Document Management System Security

Jonas Birmé
birme@cs.umu.se

January 24, 2005
20 credits

Umeå University
Department of Computing Science
SE-901 87 UMEÅ
SWEDEN

Abstract

A common demand today is that software and systems must be secure but security mechanisms are often implemented without first considering what a secure system is. A formal specification of what is allowed and not allowed in a secure system is the security policy. The SS-EN82045-1 standard is a standard that assures a certain level of quality in an electronic document management system. This thesis presents an extendable framework to specify security policies for a document management system. As a reference implementation, a security policy for a product called Zert Infologic is specified with this framework. Zert Infologic is a product that facilitates the management of product manuals in multiple languages. The framework models the security aspects: confidentiality and integrity, and are based on the Bell-LaPadula model of secure information flow and Clark-Wilson model of integrity. In this framework, security is specified with a set of rules that defines the secure and non-secure states of a system. A secure system is a system that can not reach a non-secure state. The security mechanisms to enforce the security policy for Zert Infologic are discussed and a requirement specification (with respect to security) is suggested.

Contents

1 Introduction **1**

1.1 Background . . . 1

1.2 Thesis Description . . . 1

1.2.1 Goal . . . 2

1.2.2 Purpose . . . 2

1.2.3 Method . . . 2

1.2.4 Report Outline . . . 2

1.3 Related Work . . . 2

2 Background Theory **5**

2.1 Document Management Standard . . . 5

2.1.1 Principles . . . 5

2.1.2 Document Life Cycle . . . 6

2.1.3 Initiation . . . 6

2.1.4 Development . . . 7

2.1.5 Approval . . . 7

2.1.6 Revision . . . 7

2.1.7 Archival . . . 8

2.1.8 Deletion . . . 8

2.2 Basic Theory . . . 8

2.2.1 Basic Set Theory . . . 8

2.2.2 Cartesian Product . . . 10

2.2.3 Powerset . . . 10

2.2.4 Lattice . . . 10

2.3 Defining Security . . . 11

2.3.1 Terminology . . . 12

2.3.2 Security Policy . . . 14

2.3.3 Access Control Matrix Model . . . 16

2.3.4 Discretionary Access Control . . . 18

2.3.5 Mandatory Access Control . . . 18

2.3.6 Bell-LaPadula Model . . . 18

2.3.7 Clark-Wilson Model . . . 21
2.4 Enforcing Security . . . 23
2.4.1 Authentication Mechanisms . . . 24
2.4.2 Access Control Mechanisms . . . 25
2.4.3 Non-repudiation Mechanisms . . . 27
2.4.4 Integrity Mechanisms . . . 29

3 Document Management Security Policy 31
3.1 Security Aspects of SS-EN 82045-1 . . . 31
3.1.1 Information Security . . . 31
3.1.2 Aggregated Documents . . . 32
3.1.3 Approval . . . 32
3.1.4 Archivation . . . 32
3.1.5 Storage . . . 32
3.2 Security Requirements for SS-EN 82045-1 . . . 33
3.3 Document Management Policy Framework . . . 34
3.3.1 Definitions . . . 34
3.3.2 Rule-Based Policy Specification Language . . . 37
3.3.3 Specification Functions . . . 38
3.3.4 Basic Authorization Rules . . . 39
3.3.5 Derived Authorization Rules . . . 40
3.3.6 Policy Specification . . . 41
3.4 Security Policy Validation . . . 42
3.4.1 Access Control and Information Flow . . . 42
3.4.2 Integrity . . . 43
3.4.3 Traceability . . . 43
3.4.4 Rules . . . 43
3.5 Security Policy Example . . . 46
3.5.1 Meeting Protocol Management Policy . . . 47

4 Security Policy for Infologic 51
4.1 Detailed Description of Infologic . . . 51
4.1.1 Document Structure . . . 51
4.2 Threat Analysis . . . 53
4.2.1 Integrity . . . 54
4.2.2 Information Flow . . . 54
4.3 Infologic Security Policy . . . 55
4.3.1 Information Flow . . . 57
4.3.2 Operations . . . 60
4.3.3 Derived Rules . . . 61
4.4 Infologic Security Implementation . . . 63
4.4.1 Authentication Mechanisms . . . 63

4.4.2 Access Control Mechanisms 63
4.4.3 Non-repudiation Mechanisms 63
4.4.4 Repository Integrity Mechanisms 64
4.5 Implementation Requirements for Infologic 65
4.5.1 Must 65
4.5.2 Should 65
4.5.3 Could 66
4.5.4 Would 66

5 Conclusion **67**
5.1 Future work 68

References **69**

A Set Definitions **73**

List of Figures

2.1 Example of a weak security policy 14
2.2 Example of a finite-state machine 15
2.3 Portion of an access matrix 16
2.4 Example of a basic classification system 19
2.5 Access Control List 26
2.6 Capabilities 27
2.7 An example of a Merkle Hash Tree 30

3.1 Example of a cyclic document structure 35
3.2 The two security classifications in the meeting protocol policy 47

4.1 Example of a document and its structure in Infologic 52
4.2 Example of how documents are stored in a repository in Infologic 52
4.3 A document expressed with the document management framework 55
4.4 Overview of information flow in Infologic 56
4.5 A more detailed figure of how information flow from Infologic 56
4.6 Exporting and importing documents in Infologic 58
4.7 Approval and publishing procedures 59

