a similar evaluation method to determine how worthwhile solving the preservative exposure problem would be.

As a single preservative can be used for just one specific data block, this work looks at the ratio between the size of the data and the size of its preservative. However, rather than measuring data and preservative size in terms of the number of bits, this work considers a measure of the information contained within each. This is done by using the *Kolmogorov complexity* (see Definition 3.7), to suppress any redundancy that might be discarded as a result of compression.

Definition 3.7 (Kolmogorov Complexity)

Given a string x, the *Kolmogorov complexity* $K(x)$ is the minimum number of bits into which x can be compressed without losing information. This is defined with respect to a fixed, but universal decompression scheme, given by a universal Turing machine [6].

Let $I(x)$ be the shortest description of the string x, then

$$K(x) = |I(x)|. \tag{3.3}$$

The tamper-invulnerable medium used for exposure of the preservative could potentially be used to expose the data, so there must be an upper-bound for the amount of information in the preservative. Consideration of this fact suggests that if the preservative is to be exposed via an invulnerable medium, then it must contain less information than the data it corresponds to, otherwise it becomes redundant. (One may as well expose the data via the invulnerable medium.) Therefore this work considers the possible existence of a *limited* invulnerable medium, which is restricted in the amount of information that can be exposed via it, and essentially restricting the size of the preservative. Definition 3.8 encompasses this restriction in Equation 3.4, limiting the Kolmogorov complexity of the preservative to be less than the Kolmogorov complexity of the data (see Theorem 5.2).

Definition 3.8 (Invulnerable Preservative Model)

In the vulnerable preservative model, the original data d and its original preservative p are exposed separately. The data being exposed via a vulnerable medium and the preservative via an invulnerable medium. When the information is obtained, the exposed data e is either the original data d or the tampered data f, in an identical manner to the VP model, and thefore Equation 3.1 holds. However, the exposed preservative q remains original, such that

$$q = p,$$

with the restriction that

$$K(p) < K(d). \tag{3.4}$$

3.6 Process Model

The most basic process model involves just three stages (see Figure 3.1). The *initiation* stage is where the data is produced in some way, and is therefore *original*. The *exposure* stage begins

when Alice exposes (stores or transmits) the data on a vulnerable medium. When the exposed data is tampered with by Tom, it becomes *counterfeit*. The *completion* stage begins when Bob obtains (retrieves or receives) the data from the vulnerable medium. As Bob has no way of knowing that the data has been tampered with, he accepts the counterfeit data as if it were original.

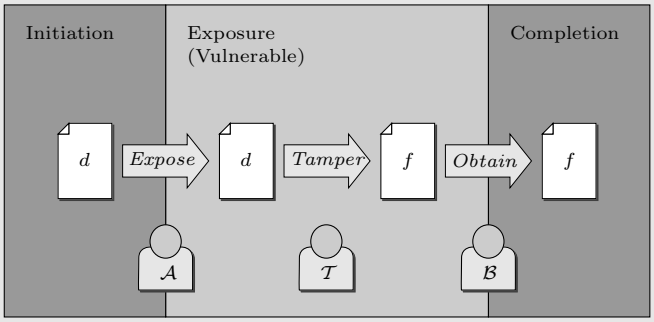

Figure 3.1: Basic Process Model

The ideal solution to this problem would be for Alice to expose the data on an invulnerable medium, rather than a vulnerable one (see Figure 3.2). By using an invulnerable medium, Tom is prevented from tampering with the data altogether, and so Bob will always obtain original data. However, the feasibility of an unlimited invulnerable medium makes this solution idealistic, rather than realistic.

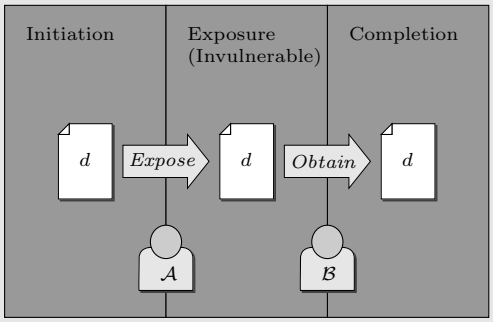

Figure 3.2: Ideal Process Model

An alternative solution is for Alice and Bob to use a signature-based integrity detection scheme to solve the problem (see Figure 3.3). This involves adding two extra stages to the process for protecting and verifying the data. Alice adds a *protection* stage before exposing the data, in which the data is *protected* using the sign function of a signature-based integrity detection scheme. Both the data and its signature are then exposed on a vulnerable medium, where they become subject to tampering by Tom. The counterfeit data and counterfeit (or

original) signature are then obtained by Bob who adds a *verification* stage before completion. In this stage, Bob uses the validate function of a signature-based integrity detection scheme to determine whether the data is original, in which case the data becomes *validated*. However, since the data is counterfeit, the validate function determines the data is *false* as the result of verification.

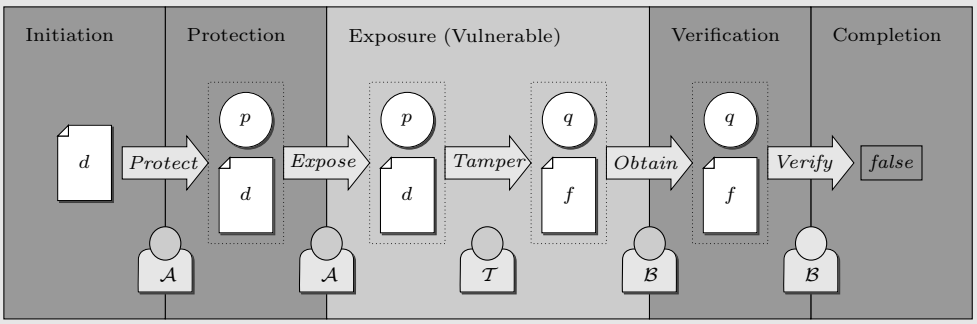

Figure 3.3: Signature-based Integrity Detection Scheme Process Model

This thesis uses a more general process model that accommodates different aspects of the previous process models (see Figure 3.4). This model is similar to that used for signature-based integrity detection schemes, but also incorporates the IP model for exposure. After initiation, the preservative is computed in the *preservation* stage — a more general stage than the protection stage of a signature-based integrity detection scheme (SIDS). In the exposure stage the data is exposed via a vulnerable medium and the preservative is exposed via a limited invulnerable medium (as in the IP model). When the data and its preservative are obtained the *resolution* stage begins — a more general stage than the verification stage of a signature-based integrity detection scheme. Within this stage Bob *resolves* any problems with the data using its preservative, thereby resulting in the original data.

Chapter Summary

The fundamentals of the data integrity problem have been discussed by defining a model for each of its main aspects. The discussion has been focussed around three areas of the problem: the participants, the data and the processes.

The participants have been discussed in terms of the attacking and defending parties and their adverse goals. The role that each participant plays in achieving their party's goal has been defined, and the idea of a trusted third-party used in the security model has been introduced. The abilities of the attacking party have been elaborated on by discussing the adversary model of this work, and a powerful adversary with the ability for unrestricted modification to data exposed via a vulnerable medium has been defined. Alongside this vulnerable medium, an invulnerable medium has been defined, where the adversary has no ability to modify the exposed data.

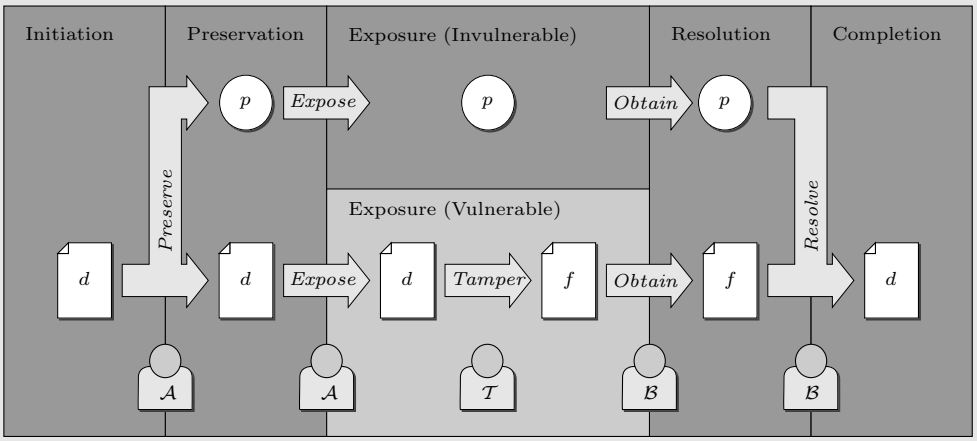

Figure 3.4: Multi-layered Document Integrity Process Model

The data model has been defined in terms of a partitionable data structure, which provides the ability to locate modifications to specific data blocks, rather than as a whole. Subsequently, the idea of a preservative has been discussed, which is determined from the data prior to exposure. The preservative can be used to resolve any data integrity problems after exposure, thereby helping to preserve the data's integrity.

The final area of the problem concerns its processes. The transaction model has further defined some aspects of the participants' abilities and has outlined the definition of a medium. The different types of media have been discussed, including the use of the term *exposure* in place of *store* and *transmit*, and the use of *vulnerable* and *invulnerable* media as respective synonyms for *tamper-vulnerable* and *tamper-invulnerable* media. Various process models have been discussed, which help to illustrate the problem of integrity preservation by classifying the various stages of the problem, and defining the participants and data involved at each stage.

Chapter 4

Taxonomy

sed quis custodiet ipsos custodes?
[but who is to guard the guards themselves?]

— Juvenal, *Satire VI* (1^{st}–2^{nd} Century)

A common issue with security schemes concerns the actual definition of *secure*. Whilst other security problems have well-defined models to define security within the problem's context, the integrity problem lacks such a model. In this chapter, a well-defined model for the *data integrity problem* is introduced. The model considers the problem as four distinct sub-problems (see Definition 4.1), allowing a formal definition to be constructed for each one.

Definition 4.1 (Data Integrity Problem)

The problem encapsulates the following sub-problems:

- Detection: Determining any modification in the exposed data;
- Location: Identifying any instances of modification in the exposed data;
- Correction: Restoring any instances of modification in the exposed data to their original state;
- Prevention: Precluding any instances of modification in the exposed data.

Two approaches are required in order to distinctly define each of the sub-problems. The first approach defines the problem in terms of its *behaviour*, and the second approach in terms of its *solvability*. Collectively, these approaches provide a formal *problem semantics* for data integrity.

The problem semantics can be used to build a new security model for determining whether or not a protocol can solve any of the sub-problems that constitute the data integrity problem. A generalized description of this security model is described in the final section of this chapter.

4.1 Behaviour Model

The behaviour model is formally defined as a set of *behaviour functions*, with one function defined for each of the sub-problems. These functions describe the sub-problems by defining the output for a given input. Informally, a behaviour function could be described as "specifying how an algorithm for solving the sub-problem should operate".

A common characteristic of the behaviour functions is that, despite the original data block (d) being used in the definition of all the functions, it is not given as an input to the functions. The original data block is only used to specify the functions' behaviour in relation to their input. In other words, it specifies how an algorithm *should* evaluate, but the definition itself cannot be used as an algorithm.

Consider the generic sub-problem Z. The behaviour function (γ) for Z might be defined $\gamma \cdot x \ ::= \ z$ (i.e. given the input x, an algorithm to solve Z should output z). Whereas an algorithm (G) for solving Z might be defined $G \cdot x \ ::= \ y$ (i.e. given input x, an algorithm G to solve Z outputs y). In either case x is the input, and therefore if $y = z$, then G meets the behaviour specification and is therefore a possible solution to Z. However, if $y \neq z$, then G is definitely not a solution to Z. For a possible solution to be an actual solution, a further type of specification must be met (see §4.2).

4.1.1 Detection Behaviour

The problem of detection is the most commonly discussed problem regarding data integrity and tampering. Both the problem and its widely-accepted solution (signature-based integrity detection schemes) have been studied in detail. However, the approach taken in this work is to construct an abstract model of the detection problem.

Essentially, detection is a process that determines modification in exposed data. That is, if the exposed data block was tampered with, the detection behaviour function should return the value *false*; otherwise the exposed data block has not been tampered with and the value *true* should be returned (see Definition 4.2). This problem requires that *only* one bit of information be determined, irrespective of the size of the original data block.

Definition 4.2 (Detection Behaviour Function)

Given an exposed data block e, the *detection behaviour function* δ evaluates to *true* if the exposed data block is original, and *false* if it has been tampered with. The function is formally defined as

$$\delta \cdot e \ ::= \ (e = d).$$

The problem with determining the integrity of exposed data from the data alone is that all of the information available is part of the data itself. In order to determine integrity, extra information regarding the original data must be included in the form of a preservative. A detection algorithm therefore operates on an exposed tuple, which is made up of both the data and its preservative.

As a result of *packaging* the data along with a preservative in a tuple, there are now four possibilities for tampering. Ideally, detection should determine whether the exposed document is the original document. However, when the exposed preservative is tampered with, there is no reliable means to determine whether the data is original. The result of this is that the only reliable means to achieve detection is when both data and preservative are original. Table 4.1 shows the different possible exposed data and preservative values, along with a statement representing them both being original. Note that when the IP model is used, the exposed preservative is original by definition (see Definition 3.8).

e	q	$e = d$	$q = p$	$(e = d) \wedge (q = p)$
d	p	*true*	*true*	*true*
d	r	*true*	*false*	*false*
f	p	*false*	*true*	*false*
f	r	*false*	*false*	*false*

Table 4.1: Ideal Versus Achievable Detection Results

To illustrate this idea, consider a simple detection algorithm D that uses cryptographic hashing. The definition for the behaviour of D is that when given an exposed data block e and its corresponding preservative q, D should return $e = d$ in accordance with Definition 4.2.

$$
\begin{aligned}
& D \cdot (e, q) \\
= \quad & \{\text{Let } D \cdot (e, q) \ ::= \ (\varpi \cdot e = p) \wedge (q = p)\} \\
& (\varpi \cdot e = p) \wedge (q = p) \\
= \quad & \{\text{Definition 3.4}\} \\
& (\varpi \cdot e = \varpi \cdot d) \wedge (q = p) \\
= \quad & \{\varpi \ ::= \ H\} \\
& (H \cdot e = H \cdot d) \wedge (q = p) \\
= \quad & \{\text{Definition 2.1, properties (iv) and (v)}\} \\
& (e = d) \wedge (q = p) \\
\Rightarrow \quad & \{\text{Weakening}\} \\
& e = d
\end{aligned}
$$

The penultimate step of this derivation will only hold under the assumption that no two different inputs can hash to the same output. In theory this does not hold for a useful cryptographic hash function, but in practice (and for the sake of this illustration) this assumption can be made. The final step is a compromise, in that a stronger statement can be achieved than is actually required to satisfy detection. The algorithm determines that both the exposed data and its preservative are original, which can be weakened to just the exposed data being original.

4.1.2 Location Behaviour

The location problem is less straightforward than that of detection. Rather than determining whether any modification has taken place in the exposed data, the problem must determine whether any modification has taken place in each of the exposed data blocks. By defining an "instance" of tampering using data blocks (see Definition 3.3), a deterministic decision can be

made as to whether each block has been modified; considering each block as an autonomous piece of data as opposed to the whole data being considered as an autonomous block.

A further complication arises because it is unclear what the output of the location process should be. This issue is resolved by considering the data model and the aim of location. Given two data blocks d and e, the aim of location is to perform a pairwise comparison on each of their data block pairs (d_i, e_i). If a data block is modified in any way, given both the original and modified data blocks, the location behaviour function should determine where modifications have occurred with respect to the original data block (see Definition 4.3). Essentially the location process is identical to the detection process, but iterated over each of the data blocks.

Definition 4.3 (Location Behaviour Function)

Given exposed data e that consists of $\#(e)$ data blocks $e_0, e_1, \ldots, e_{\#(e)-1}$, the *location behaviour function* λ indicates which blocks of the exposed data were original. The function is formally defined as

$$\lambda \cdot e \ ::= \ [\, i \in \mathbb{N} \mid 0 \leq i < \#(d) \colon \ d_i = e_i \,].$$

The location sub-problem requires that one bit of information be determined for each data block of the original data block. Therefore the information required for a solution is dependent on the size and structure of the original data block. However, *at most* n bits of information are required to determine the solution for an n-bit original data block.

4.1.3 Correction Behaviour

The correction problem is that of restoring the original data block, given an exposed data block. Therefore *exactly* n bits of information are required to determine the solution for an n-bit original data block.

Definition 4.4 (Correction Behaviour Function)

Given an exposed data block e, the *correction behaviour function* χ determines the original data block. The function is formally defined as

$$\chi \cdot e \ ::= \ d.$$

4.1.4 Prevention Behaviour

The prevention problem is fundamentally different from the other sub-problems in several ways. Essentially the adversary is inhibited from modifying the exposed data block, and so the original and exposed data blocks must be identical. In other words, if the adversary can make any modification to the data block, then prevention has not been achieved, irrespective of whether tampering can be detected, located or corrected. One might even argue that if the adversary

cannot modify the data, then they become passive (rather than active), which would contradict the adversary model given in §3.4.

Another fundamental difference of prevention is that the computation modelled by the behaviour function for the other sub-problems is done at resolution stage. However, any computation done by the defending party after exposure cannot affect the outcome of prevention.

The difference becomes more apparent when considering the prevention behaviour function (see Definition 4.5), which is defined in an identical way to correction. Essentially, both of these processes should produce the same result - the original data block.

Definition 4.5 (Prevention Behaviour Function)

Given an exposed data block e, the *prevention behaviour function* π determines the original data block. The function is formally defined as

$$\pi \cdot e \ ::= \ d.$$

This situation is one of the reasons that this model must be extended further to encompass the fundamental difference of prevention, and thereby distinguish this sub-problem.

4.2 Solvability Model

The solvability model is formally defined as a set of *solvability functions*, with one function defined for each of the sub-problems, in a similar way to the behaviour functions. These functions describe the sub-problems in terms of whether a given algorithm for solving one of the sub-problems is correct, and therefore constitutes an actual solution to the problem. Informally, the solvability functions determine whether or not the given algorithm can solve the sub-problem.

Consider the generic sub-problem Z. The solvability function (Γ) for Z might be defined $\Gamma \cdot x \ ::= \ z$, where z is a function of the equality between the result of the algorithm (G) and the result of the behaviour function (γ), except in the special case of the prevention sub-problem. In other words:

$$\Gamma \cdot x \ ::= \ G \cdot y = \gamma \cdot y \tag{4.1}$$

As a consequence of this underlying pattern the discussion given for the solvability of location (see §4.2.2) and correction (see §4.2.3) is just an iteration of that given for detection with appropriate substitutions. These sections are included for the sake of completeness and referencing.

4.2.1 Detection Solvability

Detection solvability is essentially the same as the tamper detectable property (see Definition 4.6). It verifies the solvability for results of a detection algorithm by comparing them against the results of the detection behaviour function.

Exposed data is said to be *tamper detectable* if, on input of the exposed data and its original preservative, a given detection algorithm evaluates in the same way as the detection behaviour function. This can be formally expressed as follows:

$$
\begin{aligned}
& D \cdot (e, q) = \delta \cdot e \\
= \quad & \{\text{Definition 4.2}\} \\
& D \cdot (e, q) = (d = e)
\end{aligned}
$$

Definition 4.6 (Tamper Detectable)

Given an original data element d, an exposed data block e, an exposed preservative q, and an algorithm D, the detection solvability function Δ determines whether the exposed data is *tamper detectable*. The function is formally defined as

$$\Delta \cdot (d, e, q, D) \ ::= \ D \cdot (e, q) = (d = e).$$

If Δ evaluates to *true* for an exposed data block, then the data block is tamper detectable.

4.2.2 Location Solvability

Location solvability is essentially the same as the tamper locatable property (see Definition 4.7). It verifies the solvability for results of a location algorithm by comparing them against the results of the location behaviour function.

Exposed data is said to be *tamper locatable* if, on input of the exposed data and its original preservative, a given location algorithm evaluates in the same way as the location behaviour function. This can be formally expressed as follows:

$$
\begin{aligned}
& L \cdot (e, q) = \lambda \cdot e \\
= \quad & \{\text{Definition 4.3}\} \\
& L \cdot (e, q) = [\ i \in \mathbb{N} \mid 0 \leq i < \#(d) \colon \ d_i = e_i\ ]
\end{aligned}
$$

Definition 4.7 (Tamper Locatable)

Given an original data element d, an exposed data block e, an exposed preservative q, and an algorithm L, the detection solvability function Λ determines whether the exposed data is *tamper locatable*. The function is formally defined as

$$\Lambda \cdot (d, e, q, L) \ ::= \ L \cdot (e, q) = [\ i \in \mathbb{N} \mid 0 \leq i < \#(d) \colon \ d_i = e_i\ ].$$

If Λ evaluates to *true* for an exposed data block, then the data block is tamper locatable.

4.2.3 Correction Solvability

Correction solvability is essentially the same as the tamper correctable property (see Definition 4.8). It verifies the solvability for results of a correction algorithm by comparing them against the results of the correction behaviour function.

Exposed data is said to be *tamper correctable* if, on input of the exposed data and its original preservative, a given correction algorithm evaluates in the same way as the correction behaviour function. This can be formally expressed as follows:

$$
\begin{aligned}
& C \cdot (e, q) = \chi \cdot e \\
= \quad & \{\text{Definition 4.4}\} \\
& C \cdot (e, q) = d
\end{aligned}
$$

Definition 4.8 (Tamper Correctable)

Given an original data element d, an exposed data block e, an exposed preservative q, and an algorithm C, the detection solvability function X determines whether the exposed data is *tamper correctable*. The function is formally defined as

$$X \cdot (d, e, q, C) \ ::= \ C \cdot (e, q) = d.$$

If X evaluates to *true* for an exposed data block, then the data block is tamper correctable.

4.2.4 Prevention Solvability

Prevention solvability is essentially the same as the tamper preventable property (see Definition 4.9). However, unlike the other sub-problems, tamper prevention requires that no modification occurs during exposure - it requires a preventative, rather than reactive, measure. As mentioned previously, if the adversary can make any modification to the data block, then prevention has not been achieved.

In order to solve such a problem in a preventative manner, the solution must not only adhere to the prevention behaviour function, but also ensure that the exposed data block remains unmodified (i.e. $d = e$).