List of Tables

3.1 Specification Functions 39
3.2 Utility Functions 40
3.3 Example of an access control matrix 40
3.4 Authorization Functions 41
3.5 Access control matrix in a meeting protocol management system 48
3.6 Operations in a meeting protocol management system 48

4.1 Principals in Infologic 57
4.2 Example of an access control matrix in Infologic 57
4.3 Security classifications in Infologic 58
4.4 Category combinations in Infologic 58
4.5 The security levels of the principals in Infologic 59
4.6 Operations 60

Chapter 1

Introduction

A common requirement in software development today is that the product must be secure, but with no consideration of what security is, it is difficult to validate that the product is actually secure [1]. Is a software that encrypts all data by default a secure software? When is the security of a system breached? Security in a system is often only specified by an informal consensus among the participants, a specification prone to be ambiguous [2].

Bishop, author of the book Computer Security - Art and Science [3], emphasizes that a distinction between policy and mechanism is critical in the study of computer security. A security policy specifies what is allowed and not allowed in a software or a computer system, that is, it specifies what is secure and insecure. A security mechanism is a method, tool or procedure to enforce a security policy. Without an unambiguous specification it can be difficult to implement the "right" mechanisms and detect the real threats.

1.1 Background

Zert AB[1] is a company that develops software and offers services in the area of risk management. One of the critical tasks in risk management is document management, ensuring that integrity and consistency of the documents are maintained. To facilitate the management and development of product manuals in multiple languages, Zert AB offers a product called Zert Infologic[2]. A new version of Infologic is planned to be developed and it is important this version is secure. There exists an industry standard for electronic document management systems called SS-EN 82045-1 [4]. The purpose of the standard is to ensure a certain level of quality assurance in an organization, and because a quality assured organization must be able to use this product, it is essential that Infologic also follows this standard.

1.2 Thesis Description

To define security in Infologic, the question to what is allowed and not allowed in Infologic must be answered.

[1] Official homepage at `http://www.zert.se`

[2] The shorter name Infologic will be used from now on.

1.2.1 Goal

The goal of this master thesis project is to analyze what a secure document management system is, develop a framework to specify a security policy for this type of application, obtain a security policy for Infologic, identify the security mechanisms to be implemented and develop the requirement specification (with respect to security) for Infologic.

1.2.2 Purpose

Different types of security policies focus on different security properties[3]. Already in the early 70s, Bell-LaPadula introduced a security policy with focus on confidentiality [5, 6] and both Biba [7] and Clark-Wilson have introduced policies with focus on integrity [3][4]. A combination of different types of policies yields a hybrid policy such as the Chinese Wall model [3][5] with focus on conflict of interest.

As both integrity and confidentiality are important properties in document management, it is interesting to study whether a combination of these policies can be used to develop a framework to specify security policies for a general document management system that follows the SS-EN82045-1 standard.

1.2.3 Method

First, the security aspects inferred by the SS EN-82045-1 document standard must be obtained and from these aspects, the requirements for the document management policy framework are specified. These requirements specify what must be considered when the framework is developed. Then, a threat analysis of Infologic is performed and with the framework a security policy for Infologic is written as a reference implementation to the framework. Finally the security mechanisms to enforce the Infologic security policy are discussed, and a requirement specification (with respect to security) for Infologic is specified.

1.2.4 Report Outline

The outline of this thesis report is to first present basic and general theory for the presented work. Then the developed security policy framework is introduced followed by the reference implementation, a security policy for Infologic to be delivered to Zert AB. Finally, the requirement specification for Infologic is presented followed by a conclusion.

1.3 Related Work

The importance of involving security at an early stage of software development was also identified by Doan et.al. [1] which introduced a framework for secure software design. They proposed an approach to incorporate security in object oriented software development, by assigning security levels to elements, use cases, classes and sequence diagrams in UML (unified modeling language).

A document management system must be able to store documents, for example, in a content repository. Devanbu et.al. [10] addresses the problem with publishing information

[3] These properties are described in more detail in section 2.3.1.

[4] As presented in Bishop's book "Computer Security" because the author of this thesis was unable to obtain the source article [8].

[5] Also as presented in Bishop's book "Computer Security" because the author of this thesis was unable to obtain the source article [9].

from integrity critical databases, e.g. financial information, over a public network. The proposed solution reduces the trust required of the publisher by separating the roles of owner and publisher. Trusted signatures and techniques based on Merkle Hash Trees provide authenticity and non-repudiation.

These techniques were also found to be quite useful when publishing structured documents as XML documents [11]. XML is a format often used in information exchange and in this extension to [10], it is proposed that the authenticity of path queries and selection queries can be guaranteed without the need for trusted online signing keys.

Another problem, addressed by Liu et.al. [12], is that digital signatures do not feel as authentic as traditional seals, a cultural gap exists. They propose a solution where a visual image is incorporated in the document and then the document is digitally signed, as an extension to the X.509 v3 certificate structure.

Chapter 2

Background Theory

This chapter presents the necessary theory for the work of this thesis project. The document management standard is first described, followed by some basic theory in computer science and computer security. It is recommended to read this chapter but anyone with expert knowledge in computer security can skip the basic theory and only read the description of the document management standard.

2.1 Document Management Standard

The intended purpose for the SS-EN 82045-1 standard [4] is to be used in computerized systems, e.g. electronic document management systems or other electronic systems for product data.

2.1.1 Principles

The standard defines that each document is associated with a collection of metadata, or in other words, data that identifies and describes the document. The metadata can be implemented:

- as visual parts of a document.
- in a document file transferred between document management systems.
- associated with a document in a document management system.
- as a separate collection of data stored independently of the document.

A document is a single document, a compound document, an aggregated document or a document set. For example:

- text documents.
- diagrams, maps, etc.
- hypertext documents, HTML, XML, etc.
- multimedia documents.

A single document is a document associated with its own metadata, for example a business letter or a PM. A compound document is a single document consisting of other types of documents, e.g. a document including a spreadsheet. An aggregated document is a composition of documents, where each document is associated with its own metadata. The compound document has metadata but is not necessary a separate document of its own. A hypertext document is an example of a compound document. A document set has metadata describing the purpose with the set and a list of including documents. Every document in the set has metadata.

All documents must be version controlled, a modification to the content of a published document will create a new version of that document.

2.1.2 Document Life Cycle

The life cycle of a document can be divided into the following phases:

- Initiation
- Development
- Approval
- Revision
- Archival
- Cancellation
- Deletion

A detailed description of each phase above follows.

2.1.3 Initiation

This is the phase where the document is initiated. A document must be assigned a unique identification. The identification must be stable and not depend on how the document is presented. This phase does not include adding contents to the document, that is writing the document.

The visual appearance of a document can be controlled with style sheets or other types of electronic templates. A document is equal to another document as long as the contents is the same.

A document can have a classification to simplify the process of searching for relating documents. All the necessary information are stored in the metadata of the document and SS-EN 82045-1 suggests the following information[1]:

- the owner of the document.
- a document identifier.
- the title of the document.
- date of initiation and expiration date.

[1] The complete list can be found in the SS-EN 82045-1 standard.

- a classification.
- security classification that can be changed without affecting the document version.
- access rights for persons and organizations.
- in the case with multiple languages the origin (native language) must be identified, that is which document is translated?

2.1.4 Development

This phase begins after the initiation phase and represent the activity of writing the actual document. The metadata could include:

- progress level.
- list of keywords.
- a summary.
- the source of the document.

All documents are stored in a storage or a so called electronic vault.

2.1.5 Approval

To ensure quality, every document must pass an approval process. A process defined by the organization, for example a document is first controlled internally by the organization (experts, etc) before the document is accessible by the public. The purpose is to check that the contents of a document agrees with the intended purpose of the document. A document must be under version control before the approval process can begin and it is important that all changes to a approved document are traceable. The following information is then added to the metadata:

- name of the person that approved the document and the approval date.
- comments related with the approval process.

All documents under version control are also stored in the electronic vault but the modification of the contents of a specific version must be prevented.

2.1.6 Revision

A revision to a document is either a modification of the contents or an alteration of the purpose of the document. It is important that every revision undergoes a controlled revision process. The contents of a published document version can not be modified without initiating a new document version. This adds the following information to the metadata:

- name and organization responsible for the modifications.
- what has been done.
- when it was done.

If the purpose of the document is altered only the metadata is modified and not the contents.

2.1.7 Archival

A selection of document versions are stored in a physical more compact and not modifiable form, including the associating metadata. A purpose of archiving documents is for example to fulfil legal requirements or other obligations. The contents and the metadata of an archived document must be reproducible and prevented from modification. Also, the access to the document must be controlled.

For long-term archiving purposes the metadata of the archived document is complemented with:

- expiration date.
- access rights.
- security level.
- usage of digital signature.
- backup information.

2.1.8 Deletion

When all legal obligations of a document are expired, the contents and metadata of a document can be deleted. The document is also removed from the electronic vault. A deletion should also destroy all physical instances of the document, e.g. CD-ROM, video tape, etc.

2.2 Basic Theory

This section gives some basic knowledge in mathematics and computing science but anyone with good knowledge in set theory and computing science can preferably skip this part. The material in this section is adapted from a course text book in discrete mathematics [13] and the Wikipedia[2].

2.2.1 Basic Set Theory

A *set* is an unordered collection of elements. An element in a set only exists once and ordering of the element is not considered, which means that the set $A = \{1, 2, 2\}$ is identical to $B = \{1, 2\}$ and set $C = \{1, 4\}$ is identical to $D = \{4, 1\}$. A set can have a finite or an infinite number of elements. A set with 0 elements is called the *null set* or the *empty set*, $\emptyset$. The set of natural numbers is a set of infinite number of elements.

Subsets

A *subset* A to B is a set where every element in A is also a member in B, denoted as $A \subseteq B$. A *proper subset* is a subset A to B and A is not equal to B, $A \subset B$. For example:

- The set of all men is a proper subset of all people.
- $\{1, 3\} \subset \{1, 2, 3, 4\}$

[2]Wikipedia (http://wikipedia.org) is a free-content encyclopedia started in January 2001 and is available online.

- $\{1, 2, 3, 4\} \subseteq \{1, 2, 3, 4\}$

The null set is a subset to every set $\emptyset \subseteq A$ and every set is a subset to itself, $A \subseteq A$.