The notion of *tamper preventable* might be formally expressed as follows:

$$
\begin{aligned}
& P \cdot (e, q) = \pi \cdot e \ \ \wedge \ \ d = e \\
= \quad & \{\text{Definition 4.5}\} \\
& P \cdot (e, q) = d \ \ \wedge \ \ d = e
\end{aligned}
$$

That is, there must exist an algorithm (P) that determines the same value as the prevention behaviour function (condition 1) *and* the exposed data must not be different to the original data (condition 2). However, there are two problems with this definition. The first problem is due to the assumption that the algorithm P, which is applied *after* exposure, can in some way prevent a tamperer from modifying the data *during* exposure. The second problem is that if condition 2 holds, then condition 1 becomes redundant; i.e. if the exposed data is not modified, then there is no requirement to perform any process on it. Therefore prevention solvability can be defined in terms of the second condition alone.

Definition 4.9 (Tamper Preventable)

Given an original data element d, an exposed data block e, an exposed preservative q, and an algorithm C, the detection solvability function X determines whether the exposed data is *tamper correctable*. The function is formally defined as

$$\Pi \cdot (d, e, q, P) \ ::= \ d = e.$$

If Π evaluates to *true* for an exposed data block, then the data block is tamper preventable.

4.3 Security Model

The security model simulates the tampering problem in terms of a game *played* between the attacking and defending party. Each game involves a series of challenges between the parties in order to formalize their goals and determine a winner. This process requires the help of a mutually-trusted third party ($\mathcal{J}$) to impartially judge the winner in a transparent manner, thereby convincing all parties that the game is fair.

The *general tampering game* (see Definition 4.10) is defined in terms of a *general algorithm* and a *general solvability function*. The game is used to determine the security of the algorithm (G) by using the general solvability function (Γ) to determine the winner. If Γ evaluates to *true* then the defending party wins the game and G is considered secure, but if it evaluates to *false* then the attacking party wins and G is considered to be insecure. The game consists of three *stages*, each divided into three *steps* — nine steps in total.

Definition 4.10 (General Tampering Game)

An abstract protocol played between honest and dishonest parties, with the help of a mutually-trusted third party Jude. The protocol involves the following three stages:

Stage 1		
	VP Model	IP Model
1: $\mathcal{A}$:	(d, q)	(d, q)
2: $\mathcal{A} \dashrightarrow \mathcal{J}$:	(d, q)	(d, q)
3: $\mathcal{J} \dashrightarrow \mathcal{T}$:	(d, q)	(d, q)

Stage 2		
	VP Model	IP Model
1: $\mathcal{T}$:	(f, r)	(f, r)
2: $\mathcal{T} \dashrightarrow \mathcal{J}$:	(f, r)	(f, r)
3: $\mathcal{J} \dashrightarrow \mathcal{B}$:	(f, r)	(f, q)

Stage 3		
	VP Model	IP Model
1: $\mathcal{B}$:	G	G
2: $\mathcal{B} \dashrightarrow \mathcal{J}$:	G	G
3: $\mathcal{J} \dashrightarrow *$:	$\Gamma{\cdot}(d, f, r, G)$	$\Gamma{\cdot}(d, f, q, G)$

If Jude announces *true* then Alice and Bob win the game, but if Jude announces *false*, then Tom wins.

Stage 1 allows Alice to challenge Tom with a data block and corresponding preservative. The first stage is identical in both preservative models. Similarly, stage 2 allows Tom to

respond with a modified data block and preservative challenge for Bob. However, in the IP model, Bob obtains the original preservative rather than the modified preservative from Tom's challenge. Stage 3 determines the winner by checking Bob's response to the challenge, which is an algorithm for responding to Tom's challenge. Alternatively, in the IP model, Bob is able to use the original preservative from Alice's challenge rather than Tom's preservative.

There are three steps involved in each stage. In the first step the active participant determines some value, which is then sent to Jude in the second step. The final step involves Jude sending some value to the next active participant, except in the last stage, where the value is announced (sent to all active participants). This prevents any active participant from repudiating any of their earlier actions, thus ensuring the game is fair.

The general tampering game is general in that it is not defined for a specific sub-problem. However, by specifying a corresponding algorithm and solvability function, it can be instantiated for each of the tampering sub-problems. The functions are chosen to reflect the particular game being played. That is, a detection algorithm and solvability function are used in the detection game, a location algorithm and solvability function are used for the location game, and similarly for the other games. These specifics are only required for the third stage of the game, as stages 1 and 2 are identical to the general stages given in Definition 4.10.

In summary, this approach is adopted to be able to reason about a sub-problem, which requires an instance of the general tampering game for that sub-problem. To avoid repetition, and since each sub-problem's game is almost identical to the general tampering game, only the general game is given. Constructing a sub-problem's tampering game from the general game is simply a matter of replacing G with the algorithm that potentially solves that sub-problem, and Γ with its corresponding solvability function.

Chapter Summary

A taxonomy for the data integrity problem has been presented by dividing it into four distinct sub-problems. A formal problem semantics has been defined for each sub-problem in terms of its behaviour and solvability functions. The behaviour function specifies how an algorithm for solving the sub-problem should behave in respect to its input and output. The solvability function specifies whether the output given by such an algorithm is correct. These two specification methods allow each sub-problem to be well-defined and differentiated.

A security model for the data integrity problem has been given in the form of the general tampering game. This general model can be used to construct the security model for each of the sub-problems. Each sub-problem's security model is based on the solvability function for that sub-problem, and can be used to determine the security of a given algorithm within that context.

Chapter 5

Results

After great pain, a formal feeling comes—

— Emily Dickinson, *After great pain, a formal feeling comes* (1862)

Having outlined and developed a model for the data integrity problem in the previous chapters, this chapter presents the main results surrounding the problem. The relationship between these sub-problems is determined, and this result is developed further to determine the security of existing protocols.

5.1 Taxonomy

Having formally defined the sub-problems of the data integrity problem, each sub-problem is now differentiated by its solvability function. These formal definitions have several uses, the first of which is to determine the relationship between the sub-problems.

Assuming that only the modified data blocks of the document are corrected, then the four sub-problems form a hierarchy: Locating where tampering has taken place implies that tampering has been detected; and correction of that tampering implies that the tampering has been located. Moreover, the following results hold:

Lemma 5.1 (Preventable Implies Correctable)

If a data block is tamper preventable, then it is also tamper correctable.

Proof

Assumption: $\Pi{\cdot}(d, e, q, P)$

Conclusion: $X{\cdot}(d, e, q, C)$

$$
\begin{array}{ll}
 & \Pi \cdot (d, e, q, P) \\
= & \quad \{\text{Definition of } \Pi\} \\
 & P \cdot (e, q) = d \ \wedge \ d = e \\
\Rightarrow & \quad \{\text{Weakening}\} \\
 & P \cdot (e, q) = d \\
\Rightarrow & \quad \{C \cdot (e, q) \ ::= \ P \cdot (e, q)\} \\
 & C \cdot (e, q) = d \\
\Rightarrow & \quad \{\text{Definition of } X\} \\
 & X \cdot (d, e, q, C)
\end{array}
$$

■

Lemma 5.2 (Correctable Implies Locatable)

If a data block is tamper correctable, then it is also tamper locatable.

Proof

Assumption: $X{\cdot}(d, e, q, C)$

Conclusion: $\Lambda{\cdot}(d, e, q, L)$

$$
\begin{array}{ll}
 & X \cdot (d, e, q, C) \\
= & \quad \{\text{Definition of } X\} \\
 & C \cdot (e, q) = d \\
\Rightarrow & \quad \{\text{Function application}\} \\
 & F \cdot (C \cdot (e, q), e) = F \cdot (d, e) \\
\Rightarrow & \quad \{L \cdot (e, q) \ ::= \ F \cdot (C \cdot (e, q), e)\} \\
 & L \cdot (e, q) = F \cdot (d, e) \\
\Rightarrow & \quad \{L \cdot (e, q) \ ::= \ [\ i \in \mathbb{N} \mid 0 \leq i < \#(d) \colon \ d_i = e_i \ ]\} \\
 & L \cdot (e, q) = [\ i \in \mathbb{N} \mid 0 \leq i < \#(d) \colon \ d_i = e_i \ ] \\
\Rightarrow & \quad \{\text{Definition of } \Lambda\} \\
 & \Lambda \cdot (d, e, q, L)
\end{array}
$$

■

Lemma 5.3 (Locatable Implies Detectable)

If a data block is tamper locatable, then it is also tamper detectable.

Proof
Assumption: $\Lambda{\cdot}(d, e, q, L)$

Conclusion: $\Delta{\cdot}(d, e, q, D)$

$$
\begin{array}{ll}
 & \Lambda \cdot (d, e, q, L) \\
= & \quad \{\text{Definition of } \Lambda\} \\
 & L \cdot (e, q) = [\ i \in \mathbb{N} \mid 0 \leq i < \#(d) \colon\ d_i = e_i\ ] \\
\Rightarrow & \quad \{\text{Function application}\} \\
 & F \cdot L \cdot (e, q) = F \cdot [\ i \in \mathbb{N} \mid 0 \leq i < \#(d) \colon\ d_i = e_i\ ] \\
\Rightarrow & \quad \{F \cdot x\ \ ::=\ \ \langle\ \forall x_i \in x \mid 0 \leq i < \#(x) \colon\ x_i\ \rangle\} \\
 & F \cdot L \cdot (e, q) = \langle\ \forall (d_i, e_i) \in d \mid 0 \leq i < \#(d) \colon\ d_i = e_i\ \rangle \\
\Rightarrow & \quad \{\text{Conjunction of all pairwise comparisons}\} \\
 & F \cdot L \cdot (e, q) = (d = e) \\
\Rightarrow & \quad \{D \cdot (e, q)\ \ ::=\ \ F \cdot L \cdot (e, q)\} \\
 & D \cdot (e, q) = (d = e) \\
\Rightarrow & \quad \{\text{Definition of } \Delta\} \\
 & \Delta \cdot (d, e, q, D)
\end{array}
$$

■

The apparent hierarchical relationship between these sub-problems can be shown formally by "chaining" these results together with the transitivity of implication (see Corollorary 5.1).

Corollary 5.1 (Data Integrity Problem is Hierarchical)

The data integrity problem is formed from a hierarchy of the four data integrity sub-problems in the following way:

- Solving prevention allows correction to be solved trivially;
- Solving correction allows location to be solved trivially;
- Solving location allows detection to be solved trivially.

Proof
Follows trivially from Lemma 5.1, Lemma 5.2 and Lemma 5.3, using the transitivity of implication.

■

The significance of Corollorary 5.1 is that the existence of an algorithm for solving one of the sub-problems implies the existence of an algorithm for solving all the other sub-problems that are lower in the hierarchy. For example, if one has an algorithm L that performs location according to the behaviour of Definition 4.3 and solvability of Definition 4.7, then one can

trivially construct an algorithm that performs detection according to the behaviour of Definition 4.2 and solvability of Definition 4.6 by composition of $\langle\ \forall x_i \in x \mid 0 \leq i < \#(x) \colon\ \ x_i\ \rangle$ and L.

5.2 Solvability

The next aim is to determine which of the sub-problems are solvable and under what restrictions. The most desirable solutions will exist within the VP model, where no special exposure method is required for the preservative. An algorithm that solves a sub-problem in the VP model would be considered the most ideal solution. However, if the model shows that a sub-problem is unsolvable in the VP model, the IP model will be considered as a compromise.

5.2.1 Vulnerable Preservative Model

The VP model is effectively a model of the typical real-life situation, whereby storage devices and transmission channels are vulnerable to tampering. Current tamper protection relies on signature-based integrity detection schemes, which solve the detection problem. The security model can be used to show that SIDSs are secure in the VP model by analysis of the announcement made in stage 3 of the detection game. Claim 5.1 shows how a signature-based integrity detection scheme is secure with respect to tamper detection in the VP model.

Claim 5.1 (Detection is Solvable in the VP Model)

Signature-based integrity detection schemes provide a secure solution to the detection problem in the VP model.

Proof

Assumption: $D\ ::=\ V_{\mathcal{A}}$

Conclusion: $\Delta{\cdot}(d, f, r, D)$

$$
\begin{array}{ll}
 & \Delta \cdot (d, f, r, D) \\
= & \quad \{\text{Definition of } \Delta\} \\
 & D \cdot (f, r) = (d = f) \\
= & \quad \{\text{Equation 3.1}\} \\
 & D \cdot (f, r) = \mathit{false} \\
= & \quad \{D\ ::=\ V_{\mathcal{A}}\} \\
 & V_{\mathcal{A}} \cdot (f, r) = \mathit{false} \\
= & \quad \{\text{Forgery resistance of } V_{\mathcal{A}}\} \\
 & \mathit{false} = \mathit{false} \\
= & \quad \{\text{Reflexivity of } =\} \\
 & \mathit{true}
\end{array}
$$

■

A signature-based integrity detection scheme can be used to solve detection, but if the tamperer is persistent, then the defending party will never learn anything about the original data. Therefore tamper location becomes the next goal of the defending party. If tamper

location can be solved in the VP model, then the defending party should be able to determine the unmodified data blocks, thereby learning something about the original data.

SIDSs are secure in the VP model because they are based on a publicly-verifiable relationship being established between the data and its source. This relationship is packaged together with the data in the form of a preservative. This so-called package can then be publicly verified to determine whether the relationship still holds: if so, then it can be assumed that nothing in the package has been modified; if not, then all that can be determined is that something in the package has been modified. The latter situation could have resulted from three possible causes: Either the data was modified, the preservative was modified, or both (see Table 4.1). The information is enough to determine that the *package* has been tampered with, but not enough to determine that the *data* has been tampered with.

Definition 5.1 (Data-only IDS)

A *data-only integrity detection scheme* is a scheme for integrity detection such that, given an exposed data block and its preservative, the scheme can determine whether the exposed data block is original. Formally, a data-only IDS evaluates as

$$d = e.$$

If one assumes the existence of a data-only integrity detection scheme capable of determining the presence of tampering in a single bit of data, then one can construct a scheme for tamper location and correction in arbitrary-length data. First, the data d is divided into single bits $d_0, d_1, \ldots, d_{n-1}$. Then multiple instances of the scheme are applied, such that each bit is considered individually with its own instance of the scheme. To locate the tampering, detection is performed for each of the schemes in a bitwise manner. Moreover, correction can be achieved simply by inverting the tampered bits given that location has already determined the exact bits that have been modified and the only possible modification of a bit is inversion.

This hypothetical situation highlights some important observations regarding the information that can be determined from schemes in the VP model. In particular, location can be achieved through multiple instances of detection simply by applying a detection scheme to the sub-elements of the data. Also, a data-only integrity detection scheme for determining modification in n-bit data allows the elimination of 1 of the 2^n possible data blocks. However, this can only be achieved if just the the data is vulnerable to tampering, in other words, in the IP model. Theorem 5.1 shows that no scheme for data-only integrity detection exists.

Theorem 5.1 (Non-existence of Data-only IDS)

No algorithm for data-only integrity detection exists.

Proof

Assume an IDS algorithm D exists. Being a data-only IDS is conditional on satisfying the equality in the existential quantification (i.e. implication is not sufficient), and therefore the condition is that $D \cdot (a, b) = (a = b)$ and $q = d$.

$$
\begin{aligned}
& \langle\, \exists D \mid :\ D \cdot (e, q) = (d = e) \,\rangle \Rightarrow ((D \cdot (a, b) = (a = b)) \wedge (q = d)) \\
= \quad & \{D \cdot (a, b) \ ::= \ (a = b) \text{ and } q \ ::= \ \tau \cdot d \text{ (due to VP model)}\} \\
& \langle\, \exists D \mid :\ D \cdot (e, q) = (d = e) \,\rangle \Rightarrow (((a = b) = (a = b)) \wedge (\tau \cdot d = d)) \\
= \quad & \{\text{Reflexivity of } = \text{ and Definition 3.3}\} \\
& \langle\, \exists D \mid :\ D \cdot (e, q) = (d = e) \,\rangle \Rightarrow (\mathit{true} \wedge (f = d)) \\
= \quad & \{\text{Identity of } \wedge \text{ and contradiction with Equation 3.1}\} \\
& \langle\, \exists D \mid :\ D \cdot (e, q) = (d = e) \,\rangle \Rightarrow \mathit{false} \\
= \quad & \{\text{Contrapositive of } \Rightarrow\} \\
& \neg \mathit{false} \Rightarrow \neg \langle\, \exists D \mid :\ D \cdot (e, q) = (d = e) \,\rangle \\
= \quad & \{\text{Definition of } \neg \text{ and reflexivity of } \equiv\} \\
& \mathit{true} \Rightarrow \neg \langle\, \exists D \mid :\ D \cdot (e, q) = (d = e) \,\rangle \\
= \quad & \{\text{Left identity of } \Rightarrow\} \\
& \neg \langle\, \exists D \mid :\ D \cdot (e, q) = (d = e) \,\rangle
\end{aligned}
$$

■

Having confirmed that in the VP model, *package* detection is solvable, and having determined that data-only detection is unsolvable, the next sub-problem to consider is location. The most obvious method to achieve location is to split the data into elements and use a signature-based integrity detection scheme on each element. This approach is similar to Ashman's MLDI idea, which uses hash functions rather than signature-based integrity detection schemes (although PGP [47] is suggested as a method of encrypting the hashes to ensure integrity).

This scheme allows the defending party to locate the position of tampering to the exact data blocks where the modifications occurred. The defending party then knows that the remaining data blocks are unmodified and therefore that their content can be safely utilized. However, since the adversary is aware of this scheme, they can easily prevent location by modifying every data block, reordering the data blocks, or even by modifying all the corresponding preservative elements.

Within the proposed adversary model, the problem of tamper location is unsolvable in the VP model because the adversary can modify an arbitrary number of the data blocks and hide this fact (in the sense of location) by modification of every preservative — detection is still possible, but location is not.

Claim 5.2 (Location is Unsolvable in the VP Model)

Location cannot be solved in the VP model.

Proof

Assumption: $(e = d) \wedge (q = r)$

Conclusion: $\neg\Lambda \cdot (d, e, q, L)$

$$
\begin{array}{ll}
 & \Lambda \cdot (d, e, q, L) \\
= & \quad \{\text{Definition of } \Lambda\} \\
 & L \cdot (e, q) = [\ i \in \mathbb{N} \mid 0 \leq i < \#(d) \colon\ d_i = e_i\ ] \\
= & \quad \{\text{Assumption}\} \\
 & L \cdot (d, r) = [\ i \in \mathbb{N} \mid 0 \leq i < \#(d) \colon\ d_i = d_i\ ] \\
= & \quad \{\text{Reflexivity of } =\} \\
 & L \cdot (d, r) = [\ i \in \mathbb{N} \mid 0 \leq i < \#(d) \colon\ \mathit{true}\ ] \\
= & \quad \{\text{Simplification}\} \\
 & L \cdot (d, r) = \underbrace{[\mathit{true}, \ldots, \mathit{true}]}_{\#(d)} \\
= & \quad \{\text{Theorem 5.1}\} \\
 & \underbrace{[\mathit{false}, \ldots, \mathit{false}]}_{\#(d)} = \underbrace{[\mathit{true}, \ldots, \mathit{true}]}_{\#(d)} \\
= & \quad \{\text{Simplification}\} \\
 & \mathit{false}
\end{array}
$$

■

This result can be used with the problem hierarchy to show that correction and prevention are also unsolvable in the VP model. The sketch of the proof is to assume that correction is solvable, then show (by the relevant problem hierarchy lemma) that a location solution can be constructed, which contradicts Claim 5.2 and so correction is not solvable.

Claim 5.3 (Correction is Unsolvable in the VP Model)

Correction cannot be solved in the VP model.

Proof

$$
\begin{array}{ll}
 & \neg\ X \cdot (d, e, q, C) \\
\Rightarrow & \quad \{\text{Lemma 5.2}\} \\
 & \neg\ \Lambda \cdot (d, e, q, L) \\
\Rightarrow & \quad \{\text{Claim 5.2}\} \\
 & \neg\ \mathit{false} \\
\Rightarrow & \quad \{\text{Simplification}\} \\
 & \mathit{true}
\end{array}
$$

■

The proof for prevention follows a similar line of reasoning, despite the problem's natural differences.

Claim 5.4 (Prevention is Unsolvable in the VP Model)

Correction cannot be solved in the VP model.

Proof

$\neg\ \Pi \cdot (d, e, q, P)$

$\Rightarrow$ {Lemma 5.1}

$\neg\ X \cdot (d, e, q, C)$

$\Rightarrow$ {Claim 5.3}

$\neg\ \mathit{false}$

$\Rightarrow$ {Simplification}

true ■

The general conclusion that can be made from these four results (Claims 5.1–5.4) is that tamper detection is the only solvable sub-problem in the VP model.

5.2.2 Invulnerable Preservative Model

Any solution that is secure in the VP model must also be secure in the IP model, and therefore VP is a sub-model of IP. The IP model is effectively a weaker security model than VP, since solvability in the VP model implies solvability in IP model. This can be illustrated by comparison of their detection games for a *hash-only* integrity detection protocol, which use a cryptographic hash of the data as their preservative.

It is well-known that protocols using hash-only IDSs are easily broken when considering Kerckhoffs' principle (or Shannon's maxim) — that the adversary is able to replicate the security system being used. Claim 5.5 states that this is true in the VP model, whilst Claim 5.6 shows that they are secure in the IP model.

Claim 5.5 (Hash-only IDSs are Unsecure in the VP model)

Hash-only signature-based integrity detection schemes are not secure in the VP model.

Proof

By examination of the detection solvability function from the detection game.

$$
\begin{aligned}
& \Delta \cdot (d, f, r, D) \\
= \quad & \{\text{Definition of } \Delta\} \\
& D \cdot (f, r) = (d = f) \\
= \quad & \{\text{Equation 3.1}\} \\
& D \cdot (f, r) = \mathit{false} \\
= \quad & \{D \cdot (d, q) \ ::= \ H \cdot d = q \text{ with } d := f,\ q := r\} \\
& (H \cdot f = r) = \mathit{false} \\
= \quad & \{\text{Definition of } r\} \\
& (H \cdot f = H \cdot f) = \mathit{false} \\
= \quad & \{\text{Definition of } =\} \\
& \mathit{true} = \mathit{false} \\
= \quad & \{\text{Simplification}\} \\
& \mathit{false}
\end{aligned}
$$

■

Claim 5.6 (Hash-only IDSs are Secure in the IP Model)

Hash-only signature-based integrity detection schemes are secure in the IP model.