Unions

The set of all elements which is either member in A or in B is denoted $A \cup B$ and called the union of A and B. Basic properties of unions:

- $A \cup B = B \cup A$
- $A \subseteq A \cup B$
- $A \cup A = A$
- $A \cup \emptyset = A$

For example:

- $\{1, 2\} \cup \{a, b\} = \{1, 2, a, b\}$
- $\{1, 2, a\} \cup \{a, b, c\} = \{1, 2, a, b, c\}$
- $\{1, 2\} \cup \{1, 2\} = \{1, 2\}$

Intersection

The set of common elements of two sets A and B is called the *intersection* of A and B which is denoted as $A \cap B$. Two *disjoint* sets are sets where the intersection is the empty set, $A \cap B = \emptyset$. For example:

- $\{1, 2\} \cap \{a, b\} = \emptyset$
- $\{1, 2, a\} \cap \{a, b, c\} = \{a\}$
- $\{1, 2\} \cap \{1, 2\} = \{1, 2\}$

Some basic properties of intersections:

- $A \cap B = B \cap A$
- $A \cap B \subseteq A$
- $A \cap A = A$
- $A \cap \emptyset = \emptyset$

Complement

The *relative complement* of A in B is denoted by $B \setminus A$, which is the set of all elements in B but not in A, comparable with "subtraction".
Examples:

- $\{1, 2\} \setminus \{a, b\} = \{1, 2\}$
- $\{1, 2, a\} \setminus \{a, b, c\} = \{1, 2\}$
- $\{1, 2\} \setminus \{1, 2\} = \emptyset$

2.2.2 Cartesian Product

The *cartesian product* is the set of all ordered pairs of the sets X and Y. Cartesian product of the sets X and Y is denoted as $X \times Y$. The set is defined as:

$$X \times Y = \{(x, y) | x \in X, y \in Y\}$$

An example illustrates this relation.

EXAMPLE 2.2.1: Set X is $\{A, K, Q, J, 10, 9, 8, 7, 6, 5, 4, 3, 2\}$ and set Y is $\{spades, hearts, diamons, clubs\}$ then the cartesian product of these sets, $X \times Y$ is the set:

$$\{(A, spades), (K, spades), \ldots, (2, spades), (A, hearts), \ldots, (3, clubs), (2, clubs)\}$$

This product can be generalized to the *n-ary* cartesian product which is defined as:

$$X_1 \times \ldots \times X_n = \{(x_1, \ldots, x_n) | x_1 \in X_1, \ldots, x_n \in X_n\}$$

For example, the set of all coordinates in a 3-dimensional cartesian coordinate system is $\mathbf{R} \times \mathbf{R} \times \mathbf{R}$ and $\mathbf{R}$ is the set of all real numbers.

2.2.3 Powerset

The *power set* of a set S is written as $P(S)$ or 2^S, which is the set of all subsets of S. For example the set $S = \{A, B, C\}$ has these possible subsets:

- $\{\}$ (empty set)
- $\{A\}$
- $\{B\}$
- $\{C\}$
- $\{A, B\}$
- $\{A, C\}$
- $\{B, C\}$
- $\{A, B, C\}$

Thus, the power set 2^S is $\{\{\}, \{A\}, \{B\}, \{C\}, \{A, B\}, \{A, C\}, \{B, C\}, \{A, B, C\}\}$.

2.2.4 Lattice

In mathematics, a *lattice* is a partially ordered set (poset) where any two elements have a least upper bound and a greatest lower bound.

A *partially ordered set* is a set with a special relation that formalize the ordering of elements in a set. The relation is a binary relation R over a set P. A binary relation that is *reflexive*, *antisymmetric* and *transitive*. For all a, b and c in P:

- aRa (reflexivity)

– if aRb and bRa then $a = b$ (antisymmetry)

– if aRb and bRc then aRc (transitivity)

EXAMPLE 2.2.2: Consider some set P and the relation $\leq$ on P. Then $\leq$ is a partial ordering relation if the conditions above are met, that is it is reflexive, antisymmetric and transitive (note that the $\leq$ is only an example and could be an arbitrary binary relation where the conditions holds). For all a, b and c in P.

– $a \leq a$ (reflexivity)

– if $a \leq b$ and $b \leq a$ then $a = b$ (antisymmetry)

– if $a \leq b$ and $b \leq c$ then $a \leq c$ (transitivity)

Now consider a partially ordered set $(L, \leq)$.

Definition 2.2.1. L is a lattice if for all elements x and y of a partially ordered set L, the set $\{x, y\}$ has both a least upper bound (supremum) and a greatest lower bound (infimum).

The least upper bound is the greatest element of a partially ordered set. The *greatest element* of a subset S of a partially ordered set is an element of S which is greater than or equal to any other element of S. Formally, given a partially ordered set $(P, \leq)$ then an element g of $S \subseteq P$ is the greatest element of S if

1. $s \leq g$, for all elements s of S

Infimum are dual to the concept of supremum, that is infimum of a subset of some set is the greatest element that is smaller than all other elements of the subset. Formally, the *infimum* of a subset S of a partially ordered set $(P, \leq)$ is an element l of P such that:

1. $l \leq x$ for all x in S, and
2. for any p in P such that $p \leq x$ for all x in S it holds that $p \leq l$.

To summarize, a lattice (in mathematics) is a partially ordered set in which all nonempty finite subsets have both a supremum and an infimum. The term is derived from that a Hasse diagram [13] of this particular types of posets has the shape of a lattice.

2.3 Defining Security

Security engineering is a discipline that focus on building systems to withstand threats such as malicious attacks, errors, user mistakes and misfortune. Security engineering is common compared with software engineering. An area that focus on ensuring that things can be performed. In addition security engineering also focus on the things that must not be performed. In reality, it is far more complex since requirements of systems vary and system fails because designers often protect the wrong things [2].

An article [14] published in Security Focus News[3] enlightens this complexity. The FAA (Federal Aviation Administration) was supposed to perform a certification of the computer security at major traffic control centers in the U.S, but when the security certification was

[3]Security Focus is a vendor-neutral site that provides security information to all members of the security community. Security Focus News offers news stories related to security and written by the SF staff.

audited by the Department of Transportation's Office of Inspector General, it was found that the systems were still not sufficiently secured against attacks. The problem was how the certification was performed, and a major complaint was that the certification process was limited to only developmental systems and that deployed systems were overlooked. Another issue was that only major computer servers were checked and not end-user computers, with the result that many workstations on the network were never checked for vulnerabilities.

The notion that the security was insufficient was verified by an incident where controllers were not able to communicate with the planes for three hours. The computer controlling the radio system has crashed, and Los Angeles Times reported that it was caused by a worker neglecting to perform the monthly reboot of the computer, with the consequence that the computer was automatically shutdown after 49.7 days.

With the wider usage of computers and the increasing connectivity, the society is faced with a new type of threats. Attacks today is not only performed by hackers, hacking systems for fun, it has emerged to become a tool for terrorists. For example, a computer controlling the life support system for the South Pole Station, housing 50 scientists, was hacked during the most dangerous season in South Pole [15]. The station received an email demanding money and threatened to sell obtained information. In this case the attacker was captured before any harm was done, but this emphasize the importance of secure systems in every domain of the society.

But, what do we actually mean when we discuss computer security? What is a secure system, is there any definition of a secure system, and is it possible to have a complete secure community? What differs from security in the non-virtual society?

2.3.1 Terminology

Frequently used terms in computer security are defined here (from [2, 3, 16]):

An *object* is an entity that should be protected or in other ways relevant to the system and a *subject* is a user or a program (process). When both objects or subjects are referred to, they are called *principals*. Other presented definitions in section 2.3 are taken from [3, 16] unless otherwise noted.

There are three basic aspects in computer security: confidentiality, integrity and availability. Anderson [2] argues that there are more aspects than these, but this thesis will use Bishop's [16] view on security with only three aspects.

Confidentiality

The *confidentiality* aspect concern all access restrictions to information and resources. Information can be hidden or scrambled to limit the access to only concerned parties, or the information is protected by a virtual barrier enforced with for example the access control of an operating system.

Existence of data may also apply to confidentiality, because the existence may reveal more than the data itself. For example the number of employees who distrust a manager may not be as important as for the managing group to know that such poll was performed. Another example, if your organization normally never encrypt email, someone monitoring the traffic could conclude that something is about to happen when suddenly all emails are encrypted.

Hiding resources is also an important part of confidentiality, because the notion of system configuration and what operating systems are used, can help an attacker to find the weakest link in your protection.

Contrary to Anderson [2], secrecy and privacy is covered by the confidentiality aspect. With *secrecy*, it means to limit the amount of people having access to the information and *privacy* is the ability to protect personal secrets. Bishop makes no distinction between secrecy, privacy and confidentiality. The formal definition of this property follows.

Definition 2.3.1. Let X be a set of entities and let I be some information. Then I has the property of *confidentiality* with respect to X if no member of X can obtain information about I.

Information is confidential to an entity if the entity is a member of the group that may not access the information. Thus, some entities have access to the information and the entities that do not, make up the set X.

Integrity

How much data can be trusted or the trustworthiness of data and resources is the aspect of *integrity*. It can be divided into data integrity, that is the integrity of the contents, and origin integrity, the source of the data is correct, which is also called *authentication*. Integrity violations can either be *prevented* or *detected*.

Prevention mechanisms tries to prohibit any unauthorized operations that attempts to change the data. A user can try to change data for which he or she is not authorized for or an authorized user can try to change the data in other ways than allowed, i.e. authorized user performing an unauthorized operation. Bishop gives a good example to explain this difference. Suppose an accounting system is running on a computer and someone hacks into the system and tries to modify the data, that is an unauthorized user tries to violate the integrity of the system. But what if an hired accountant tries to steal money by modifying the data and virtually transfer money to his own account, that is an authorized user tries to perform an unauthorized operation.

The detection mechanism tries not to prevent modification, rather making it possible to detect any modification violating the integrity. The data may be reported to be no longer trustworthy.

Definition 2.3.2. Let X be a set of entities and let I be some information or a resource. Then I has the property of *integrity* with respect to X if all members of X trust I.

Information I could also be information about origin or identity, the members of X trust that this information is correct and this type of integrity is also known as authentication. When I is a resource this definition defines that the resource is working correctly (as specified).