Proof

By examination of the detection solvability function from the detection game.

$$
\begin{aligned}
& \Delta \cdot (d, f, q, D) \\
= \quad & \{\text{Definition of } \Delta\} \\
& D \cdot (f, q) = (d = f) \\
= \quad & \{\text{Equation 3.1}\} \\
& D \cdot (f, q) = \mathit{false} \\
= \quad & \{D \cdot (d, q) \ ::= \ H \cdot d = q \text{ with } d := f,\ q := q\} \\
& (H \cdot f = q) = \mathit{false} \\
= \quad & \{\text{Definition of } q\} \\
& (H \cdot f = H \cdot d) = \mathit{false} \\
= \quad & \{\text{Second preimage resistance of } H\} \\
& \mathit{false} = \mathit{false} \\
= \quad & \{\text{Reflexivity of } =\} \\
& \mathit{true}
\end{aligned}
$$

■

Since hash-only signature-based integrity detection schemes are secure in the IP model, but not secure in the standard VP model, then solutions for the other sub-problems may also exist in the IP model. Corollorary 5.1 showed that by solving a sub-problem in the hierarchy,

it then becomes trivial to construct solutions to all problems lower down in the hierarchy. Therefore a sensible approach is to determine whether the prevention sub-problem is solvable, and thus be able to solve all of the other sub-problems trivially from that solution. However, as previously discussed, there is a fundamental difference between prevention and the other sub-problems, whereby prevention is only solvable when the data remains unmodified. This contradicts the adversary model, which states that the adversary *can* modify the data: An adversary with unrestricted write access to the medium can always win the prevention game. Therefore prevention cannot be achieved within this model.

In practice, prevention requires some form of physical protection of the medium, such as a dedicated communication channel or safe storage for a device. Effectively the *data* is being exposed via a tamper-invulnerable medium, which would contradict the preservative model definition (see Definition 3.8). This would appear to be an expensive solution to the problem that is not always available (i.e. on the Internet). Moreover, one might ponder the philosophical question, "Will it *ever* be possible to store or transmit data in a manner such that it cannot be destroyed?" and consider whether absolute prevention is achievable in the real world. However, this debate is beyond the scope of this work; being more relevant to censorship resistance (see §2.8).

The best alternative to solving the prevention sub-problem is to solve that of correction, since either solution should yield tamper-free data. There appears to be two general methods to solve the correction sub-problem. The first method involves distributing multiple copies of the data and exposing them via multiple media. The second method is to compute a preservative from the data before exposing a single copy of it, and then use the preservative to resolve any problems.

The distributive method assumes that at least one distributed copy remains unmodified, and is therefore similar to censorship resistance. Maintaining privacy alongside integrity seems more difficult using a distributed scheme, as the privacy of every distributed copy must be maintained.

The preservative method assumes that a predetermined relationship between the data and its preservative can be used to determine information for solving the integrity problem. A signature-based integrity detection scheme, for example, allows the detection sub-problem to be solved in this way, but the idea might be used to solve the location and correction sub-problems too. However, §5.2.1 showed that only the detection sub-problem is solvable in the VP model. Therefore the IP model must be adopted and the preservative exposure problem (see Definition 3.6) must be considered.

As a result of the preservative exposure problem, the IP model is defined such that the information contained in the preservative must be less than that contained in the data (see Definition 3.8). Theorem 5.2 is a formal statement of this requirement.

Theorem 5.2 (Preservative Upper Bound)

If the preservative contains at least as much information as the data, then it becomes redundant to the scheme.

Proof

Let $\epsilon\cdot(x,t)$ denote that data x is exposable on the medium t and $\varepsilon\cdot(|x|,t)$ denote that $|x|$ bits of data are exposable on the medium t. Both functions return *true* if exposure on the given medium is possible, and *false* if it is not possible.

If the preservative p can be exposed on an invulnerable medium t and $K(p)$ is at least $K(d)$, then d can be exposed on the medium t, and therefore p is redundant the scheme.

$$
\begin{array}{ll}
 & \epsilon\cdot(p,t) \ \wedge \ (\ K(p) \geq K(d)\ ) \\
= & \quad \{\text{Definition of } \epsilon \text{ and Equation 3.3}\} \\
 & \varepsilon\cdot(|I(p)|,t) \ \wedge \ (\ |I(p)| \geq |I(d)|\ ) \\
= & \quad \{\text{Let } z \ ::= \ I(p) \text{ and } x \ ::= \ I(d)\} \\
 & \varepsilon\cdot(|z|,t) \ \wedge \ (\ |z| \geq |x|\ ) \\
= & \quad \{\text{Let } z \ ::= \ x+y, \text{ where } y \in \mathbb{N}\} \\
 & \varepsilon\cdot(|x+y|,t) \ \wedge \ (\ |x+y| \geq |x|\ ) \\
= & \quad \{y \in \mathbb{N}\} \\
 & \varepsilon\cdot(|x+y|,t) \ \wedge \ \mathit{true} \\
= & \quad \{\text{Identity of } \wedge\} \\
 & \varepsilon\cdot(|x+y|,t) \\
= & \quad \{\langle\ \forall i,j \mid i,j \in \mathbb{N}\colon \ \varepsilon\cdot(|i+j|,t) \ \equiv \ \varepsilon\cdot(|i|,t) \ \wedge\ \varepsilon\cdot(|j|,t)\ \rangle\} \\
 & \varepsilon\cdot(|x|,t) \ \wedge \ \varepsilon\cdot(|y|,t) \\
\Rightarrow & \quad \{\text{Weakening}\} \\
 & \varepsilon\cdot(|x|,t) \\
\Rightarrow & \quad \{\text{Definition of } x\} \\
 & \varepsilon\cdot(|I(d)|,t) \\
\Rightarrow & \quad \{\text{Definition of } \varepsilon\} \\
 & \epsilon\cdot(d,t) \qquad \blacksquare
\end{array}
$$

Correction in the IP model involves recovery of all the information contained within the original data from the exposed data and the original preservative. However, the adversary can remove all the information in the exposed data thus information recovery must be done from the original preservative alone. Theorem 5.2 and the Kolmogorov-Kerckhoffs Claim (derived from the Kolmogorov complexities surrounding Kerckhoffs' principle (see Claim 5.7)), can be used to show that no deterministic correction algorithm exists.

Claim 5.7 (Kolmogorov-Kerckhoffs Claim)

For any function F with arguments $x_0, x_1, \ldots, x_n$

$$K(F \cdot (x_0, x_1, \ldots, x_n)) \quad \leq \quad K(x_0, x_1, \ldots, x_n).$$

Proof

Informally, the Kolmogorov complexity is a measure of information, and Kerckhoffs' principle effectively states that all information should exist within a function's arguments, rather than its definition. Therefore a function should (at most) retain the information of its arguments. ■

Theorem 5.3 (Non-existence of Deterministic Correction)

No deterministic correction algorithm exists in the IP model.

Proof

$\neg\langle\, \exists C \mid :\ X \cdot (d, e, q, C)\, \rangle$
$=$ {Definition of X}
$\neg\langle\, \exists C \mid :\ d = C \cdot (e, q)\, \rangle$
$=$ {IP Model}
$\neg\langle\, \exists C \mid :\ d = C \cdot (e, p)\, \rangle$
$=$ {Let $e := p$}
$\neg\langle\, \exists C \mid :\ d = C \cdot (p, p)\, \rangle$
$\Rightarrow$ {Kolmogorov complexity}
$\neg\langle\, \exists C \mid :\ K(d) = K(C \cdot (p, p))\, \rangle$
$\Rightarrow$ {Claim 5.7}
$\langle\, \exists C \mid :\ K(d) \leq K(p, p)\, \rangle$
$\Rightarrow$ {Kolmogorov complexity}
$\langle\, \exists C \mid :\ K(d) \leq K(p)\, \rangle$
$\Rightarrow$ {Contradiction of Equation 3.3}
$\neg\langle\, \exists C \mid :\ \mathit{false}\, \rangle$
$\Rightarrow$ {One point rule, reflexivity of $\equiv$}
true ■

Theorem 5.3 states that a correction algorithm cannot determine the original data with absolute certainty, as the preservation process must be many-to-one to satisfy Equation 3.4. As well as solving the problem of determining the correct data, such an algorithm must also deal with the problem of attacks constructed to exploit this feature. For example, suppose that both d and f are possibilities, then there should be a small probability of an attacker being able to determine f. Perhaps the most obvious solution is to develop a probabilistic algorithm that would suggest one possible data candidate to be far more feasible than the others, whilst ensuring that a polynomially-bounded adversary cannot determine a false candidate.

Chapter Summary

Having defined each distinct sub-problem, the hierarchical relationship between these problems has been determined. This showed that solving any sub-problem in the hierarchy is essentially equivalent to solving all sub-problems lower in the hierarchy. Therefore solving the highest possible sub-problem, should be the goal of the defending party.

Using the previously defined model, the solvability of each sub-problem has been determined for both the VP and IP models. It has been shown that detection is the only solvable sub-problem in the VP model, which is essentially a result of the information required to solve each problem versus the abilities of the adversary in manipulating information. The difference between package and data-only integrity detection has been highlighted, with non-existence of the latter being proven. In the VP model it has been shown that prevention cannot be achieved under the defined adversary model: Instead, the possibility of correction has been discussed, which led to a further discussion on restricting the size of the preservative, and that correction cannot be deterministic.

The solvability results show that integrity correction in the IP model is the best result possible, and that a scheme for solving this problem will be probabilistic at best. This probabilistic method is the idea behind Ashman's MLDI and becomes the focus of Part III.

Part III

Multi-layered Document Integrity

. . . there is always a well-known solution to every human problem - neat, plausible, and wrong.

— Henry Louis Mencken, *Prejudices: Second Series* (1920)

Chapter 6

Fundamentals

and the leaves of the tree were for the healing of the nations.

— John of Patmos, *Book of Revelation* (1^{st} Century)

Multi-layered document integrity schemes consist of two stages: An initial *preservation* stage that involves deriving the preservative from the original data before exposure; and a terminal *resolution* stage that involves resolving any integrity problems in the exposed data. It is assumed that tampering of the data can only occur between these two stages, and that the preservative remains free from tampering.

This chapter begins to develop Ashman's MLDI idea into a rudimentary multi-layered document integrity scheme. The discussion focusses around introducing preliminary algorithms for preservation and resolution, justifying their design against the fundamental principles underlying the idea.

6.1 Deterministic Model

The *deterministic model* provides a simplified model for introducing some of the principles involved in MLDI. In this model hash functions are deterministic, and therefore invertible. For a hash function to be invertible, it must be free of collisions. Such collision-free hash functions are known as *perfect hash functions* [13] (see Definition 6.1).

Definition 6.1 (Perfect Hash Function)

A perfect hash function (PHF), denoted $\overleftrightarrow{H}$, is a hash function that is non-compressive and collision free.

A *perfect one-way hash function* (not to be confused with a perfect*ly* one-way hash function [8]) combines the cryptographic properties of a cryptographic hash function (CHF) with the collision free property of a PHF. The resulting hash function exhibits the non-compressive property of a PHF, rather than the compressive property of a CHF (see Definition 6.2).

Definition 6.2 (Perfect One-way Hash Function)

A perfect one-way hash function (POHF), denoted $\overrightarrow{H}$, is a perfect hash function that is preimage resistant (i.e. computationally one-way).

Using POHFs in the deterministic model means that if data is found to have a matching hash to the original data's hash, then that data must be original. This property is derived from a POHF being collision free (see Definition A.8):

$$
\begin{aligned}
& \langle\, \forall d, d_i \mid d \neq d_i :\ \vec{H}\cdot d \neq \vec{H}\cdot d_i \,\rangle \\
= \quad & \{\text{Trading}\} \\
& \langle\, \forall d, d_i \mid :\ d \neq d_i \Rightarrow \vec{H}\cdot d \neq \vec{H}\cdot d_i \,\rangle \\
= \quad & \{\text{Definition of } \neq\} \\
& \langle\, \forall d, d_i \mid :\ \neg(d = d_i) \Rightarrow \neg(\vec{H}\cdot d = \vec{H}\cdot d_i) \,\rangle \\
= \quad & \{\text{Contrapositive of } \Rightarrow\} \\
& \langle\, \forall d, d_i \mid :\ \vec{H}\cdot d = \vec{H}\cdot d_i \Rightarrow d = d_i \,\rangle
\end{aligned}
$$

6.1.1 Preservation

The preservation process in MLDI uses an *Ashman hash tree* as the preservative. Constructing the preservative involves the *decomposition* and *hashing* of the document to derive the hash tree. The document can be considered a data block (see Definition 3.2), and hence decomposition is done in terms of sub-elements.

The fundamental idea behind data correction in MLDI is that given the hash of an unknown data block, one can determine the unknown data block in reasonable time. However, if the perfect hash function is preimage resistant, then the best algorithm to determine an original data block from its hash is an exhaustive search through all 2^n possible data blocks (i.e. $d_0, d_1, \ldots, d_{2^n-1}$), hashing each one and comparing the result to the original hash. For cryptographic hash functions this is computationally infeasible, but the use of small preimage hashing is suggested for MLDI, and using a small value of n makes this practically feasible.

By applying a publicly known constraint on the possible values of d to limit the preimage size, it becomes possible to conduct an exhaustive search for h to determine d. For example, if one knows that d is a single byte, then there are only 2^8 different possible values for d, and 256 hash computations are required (at most) to reproduce a matching hatch and determine d. When known constraints have been applied on the set of data blocks from which the hash preimage comes, the data block is said to be *bounded* (see Definition 6.3).

Definition 6.3 (Bounded)

A leaf data block is bounded if it is an element of a well-defined finite set of values $\mathbb{B}^b$. A non-leaf data block is bounded if all of its children are bounded.

The term *unbounded* is used to describe data blocks with unknown bounds, for example, all data blocks from infinite sets are unbounded (see Example 6.1).

Example 6.1 (Bounded and Unbounded Data Blocks)

A byte is bounded, having 256 possible values; whereas a sentence is unbounded, since arbitrary-length sentences are feasible and therefore the data block's set size is effectively infinite.

An exhaustive search through large search spaces is infeasible in practice, and so a stronger notion of bounding is required for an original data block to be determined. The notion of *practically bounded* enforces a practical limit on the size of the search space (see Definition 6.4): The size being defined at the application level due to its dependence on pragmatic factors (i.e. current hardware benchmarks).

Definition 6.4 (Practically Bounded)

A leaf data block is practically bounded if it is bounded and if an exhaustive search through all 2^b elements in the set $\mathbb{B}^b$ is practically feasible. A non-leaf data block is practically bounded if all of its children are practically bounded.

If a practically-bounded data block's hash is known, then it is feasible to determine the original data block by computing the hash for all possible data blocks within the same bounds, and then comparing them to the original hash. Therefore data blocks must be decomposed into practically-bounded data blocks to be tamper correctable. This is achieved through a process known as *auto-bounding*.

Auto-bounding can be done in a number of ways, according to varying constraints. The two most obvious approaches to auto-bounding are linear and recursive. *Linear data division* will produce a list of data blocks. It could be done by specifying a maximum block size and dividing the data into a sequence of blocks with this size. For example, by using 4 bytes for the block size, auto-bounding would decompose the data into a number of 4-byte data blocks. Alternatively, *recursive data division* will produce a tree structure of data blocks, where blocks contain sub-blocks, which may contain further sub-blocks, and so on. This might be done by specifying a fixed or maximum number of sub-blocks that each block can be divided into and specifying a maximum block size for leaf node blocks, to determine when to stop dividing. For example, dividing raw data into two equally-sized sub-blocks, and repeating this process on the newly formed sub-blocks until they are (at most) 4 bytes in size.

The time complexity of auto-bounding is dependent upon the number of modified parts and their distribution (see Remark 6.1). Table 6.1 shows that, overall, the recursive approach to data division has more favourable time complexities.

Number of Modified Parts (m)	**List Search**	**k-ary Tree Search**
0	$O(n)$	$O(1)$
1	$O(n)$	$O(\log_k n)$
$1 < m < n$ **(best)**	$O(n)$	$O(\log_k n)$
$1 < m < n$ **(worst)**	$O(n)$	$O(n)$
n	$O(n)$	$O(n)$

Table 6.1: Time Complexities of Searching for m Modified Blocks

Remark 6.1 (Search Time Complexities: List Versus Tree)

Assuming that any number of blocks could have been modified but only one block has been modified, then a linear list search for a single modified block in a list of length n, takes $O(n)$ (average and worst case). Alternatively, a recursive k-ary tree search takes $O(\log_k n)$ (average case and worst case).

Assuming that any number of blocks could have been modified and that all the blocks have been modified, then a linear list search for all modified blocks in a list of length n, takes $O(n)$ (average and worst case). Alternatively, a recursive k-ary tree search takes $O(n)$ (average and worst case).

Irrespective of the distribution, assuming that the number of modified parts is between one and all, then the complexity for a linear list search is $O(n)$ (average and worst case). However, for a recursive k-ary tree search, the time complexity is bounded between $O(\log_k n)$ and $O(n)$ (average and worst case).

A data block is modelled as a tree, whereby the only vertex of in-degree 0 (i.e. no parent) is the *root* node (all other vertices have in-degree 1) and the only vertices with out-degree 0 (i.e. no children) are the *leaf* nodes. Every vertex is connected to the root by a directed path. If a node is not a leaf, then it is an *internal* node and is the parent of one or more child nodes. All children of the same node are *siblings*. The *level* of a vertex in a tree is the length of the directed path connecting it to the root. The *depth* of a rooted tree is the greatest level of any leaf [7].

A *k-ary tree* is a tree with at most k children for each node. A *full* k-ary tree is a k-ary tree that has *exactly* k children for each non-leaf node. A k-ary tree with depth l is *balanced* if all of its leaves are at level l or level $l-1$ and every node at a level less than $l-1$ has k children. These definitions provide the basis for defining a *perfect k-ary tree*, which is both flexible and complex enough to model data blocks.

Definition 6.5 (Perfect *k*-ary Tree)

A tree with all leaf nodes at the same depth and all internal nodes having exactly k children [5].

Definition 6.5 describes a k-ary tree that is both full and balanced. The model assumes that all data blocks are structured as perfect k-ary trees to provide a generic model when the data is a perfect tree structure, and a worst case otherwise; with k being the greatest number of children of any single data block and the depth being the greatest level of any data block. Figure 6.1 illustrates how a perfect 3-ary tree of depth 2 can be considered as a worst case for a non-perfect tree with a greatest branching factor of 3 and a depth of 2.

For simplicity, data blocks are considered to be initially unstructured, and therefore must be bounded in the preservation stage. Algorithm 6.1 takes an unstructured data block as input and returns an equivalent practically-bounded data block with a perfect k-ary tree structure as

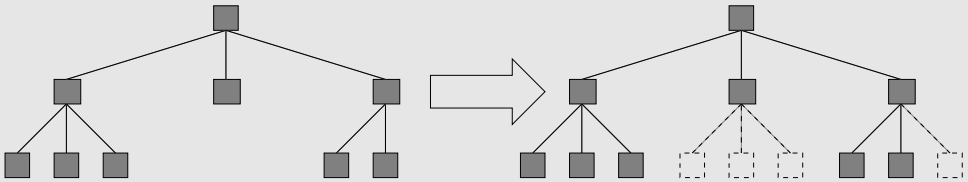

Figure 6.1: Perfect k-ary Tree as a Worst Case

output, where every internal node has a data size greater than b, and at most b for every leaf node.

Algorithm 6.1 (Autobound)

Given b and k, a leaf data block is bounded by the following algorithm:

```
Constants: b, k
begin Autobound(data)
    list := []
    dataSize := |data|
    if dataSize ≤ b then
        list.add(data)
    else
        for i = 0 to k-1 do
            blockSize := ⌈dataSize / (k−i)⌉
            dataSize := dataSize − blockSize
            block := data.range(k · i, blockSize)
            list.add(Autobound(block))
        end
    end
    return list
end
```

Examples 6.2 and 6.3 illustrate how Algorithm 6.1 handles decomposition of a data block when division is exact and "best fit" respectively. In Example 6.2, the data is divided exactly by k, and the resulting child data blocks are within $\mathbb{B}^b$. The structure of the result is shown more clearly in Figure 6.2.

Example 6.2 (Exact Division Auto-bounding)

Suppose the following data is auto-bounded using Algorithm 6.1 with $b := 4$ and $k := 4$:

`PAY␣HIM␣10K␣MORE`

The result would be:

`[[PAY␣],[HIM␣],[10K␣],[MORE]]`

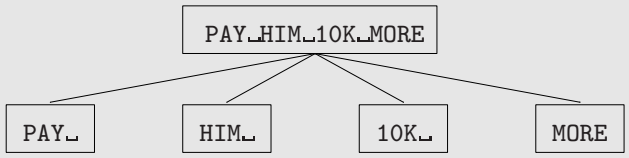

Figure 6.2: Bounded data block with $b := 4$ and $k := 4$

In Example 6.3, the data cannot be divided exactly by k, but the resulting child data blocks are still within $\mathbb{B}^b$. The resulting structure is shown more clearly in Figure 6.3.

Example 6.3 (Best Division Auto-bounding)

Suppose the following data is auto-bounded using Algorithm 6.1 with $b := 4$ and $k := 3$:

`PAY␣HIM␣10K␣MORE`

The result would be:

`[[[PA],[Y␣],[HI]], [[M␣],[10],[K]], [[␣M],[OR],[E]]]`

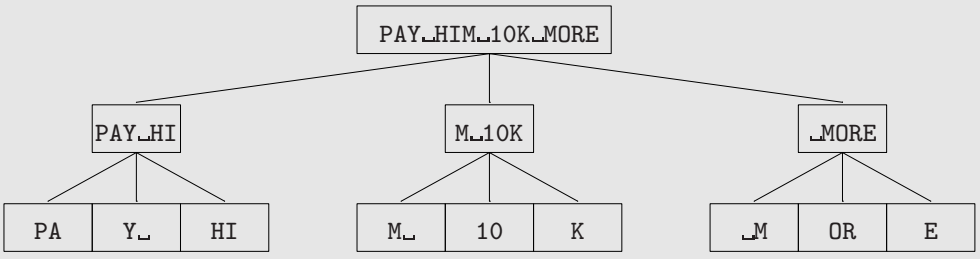

Figure 6.3: Bounded data block with $b := 4$ and $k := 3$

Once a data block is in the form of a perfect k-ary tree, its corresponding hash tree is determined by hashing the content at each node. This may be done by processing a bounded

data block with an algorithm that traverses every element hashing its data, or by extending the auto-bounding algorithm to compute the hashes of each sub-data block as they are determined. In either case, the result is a perfect k-ary tree, where the data is a hash of the corresponding data from the bounded data block.

The initial part of this processes, common to all preservation algorithms, is to precompute the practically bounded data block and preserve it in some way (see Algorithm 6.2).

Algorithm 6.2 (Preservation)

Given b, k and input data *data*, its corresponding hash tree is computed by the following algorithm:

```
Constants: b, k
begin Preserve(data)
    node := Autobound(data)
    return XPreserve(node)
end
```

The general preservation process for a single node is to hash all the node's data as a whole, then append each of its processed child nodes. The method of hashing and the method of processing the child nodes are left undefined (see Algorithm 6.3).