Availability

The ability to access information or a resource is referred to *availability*. A service can deliberately be blocked by an attacker making this aspect a part of computer security. These attempts are called *denial of service attacks*, or DOS-attacks for short, and can be difficult to identify and seperate from increased normal-usage. For example, if you are running a popular web site and suddenly one day the site hits a peak in load. Has the site become very popular and you have not enough servers to handle this increased popularity, or is it an attacker performing a denial of service attack? By manually analyzing every request, you would be able to conclude whether it is an attack or not, but it is difficult for a mechanism to prevent and detect the attack, because it would require usage-pattern analysis.

Definition 2.3.3. Let X be a set of entities and let I be a resource. Then I has the property of *availability* with respect to X if all members of X can access I.

2.3.2 Security Policy

An approach to security engineering is to structure the work in the order: *threat model - security policy - security mechanisms* [2]. A more software engineering adapted approach is to use a security engineering life cycle, which begins with analyzing the threats, define the policy, specify the security policy in a specification, design the system to follow the policy, implement mechanisms to ensure that users can not violate the policy and (as in software engineering) support maintenance [3].

Common with both approaches is that the most neglected part is the security policy and many organizations define the policy from vague statements [2]. A security policy should clearly state what is allowed and not allowed by the system and its participants. An example of a weak policy is found in Figure 2.1.

This policy is approved by Management.
All staff should obey this security policy.
Data shall be available to those with a "need-to-know".
All breaches of this policy shall be reported at once to Security.

Figure 2.1: An example of a weak security policy (adapted from Anderson's book [2])

This policy adds more questions than answers, for example what or who determines which staff members need to know and are allowed to access the data? The policy also includes a mechanism with the statement that "all staff should obey...". Is this enforced by the system or is it up to the staff members to be honest? It is clear that a distinction between policy and mechanism must be made.

Definition 2.3.4. A *security policy* is a statement of what is, and what is not, allowed.

Definition 2.3.5. A *security mechanism* is a method, tool, or procedure for enforcing a security policy.

The difference can be illustrated with an example. A university has a policy that states that copying the work of another student is not allowed, in other words cheating is not allowed. The computer system at this university has a mechanism that prevents others from reading a user's file. A student, Anna, has failed to use this mechanism correctly and forgot to protect her solution for an assignment, and the desperate student Johan (who must hand in the solution to the very same assignment in only one hour and has not even started) copies Anna's assignment. Security is clearly breached since this is an attempt of cheating, but which one of them has breached the security?

Despite the fact that Anna has failed to protect her files, which should be fairly simple in this case, she has not breached security because the policy never states that a user must protect his files. Johan, on the other hand, has breached security because the policy clearly states that it is not allowed to copy another student's work. In this case it would be simple for Anna to protect her work, but this may not always be the case. For example, if the solution was supposed to be mailed to the teacher, the network may not have adequate mechanisms to protect her work.

It is also possible to have a very weak security policy even when the mechanisms are sound [17]. For example the policy could state that all access must be authenticated and enforcing this authentication with passwords is a fairly strong mechanism, but a user could systematically always choose the same password as her login name without violating the policy.

Security policies may also be presented formally as a list of secure and non-secure states. Consider a computer system as a finite-state automaton with a set of transitions functions then a security policy can be defined as:

Definition 2.3.6. A *security policy* is a statement that partitions the states of the system into a set of *authorized*, or *secure*, states and a set of *unauthorized*, or *non-secure*, states.

And the definition of a secure system follows directly:

Definition 2.3.7. A *secure system* is a system that starts in an authorized state and cannot enter an unauthorized state.

An interesting fact derived from these definitions is that the very same system may be secure under one policy but not under another policy. If the environment changes and the policy too, the system that used to be secure may no longer be secure. An example of a finite-state machine is presented in Figure 2.2. A security policy partitions the states into a set of secure states $A = \{s_1, s_2\}$ and insecure states $UA = \{s_3, s_4\}$. Is this system a secure system?

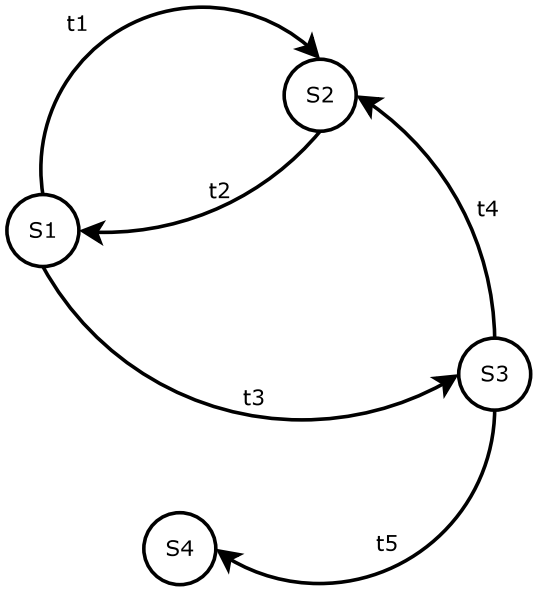

Figure 2.2: Example of a finite-state machine and in this example the secure states are s_1 and s_2

No, because there exists a transformation t_3 that takes the system from a secure state s_1 to a (by the policy specified) unsecure state s_3. Then, a formal definition of security breach can be introduced.

Definition 2.3.8. A *breach of security* occurs when a system enters an unauthorized state.

The set of all possible states in a system is the set P and the subset Q of P is the set of all authorized states. A secure system has a system state that is always in Q, and if the system is in $P \setminus Q$ the system is unsecure. The function of a *security policy* is to define Q and *security mechanisms* should prevent a system from entering a state in $P \setminus Q$. A model that can precise these states in P[4] are the access control matrix model introduced by Lampson in 1971 [18].