Algorithm 6.3 (Preserve Block)

The corresponding hash tree node for a data node is computed by the following algorithm:

```
begin PreserveNode(node)
    list := []
    list.add(XHash(node.data()))
    for i = 1 to #(node) do
        list.add(XPreserveNode(node.data(i)))
    end
    return list
end
```

In order to define a concrete preservation algorithm, a definition must be given for each of the undefined methods. The deterministic preservation algorithm (see Algorithm 6.4) defines preservation solely in terms of the generic algorithm using a perfect one-way hash function. Refer to Algorithm B.2 for a full definition of this algorithm.

Algorithm 6.4 (Deterministic Preservation)

Given b, k and input data *data*, its corresponding deterministic hash tree is computed by the algorithm defined as follows:

`XPreserve` ::= `PreserveNode`, where `XHash` ::= $\overleftrightarrow{H}$
`XPreserveNode` ::= `PreserveNode`, where `XHash` ::= $\overleftrightarrow{H}$

Figure 6.4 shows the deterministic Ashman hash tree that corresponds to the bounded data block from Example 6.2.

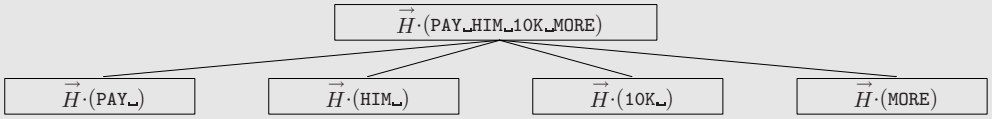

Figure 6.4: An Ashman Hash Tree of a Bounded data block

Preservation is concluded when the preservative (i.e. the Ashman hash tree) has been determined. The original data block and its preservative can now be exposed as described in Figure 3.4.

6.1.2 Resolution

The resolution stage is a three-step process that follows exposure, attempting to resolve the three solvable sub-problems. The detection step determines whether location is required, and the location step determines which of the data blocks require correction. Each sub-problem is solved using a different well-known principle.

Detection is achieved by comparing the hash tree's root hash with a hash of the exposed data block. The principle behind this solution is the collision resistance of cryptographic hash functions. Location is achieved using recursive detection over each sub-element and its corresponding hash element from the preservative. Search trees are the principle behind this solution. Correction is achieved by hashing all possible data blocks until a match is found with the corresponding original hash. This exhaustive search method is effectively impossible over an unbounded data block and computationally infeasible over a bounded leaf data block, unless the data block is small. However, a *divide and conquer* approach allows the search space to be drastically reduced.

The detection and location principles are almost identical to those used in hash-only integrity detection schemes and Merkle hash trees (see §2.7.2). Cryptographic hashes constitute the preservative, which is exposed via a trusted medium (effectively the same as the IP exposure model). Any data obtained via a tamper-vulnerable medium can have its integrity detected and located by comparison of its hash tree with the original hash tree (the preservative). Due to the collision free property of the POHF, any tampered leaf data block will be indicated by all

nodes in the path between itself and the root node. Thus any nodes that indicate no tampering do not need to be expanded further.

For the purpose of simplification, all three steps are combined into a single process known as *resolution*. The defending party uses resolution to maintain data integrity. If the data is original, then this is determined by the detection process of resolution and the original data is returned. If the data is tampered with, then this is determined by the detection process and located by iterating detection over increasingly smaller data blocks. Data blocks where tampering has been identified are restored by the correction process, all encapsulated within resolution, which also returns the original data. Furthermore, only tampered data requires location and only tampered parts of the data require correction. Therefore the processes of detection, location and correction are not discrete: Detection is a step of the location process, and location is a step of the correction process. Despite this simplification, the main discussion of resolution concerns the process of correction.

When tampering has been located to one or more of the leaf nodes in the hash tree, each of these nodes can be corrected individually. To correct an individual leaf node its bounds must be known, which is essentially the size of the leaf space used to create the hash tree (i.e. b) (see Definition 6.6). Knowing b allows an exhaustive search through all possibilities in $\mathbb{B}^b$ (see Definition 6.7), and provided that b is small enough, the search can be performed in reasonable time. Each individual search ends when a match is found, as the assumption about collision free hashing means no other match can exist. The original data has been recovered when all the tampered leaf data blocks are corrected.

Definition 6.6 (Data Block Search Space)

The set of all possible values that a data block comes from. In the context of the whole data d the space is the *root search space* $\mathbb{B}^n$. In the context of the smallest division of the data d_i the space is the *leaf search space* $\mathbb{B}^b$.

Definition 6.7 (Possibility)

A data block d_i is a *possibility* for an unknown data block $d \in \mathbb{B}^n$, if $d_i \in \mathbb{B}^n$.

An exhaustive search through all possibilities for the whole data is computationally infeasible. However, the search space can be divided into a collection of smaller, individually-verifiable search spaces by auto-bounding the data, reducing the overall computational cost of the search.

Example 6.4 (Search Space Reduction)

There are 2^{128} possibilities for a 128-bit data block. However, if the data block was bounded using a quad tree, with a 32-bit maximum size for each leaf data block, then the search space is reduced to four individual searches through 2^{32} possibilities. The total number of possibilities is now 2^{34}, which has reduced the search space by a factor of 2^{94}.

The search space reduction highlighted in Example 6.4 can be generalized by considering a data block of length n, (i.e. a 128-bit string has $n := 128$). Initially there are 2^n possibilities for an exhaustive search. Assuming that the data block is structured as a k-ary perfect tree of depth l in which the length of data at any two leaf nodes differs by at most 1, then the number of possibilities for initial values of l is given in Table 6.2.

Depth	Leaves	Leaf Search Space	Root Search Space
0	1	2^n	2^n
1	k	$2^{\lceil \frac{n}{k} \rceil}$	$k \cdot 2^{\lceil \frac{n}{k} \rceil}$
2	k^2	$2^{\lceil \frac{n}{k^2} \rceil}$	$k^2 \cdot 2^{\lceil \frac{n}{k^2} \rceil}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
l	k^l	$2^{\lceil \frac{n}{k^l} \rceil}$	$k^l \cdot 2^{\lceil \frac{n}{k^l} \rceil}$

Table 6.2: Search Space Reduction

As the depth of tree increases the number of possibilities decreases, reducing the search space. However, there exists a point where it becomes impossible to divide the data any further: where a leaf data block consists of a single bit. Moreover, as the depth of the tree increases, so does the size of the hash tree, and the limit on the size of the preservative must be considered.

The deterministic resolution algorithm (see Algorithm 6.10) is based around the assumption that any modified data blocks (or sub-data blocks) are determined by their corresponding hash, irrespective of the preimage size. As with preservation, the initial process of resolution is common to all resolution algorithms. The process involves checking that a single result for the original data has been determined. If zero or multiple results are found, then a *null* value is returned to indicate an error (see Algorithm 6.5).

Algorithm 6.5 (Resolution)

From b and k, given exposed data *data* and its corresponding original preservative p, the original data is computed by the following algorithm:

```
Constants: b, k
begin Resolve(data, p)
    node := Autobound(data)
    results := XResolve(node, p)
    if #(results) = 1 then
        return results.data()
    else
        return null
    end
end
```

The first step in resolving the exposed data is to detect whether it was tampered with or not by comparison of the hash of the node's data and its corresponding original hash. If the

hashes match, then the node's data is assumed to be original, otherwise it must be corrected (see Algorithm 6.6).

Algorithm 6.6 (Detection)

Given exposed data and its corresponding original preservative, the following algorithm returns a list of all data of with relevant matching hash(es) to the original data.

```
begin Detect(node, p)
    if XHash(node.data()) = p.hash() then
        return node
    else
        return Correct(node, p)
    end
end
```

The general correction process involves two steps. First a set of all possibilities must be determined for the current node, then each possibility must be tested to determine if it is a *candidate* for the current node (see Algorithm 6.7).

Algorithm 6.7 (Correction)

Given exposed data and its corresponding original preservative, the following algorithm returns a list of candidates for the original data.

```
begin Correct(node, p)
    possibilities := XFindPossibilities(node, p)
    return XFindCandidates(possibilities, p)
end
```

The set of possibilities at a leaf node is the set of all data within the bounds b. Only one possibility will match in the deterministic model (i.e. the original data), and this is formed by concatenating (in order) the matching possibility for each leaf node (see Algorithm 6.8).

Algorithm 6.8 (Deterministic Find Possibilities)

Given exposed data and its corresponding original preservative, the following algorithm returns a set of possibilities for the original data.

```
Constants: b, k
XFindPossibilities ::= DFindPossibilities
begin DFindPossibilities(node, p)
    if #(p) = 1 then
        return 𝔹^b
    else
        block := null
        for i = 1 to k do
            block := block‖XResolveNode(node.data(i), p.hash(i))
        end
        return {block}
    end
end
```

Candidates are possibilities that have the same hash as the original data, but in the deterministic model only the original data will have this hash. The list of candidates is determined by hashing each possibility from the given set of possibilities until one matches the corresponding hash in the hash tree. The process ends once a match is found, because the only candidate in the deterministic model is the original data (see Algorithm 6.9).

Algorithm 6.9 (Deterministic Find Candidates)

Given a set of possibilities for the original data and its corresponding original preservative, the following algorithm returns a list of candidates for the original data.

```
XFindCandidates ::= DFindCandidates
begin DFindCandidates(possibilities, p)
    foreach possibility ⇐ possibilities do
        if XHash(possibility) = p.hash() then
            return [possibility]
        end
    end
end
```

The deterministic resolution algorithm (see Algorithm 6.10) defines each of the abstract algorithms in the generic resolution algorithm. The algorithm is based on depth-first recursion of the generic detection and correction algorithms, using a perfect one-way hash function for all hashing. Refer to Algorithm B.3 for a full definition of this algorithm.

Algorithm 6.10 (Deterministic Resolution)

Given exposed data and its corresponding original preservative, the following algorithm returns the original data.

Constants: b, k
`XResolve` ::= `Detect`, where `XHash` ::= $\overleftrightarrow{H}$
`XFindPossibilities` ::= `DFindPossibilities`
`XFindCandidates` ::= `DFindCandidates`, where `XHash` ::= $\overleftrightarrow{H}$
`XResolveNode` ::= `XResolve`

Example 6.5 (Deterministic Resolution)

Suppose that a data block d is made up of four data blocks (d_0, d_1, d_2, d_3) and structured as binary tree such that the root data block is $d_0 \| d_1 \| d_2 \| d_3$, its children are $d_0 \| d_1$ and $d_2 \| d_3$ and its leaves are each individual block. Algorithm 6.4 can be used to determine its corresponding hash tree p.

If d is tampered with so that d_1 is replaced with f_1, then Algorithm 6.10 will determine that the root is modified. At level 2 it will determine that the left child is modified and check its children before checking the right child. At level 3 it will determine that the left child is original, but the right child is modified. Since the right child is a modified leaf node it will then begin an exhaustive search through all possibilities until it checks d_1, which it will return as the match. As there are no more children, it assumes that the left child at level 2 is now corrected and checks the right child at level 2. As this level 2 node is original there is no need to check its children, and since there are no further nodes at this level it assumes to have corrected the parent, which is the root.

The scenario described in Example 6.5 is illustrated in Figure 6.5, which shows checked nodes determined to be original with a "✓", checked nodes determined to be modified with a "✗", and unchecked nodes with a "**?**".

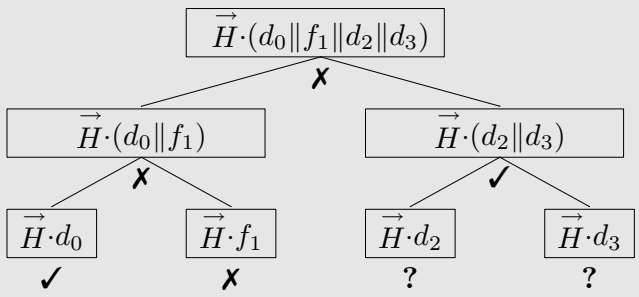

Figure 6.5: Binary Hash Tree with One Tampered Leaf Node Identified

Using MLDI employs a divide-and-conquer technique to reduce the correction problem from an exhaustive search of 2^n possibilities to an exhaustive search of 2^b possibilities when tampering has been located to a single block, or k^l (where $l \ ::= \ \log_k \lceil \frac{n}{b} \rceil$) individual exhaustive searches when all blocks contain tampering. Without MLDI, an unoptimized method would have to correct tampering before it could be located. Therefore the unoptimized time complexity for location is equal to that of correction. However, MLDI can achieve block-level location in $O(\log n)$ for a single instance, and in $O(n)$ when all blocks contain tampering. In any case, the detection process requires a single hash computation. These time complexities are summarized in Table 6.3.

Process	**Tampered Blocks**	**Unoptimized Method**	**Deterministic MLDI**
Detect	1	$O(1)$	$O(1)$
	all	$O(1)$	$O(1)$
Locate	1	$O(2^n)$	$O(\log_k n)$
	all	$O(2^n)$	$O(n)$
Correct	1	$O(2^n)$	$O(1)$
	all	$O(2^n)$	$O(n)$
Overall	1	$O(2^n)$	$O(\log_k n)$
	all	$O(2^n)$	$O(n)$

Table 6.3: Time Complexities for Algorithms in the Deterministic Model

Chapter Summary

The fundamental principles of multi-layered document integrity have been introduced, including the central idea concerning the divide-and-conquer approach to location and correction. The notion of bounding the data and structuring it as a perfect k-ary tree to compute a hash tree as a preservative has also been discussed.

The deterministic model has presented multi-layered document integrity in a simplified manner, introducing generic algorithms for preservation and resolution, and using them to build a deterministic solution. The time complexities of deterministic MLDI indicate that it could provide a feasible solution to the data integrity problem in terms of computational complexity. Despite the favourable time complexities, the hash tree size has not been considered in the non-compressive deterministic model, and therefore contravenes the previous result that correction must be non-deterministic. As such, the use perfect one-way hashing to provide a simplified deterministic model must be readdressed.

Chapter 7

Realization

Between falsehood and useless truth there is little difference. As gold which he cannot spend will make no man rich, so knowledge which cannot apply will make no man wise.

— Dr Samuel Johnson, *The Idler* No. 84 (November 1759)

Some of the fundamental ideas behind MLDI were introduced in the deterministic model, where hash collisions are non-existent. However, the use of perfect one-way hashing means the image size is at least the preimage size (due to the non-compressive property). This results in the hash tree's root node being at least the size of the data, and therefore the preservative must be at least the size of the original data (without even considering any children in the hash tree). The deterministic model for preservation and resolution conflicts with the invulnerable preservative model for exposure (see Definition 3.8). This chapter introduces an alternative probabilistic model that aims to solve this problem.

7.1 Probabilistic Model

The *probabilistic model* is based on compressive hash functions, whereby collisions must exist due to the pigeonhole principle (see Definition 7.1). In practice, the use of a cryptographic hash function minimizes the problem of collisions by providing compression whilst remaining collision resistant (see Definition A.7).

Definition 7.1 (Pigeonhole Principle)

If n discrete objects are to be allocated to m containers, then at least one container must hold no fewer than $\lceil \frac{n}{m} \rceil$ objects [25].

7.1.1 Collision Resistant Model

The *collision resistant model* is based on the use of cryptographic hash functions. This model assumes that collisions exist in theory, but do not occur in practice. Moreover, the hash function will always identify tampered data blocks, thereby preventing an attacker from determining a collision. However, when tampered data blocks are identified, all possibilities must be checked for a match rather than accepting the first match. These differences necessitate several modifications to the deterministic algorithms.

The preservation algorithm is identical to that of deterministic preservation, except that a cryptographic hash function is used in place of a perfect one-way hash function (see Algorithm 7.1). The preservative is now constructed from fixed-length cryptographic hashes and therefore is generally smaller in overall size than a corresponding deterministic preservative. Refer to Algorithm B.4 for a full definition of this algorithm.

Algorithm 7.1 (Collision Resistant Preservation)

Given $\mathbb{B}^b$ and k, the Ashman hash tree for a leaf data block is produced by the following algorithm:

`XPreserve` ::= `PreserveNode`, where `XHash` ::= H
`XPreserveNode` ::= `PreserveNode`, where `XHash` ::= H

The same principles underlie solving the detection and location problem, due to the assumption that collisions do not occur. The existence of collisions dictates that the original value of a block can be any one of several colliding values. The deterministic resolution algorithm terminates as soon as a match is found, but if there are multiple possibilities that match, then all of these must be found. Multiple possibilities that share the same hash as the original data block are defined as *candidates* for the original data block (see Definition 7.2).

Definition 7.2 (Candidate)

A possibility c is a *candidate* for a data block d if its hash matches the corresponding data block's hash from the hash tree ($H \cdot c = H \cdot d$ and $c, d \in \mathbb{B}^n$). The set of candidates is denoted $\mathbb{C}$, such that $c \in \mathbb{C}$.

The full set of candidates for a leaf data block will always contain the original leaf data block. Therefore, when the set of candidates has been determined, a method is required to identify which candidate is the original. By testing each candidate at the parent level, it should be possible to eliminate some of the candidates, and iteration of this technique should allow the original data to be determined.

To illustrate the idea of testing candidates at the parent level, consider the example illustrated in Figure 6.5 where one of the four data blocks (d_1) had been tampered with. Assume that when correcting this block the exhaustive search determines a set of eight candidates from all the possibilities. These eight candidates all have matching hashes, which also match the corresponding original hash from the hash tree. To determine which is the original, each candidate $c_0, c_1, \ldots, c_7$ can be concatenated with its verified siblings (or sibling in this case) to form a set of possibilities for the parent data block $d_0\|c_0, d_0\|c_1, \ldots, d_0\|c_7$. Each parent possibility can be hashed and compared against the corresponding original parent hash from the hash tree. Since one of the candidates is the original value for d_1, then the hash of (at least) one of the parent possibilities ($d_0\|c_0, d_0\|c_1, \ldots, d_0\|c_7$) will match the original parent hash ($H \cdot (d_0\|d_1)$).

Even if multiple parent candidates match, by iterating this process upwards the set of candidates should be reduced, since only candidates (as opposed to all possibilities) need to be

considered as potential original data blocks after the leaf node level. Eventually the number of candidates should be reduced to one: the original data block.

The principle behind correction, when collision-resistant hash functions are used, is based around the *candidate reduction process*. To model this process it is assumed that whenever a set of candidates is determined or reduced via comparisons with a corresponding hash in the original hash tree, the process is behaving as a *filter*. The repeated filtering of possibilities to candidates should eventually determine the original data.

Ideally a cryptographic hash function should map its input values to output values with a discrete uniform distribution, albeit in a seemingly random manner. As such, the frequency of each possible output value, the *collision frequency*, should be approximately constant and equal to the size of the domain divided by the size of the range. The collision frequency can be used to provide an approximation for how many candidates a filter produces. For example, a hash function with a 16-bit preimage and an 8-bit hash will have a collision frequency of approximately $2^{16}/2^8 = 2^8$. If the preimage size changes and the hash size remains the same, then the collision frequency will change accordingly. However the probability of a possibility (in the sense of Definition 6.7) or candidate filtering through a stage remains the same, as the distribution is approximately uniform.

To reason about the candidate reduction process it is necessary to make an assumption as to the probability of a possibility filtering through one stage to become a candidate. A preimage can map to any of the possible hashes and does so with uniform distribution, since only possibilities that collide with the original hash are candidates, then the collision probability can be assumed to be that given in Definition 7.3.

Definition 7.3 (Hash Collision Probability Assumption)

The probability u of a possibility d_i having a hash collision with the original data d (i.e. $\Pr(H \cdot c = H \cdot d)$) is approximately

$$\frac{1}{2^{|h|}},$$

where h is an image of the hash function H. To reason about the number of candidates, this approximation is assumed as the equality

$$u = 2^{-|h|}. \tag{7.1}$$

Each single filtering process should remove some of the possibilities as potential original values of a data block. The remaining possibilities (candidates) must then be combined with their siblings to form a set of possibilities for the parent. If the same hash function is used throughout, then the collision probability remains constant throughout the process. By modelling each filter process as a discrete event the probability of a single instance of tampering passing through i levels is $\frac{1}{2^{(i \cdot |h|)}}$. Therefore it becomes increasingly improbable for a single instance of tampering to remain uncorrectable over multiple levels.

The same principle does not hold when considering multiple instances of tampering. If there are non-verified siblings for which the original data blocks are still undetermined (i.e.

have candidates), then this could potentially result in a larger set of possibilities/candidates at the parent level (see Example 7.1).

Example 7.1 (Potential for Candidate Increases)

Supposing a binary tree has two sibling leaf nodes, each having 1024 possibilities (i.e. 2048 possibilities in total). If each node is determined to have a set of 100 candidates, then there are $100 \cdot 100 = 10000$ possibilities for the parent. Since this is greater than the original number of possibilities for the two leaf nodes combined and the hash collision probability remains constant, then there will be more than 100 candidates at the parent node.

When multiple leaf nodes contain tampering, a set of candidates can exist for multiple siblings. In this case the possibilities to be tested at the parent level are the elements of the set formed by the Cartesian product of the sibling candidates. Consider a perfect k-ary tree with the parent's branches labelled 0 through $k-1$. If $\mathbb{C}_i$ is the set of candidates on branch i at the child data block, then the set of parent possibilities is determined by all possible ordered combinations of the child candidates, given by $\mathbb{C}_0 \times \mathbb{C}_1 \times \ldots \times \mathbb{C}_{k-1}$.

Two parts of the deterministic resolution algorithm must be revised to reflect the presence of hash collisions, and cryptographic hashing must be used in place of perfect one-way hashing. The principle behind the revised algorithm is to assume that all verified nodes are original and all non-verified nodes have been modified, and therefore should be corrected by considering all possibilities.

The probabilistic algorithm for determining the set of possibilities is based on the deterministic algorithm, but differs in the case of internal nodes. When determining the set of possibilities for an internal node the algorithm computes the Cartesian product of the candidate sets from each child node (see Algorithm 7.2). An algorithm for computing the Cartesian product can be found in Appendix B (see Algorithm B.1).

Algorithm 7.2 (Probabilistic Find Possibilities)

Given exposed data and its corresponding original preservative, the following algorithm returns a set of possibilities for the original data.

```
Constants: b, k
XFindPossibilities ::= PFindPossibilities
begin PFindPossibilities(node, p)
    if #(p) = 1 then
        return 𝔹^b
    else
        list := []
        for i = 1 to k do
            list.add(XResolve(node.data(i), p.hash(i)))
        end
        return CartesianProduct(list)
    end
end
```

Determining the candidates from the set of possibilities in the deterministic model involves hashing *each* possibility in the set until a match was found. In the probabilistic model, *every* possibility must be hashed to determine a list of all matching possibilities. Algorithm 7.3 performs the exhaustive search to determine this list of candidates.