2.3.3 Access Control Matrix Model

The access control matrix is a simple framework to describe the relationship between subjects and objects in a system. For example, this model is used in both operating systems and database systems.

An access control matrix is a systematic way of describing shared data or resources and controlling the access to these shared things. Lampson defined a system with a set of objects O, a set of subjects[5] S and an access function A which is called an access matrix . An object is an entity to be protected, a subject is an entity which has access to an object and which subjects can access which objects are determined by the access matrix. The rows of the access matrix are labeled by the subjects and the columns by the objects, thus element $A[i, j]$ specifies the access subject i has to object j. An element consists of a set of strings that specifies what type of access a subject has (access attributes or access permissions). For example "read" and "write" are typical access attributes but others can be specified as well. Figure 2.3 shows an example of an access matrix.

	Subject 1	Subject 2	Subject 3	File 1	File 2	Process 1
Subject 1	owner, control	owner, control	call	owner, read, write		
Subject 2			call	read	write	wakeup
Subject 3			owner, control	read	owner	

Figure 2.3: Portion of an access matrix (adapted from [18])

In this example, `Subject 1` has *read* access to `File 1` and has *owner* access to both itself and `Subject 2`. It shows that a subject can also be an object if access to a subject must also be controlled.

Harrison et.al. [19] introduced a model to modify an access matrix by defining a set of *primitive operations* that alters the matrix. Formally an operation moves the system in state (S, O, A) to another state (S', O', A'). The operations Harrison et.al. defined are:

1. Operation `enter` r `into` $a[s, o]$

   Add the right r to the cell $a[s, o]$ in the access matrix. If $a[s, o]$ already contains r the operation has no effect.

[4] The set is called P from the term *protected states*, that is the states which are protected by the access control model.

[5] Lampson originally called the set of subjects for the domain but the more "modern" term subject will be used in this thesis.

2. Operation `delete` r `from` $a[s, o]$

   Remove the right r from the cell $a[s, o]$ and in the same way as in `enter` if r is not in $a[s, o]$ the state is unchanged.

3. Operation `create subject` s

   Adds a new row to the access matrix for subject s. The subject must not already exist in the matrix.

4. Operation `create object` o

   Adds a new column for object o if the object not already exists.

5. Operation `destroy subject` s

   Removes a subject s from the access matrix.

6. Operation `destroy object` o

   Removes an object o from the matrix.

These operations can then be used to create new commands. For example creating a file in the UNIX file system could be performed by executing these primitive operations.

```
command create_file(p, f)
  create object f;
  enter own into a[p, f];
  enter r into a[p, f];
  enter w into a[p, f];
end
```

In the example, a process p creates a file f with read (r) and write (w) rights by the owner, and creating a new process q by an existing process p could be performed with the following command:

```
command spawn_process(p, q)
  create subject q;
  enter own into a[p, q];
  enter r into a[p, q];
  enter w into a[p, q];
  enter r into a[q, p];
  enter w into a[q, p];
end
```

The new process q is owned by the spawned process p and the r and w rights make it possible for the processes to communicate with each other. Both examples above were adapted from Bishop's book in computer security [16].

A security policy can use two types of access control: discretionary access control or mandatory access control and these can be used separately or combined.

2.3.4 Discretionary Access Control

In discretionary access control (DAC) it is left to the owner's own discretion to determine the access to her files. This is the most common type of access control. For example, suppose a person is writing a self biography. She controls who is allowed or not allowed to read the biography. Her closest friend and mother are allowed to read it, but not her colleagues at work.

Definition 2.3.9. If an individual user can set an access control mechanism to allow or deny access to an object, that mechanism is a *discretionary access control* (DAC).

2.3.5 Mandatory Access Control

Access control where the owner can not control who can access the owned objects is called a mandatory access control.

Definition 2.3.10. When a system mechanism controls access to an object and an individual user cannot alter that access, the control is a *mandatory access control* (MAC).

Information about both the subject and the object which the subject wants to access, is used to determine (by an external mechanism) whether the access is allowed or not. For example, a court could access a person's criminal register without asking for consent by the person in question.

Now, some existing security policies, each with focus on one of the three security properties described in section 2.3.1 are presented. A security policy with focus on confidentiality is simply called a confidentiality policy and a security policy dealing with integrity is called an integrity policy.

2.3.6 Bell-LaPadula Model

A confidentiality policy should prevent the disclosure of information and restrict the information flow. It is also referred to as an *information flow policy*. A model that has influenced a lot of other models are the Bell-LaPadula model [5, 6] and based on this model, Denning [20] provided a unifying view of all systems that restrict information flow. There are typing systems, based on this model, to capture the flow of information in a programming language [21] and compile-time certification of secure information flow [22].

This model corresponds to the security classifications used in military systems. The phrase "information is on a need-to-know basis", used in more than one Hollywood movie, comes from this style of classifications. It is also called multilevel security, because information and subjects are placed in levels and information can only flow from a lower level to a subject on a higher level or same level.

Information at a higher level are more sensitive (and more important to keep confidential) than information at a lower level. A subject has a *security clearance* and an object a *security classification*. Thus, objects at a higher security classification are not readable by a subject of lower clearance. An example of a classification system is shown in Figure 2.4 on the next page.