Algorithm 7.3 (Probabilistic Find Candidates)

Given a set of possibilities for the original data and its corresponding original preservative, the following algorithm returns a list of candidates for the original data.

```
XFindCandidates ::= PFindCandidates
begin PFindCandidates(possibilities, p)
    candidates := []
    foreach possibility ⇐ possibilities do
        if XHash(possibility) = p.hash() then
            candidates.add(possibility)
        end
    end
    return candidates
end
```

The collision resistant resolution algorithm (see Algorithm 7.4) is almost identical to its deterministic counterpart. However, cryptographic hashing is used throughout, and probabilistic algorithms replace the previous deterministic algorithms for determining the possibilities and candidates. Refer to Algorithm B.5 for a full definition of this algorithm.

Algorithm 7.4 (Collision Resistant Resolution)

Given exposed data and its corresponding original preservative, the following algorithm returns the original data.

Constants: b, k
`XResolve` ::= `Detect`, where `XHash` ::= H
`XFindPossibilities` ::= `PFindPossibilities`
`XFindCandidates` ::= `PFindCandidates`, where `XHash` ::= H
`XResolveNode` ::= `XResolve`

The collision resistant model employs cryptographic hashing in order to realize the assumption that collisions exist, but do not occur. This approach is based on the second preimage resistance of cryptographic hash functions, making it computationally infeasible for an adversary to determine a counterfeit data block at any stage of the process due to the large search space. However, if the data blocks are practically bounded for correction purposes, then the search space must be small enough for an attacker to determine counterfeit blocks (i.e. collisions or candidates). Given a hash node from the preservative, an attacker can use the correction algorithm (see Algorithm 6.7) with the probabilistic algorithms for finding possibilities and candidates to determine a list of collisions for the corresponding data. Any of the colliding values can then be used to create a counterfeit for the original data, which will be detected by the collision resistant algorithm, but cannot be located or corrected by it. The assumptions concerning collisions must be weakened further to incorporate both their existence and their occurrence.

7.1.2 Collision Prone Model

The *collision prone model* assumes both the existence and the occurrence of hash collisions. The principles behind solving the detection and location problems still hold, due to the hash's collision resistance. However, the collision resistance is weakened at each successive level in the hash tree as the preimage size decreases, approaching (and possibly surpassing) the image size due to bounding. Consequently, it becomes more feasible for an adversary to determine collisions, which can be substituted for the original lower-level data blocks.

If an attacker can feasibly create a counterfeit data block that contains tampered elements that collide at one level, then this possibility needs to be incorporated into the resolution algorithm. If a data block's hash and its corresponding original hash from the hash tree match, then it cannot be assumed that the data block is original. To show that a data block is original it must be determined to be the *only* candidate, which involves a search throughout all of its child possibilities. However, the algorithm must also handle multi-level collisions, where a tampered data block collides at multiple consecutive levels throughout the hash tree.

A further problem with the collision resistant algorithm is the hash tree size. Using cryptographic hashes to build the hash tree has the advantage of collision resistance, but the resulting hash tree is still relatively large in size. The most trivial solution to this problem is to reduce the size of the hashes at the cost of increasing the collision frequency for each node.

The latter problem is addressed using *bounded-preimage hashing* (see Definition 7.4) in place of cryptographic hashing for each algorithm. The former problem is addressed by removing the detection and location processes from the resolution algorithm, which results in an algorithm that indiscriminately traverses every node in the hash tree, considering every possibility and determining every candidate. By expanding every node fully, the algorithm will never accept a counterfeit node as genuine.

Definition 7.4 (Bounded-preimage Hash Function)

A *bounded-preimage hash function* $\vec{H}$ is a cryptographic hash function with a practically-bounded preimage, such that all inputs are in $\mathbb{B}^b$, allowing an exhaustive search of $\mathbb{B}^b$ within a practically-feasible time period.

The preservation algorithm is identical to the previous preservation algorithms, except for the use of bounded-preimage hashing (see Algorithm 7.5). Refer to Algorithm B.6 for a full definition of this algorithm.

Algorithm 7.5 (Collision Prone Preservation)

Given b and k, the Ashman hash tree for a leaf data block is produced by the following algorithm:

`XPreserve` ::= `PreserveNode`, where `XHash` ::= $\vec{H}$
`XPreserveNode` ::= `PreserveNode`, where `XHash` ::= $\vec{H}$

The problem of multi-level collisions is addressed by assuming that all nodes have been tampered with, and therefore the possibility and candidate lists for every node must be determined. Algorithm 7.6 replaces the detection-correction process with correction alone, and therefore performs indiscriminate resolution by traversing and correcting all nodes, irrespective of whether they have been tampered with. Refer to Algorithm B.7 for a full definition of this algorithm.

Algorithm 7.6 (Collision Prone Resolution)

Given exposed data and its corresponding original preservative, the following algorithm returns the original data.

Constants: b, k
`XResolve` ::= `Correct`, where `XHash` ::= $\vec{H}$
`XFindPossibilities` ::= `PFindPossibilities`
`XFindCandidates` ::= `PFindCandidates`, where `XHash` ::= $\vec{H}$
`XResolveNode` ::= `XResolve`

Assuming that the exposed data has been tampered with at every level means that it is never referenced. Algorithm 7.6 requires the exposed data and original preservative as arguments, for generality rather than necessity. In practice, the exposed data is not required as an input to the collision-prone resolution algorithm.

The use of bounded-preimage hashing provides more flexibility in the size of the hash tree, enabling the preservative to be smaller than the data. Aside from the increased risk of multi-level collisions, the lack of detection and correction results in an inefficient resolution algorithm. This is particularly prevalent when the exposed data is original, because the algorithm expands every node reconstructing the data from the preservative alone.

7.1.3 Hybrid Model

The *hybrid model* is a compromise between the two contrasting probabilistic models. The MLDI idea postulated by Ashman uses a cryptographic hash at the root of the hash tree to prevent counterfeits from filtering through all layers, whilst keeping the overall size of the hash tree to a minimum by using "small hashes" (i.e. bounded-preimage hashes).

The root node is considered as a special case in the hybrid model and must therefore be handled separately in both preservation and resolution (see Algorithm 7.7). Refer to Algorithm B.8 for a full definition of this algorithm.

Algorithm 7.7 (Hybrid Probabilistic Preservation)

Given b and k, the Ashman hash tree for a leaf data block is produced by the following algorithm:

`XPreserve` ::= `PreserveNode`, where `XHash` ::= H
`XPreserveNode` ::= `PreserveNode`, where `XHash` ::= $\vec{H}$

The original algorithm outlined by Ashman [1] has been interpreted into the so-called *Ashman-style resolution* algorithm (see Algorithm 7.8). Refer to Algorithm B.9 for a full definition of this algorithm.

Algorithm 7.8 (Ashman-style Resolution)

Given exposed data and its corresponding original preservative, the following algorithm returns the original data.

Constants: b, k
`XResolve` ::= `Detect`, where `XHash` ::= H
`XFindPossibilities` ::= `PFindPossibilities`
`XFindCandidates` ::= `PFindCandidates`, where `XHash` ::= H
`XResolveNode` ::= `Detect`, where `XHash` ::= $\vec{H}$
`XFindCandidates` ::= `PFindCandidates`, where `XHash` ::= $\vec{H}$

Ashman's algorithm performs detection using cryptographic hashing at the root node and location with bounded-preimage hashing elsewhere. As previously mentioned, when bounded pre-image hashing is used for location, an attacker can determine multi-layer collisions, which remain undetectable. Therefore a final hybrid resolution algorithm is given that uses cryptographic hashing for detection and, without performing location, fully corrects the data if tampering is indicated (see Algorithm 7.9). Refer to Algorithm B.10 for a full definition of this algorithm.

Algorithm 7.9 (Hybrid Resolution)

Given exposed data and its corresponding original preservative, the following algorithm returns the original data.

Constants: b, k
`XResolve` ::= `Detect`, where `XHash` ::= H
`XFindPossibilities` ::= `PFindPossibilities`
`XFindCandidates` ::= `PFindCandidates`, where `XHash` ::= H
`XResolveNode` ::= `Correct`, where `XHash` ::= $\vec{\vec{H}}$
`XFindCandidates` ::= `PFindCandidates`, where `XHash` ::= $\vec{\vec{H}}$

The hybrid resolution algorithm uses bounded-preimage hashing in a similar manner to Ashman's algorithm, allowing correction to be performed whilst minimizing the preservative size. However, the purpose of algorithms in the hybrid model is only in providing a basis for reasoning about MLDI schemes: The design of compression- and performance-optimized algorithms is left as an open problem (see §9.3).

7.2 Attack Models

There are several approaches an adversary might take in attacking a MLDI scheme; and the nature of these approaches depends on what the adversary is attempting to achieve and the resources they have available. For example, a powerful adversary might have the computational resources to determine a counterfeit that is undetectable by a MLDI scheme, whereas a weaker adversary might attempt only to make one data block uncorrectable, disregarding the fact that their tampering will be detected.

Attacks can be classified into two categories: *overt* and *covert*. A tampered node is indicated by the presence of one or more f blocks. If the hash of a tampered node matches the corresponding hash in the original hash tree (indicated by a ✓ below), the counterfeit node forms the basis of a *covert* attack. Alternatively, an *overt* attack is formed by an easily-detectable tampered node where the corresponding hashes do not match (indicated by a ✗ below). A non-root, non-leaf node is referred to as a *mid* node. Counterfeit nodes can be either *single-* or *multi-level*, with the latter meaning a counterfeit which has a child or parent also being a counterfeit (i.e. counterfeits on consecutive levels).

The attacks are presented by means of an example, based on data split into four blocks. The hybrid preservation algorithm (see Algorithm 7.7) is used to create a binary hash tree with

a cryptographic root hash and bounded-preimage hashes throughout the rest of the tree. As such, it is assumed that determining a single-level counterfeit leaf or mid node is practically feasible, as is determining a multi-level counterfeit within the leaf and mid nodes. However determining a root counterfeit is assumed to be computationally infeasible.

7.2.1 Overt Attacks

The most basic form of attack is for an adversary to arbitrarily modify a single block of the data. In a *tampered leaf node* attack, the modified block is present in all nodes along the path between the root and itself, and the corresponding hashes will indicate this. Figure 7.1 illustrates this type of attack on a simple hash tree where the original block d_0 has been replaced with tampered block f_0. A resolution algorithm should be trivially secure against a tampered leaf node attack.

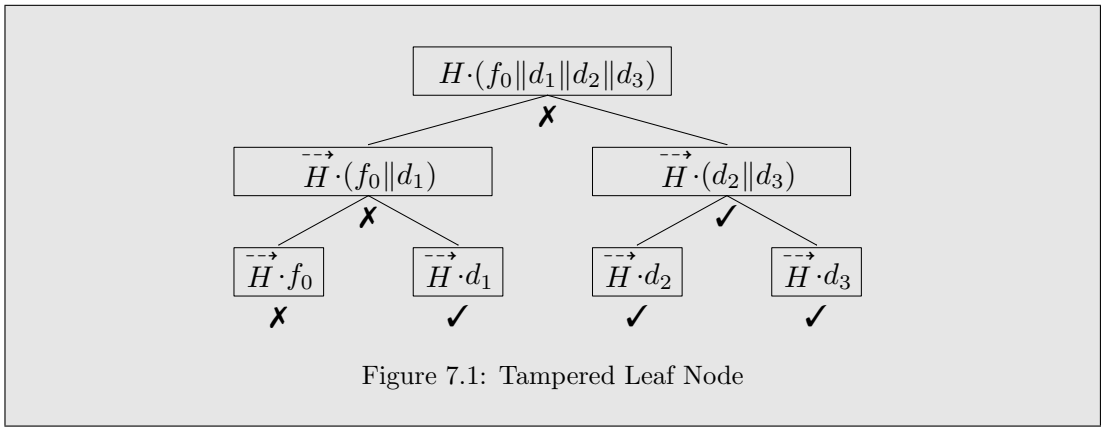

Figure 7.1: Tampered Leaf Node

A similar attack involves modifying two or more blocks that correspond to siblings in the hash tree. In a *tampered leaf siblings* attack, the modified blocks are present in their parent node and all nodes from it to the root. Figure 7.2 illustrates the case when two leaf siblings, (d_0 and d_1), are replaced by two tampered leaf siblings (f_0 and f_1). A resolution algorithm designed with the assumption that correction of a single tampered child node will automatically correct its parent, will not be secure against this form of attack.

There are two methods for overcoming a tampered leaf siblings attack. The pre-emptive method checks all children of a tampered node before determining its candidates. This method is generally less efficient in terms of the number of hash comparisons, but less complex to implement as the tree traversal is more straightforward. The post-emptive method checks all remaining children of a tampered node after determining that none of the currently established candidates are correct. This method is generally more efficient, but requires a more complex tree-traversal algorithm. Either method provides a solution to the problem; each having advantages over the other.

The final tampering attack illustrates that tampering does not necessarily involve a direct change in content: instead, it might involve a change in the block ordering. In a *transposed leaf siblings* attack, blocks that correspond to two or more sibling nodes are reordered, therefore

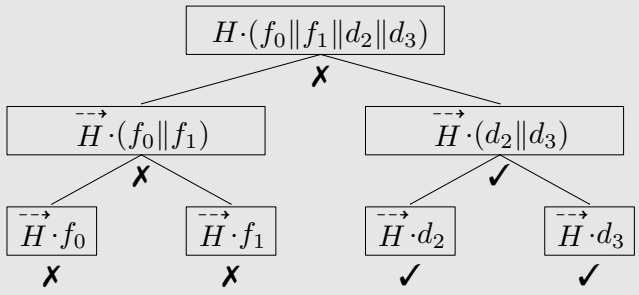

Figure 7.2: Tampered Leaf Siblings

indicating modification of the nodes involved. Figure 7.3 illustrates a simple hash tree in which blocks d_0 and d_1 have been swapped.

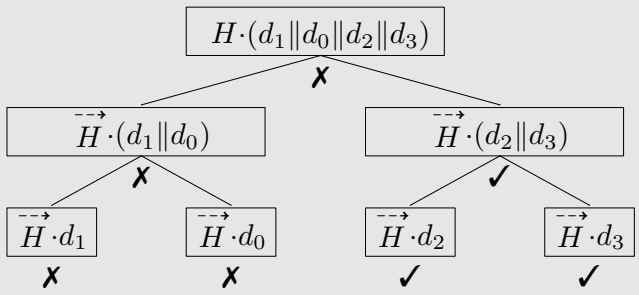

Figure 7.3: Transposed Leaf Siblings

Securing a resolution algorithm against the transposition of sibling nodes is achieved in the same manner as the general case of tampered leaf siblings. However, an algorithm could be optimized to overcome this type of attack without the need for exhaustive searching.

7.2.2 Covert Attacks

The more serious form of attack that MLDI schemes face involve counterfeits; where a tampered node appears to be original. An adversary can construct a *counterfeit leaf node* attack, by replacing a single original data block d_0 with a tampered block f_0 that has a matching hash (see Figure 7.4). This is achieved using a similar method to that of correction, by an exhaustive search through the set of leaf node possibilities ($\mathbb{B}^b$). A simple resolution algorithm would identify tampering in the parent node, but then fail to identify which of its child nodes were tampered with.

To overcome this problem a resolution algorithm's design must assume that any tampered mid node must be the result of a descendant tampered leaf node. Therefore each of the descendant nodes is a potential counterfeit, and their candidates must be determined and checked at

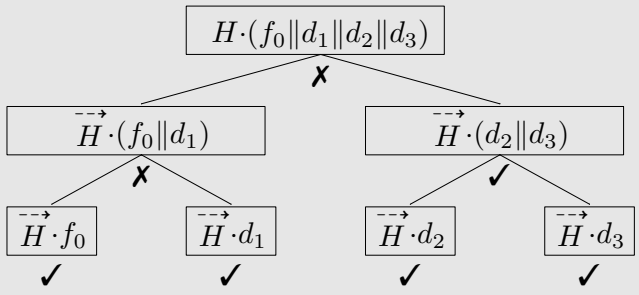

Figure 7.4: Counterfeit Leaf Node

the parent level. However, an exhaustive search is performed on any descendant nodes that do not contain tampering, which might be avoided using an alternative technique. In the example illustrated in Figure 7.4, the hash tree indicates that data blocks f_0 and d_1 are both original, therefore an exhaustive search must be performed on both of them, despite only f_0 containing tampering.

A similar situation occurs when an adversary can construct a *counterfeit mid node* attack, whereby the tampered data block forms a counterfeit at a mid node. However, an exhaustive search of all descendant possibilities from a mid node requires many more hash computations than at a leaf node.

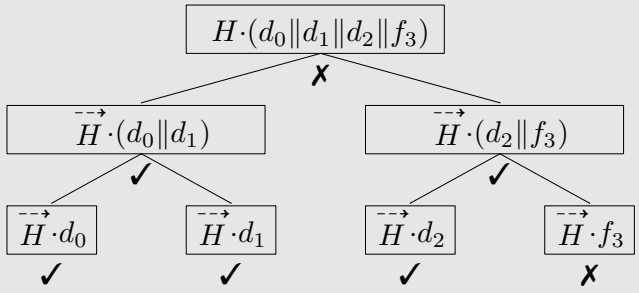

Figure 7.5: Counterfeit Mid Node

By assuming that counterfeits cannot occur over multiple consecutive levels, the problem can be solved by expanding all nodes a further level to determine which child nodes indicate tampering. In the example shown in Figure 7.5, the expansion of the left-hand mid node would indicate that data blocks d_0 and d_1 are original, therefore the right-hand mid node would be expanded, and the source of tampering would be located to data block f_3.

A *multi-level counterfeit* attack is based on the assumption that counterfeits can occur on two or more consecutive levels, which might be feasible when bounded-preimage hashing is used. In this case, an exhaustive search of all possibilities below the root node is necessary, but due to the complexity of this task, optimization techniques would be beneficial.

The situation is similar to that of the previous problem, but the algorithm can not determine any false candidates by checking their children for modifications as the children themselves are false candidates (see Figure 7.6).

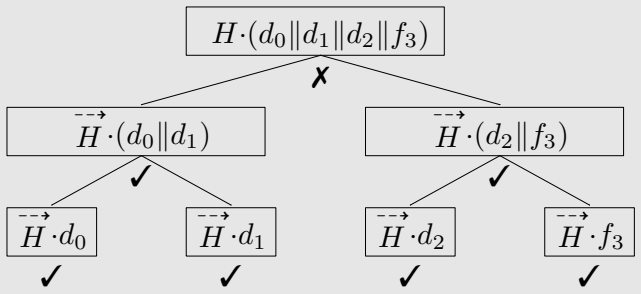

Figure 7.6: Multi-level Counterfeit

The solution to this problem is simply to recursively check all children of a parent with at least one false child candidate. In the case of leaf data blocks (with no children to check), all leaf candidates are found, and the true candidates are found by checking the parent candidates recursively.

Other types of attack that an algorithm designer should be aware of involve a combination of the previously mentioned attacks. For example, it may be feasible for an attacker to construct a multi-level counterfeit attack that contains a *dummy* node: indicating the presence of tampering in one part of the hash tree when further tampering is present in another part of the tree, but "hidden" behind a multi-level counterfeit. Figure 7.7 illustrates a simple form of the dummy attack, where data blocks f_0 and f_1 form an easily-locatable dummy and data block f_3 forms a less obvious multi-level counterfeit.

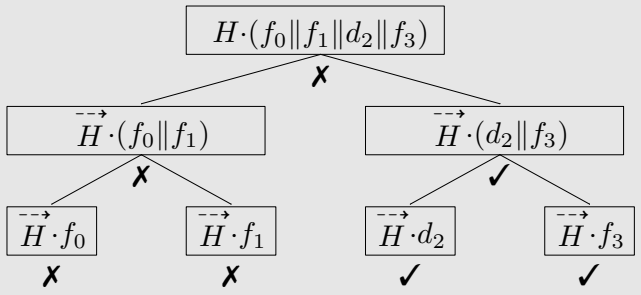

Figure 7.7: Multi-level Counterfeit with Dummy

Again, the obvious solution to this problem is an exhaustive search for all candidates beyond the root node, but this has a relatively high computational cost so optimization techniques should be considered §9.3.1.

Chapter Summary

Based on the assumption that collisions exist, the probabilistic nature of multi-layered document integrity has been addressed by the introduction of two opposing collision models. These models differ in their assumption regarding the occurrence of collisions. The collision resistant model assumes that collisions do not occur, despite their existence. Therefore an attacker is unable to substitute data blocks which remain undetected and/or cannot be located. The collision prone model assumes that collisions do occur and, in the extreme case, every node must be corrected to ensure substituted data blocks are corrected. Ultimately this means ignoring the indicators for detection and location. A compromise solution that is both secure and efficient, has been presented in the form of the hybrid model, which combines cryptographic hashing at the root node with bounded-preimage hashing elsewhere.

The hash collision probability has been defined, along with the notion of a node having multiple candidates when hashes collide. The idea of the candidate reduction process has been described, including the potential for an increase in candidates when sibling nodes in the hash tree contain tampered blocks. To realize the invertible hashing described by Ashman, this thesis has introduced the concept of practically bounding the preimage of the hash function. A side-effect of bounded-preimage hashing is a large increase in the collision probability. The higher collision probability makes the hash tree more susceptible to collisions, at both single and multiple levels in the tree.

Collisions are not only a problem from the point of view of determining the original data, but also from a security aspect. Various tampering attacks have been introduced, which can be classed as overt or covert, dependent on the hash tree's ability to indicate them. Two adversarial aims have been discussed for attacking MLDI schemes: those which address the data integrity problem directly, and those which attempt to make resolution less efficient. It is suggested, therefore, that the security of MLDI should be considered from two perspectives: whether the algorithm can resolve the integrity problems; and, if so, whether the problems can be resolved in a reasonable time period.

Chapter 8

Results

He said the only way you can get around it is to come up with believable limitations, and you have to be very specific about what those limitations are right at the outset.

— David Eddings [*attributed in an interview*]

The remaining results of this thesis are presented in this chapter, which (for the most part) relate to the previous two chapters. however, several overall results are also presented. Section 8.1 describes the four major constraints of multi-layered document integrity that all MLDI schemes must adhere to. Section 8.2 discusses the ability of such schemes to meet these constraints, thereby determining the efficacy of MLDI as a solution to the data integrity problem.

8.1 Constraints

Unless otherwise stated, the model of multi-layered document integrity used throughout this section is based on a hash tree with identically-sized hashes at every node. This model is a minor simplification of the hybrid hash tree described in Algorithm 7.7, which allows for a more elegant expression of definitions, propositions, and their related proofs. Regardless of this simplification, all results still hold in the less elegant model.

8.1.1 Preservative Size

Theorem 5.2 showed that the amount of information in the preservative must be limited by the amount of information in the data. The limitation forms part of the definition of the IP model (see Equation 3.4), which effectively states that the preservative size must be less than that of the original data (see Definition 8.1).

Definition 8.1 (Preservative Size Constraint)

The constraint that limits the size of the preservative relative to the size of the data such that:

$$|p| < |d| \tag{8.1}$$

In the context of multi-layered document integrity, the preservative size constraint limits the size of the hash tree to being smaller than the data from which it was derived. The hash

tree size is the product of the hash function's image size and the number of nodes in the tree, which can be determined by summing the number of nodes at each level (see Claim 8.1).

Claim 8.1 (Number of Nodes in a Level)

The number of nodes in a perfect k-ary tree at level i is given by k^i.

Proof
The proof (by induction on i) is trivial and covered in many introductory discrete mathematics textbooks. ■

The size of a hash tree is determined by the product of the hash function's image size and the sum of the number of nodes at all levels in the tree (as defined by its depth) (see Theorem 8.1). No structural data (i.e. any pointers) is necessary due to the fixed node size and regular structure of a perfect k-ary tree (see Remark 8.1).

Theorem 8.1 (Hash Tree Size)

The size of a hash tree with hashes of size $|h|$, a branching factor k and a depth l is given by:

$$|h| \cdot \left(\frac{k^{l+1} - 1}{k - 1} \right) \tag{8.2}$$

Proof

$$\begin{aligned}
& |p| \\
= \quad & \{p \text{ is constructed of a number of equally-sized hashes}\} \\
& [\text{single hash size}] \cdot [\text{number of hashes}] \\
= \quad & \{\text{number of hashes} = \text{sum of number of hashes at each level}\} \\
& |h| \cdot ([\text{sum of number of hashes at each level}]) \\
= \quad & \{ \ l \text{ levels after the root (level 0) and } k^i \text{ hashes at level } i\} \\
& |h| \cdot \langle \ \Sigma i \mid 0 \leq i \leq l : \ \ k^i \ \rangle \\
= \quad & \{\text{Geometric series}\} \\
& |h| \cdot \left(\tfrac{k^{l+1}-1}{k-1} \right)
\end{aligned}$$

■

Remark 8.1 (Efficient Storage of a k-ary Tree)

A perfect k-ary tree can be stored in an array, where the x-th child of the node stored in array position i is at position $(k \cdot i) + x$. In terms of raw data, if each node contains $|h|$, then the x-th child of the node stored at offset $i \ \cdot |h|$ is at offset $((k \cdot i) + x)|h|$.

Using a bounded-preimage hash function with a relatively small image will reduce the overall size of the hash tree, but has an impact on the probability of candidate collisions (see Definition 7.3), which affects the candidate reduction process.

The tree's depth and branching factor determine the structure of the tree, which is dependent upon the computational bounds of the tree (see Definition 6.4). In general, the greater the

branching factor of a tree, the smaller its depth, and the less nodes it has. Therefore increasing the branching factor generally reduces the overall size of the hash tree. However, the branching factor of the tree has an impact on the number of internal candidates, thereby affecting the candidate reduction process.

The hash tree must be optimized to meet the preservative size constraint, but adjusting any of the factors that influence the hash tree size will also affect the candidate reduction process.

8.1.2 Determinability

The candidate reduction process must determine a single data block as the original data block. If more than one possibility exists, then one or more counterfeit data blocks exist that the scheme is unable to detect. Unless correction is deterministic, an attacker can determine the entire set of counterfeit data blocks in a reasonable time using a slightly modified resolution algorithm that returns all matching results rather than a *null* result.

To model the candidate reduction process for determinability, functions are defined to quantify both the number of possibilities and the number of candidates at a given level in the hash tree.

Definition 8.2 (Possibility Quantification Function)

The *possibility quantification function* $\alpha{\cdot}i$ determines the number of data block possibilities at a given hash tree level i.

Definition 8.3 (Candidate Quantification Function)

The *candidate quantification function* $\beta{\cdot}i$ determines the number of data block candidates at a given hash tree level i.

Definition 8.3 is used to define the determinability constraint, effectively stating that for a scheme to be deterministic, at most one candidate can filter through the tree (see Definition 8.4).

Definition 8.4 (Determinability Constraint)

The constraint that limits the number of candidates determined by the hash tree such that:

$$\beta \cdot 0 \leq 1 \tag{8.3}$$

The candidate reduction model can be constructed using Definition 8.2 and Definition 8.3 to formulate the number of possibilities and candidates corresponding to each position in the hash tree. Any two nodes at the same level in the tree will have an identical number of possibilities and candidates, since the tree is assumed to be a perfect k-ary tree. The candidate reduction model is illustrated in Figure 8.1, where the candidates are indicated in the top part of the node and the possibilities indicated in the bottom part.

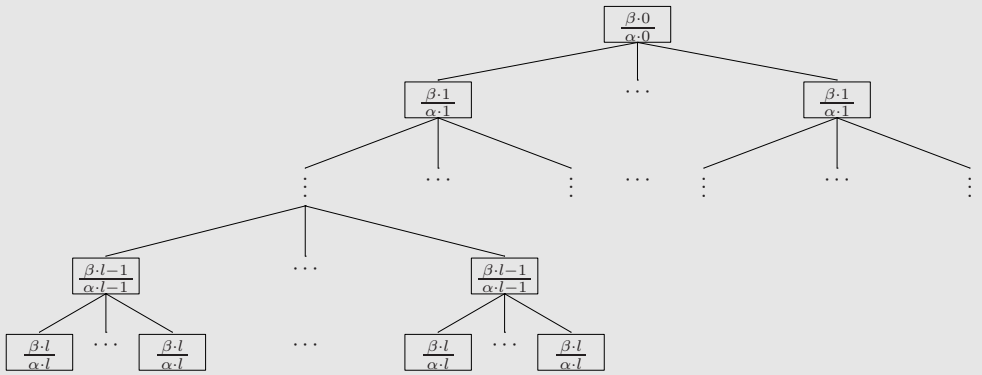

Figure 8.1: Candidate Reduction Model

Two axioms can be determined from the model: the *reduction axiom* and the *enlargement axiom*. The reduction axiom concerns any single node, for which the number of candidates is equal to the product of the number of possibilities and the probability of a hash collision (see Definition 8.5).

Definition 8.5 (Reduction Axiom)

Any single node in the candidate reduction model with x possibilities and a hash collision probability of u has $u \cdot x$ candidates.

$$\beta \cdot i = u \cdot \alpha \cdot i \tag{8.4}$$

The enlargement axiom concerns a parent node and its children, for which the number of possibilities for the parent is equal to the product of the number of candidates for each child. The property is simplified by all parents having k children, and siblings having an equal number of candidates (see Definition 8.6).

Definition 8.6 (Enlargement Axiom)

Any single parent node in the candidate reduction model with k children, each having x candidates, has x^k possibilities.

$$\alpha \cdot i = (\beta \cdot (i+1))^k \tag{8.5}$$

A combination of the two axioms can determine an iterative function for candidate reduction, which can be used to determine the condition required for a reduction in the total number of possibilities (see Theorem 8.2).

Theorem 8.2 (Reduction Theorem)

In order for the number of possibilities to be reduced at any stage the following must hold:

$$\frac{k-1}{k} < \frac{|h|}{|d|} \tag{8.6}$$

Proof

$\alpha \cdot (i-1) < \alpha \cdot i$

$=$ {Definition of α}

$(\beta \cdot i)^k < \alpha \cdot i$

$=$ {Definition of β}

$(\alpha \cdot i \cdot 2^{-|h|})^k < \alpha \cdot i$

$=$ {Definition of α at leaf with bounding $|d|$}

$(2^{|d|} \cdot 2^{-|h|})^k < 2^{|d|}$

$=$ {Indices laws}

$(2^{|d|-|h|})^k < 2^{|d|}$

$=$ {Indices laws}

$2^{k(|d|-|h|)} < 2^{|d|}$

$=$ {Application of logarithm function is monotonic}

$\log 2^{k(|d|-|h|)} < \log 2^{|d|}$

$=$ {Logarithm power law}

$k(|d|-|h|) \cdot \log 2 < |d| \cdot \log 2$

$=$ {Multiplication of $\frac{1}{\log 2}$ is monotonic as $\frac{1}{\log 2} > 0$}

$k(|d|-|h|) < |d|$

$=$ {Monotonicity of addition}

$k|d| - |d| < k|h|$

$=$ {Factorization}

$(k-1)|d| < k|h|$

$=$ {Monotonicity of multiplication}

$\frac{k-1}{k} < \frac{|h|}{|d|}$ ■

Furthermore, the iterative function can also be used to determine the number of candidates at the root node (see Theorem 8.3) from the number of possibilities at a leaf node (see Lemma 8.1).

Lemma 8.1 (Leaf Node Possibilities)

The number of data block possibilities at a single leaf in a perfect tree at depth l is given by:

$$\alpha \cdot l = 2^{\left(\frac{n}{k^l}\right)}. \tag{8.7}$$

Proof
By induction:

For the base case, let $l := 0$ and note that this is a string of length n in base 2.

$$\begin{aligned}
& \alpha \cdot 0 \\
= & \quad \{\text{Definition of } \alpha\} \\
& 2^{\left(\frac{n}{k^0}\right)} \\
= & \quad \{\text{Indices laws}\} \\
& 2^{\left(\frac{n}{1}\right)} \\
= & \quad \{\text{Simplification}\} \\
& 2^n
\end{aligned}$$

For the inductive case, assume $\alpha \cdot l = 2^{\left(\frac{n}{k^l}\right)}$. At depth $l+1$ the size of a single leaf decreases by a factor of k to $\frac{n}{k^{l+1}}$. Therefore the number of possibilities decreases to $2^{\left(\frac{n}{k^{l+1}}\right)}$ (i.e. by a factor of $2^{\left(\frac{n}{k^{l+1}} - \frac{n}{k^l}\right)}$).

$$\begin{aligned}
& (\alpha \cdot l) \cdot 2^{\left(\frac{n}{k^{l+1}} - \frac{n}{k^l}\right)} \\
= & \quad \{\text{Definition of } \alpha\} \\
& 2^{\left(\frac{n}{k^l}\right)} \cdot 2^{\left(\frac{n}{k^{l+1}} - \frac{n}{k^l}\right)} \\
= & \quad \{\text{Indices laws}\} \\
& 2^{\left(\frac{n}{k^{l+1}}\right)} \\
= & \quad \{\text{Definition of } \alpha\} \\
& \alpha \cdot (l+1)
\end{aligned}$$

■

Theorem 8.3 (Root Node Candidates)

The number of candidates at the root node in a perfect k-ary tree of depth l is given by:

$$\beta \cdot 0 = 2^{n-\left(|h| \cdot \left(\frac{k^{l+1}-1}{k-1}\right)\right)}. \tag{8.8}$$

Proof

$$\begin{aligned}
& \beta \cdot 0 \\
= & \quad \{\text{Combination of Definition 8.5 and Definition 8.6}\} \\
& u \cdot (\beta \cdot 1)^k \\
= & \quad \{\text{Combination of Definition 8.5 and Definition 8.6}\} \\
& u \cdot (u \cdot (\beta \cdot 2)^k)^k \\
= & \quad \{\text{Combination of Definition 8.5 and Definition 8.6}\} \\
& \underbrace{u \cdot (\ldots u \cdot (}_{l} \beta \cdot l \underbrace{)^k \ldots)^k}_{l} \\
= & \quad \{\text{Indices laws}\} \\
& u \cdot u^k \cdot u^{k^2} \cdot \ldots \cdot u^{k^{l-1}} \cdot (\beta \cdot l)^{k^l} \\
= & \quad \{\text{Definition 8.5}\} \\
& u \cdot u^k \cdot u^{k^2} \cdot \ldots \cdot u^{k^{l-1}} \cdot (u \cdot \alpha \cdot l)^{k^l} \\
= & \quad \{\text{Indices laws}\} \\
& u \cdot u^k \cdot u^{k^2} \cdot \ldots \cdot u^{k^{l-1}} \cdot u^{k^l} \cdot (\alpha \cdot l)^{k^l} \\
= & \quad \{\text{Indices laws}\} \\
& u^{\left(1+k+k^2+\ldots+k^{l-1}+k^l\right)} \cdot (\alpha \cdot l)^{k^l} \\
= & \quad \{\text{Geometric series}\} \\
& u^{\left(\frac{k^{l+1}-1}{k-1}\right)} \cdot (\alpha \cdot l)^{k^l} \\
= & \quad \{\text{Equation 7.1 and Lemma 8.1}\} \\
& \left(2^{-|h|}\right)^{\left(\frac{k^{l+1}-1}{k-1}\right)} \cdot \left(2^{\left(\frac{n}{k^l}\right)}\right)^{k^l} \\
= & \quad \{\text{Indices laws}\} \\
& 2^{-|h| \cdot \left(\frac{k^{l+1}-1}{k-1}\right)} \cdot 2^n \\
= & \quad \{\text{Indices Laws}\} \\
& 2^{n-\left(|h| \cdot \left(\frac{k^{l+1}-1}{k-1}\right)\right)}
\end{aligned}$$

■

The number of possibilities for a single leaf ($\alpha \cdot l$) is directly related to the size of a bounded data block, which is a function of the tree's branching factor, its depth, and the size of the overall data size (see Table 6.2). To reduce the number of leaf possibilities without reducing the data size, the tree structure must be adjusted by increasing either the branching factor or the depth. In either case, this affects the size of the hash tree.

In the uniform distribution model, increasing the size of the hash image is the only method to reduce the hash collision probability. Therefore the size of the hash tree will be affected by reducing the hash collision probability. The hash tree must be optimized to meet the determinability constraint, but adjusting any of the factors that influence the candidate reduction process will also affect the hash tree size.

8.1.3 Time Complexity

A further constraint on multi-layered document integrity is for resolution to be achieved within a reasonable time period. In the context of this thesis, a reasonable time period is defined as polynomial time (see Definition 8.7).

Definition 8.7 (Time Complexity Constraint)

The constraint that limits the time complexity of a resolution algorithm R such that:

$$R \in O(n^c), \tag{8.9}$$

from some constant $c > 1$.

8.1.4 Security

In Part II, precise definitions for the integrity detection, location, and correction problems were given. These definitions can be used to determine the security of MLDI schemes as shown in §4.3. Since MLDI does not provide any obvious advantages over the existing schemes for tamper detection (i.e. hash-only and signature-based IDSs) or for tamper location in the IP model (i.e. Merkle hash trees), a security constraint is defined in terms of MLDI schemes providing security against the integrity correction problems (see Definition 8.8).

Definition 8.8 (Security Constraint)

The minimum security requirement for a multi-layered document integrity scheme R in the IP model is that R provides security against the data integrity correction problem for all original data d and tampered data f, where $p ::= \varpi \cdot d$.

$$\langle\, \forall d, f \mid :\ X \cdot (d, f, p, R) \,\rangle$$

8.2 Efficacy

In order for MLDI to be effective, the following constraints must be satisfied:

- Preservative size constraint (see Definition 8.1);
- Determinability constraint (see Definition 8.4);
- Time complexity constraint (see Definition 8.7);
- Security constraint (see Definition 8.8).

The first two of these constraints have a direct impact upon each other, whilst the other two are more weakly related in that the time complexity of correction is linked to the time complexity of constructing an attack.

8.2.1 Preservative Size Versus Determinability

As previously mentioned, reducing the preservative size will affect the candidate reduction process (i.e. the determinability) and increasing the determinability will affect the preservative size. In fact, Theorem 8.4 states that the preservative size and determinability constraints contradict.

Theorem 8.4 (Preservative Size and Determinability Constraints Contradict)

Multi-layered document integrity is unable to determine the original data under the constraint on the preservative size.

Proof

The constraints on the preservative size (see Definition 8.1) and the determinability (see Definition 8.4) contradict.

$$
\begin{aligned}
& [|p| < |d|] \wedge [\beta \cdot 0 \leq 1] \\
= \quad & \{\text{Equation 8.2, Equation 8.8}\} \\
& \left[|h| \cdot \left(\tfrac{k^{l+1}-1}{k-1}\right) < |d|\right] \wedge \left[2^{n-\left(|h| \cdot \left(\frac{k^{l+1}-1}{k-1}\right)\right)} \leq 1\right] \\
= \quad & \{\text{Multiplication by } 2^{|h| \cdot \left(\frac{k^{l+1}-1}{k-1}\right)} \text{ is monotonic}\} \\
& \left[|h| \cdot \left(\tfrac{k^{l+1}-1}{k-1}\right) < |d|\right] \wedge \left[2^{n} \leq 2^{|h| \cdot \left(\frac{k^{l+1}-1}{k-1}\right)}\right] \\
= \quad & \{\text{Logarithm function is monotonic}\} \\
& \left[|h| \cdot \left(\tfrac{k^{l+1}-1}{k-1}\right) < |d|\right] \wedge \left[\log\left(2^{n}\right) \leq \log\left(2^{|h| \cdot \left(\frac{k^{l+1}-1}{k-1}\right)}\right)\right] \\
= \quad & \{\text{Logarithm power law}\} \\
& \left[|h| \cdot \left(\tfrac{k^{l+1}-1}{k-1}\right) < |d|\right] \wedge \left[n \cdot \log 2 \leq |h| \cdot \left(\tfrac{k^{l+1}-1}{k-1}\right) \cdot \log 2\right] \\
= \quad & \{\text{Multiplication by } \tfrac{1}{\log 2} \text{ is monotonic}\} \\
& \left[|h| \cdot \left(\tfrac{k^{l+1}-1}{k-1}\right) < |d|\right] \wedge \left[n \leq |h| \cdot \left(\tfrac{k^{l+1}-1}{k-1}\right)\right] \\
= \quad & \{n = |d|, \text{ let } x \ ::= \ |h| \cdot \left(\tfrac{k^{l+1}-1}{k-1}\right)\} \\
& [x < n] \wedge [n \leq x] \\
= \quad & \{\text{Contradiction}\} \\
& \mathit{false}
\end{aligned}
$$

■

Theorem 8.4 is based on the underlying fine detail of MLDI schemes, and the same result can be shown more trivially from an information-theoretic approach. The preservative can be considered as a compression of the data, such that

$$p \ ::= \ K(d), \tag{8.10}$$

since all the information in the data must be recoverable from the preservative alone (i.e. deterministic) when the adversary is arbitrary, malicious, and persistent.

From the definition of the IP model for exposure, the information in the preservative must be less than the information in the data, such that

$$K(p) < K(d),$$

which derives to the contradiction when Equation 8.10 is substituted:

$$K(d) < K(d).$$

Therefore, if the data is vulnerable to tampering, then recovery of the original data cannot be deterministic. Irrespective of this rather damning result, it might be useful to determine whether the other constraints can be met.

8.2.2 Time Complexity

The resolution process can be categorized in different ways, since the degree of tampering affects its performance and the process itself consists of three sub-processes. For example, if no data blocks are modified, then the detection process is $O(1)$ and the other processes are unnecessary.

The algorithm for determining leaf node candidates (see Algorithm 7.3) involves an exhaustive search, which typically has exponential time complexity. However, the use of bounded data blocks allows a single block to be searched in constant time with 2^n comparisons, which is independent of the input size n. In the case where one data block contains tampering, prior tamper location ensures that only a single data block requires an exhaustive search; if all were tampered with, the number of searches is proportional to the input size.

Having performed the exhaustive searching at the leaf nodes, the remaining candidates need to be filtered out through the depth of the tree, which is logarithmic in respect to the length of the input data. Table 8.1 shows the time complexities for the Ashman-style and hybrid resolution algorithms in the case where a single data block is tampered with and all data blocks are tampered with. The time complexities are based on the assumption that no counterfeits exist, and that the hash functions operate in constant time, as is typical for most hash functions.

Process	**Tampered Blocks**	**Ashman-style Resolution**	**Hybrid Resolution**
Detect	1	$O(1)$	$O(1)$
	all	$O(1)$	$O(1)$
Locate	1	$O(\log_k n)$	$O(n)$
	all	$O(n)$	$O(n)$
Correct	1	$O(\log_k n)$	$O(n \log_k n)$
	all	$O(n \log_k n)$	$O(n \log_k n)$
Overall	1	$O(\log_k n)$	$O(n \log_k n)$
	all	$O(n \log_k n)$	$O(n \log_k n)$

Table 8.1: Time Complexities for Algorithms in the Probabilistic Model

The overall time complexities of both hybrid resolution algorithms are linearithmic, and therefore within the time complexity constraint of polynomial time.

8.2.3 Security

The Ashman-style resolution algorithm (see Algorithm 7.8) aims to provide secure tamper detection through cryptographic hashing at the root node. Its tamper detection security is equivalent to that of a hash-only IDS, and therefore considered secure under the same assumptions as Claim 5.6.

One might question whether the additional information in the hash tree (i.e. the non-root nodes) provide extra information to an adversary, rendering the tamper detection security of MLDI less than that of a hash-only IDS. In terms of preimage resistance, an attacker can determine what the input was more easily from the hash tree; after all, that is the purpose of the hash tree in tamper correction. Furthermore, privacy of the original data is beyond the scope of MLDI (see §1.2). However, in terms of collision resistance and second-preimage resistance, this is left as an open question. From one perspective, the attacker has more information to utilize than with a hash-only IDS, but from another perspective, this information reduces the total number of possibilities: If collisions were evenly distributed, then this process would impede the adversary's ability to find collisions. Furthermore, the nature of such attacks has no obvious correlation to the design of hash functions, making it increasingly unlikely to be successful.

The tree of hashes corresponding to data blocks aims to provide secure tamper location. However, cryptographic hashing is only used at the root node, below which, bounded preimage hashing is used. This weaker form of hashing allows an adversary to construct a counterfeit leaf node attack in reasonable time using an algorithm to determine a list of candidates (see Algorithm 8.1).

Algorithm 8.1 (Counterfeit)

Given exposed data and its corresponding original preservative, the following algorithm returns a list of candidates for the original data.

Constants: b, k
begin `Counterfeit`(*data*, p)
  node := `Autobound`(*data*)
  return `Correct`(*node*, p)
end

Counterfeits must exist due to the pigeonhole principle and the preservative size constraint §7.1. The algorithm allows an adversary to determine a list of *all* candidates in the same time it takes the resolver to perform correction (when all nodes were tampered with); and less time when only a single counterfeit is required. As the Ashman-style resolution algorithm cannot locate counterfeits, it is unable to correct them, since correction relies on successful location.

The hybrid resolution algorithm solves the detection problem identically to the Ashman-style algorithm. In the case of location, however, it does not rely on indications from bounded-preimage hashing to locate tampered data blocks: The algorithm effectively skips the location process, assuming all blocks to contain tampering. The algorithm will compute all possible candidates for every node and, assuming the determinability constraint is met, reduces the can-

didates to determine the original data. After the initial detection process, the hybrid algorithm only uses the original preservative to reconstruct the original data, so correction is achieved irrespective of any attack performed on the data.

It is debatable as to whether the hybrid algorithm is tamper location secure, since the obvious location process is omitted. However, the fact that the hybrid algorithm solves the correction problem implies that the location problem is also solved (via the problem hierarchy model): Having determined the original data through correction, a simple block-wise comparison with the exposed data identifies the modified blocks.

Chapter Summary

Multi-layered document integrity in general is subject to four types of constraint: an upper-bound on the preservative size; an upper-bound on the number of root node candidates; an upper-bound on the time complexity of resolution; and a lower-bound on the security.

The preservative size has been shown to be equivalent to the size of the hash tree and, aside from the size of the data, is largely dependent on the preimage size and image size of the bounded-preimage hash function. The number of root node candidates (i.e. the determinability) has also been shown to be dependent on the same factors, resulting in a contradiction when attempting to meet both constraints - the problem is unsatisfiable.

Assuming that tampering has occurred to some extent, the time complexity of resolution has been shown to be $O(\log_k n)$ for the Ashman-style algorithm in the best case ($O(n \log_k n)$ for the hybrid algorithm) and $O(n \log_k n)$ for the worst case (for both algorithms); and therefore meets the constraint of being within polynomial time.

The security constraint dictated that correction must be solvable (as defined in Part II of the thesis) to improve upon existing research. It has been shown that the Ashman-style algorithm does not achieve this, since an adversary can determine counterfeit data blocks, which the algorithm fails to locate. Moreover, the adversary can construct this form of attack in reasonable time using a modified resolution algorithm. However, at the cost of computation time, counterfeit attacks are located and corrected by the hybrid algorithm; although extra computational time does not affect the overall time complexity.

If all leaf node elements were tampered with then the performance of the collision prone resolution algorithm has been shown to be similar to that of the collision resistant resolution algorithm. However, the performance of the collision prone resolution algorithm is independent of the number of tampered leaf node elements, so will perform similarly even when no leaf node elements are tampered with.

Part IV

Conclusion

Great is the art of beginning, but greater the art is of ending;

— Henry Wadsworth Longfellow, *Elegiac Verse* (1882)

Chapter 9

Conclusions

Compromise is but the sacrifice of one right or good in the hope of retaining another — too often ending in the loss of both.

— Tryon Edwards [*attributed in The New Dictionary of Thoughts* (1963)]

An overview of the outcomes of this research is presented here, beginning with a summary of the previous chapters, outlining some of the important definitions and significant results. The contributions of this work are stated, and the potential for future work is discussed. The thesis concludes with a discussion of the work and its general results.

9.1 Summary

The common consensus that data integrity concerns only the problem of tamper detection has led to spurious claims of systems that can guarantee tamper-free data. Furthermore, this view of data integrity does not encapsulate the true meaning of integrity, where maintaining the truth is equally as important as determining it. The idea of multi-layered document integrity postulated by Ashman aims to provide a level of data integrity beyond that of detection, thus furthering the need for a more general notion of the problem.

This thesis aimed to conceptualize the general data integrity problem, and determine whether multi-layered document integrity can provide a feasible solution. The aims and objectives were discussed in detail, and the scope of this thesis was clearly defined; differentiating the focus from a variety of schemes involved in the processes of data modification. The main contributions of this thesis were outlined and its significance was discussed. A description of the various notational styles used throughout this thesis were provided for the less-obvious conventions used in presenting this research.

Various methods of addressing modification in data were discussed through a series of problems, describing the general or widely-accepted solution of each. Whilst revision control and error detection/correction schemes provide useful methods of monitoring changes, neither are suitable in the presence of a malicious adversary. The problems that followed, introduced the common cryptographic primitives used for (and related to) solving parts of the integrity problem.

Ashman's idea of multi-layered document integrity was discussed as a solution to the tamper correction problem (as well as detection and location). However, several problems were noted within MLDI that had to be resolved to develop it into a concrete algorithm that can be

implemented. The author believed that the most fundamental problem is the lack of a solid theoretical foundation, preventing the idea and its algorithm being described clearly. Despite the problems with MLDI, it seemed to be a reasonable approach to solving the tamper correction problem, and therefore deserved further development.

The fundamentals of the data integrity problem were discussed by defining a model for each of its main aspects. The discussion was focussed around three areas of the problem: the participants, the data and the processes.

The participants were discussed in terms of the attacking and defending parties and their opposing goals. The role that each participant plays in achieving their party's goal was then defined, and the idea of a trusted third-party used in the security model was introduced. The abilities of the attacking party were elaborated on by discussing the adversary model of this work and a powerful adversary with the ability for unrestricted modification to data exposed via a vulnerable medium was defined. Alongside this vulnerable medium, an invulnerable medium was defined, where the adversary has no ability to modify the exposed data.

The data model was defined in terms of a partitionable data structure, which provides the ability to locate modifications to specific data blocks, rather than as a whole. Subsequently, the idea of a preservative was discussed, which is determined from the data prior to exposure. The preservative can be used to resolve any data integrity problems after exposure, thereby helping to preserve the data's integrity.

The final area of the problem concerned its processes. The transaction model further defined some aspects of the participants' abilities and gave the definition of a medium. Then the different types of media were discussed, including the use of the term *exposure* in place of *store* and *transmit*, and the use of *vulnerable* and *invulnerable* media as respective synonyms for *tamper-vulnerable* and *tamper-invulnerable* media. Various process models were discussed, which help to illustrate the problem of integrity preservation by classifying the various stages of the problem, and defining the participants and data involved at each stage.

A taxonomy for the data integrity problem was presented by dividing it into four distinct sub-problems. A formal problem semantics has been defined for each sub-problem in terms of its behaviour and solvability functions. The behaviour function specifies how an algorithm for solving the sub-problem should behave in respect to its input and output. The solvability function specifies whether the output given by such an algorithm is correct. These two specification methods allow each sub-problem to be well-defined and differentiated.

A security model for the data integrity problem was given in the form of the general tampering game. This general model can be used to construct the security model for each of the sub-problems. Each sub-problem's security model was based on the solvability function for that sub-problem, and can be used to determine the security of a given algorithm within that context.

Having defined each distinct sub-problem, the hierarchical relationship between these problems was determined. This showed that solving any sub-problem in the hierarchy is essentially equivalent to solving all sub-problems lower in the hierarchy. Therefore solving the highest possible sub-problem, should be the goal of the defending party.

Using the previously defined model, the solvability of each sub-problem was determined for both the VP and IP models. It was shown that detection is the only solvable sub-problem in the

VP model, which is essentially a result of the information required to solve each problem versus the abilities of the adversary in manipulating information. The difference between package and data-only integrity detection was highlighted, with non-existence of the latter being proven. In the VP model it was shown that prevention cannot be achieved under the defined adversary model: Instead, the possibility of correction was discussed, which led to a further discussion on restricting the size of the preservative and that correction cannot be deterministic.

The solvability results show that integrity correction in the IP model is the best result possible, and that a scheme for solving this problem will be probabilistic at best. This probabilistic method is the idea behind Ashman's MLDI and became the focus of Part III, which introduced the fundamental principles of multi-layered document integrity. In particular, the central idea concerning the divide-and-conquer approach to location and correction was discussed; including the notion of bounding the data and structuring it as a perfect k-ary tree to compute a hash tree as a preservative.

The deterministic model presented multi-layered document integrity in a simplified manner, introducing generic algorithms for preservation and resolution, and using them to build a deterministic solution. The time complexities of deterministic MLDI indicated that it could provide a feasible solution to the data integrity problem in terms of computational complexity. Despite the favourable time complexities, the hash tree size was not considered in the non-compressive deterministic model, and therefore contravened the previous result that correction must be non-deterministic. As such, the use perfect one-way hashing to provide a simplified deterministic model had to be readdressed.

Based on the assumption that collisions exist, the probabilistic nature of multi-layered document integrity was addressed by the introduction of two opposing collision models. These models differ in their assumption regarding the occurrence of collisions. The collision resistant model assumes that collisions do not occur, despite their existence. Therefore an attacker is unable to substitute data blocks which remain undetected and/or cannot be located. The collision prone model assumes that collisions do occur and, in the extreme case, every node must be corrected to ensure substituted data blocks are corrected. Ultimately this means ignoring the indicators for detection and location. A compromise solution that is both secure and efficient, was presented in the form of the hybrid model, which combines cryptographic hashing at the root node with bounded-preimage hashing elsewhere.

The hash collision probability was defined, along with the notion of a node having multiple candidates when hashes collide. The idea of the candidate reduction process was described, including the potential for an increase in candidates when sibling nodes in the hash tree contain tampered blocks. To realize the invertible hashing described by Ashman, this thesis introduced the concept of practically bounding the preimage of the hash function. A side-effect of bounded-preimage hashing is a large increase in the collision probability. The higher collision probability makes the hash tree more susceptible to collisions, at both single and multiple levels in the tree.

Collisions are not only a problem from the point of view of determining the original data, but also from a security aspect. Various tampering attacks were introduced, which can be classed as overt or covert, dependent on the hash tree's ability to indicate them. Two adversarial aims were discussed for attacking MLDI schemes: those which address the data integrity problem directly, and those which attempt to make resolution less efficient. It is suggested, therefore,

that the security of MLDI should be considered from two perspectives: whether the algorithm can resolve the integrity problems; and, if so, whether the problems can be resolved in a reasonable time period.

The four types of constraint on the general idea of multi-layered document integrity were defined as: an upper-bound on the preservative size; an upper-bound on the number of root node candidates; an upper-bound on the time complexity of resolution; and a lower-bound on the security.

The preservative size was shown to be equivalent to the size of the hash tree and, aside from the size of the data, is largely dependent on the preimage size and image size of the bounded-preimage hash function. The number of root node candidates (i.e. the determinability) was also shown to be dependent on the same factors, resulting in a contradiction when attempting to meet both constraints - the problem was determined to be unsatisfiable.

Assuming that tampering has occurred to some extent, the time complexity of resolution was shown to be $O(\log_k n)$ for the Ashman-style algorithm in the best case ($O(n \log_k n)$ for the hybrid algorithm) and $O(n \log_k n)$ for the worst case (for both algorithms); and therefore meets the constraint of being within polynomial time.

The security constraint dictated that correction must be solvable (as defined in Part II of the thesis) to improve upon existing research. It was shown that the Ashman-style algorithm does not achieve this, since an adversary can determine counterfeit data blocks, which the algorithm fails to locate. Moreover, the adversary can construct this form of attack in reasonable time using a modified resolution algorithm. However, at the cost of computation time, counterfeit attacks are located and corrected by the hybrid algorithm; although extra computational time does not affect the overall time complexity.

If all leaf node elements were tampered with then the performance of the collision prone resolution algorithm was shown to be similar to that of the collision resistant resolution algorithm. However, the performance of the collision prone resolution algorithm is independent of the number of tampered leaf node elements, so will perform similarly even when no leaf node elements are tampered with.

9.2 Contributions

The main contributions of this research lie in two areas of data integrity, as reflected by the title and structure of the thesis:

- Part II introduces the first formal model for the generalized data integrity problem (as far as the author is aware).
- Part III provides a detailed discussion of, and underlying model for, Ashman's concept of multi-layered document integrity.

Aside from the literature review, the contributions from the two models can be defined in terms of their propositions, corresponding proofs, and related original definitions. The most significant contributions are:

- **A general review of the literature concerning the modification of data**: This review defined several well-known problems concerning data modification, categorizing the

modification in terms of two factors: accidental versus deliberate; and authorized versus unathorized. The generally accepted solution to each problem was discussed, and the focus became the overlap of deliberate and unauthorized modification — data integrity. After differentiating MLDI from existing schemes for solving the detection and location problems, Ashman's original MLDI paper was discussed in depth.

- **A formal hierarchical definition of data integrity problem**: By defining each of the data integrity sub-problems with a formal problem semantics, the relationship between each sub-problem could be determined. This relationship was proven to be a hierarchy, in which the solution to any sub-problem can be trivially reduced to solve sub-problems lower in the hierarchy. Furthermore, the non-existence of a solution to any sub-problem implies the non-existence of solutions higher in the hierarchy.

- **Establishing that detection is the only solvable problem in the vulnerable preservative model**: In the VP model, it was shown that signature-based integrity detection schemes provide a solution to the detection problem, but that the location problem is unsolvable, as were correction and prevention from the problem hierarchy.

- **Establishing the non-existence of data-only integrity detection**: The idea of data-only integrity was defined, which concerns the integrity of an atomic piece of information, unreliant on a preservative. It was shown that no scheme can determine the integrity of data alone; it is only possible to determine integrity of data packaged with a preservative.

- **Establishing an underlying model for multi-layered document integrity**: This model allowed a clear discussion of the issues surrounding the efficacy of MLDI and how these issues can be resolved.

- **A well-defined set of algorithms for preservation and resolution in MLDI**: The algorithms have been decomposed into several fundamental functions, which were presented alongside a detailed discussion and justification of their design.

- **Outlining several feasible attacks against MLDI algorithms**: The attacks outline some of the security threats to MLDI schemes and serve as an aid in the design of algorithms.

- **Establishing the efficacy of multi-layered document integrity**: The four constraints on MLDI were formally defined, and used to determine its efficacy. It was shown that Ashman's original MLDI algorithm is not secure against problems other than detection, since it cannot locate counterfeit nodes that an adversary can determine in a reasonable amount of time. Moreover, it was determined that the constraints of preservative size and determinability contradict, rendering the general concept of MLDI ineffective.

- **Establishing that detection is the only solvable problem in the invulnerable preservative model**: The main result concerning the efficacy of MLDI was generalized from an information-theoretic point of view, to establish that detection is the only solvable data integrity sub-problem, irrespective of the preservative exposure model.

9.3 Future Work

The results of this thesis are relatively conclusive in that little scope remains for incremental research. Perhaps the most significant area of future work concerns extending the problem model to cover schemes that preserve integrity through the use of data distribution. The result concerning the efficacy of MLDI, seriously limits its capacity for future research, and the work of this thesis suggests that future work on MLDI is unlikely to have any significant impact on data integrity research. Despite this, some of the more obvious research problems posed by this study, may have an impact in other research fields.

9.3.1 Optimization of the Resolution Algorithm

One of the more obvious areas of incremental research surrounding MLDI is to design and implement more efficient resolution algorithms, whilst maintaining security against counterfeits. The two resolution algorithms presented in §7.1.3 serve as complementary best and worst cases: the Ashman-style algorithm being efficient, but insecure; and the hybrid algorithm being secure, but inefficient.

The correction process of resolution involves bounded exhaustive searches performed at every tampered leaf node. These searches are unnecessarily costly for two reasons: all possibilities within the bound must be hashed; and an identical search and corresponding hashing is repeated for each tampered leaf node. Using a pre-computed lookup table for fixed bounding size and indexed by hash image would drastically improve the time taken for correction at the cost of some disk space [39, 27]. The financial feasibility of using purpose-built hardware with a built-in lookup table could also be investigated.

Alternatively, one or more lookup tables for various bounding sizes might be both computed and stored in a distributed manner to reduce the initial computation time and overall storage space. In this situation, a user would request the candidates for a data block by submitting the corresponding hash from the hash tree. The candidate list can be determined in constant time from a lookup in the distributed lookup table, removing the need for a costly exhaustive search at the leaf node.

The location process of resolution is subject to so-called covert attacks that involve counterfeit nodes. When counterfeit nodes are present, especially in the case of multi-level counterfeits, it becomes increasingly difficult to identify which parts of the data contain tampered blocks. The problem is essentially a tree searching problem, where a strategy is required in determining which nodes to expand and continue searching.

An optimized location algorithm could be designed by employing search heuristics, which prioritize the set of nodes that can be expanded after each step of location, with an estimate of whether a sub-tree contains tampering. Possible considerations for optimizing location are:

- Number of data blocks: If one sub-tree contains more data blocks than another, then it might be more prone to tampering since it potentially contains more candidates. However, the converse might be true, since the sub-tree with fewer data blocks requires less time for an adversary to construct a brute-force attack against it.

- Number of processed data blocks: As in the previous case, but subtrees containing verified or, at least, processed (i.e. where some/all possible candidates have been determined) should be less prone to tampering.
- Number of processed levels: A sub-tree with i matching ancestor nodes should be more likely to contain tampering than a sub-tree with $i + 1$ matching ancestor nodes.

Interesting points of this problem are that any number of data blocks can contain tampering and all must be determined to constitute a solution. It should be possible to determine whether the problem has been solved at any point in the search, by comparison of the root hash, which is assumed to be collision resistant.

9.3.2 Bounded-preimage Hash Functions

The concept of a weakened invertible hash function was originally described by Ashman [1], and defined more formally in this thesis as a bounded-preimage hash function (see Definition 7.4). If schemes for MLDI were implemented, then such functions would need to be designed. The design and analysis of bounded-preimage hash functions would constitute a new area of research.

Under supervision of the author, the M.Sc. thesis of Drakoulis carried out initial work in this area [17]; investigating several designs of bounded-preimage hash functions based around the MD5 and SHA-1 cryptographic hash functions. The central idea was to first compute a cryptographic hash of the bounded data block and then compress the output to form a weaker bounded-preimage hash.

The results of the thesis suggest that cryptographic hash function provide a suitable basis for bounded-preimage hash functions with a reasonably uniform collision frequency. However, it seems feasible that much more efficient algorithms with a similar or more uniform collision frequency can be realized without pre-computing the cryptographic hash.

9.3.3 Distributive Data Integrity Model

This thesis has established that detection is the only solvable problem of the data integrity problem for both the VP and IP exposure models. However, this result is based on the assumption that the data is a single atomic piece of information that cannot be distributed. In reality, it is feasible to make multiple instances of the data, and expose each instance via a different medium, and recent trends in computing have seen the popularization of this methodology for both storage and transmission of data.

Peer-to-peer schemes such FreeNet [11] and BitTorrent [12] allow users to upload and download files from other users. Furthermore, these schemes suffer from a variety of attacks aiming to compromise data integrity, and various methods have been proposed to secure against these attacks. This area of research is closely related to that of censorship-resistance schemes (see §2.8), and solutions are typically based on the idea of Byzantine fault tolerance [9].

Data distribution-based schemes form a preventive (rather than reactive) approach by assuming that one or more distributed instances of the data remains unchanged. Rather than determining the original data from any possibility (of the given size), it is determined from a

relatively small number of distributed instances, of which one or more could be original. Consequently the preservative only has to contain enough information to solve the detection problem (or multiple instances of the detection problem for partitioned data). As such, the preservative can be much smaller than the data that it is preserving, and the conditions on the IP exposure model are trivially satisfied.

The model proposed in this thesis could be extended to incorporate the idea of distributive data integrity by adjusting the definitions of the underlying models, and the resulting problem taxonomy. The transaction model would use the IP exposure model to obtain an initial preservative and, if necessary, obtain further preservative partitions via the VP model, which then can be verified against the initial one. After preservative partitions have been verified, they can be used to verify the integrity of the corresponding data partitions that are exposed as part of the VP model. Only the initial preservative requires exposure via the IP model, and this preservative can be a fixed-length cryptographic hash.

In addition to multiple partitions in the data, the data model would have to encompass multiple instances for each partition. Furthermore, any number of instances for each partition could have been tampered with, so the adversary model might be required to consider the ratio between tampered and original instances of each data partition. Essentially, the preservative model would be the same as that defined in Chapter 3, despite the preservative being smaller. However the process model would require additional steps for establishing the preservative, and would incorporate multiple sources, as opposed to a single source.

Having extended the underlying model, the problem taxonomy could be adjusted to incorporate these extensions. The changes would stem primarily from the assumption that one or more distributed instances are original, and involve the identification of at least one original instance of the data, albeit from multiple sources. The general notion of the main results will still apply, but some of the definitions and assumptions will change. One of the more obvious changes would be in the problem definitions, as multiple distributed instances must be considered. For example, the prevention problem might be redefined to permit modification in instances, provided at least one instance of each partition remains original. Alternatively, an absolute definition of prevention might stipulate that all instances of the data must remain original.

Extending the data integrity model proposed in this thesis to accommodate distributed exposure models might provide a better understanding of the security issues involved with distributed data integrity. Such work could be useful in determining the feasibility of certain types of attack and/or methods for securing against them.

9.4 Discussion

This thesis has investigated the notion of data integrity beyond tamper detection within the context of non-distributed data exposure and a powerful, but realistic, adversary. To verify Ashman's claim that MLDI can provide location and correction of tampering, the formal model for the general data integrity problem was developed. The model introduced the idea of vulnerable and invulnerable exposure of the preservative, showing that only tamper detection was possible in the former, and the preservative exposure problem applied to the latter.

After defining a model for MLDI and implementing Ashman's idea in concrete algorithms, the four constraints of MLDI were defined. These constraints were shown to be unsatisfiable within the context of the problem model, due to a fundamental result concerning the ability to preserve information in the presence of a powerful adversary. Based on this result, it was determined that MLDI is an ineffective method for preserving data integrity.

One of the arguments for MLDI is that it can use the vulnerable preservative model with the aid of some public-key cryptosystem or digital-signature scheme. However, this can only solve the the detection problem, since the adversary can tamper with the data or preservative in an arbitrary manner. A further argument is that in the invulnerable preservative model, the preservative does not have to be smaller than the data, and it can be compressed, and the compressed preservative will be smaller than the data. However, the relative size of the data and preservative is defined in terms of the Kolmogorov complexity, so any compression is irrelevant.

It is easy to entertain the idea that tamper location is possible if one assumes a less malicious adversary where some, but not all, of the blocks are tampered with. However, MLDI is not suitable for this, since the bounded-preimage hashing employed to provide tamper correction, is the very thing that weakens the idea against solving location. Alternatively, each block might be protected individually using a Signature-based IDS: Location is simply a matter of block-by-block verification of the data. This common misconception stems from the assumption that the data integrity problem is all about detection and that attacks are concerned with producing counterfeits. However, if the defending party adopts a scheme to solve the location problem, then the attacking party is not necessarily concerned with producing counterfeits to attack the scheme, since any method that obstructs the defender's goals constitutes an attack. If Kerckhoffs' Principle is assumed (i.e. the adversary knows the system), then it must be assumed that the attacking party is "playing the same game" as the the defending party.

In conclusion, the formal model for the data integrity problem presented in Part II has been used to prove that the problems of tamper location, correction, and prevention are not solvable in a realistic situation, where the preservative is exposed with the data. Furthermore, the results of Part III showed that even if the preservative is exposed in a tamper-free manner, it is not possible to solve these problems in a realistic situation, due to the preservative exposure problem. However, it is important to note that the results outlined in the list of contributions are valid for the models defined within this thesis. In particular, the results are limited to the context of non-distributed data integrity, but suggested future work described how the models could be extended to incorporate the more realistic notion of distributed data integrity.

Appendix A

Function Properties

The following definitions are given here for the sake of readability, brevity, and completeness in the main text.

Definition A.1 (Easy to Compute)

A function F is *easy to compute* if the algorithm operates in deterministic polynomial time.

Definition A.2 (Hard to Compute)

A function F is *hard to compute* if the algorithm operates in non-deterministic polynomial time.

Definition A.3 (Compressive)

A function F is *compressive* if the (typically fixed-size) output $F{\cdot}x$ is smaller than its input x.

Definition A.4 (Non-compressive)

A function F is *non-compressive* if $|F \cdot m| \geq |m|$.

Definition A.5 (Preimage Resistant)

A function F is *preimage resistant* if given m, the algorithm for finding $F{\cdot}m$ operates in deterministic polynomial time, and given $F{\cdot}m$, the algorithm for finding m operates in non-deterministic polynomial time.

Definition A.6 (Second Preimage Resistant)

A function F is *second preimage resistant* if given the pair $(m_1, F \cdot m_1)$, the algorithm for finding m_2 such that $m_1 \neq m_2$ and $F \cdot m_1 = F \cdot m_2$ operates in non-deterministic polynomial time.

Definition A.7 (Collision Resistant)

A function F is *collision resistant* if the algorithm for finding any pair (m_1, m_2) such that $m_1 \neq m_2$ and $F \cdot m_1 = F \cdot m_2$ operates in non-deterministic polynomial time.

Definition A.8 (Collision Free)

A function F is *collision free* if $\langle\, \forall m_1, m_2 \mid m_1 \neq m_2 :\ F \cdot m_1 \neq F \cdot m_2 \,\rangle$.

Definition A.9 (Self-invertible)

A function F is *self-invertible* if $\langle\, \forall m \in \mathbb{B}^n \mid :\ F \cdot F \cdot m = m \,\rangle$.

Definition A.10 (Invertible)

A function F is *invertible by* G if $\langle\, \forall m \mid :\ G \cdot F \cdot m = m \,\rangle$.

Definition A.11 (Universal Validity)

A digital signature scheme comprising of a private sign function $S_{\overline{\mathcal{A}}}$ and a public validate function $V_{\mathcal{A}}$ has *universal validity* if $\langle\, \forall m \mid :\ V_{\mathcal{A}} \cdot (m, S_{\overline{\mathcal{A}}} \cdot m) \,\rangle$.

Definition A.12 (Forgery Resistant)

A digital signature scheme comprising of a private sign function $S_{\overline{\mathcal{A}}}$ and a public validate function $V_{\mathcal{A}}$ is *forgery resistant* if, given the pair (m_1, s) such that $V_{\mathcal{A}} \cdot (m_1, s)$, the algorithm for finding any pair (m_2, s) such that $m_1 \neq m_2$ and $V_{\mathcal{A}} \cdot (m_2, s)$ operates in non-deterministic polynomial time.

Appendix B

Algorithms

B.1 Preliminaries

Any *constants* are assumed to be publicly-known, but algorithm-independent values that are assigned at the application level. For example, it might be decided that all schemes will use a branching factor of 2 (i.e. $k := 2$).

Three primitive data structures are used: strings, sets and lists. A string is a simple bitstring of with typical length and concatenation operations. When the `.range(off, len)` function is applied to a string it returns the substring that starts at the given offset *off* (inclusive and zero-indexed) and ends at the point when the substring is of the given length of characters *len*. A set with elements $x_0, x_1, \ldots, x_{n-1}$ is denoted $x_0, x_1, \ldots, x_{n-1}$, and a list of the same elements is denoted $[x_0, x_1, \ldots, x_{n-1}]$. The `.add(...)` function is used to add the given element to the specified set or list and, in the case of a list, the element is added to the end. For example, $[x_0, x_1, \ldots, x_{n-1}]$`.add(`$x_n$`)` is $[x_0, x_1, \ldots, x_n]$

The majority of the algorithms are based on tree-like data structures implemented as recursive lists, such that a list's elements might also be lists. Consider the two representations for the encryption key given in Example 3.1:

$$x := \texttt{[10011000101010111010000100101110]}$$

$$y := \texttt{[[10011000]}, \texttt{[10101011]}, \texttt{[10100001]}, \texttt{[00101110]]}$$

Both representations have the same data, but different structures. This is reflected by the function `.data()`, which returns "10011000101010111010000100101110" in either case. However, when an integer argument i is given, the function attempts to return the ith child element (with 1 being the first element), but returns a `null` value if no such child element exists. For example, x`.data(1)` returns the same as x`.data()`, but y`.data(1)` will return "`[10011000]`".

In contrast to the tree structure used for data whereby parent data is defined in terms of its children, the hash tree data structure uses a different indexing system to encompass the fact that a parent hash is not directly derived from its child hashes. The first element of a list used to implement a hash tree is reserved for the current level hash, and followed by elements for its children. For example, the corresponding hash tree for y, might be:

$$y := \texttt{[0110}, \texttt{[11]}, \texttt{[10]}, \texttt{[01]}, \texttt{[01]]}$$

As such, the `.hash()` function would return "`0110`", whereas `.hash(1)` would return "`[11]`". Note that whilst $\#(y)$ can be used to correctly determine the number of child data blocks in block y, it will also count the root hash when applied to a hash tree.

The algorithm descriptions that appear in the main text use abstract functions (denoted with an `X` suffix) to represent functions with definition-specific implementations. For example, the `XHash` function denotes an abstract hash function, which might be implemented as H, $\vec{H}$, $\overleftrightarrow{H}$, etc. depending on the required definition. Furthermore, an abstract function (the parent) that is itself implemented in terms of an abstract function (the child), must provide a definition for both parent and child. For example, in Algorithm 7.4 defining `XResolve` as `Detect` requires that `XHash` is defined as H, since `Detect` is defined in terms of `XHash`.

B.2 Full MLDI Algorithms

Algorithm B.1 (Cartesian Product of n Sets)

The following algorithm is used in the resolution process for which hash collisions are present. The algorithm constructs the set of parent possibilities from the set of k child candidate sets.

```
Constants: k
begin CartesianProduct(candidates)
    possibilities := candidates.data(1)
    if |candidates| > 1 then
        for i = 2 to k do
            list := []
            foreach p ⇐ possibilities do
                foreach c ⇐ candidates.data(i) do
                    list.add(p ‖ c)
                end
            end
            possibilities := list
        end
    return possibilities
end
```

Algorithm B.2 (Full Deterministic Preservation)

Given b, k and input data *data*, its corresponding hash tree is computed by the following algorithm:

```
Constants: b, k
begin Preserve(data)
    node := Autobound(data)
    return PreserveNode(node)
end
begin PreserveNode(node)
    list := []
    list.add(H↔(node.data()))
    for i = 1 to #(node) do
        list.add(PreserveNode(node.data(i)))
    end
    return list
end
```

Algorithm B.3 (Full Deterministic Resolution)

From b and k, given exposed data *data* and its corresponding original preservative p, the original data is computed by the following algorithm:

```
Constants: b, k
begin Resolve(data, p)
    node := Autobound(data)
    results := Detect(node, p)
    if #(results) = 1 then
        return results.data()
    else
        return null
    end
end
begin Detect(node, p)
    if H↔(node.data()) = p.hash() then
        return node
    else
        return Correct(node, p)
    end
end
begin Correct(node, p)
    if #(p) = 1 then
        possibilities := 𝔹^b
    else
        block := null
        for i = 1 to k do
            block := block‖Detect(node.data(i), p.hash(i))
        end
        possibilities := {block}
    end
    foreach possibility ← possibilities do
        if H↔(possibility) = p.hash() then
            return [possibility]
        end
    end
end
```

Algorithm B.4 (Full Collision Resistant Preservation)

Given b, k and input data *data*, its corresponding hash tree is computed by the following algorithm:

```
Constants: b, k
begin Preserve(data)
    node := Autobound(data)
    return PreserveNode(node)
end
begin PreserveNode(node)
    list := []
    list.add(H(node.data()))
    for i = 1 to #(node) do
        list.add(PreserveNode(node.data(i)))
    end
    return list
end
```

Algorithm B.5 (Full Collision Resistant Resolution)

From b and k, given exposed data *data* and its corresponding original preservative p, the original data is computed by the following algorithm:

```
Constants: b, k
begin Resolve(data, p)
    node := Autobound(data)
    results := Detect(node, p)
    if #(results) = 1 then
        return results.data()
    else
        return null
    end
end
begin Detect(node, p)
    if H(node.data()) = p.hash() then
        return node
    else
        return Correct(node, p)
    end
end
begin Correct(node, p)
    if #(p) = 1 then
        possibilities := 𝔹^b
    else
        list := []
        for i = 1 to k do
            list.add( Detect(node.data(i), p.hash(i)))
        end
        possibilities := CartesianProduct(list)
    end
    candidates := []
    foreach possibility ← possibilities do
        if H(possibility) = p.hash() then
            candidates.add(possibility)
        end
    end
    return candidates
end
```

Algorithm B.6 (Full Collision Prone Preservation)

Given b, k and input data *data*, its corresponding hash tree is computed by the following algorithm:

```
Constants: b, k
begin Preserve(data)
    node := Autobound(data)
    return PreserveNode(node)
end
begin PreserveNode(node)
    list := []
    list.add(⃗H(node.data()))
    for i = 1 to #(node) do
        list.add(PreserveNode(node.data(i)))
    end
    return list
end
```

Algorithm B.7 (Full Collision Prone Resolution)

From b and k, given exposed data *data* and its corresponding original preservative p, the original data is computed by the following algorithm:

```
Constants: b, k
begin Resolve(data, p)
    node := Autobound(data)
    results := Correct(node, p)
    if #(results) = 1 then
        return results.data()
    else
        return null
    end
end
begin Correct(node, p)
    if #(p) = 1 then
        possibilities := 𝔹^b
    else
        list := []
        for i = 1 to k do
            list.add(Correct(node.data(i), p.hash(i)))
        end
        possibilities := CartesianProduct(list)
    end
    candidates := []
    foreach possibility ← possibilities do
        if H(possibility) = p.hash() then
            candidates.add(possibility)
        end
    end
    return candidates
end
```

Algorithm B.8 (Full Hybrid Probabilistic Preservation)

Given b and k, the Ashman hash tree for a leaf data block is produced by the following algorithm:

```
Constants: b, k
begin Preserve(data)
    node := Autobound(data)
    return RootPreserveNode(node)
end
begin RootPreserveNode(node)
    list := []
    list.add(H(node.data()))
    for i = 1 to #(node) do
        list.add(PreserveNode(node.data(i)))
    end
    return list
end
begin PreserveNode(node)
    list := []
    list.add(H⃗(node.data()))
    for i = 1 to #(node) do
        list.add(PreserveNode(node.data(i)))
    end
    return list
end
```

Algorithm B.9 (Full Ashman-style Resolution)

From b and k, given exposed data *data* and its corresponding original preservative p, the original data is computed by the following algorithm:

```
Constants: b, k
begin Resolve(data, p)
    node := Autobound(data)
    results := RootDetect(node, p)
    if #(results) = 1 then
        return results.data()
    else
        return null
    end
end
begin RootDetect(node, p)
    if H(node.data()) = p.hash() then
        return node
    else
        return RootCorrect(node, p)
    end
end
begin RootCorrect(node, p)
    if #(p) = 1 then
        possibilities := 𝔹^b
    else
        list := []
        for i = 1 to k do
            list.add( Detect(node.data(i), p.hash(i)))
        end
        possibilities := CartesianProduct(list)
    end
    candidates := []
    foreach possibility ⇽ possibilities do
        if H(possibility) = p.hash() then
            candidates.add(possibility)
        end
    end
    return candidates
end
begin Detect(node, p)
    if H⃗(node.data()) = p.hash() then
        return node
    else
        return Correct(node, p)
    end
end
begin Correct(node, p)
    if #(p) = 1 then
        possibilities := 𝔹^b
    else
        list := []
        for i = 1 to k do
            list.add( Detect(node.data(i), p.hash(i)))
        end
        possibilities := CartesianProduct(list)
    end
    candidates := []
    foreach possibility ⇽ possibilities do
        if H⃗(possibility) = p.hash() then
            candidates.add(possibility)
        end
    end
    return candidates
end
```

Algorithm B.10 (Full Hybrid Resolution)

Given exposed data and its corresponding original preservative, the following algorithm returns the original data.

```
Constants: b, k
begin Resolve(data, p)
    node := Autobound(data)
    results := RootCorrect(node, p)
    if #(results) = 1 then
        return results.data()
    else
        return null
    end
end
begin RootCorrect(node, p)
    if #(p) = 1 then
        possibilities := 𝔹^b
    else
        list := []
        for i = 1 to k do
            list.add( Correct(node.data(i), p.hash(i)))
        end
        possibilities := CartesianProduct(list)
    end
    candidates := []
    foreach possibility ← possibilities do
        if H(possibility) = p.hash() then
            candidates.add(possibility)
        end
    end
    return candidates
end
begin Correct(node, p)
    if #(p) = 1 then
        possibilities := 𝔹^b
    else
        list := []
        for i = 1 to k do
            list.add( Correct(node.data(i), p.hash(i)))
        end
        possibilities := CartesianProduct(list)
    end
    candidates := []
    foreach possibility ← possibilities do
        if H⃗(possibility) = p.hash() then
            candidates.add(possibility)
        end
    end
    return candidates
end
```

References

[1] Ashman H. L. Hashes DO Grow on Trees - Document Integrity at Every Level. In *Sixth Australian World Wide Web Conference (AusWeb 2000)*, Southern Cross University, Cairns, June 2000.

[2] Aspnes J., Feigenbaum J., Yampolskiy A. and Zhong S. Towards a theory of data entanglement. In *Ninth European Symposium on Research in Computer Security*, volume 3193, pages 177–192, Berlin, September 2004. Springer-Verlag.

[3] Bakhtiari S., Safavi-Naini R. and Pieprzyk J. Cryptographic Hash Functions: A Survey. Technical Report 95-09, Department of Computer Science, University of Wollongong, July 1995.

[4] Balusani R. M. S. Active certificates: A new paradigm in digital certificate management. In *2002 International Conference on Parallel Processing Workshops (ICPPW'02)*, page 30, 2002.

[5] Black P. E. *"perfect **k**-ary tree", from Dictionary of Algorithms and DataStructures*. NIST, January 2006.

[6] Black P. E. and Algorithms and Theory of Computation Handbook C. *"Kolmogorov complexity", from Dictionary of Algorithms and Data Structures*. NIST, April 2006.

[7] Black P. E. and Algorithms and Theory of Computation Handbook C. *"tree", from Dictionary of Algorithms and Data Structures*. NIST, January 2006.

[8] Canetti R., Micciancio D. and Reingold O. Perfectly one-way probabilistic hash functions (preliminary version). In *STOC '98: Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 131–140, New York, NY, USA, 1998. ACM Press.

[9] Castro M. and Liskov B. Practical byzantine fault tolerance. In *OSDI '99: Proceedings of the third symposium on Operating systems design and implementation*, pages 173–186, Berkeley, CA, USA, 1999. USENIX Association.

[10] Chor B., Goldwasser S., Micali S. and Awerbuch B. Verifiable secret sharing and achieving simultaneity in the presence of faults. In *Proceedings of the 26th IEEE Conference on the Foundations of Computer Science (FOCS)*, pages 383–395, October 1985.

[11] Clarke I., Hong T. W., Miller S. G., Sandberg O. and Wiley B. Protecting free expression online with freenet. *IEEE Internet Computing*, 6(1):40–49, 2002.

[12] Cohen B. Incentives build robustness in bittorrent. Technical report, bittorrent.org, 2003.

[13] Cormen T. H., Leiserson C. E., Rivest R. L. and Stein C. *Introduction to Algorithms*, chapter 11.5: Perfect hashing, pages 245–249. MIT Press and McGraw-Hill, second edition, 2001.

[14] Deswarte Y., Blain L. and Fabre J.-C. Intrusion Tolerance in Distributed Computing Systems. In *IEEE Symposium on Security and Privacy*, pages 110–121, 1991.

[15] Diffie W. and Hellman M. E. New Directions In Cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, 1976.

[16] Dobbertin H. The Status of MD5 After a Recent Attack. Technical Report 2, RSA Laboratories, Summer 1996.

[17] Drakoulis I. Analysis of tamper-tolerant hash functions. Master's thesis, School of Computer Science and Information Technology, University of Nottingham, UK, September 2006.

[18] Ellison C., Hall C., Milbert R. and Schneier B. Protecting secret keys with personal entropy. *Future Generation Computer Systems*, 16(4):311–318, 2000.

[19] (FIPS) F. I. P. S. Secure Hash Standard (SHS). Technical Report 1, National Institute of Standards and Technology (NIST), 1995.

[20] (FIPS) F. I. P. S. Secure Hash Standard (SHS). Technical Report 2, National Institute of Standards and Technology (NIST), August 2002.

[21] Goldreich O., Micali S. and Wigderson A. Proofs that yield nothing but their validity or all languages in np have zero-knowledge proof systems. *Journal of the ACM*, 38(3):690–728, 1991.

[22] Goldwasser S., Micali S. and Rivest R. L. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308, 1988.

[23] Gollmann D. *Computer Security*. Worldwide Series in Computer Science. John Wiley & Sons Ltd, 1999.

[24] Gries D. and Schneider F. B. *A logical approach to discrete math*. Springer-Verlag New York, Inc., New York, NY, USA, 1993.

[25] Grimaldi R. P. *Discrete and Combinatorial Mathematics: An Applied Introduction*. Addison Wesley, fourth edition, December 1998.

[26] Hamming R. W. Error Detecting and Error Correcting Codes. *The Bell System Technical Journal*, 26(2):147–160, April 1950.

[27] Knuth D. E. *The Art of Computer Programming*, volume 1. Fundamental Algorithms. Addison-Wesley, second edition, 1998.

[28] Manuel B. Coin flipping by telephone a protocol for solving impossible problems. *SIGACT News*, 15(1):23–27, 1983.

[29] Menezes A. J., van Oorschot P. C. and Vanstone S. A. *Handbook of Applied Cryptography*, volume 6 of *Discrete Mathematics and Its Applications*. CRC Press, fifth printing (august 2001) edition, 1996.

[30] Merkle R. *Secrecy, authentication, and public key systems*. PhD thesis, Stanford University, 1979.

[31] Moss B. and Ashman H. Hash-Tree Anti-Tampering Schemes. In *IEEE International Conference on Information Technology and Applications (ICITA 2002)*, Bathurst, Australia, November 2002.

[32] Perng G., Reiter M. K. and Wang C. Censorship Resistance Revisited. In et al. M. B., editor, *Proceedings of Information Hiding Workshop (IH 2005)*, volume 3727 of *Lecture Notes in Computer Science*, pages 62–76, Berlin Heidelberg, June 2005. Springer-Verlag.

[33] Pretzel O. *Error-Correcting Codes and Finite Fields*. Oxford Applied Mathematics and Computing Science. Clarendon Press, August 1992.

[34] Reed I. S. and Solomon G. Polynomial codes over certain finite fields. *SIAM Journal of Applied Math.*, 8(2):300–304, June 1960.

[35] Rivest R. The MD5 Message-Digest Algorithm. IETF RFC 1321, MIT Laboratory for Computer Science and RSA Data Security, Inc., April 1992.

[36] Roskos J. E., Welke S. R., Boone J. and Mayfield T. A taxonomy of integrity models, implementations and mechanisms. In *13th NIST-NCSC National Computer Security Conference*, NCSC National Computer Security Conference, pages 526–540, Washington D.C., October 1990. NIST.

[37] Sandhu R. Terminology, Criteria and System Architectures for Data Integrity. In Ruthberg Z. G. and Polk W. T., editors, *Invitational Workshop on Data Integrity*, volume 500 of *Special Publication*, pages 1–14. National Institute of Standards and Technology NIST, September 1989. Section A.4.

[38] Sandhu R. S. On Five Definitions of Data Integrity. In Keefe T. and Landwehr C., editors, *IFIP Workshop on Database Security - Status and Prospects*, volume 7 of *Database Security*, pages 257–267. North-Holland, 1993.

[39] Sedgewick R. *Algorithms in Java, Parts 1-4*. Addison-Wesley, third edition, July 2002.

[40] Stubblefield A. and Wallach D. S. Dagster: Censorship-resistant publishing without replication. Technical Report TR01-380, Rice University, 2001.

[41] Tichy W. F. RCS–A System for Version Control. *Software-Practice & Experience*, 15(7):637–654, 1985.

[42] Tsudik G. Message Authentication with One-Way Hash Functions. *ACM Computer Communication Review*, 22(5):29–38, October 1992.

[43] Waldman M. and Mazières D. Tangler: A censorship-resistant publishing system based on document entanglements. In *8th ACM Conference on Computer and Communications Security*, pages 126–135, 2001.

[44] Waldman M., Rubin A. D. and Cranor L. F. Publius: A robust, tamper-evident, censorship-resistant, web publishing system. In *9th USENIX Security Symposium*, 2000.

[45] Wang X., Feng D., Lai X. and Yu H. Collisions for Hash Functions MD4, MD5, HAVAL-128 and RIPEMD. Technical report, Cryptology ePrint Archive, 2004.

[46] Wang X., Yin Y. L. and Yu H. *Advances in Cryptology CRYPTO 2005*, volume 3621/2005 of *Lecture Notes in Computer Science*, chapter Finding Collisions in the Full SHA-1, pages 17–36. Springer Berlin / Heidelberg, Santa Barbara, California, USA, August 2005.

[47] Zimmermann P. R. *The Official PGP User's Guide*. MIT Press, Cambridge, MA, USA, 1995.

Colophon

This thesis was typeset using LaTeX 2_ε, edited using TeXnicCenter (v1 Beta 7.01), and compiled using pdfeTeX distributed with MikTeX (v2.4). The figures were also typeset using LaTeX 2_ε: the majority of tree diagrams were produced using the xyling package; and the others with the pstricks package aided by jPicEdt (v1.4pre4). BibTeX (v0.99c) was used to typeset the references.

Layout of the main text is based around a customized version of the memoir class, using the epigraph, lettrine, hyperref, ntheorem and algorithm2e packages.

The main body text is set in the Computer Modern font at 10pt, and the type1cm package was used for the *dropped capital* at the start of each chapter. Other fonts include the AMS Euler script font used to denote parties, and the JD font used for the dedication page.