This model combines discretionary access control with mandatory access control in the order that if mandatory access was removed from the model, access are still controlled discretionary. To be able to really enforce the "need-to-know" principle, information flow at the same level must also be restricted, the model is expanded with a set of *categories*. All classifications are also assigned a category where the category should describe the type

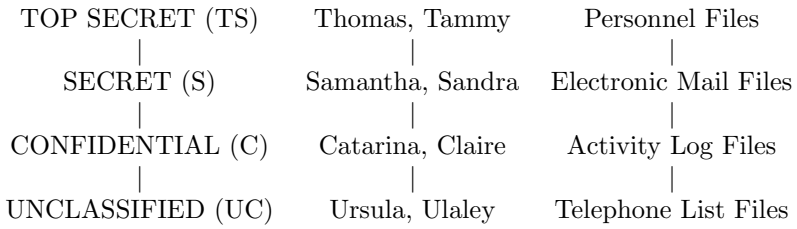

Figure 2.4: Example of a basic classification system (adapted from [16]). The most sensitive level is at the top and subjects and objects are associated to the classification on the same row

of information, or the class of information. Objects can also be placed in more than one category at a time. The combination of a security classification (or clearance) and a security category form a *security level*. A subject is said to be cleared into a security level and an object is classified with a security level.

Let $\{\texttt{NUC}, \texttt{EUR}, \texttt{US}\}$ be the set of all categories in a classification system. A person with access to the category set $\{\texttt{NUC}, \texttt{US}\}$ does not have access to an entity of the category `EUR`. Even if this person has a higher security clearance than the entity, the person is not allowed to read the object. However, if the object is in any of the combinations of $\emptyset, \{\texttt{NUC}\}, \{\texttt{US}\}$ this person may access the information. This relation is defined and called *dom* (dominates).

Definition 2.3.11. The security level (L, C) *dominates* the security level (L', C') if and only if $L' \leq L$ and $C' \subseteq C$.

This forms a lattice[6] under the $\subseteq$ ordering relation [20].

Example 2.3.1: (adapted from [3]). George has the security level (SECRET, { `NUC, EUR` }), document d_1 is assigned (CONFIDENTIAL, { `NUC` }), document d_2 has classification (SECRET { `EUR, US` }), and document d_3 is classified to (SECRET, { `EUR` }).

a) George *dom* d_1 because CONFIDENTIAL $\leq$ SECRET and { `NUC` } $\subseteq$ { `NUC, EUR` }.

b) George $\neg$ *dom* d_2 as { `EUR, US` } $\subsetneq$ { `NUC, EUR` }.

c) George *dom* d_3 as SECRET $\leq$ SECRET and { `EUR` } $\subseteq$ { `NUC, EUR` }.

Bell-LaPadula defined two conditions called *simple security condition* and **-property* (pronounced star-property). Simple security condition is also called "no reads up" (NRU) property, which means that information is not allowed to flow from higher to lower level. Let S be the set of subjects and O be the set of objects, the simple security condition can be defined as:

Definition 2.3.12. S can read O if and only if S *dom* O and S has discretionary read access to O.

The next condition, the *-property, is often referred to as "no writes down" (NWD) specifying that it is not allowed by a subject of higher clearance to write to a document at a lower level. If this was allowed, information at a higher level could be transferred to a subject at a lower level, with the help of a malicious subject adding the higher classified information to a document at a lower (and same) level as the low-level subject.

[6] See subsection 2.2.4 but showing that the relation forms a lattice is left as an exercise for the reader.

Definition 2.3.13. S can write to O if and only if O *dom* S and S has discretionary write access to O.

A *secure system* is a system where both simple security condition and *-property holds. Bell-LaPadula formalizes this with a *basic security theorem* (according to [3]):

Theorem 2.3.1. *Let Σ be a system with a secure initial state σ_0 and let T be a set of state transformations. If every element of T preserves, the simple security condition and the *-property, then every $\sigma_i, i \geq 0$, is secure.*

Let us consider a classification system with only one category and two security clearances: HIGH and LOW. What is the effect of changing the current classification of a subject or an object? If the classification of an object is raised from LOW to HIGH, then this object is no longer available to some subjects and if an object on HIGH clearance is lowered to LOW then everyone can access the object.

Raising classification (upgrading) is actually not a problem with this model, because the model only deals with manipulation of objects once their classifications are set. How and when the information is classified is not defined by the model. Allowing upgrading may violate the goals of the information flow system, since the goal of upgrading an object is to restrict access to the information. But, subjects at the lower level may already have accessed the information in prior the upgrade, and the goal of restricting the information might be violated.

A problem occurs when lowering a classification (declassification) because this is similar to a write-down. A document classified as HIGH and declassified to LOW has the effect that subjects at LOW level now can read the information written to the document. Information is written to a lower level, which is not allowed by the *-property. This problem is called the *declassification problem*.

The solution to this problem is called the *principle of tranquility*, a principle that prevents that subjects and objects change their security levels. In practice, this inflexibility could be a great limitation and is thereof divided into two forms.

Definition 2.3.14. The *principle of strong tranquility* states that security levels do not change during the lifetime of the system.

Definition 2.3.15. The *principle of weak tranquility* states that security levels do not change in a way that violates the rules of a given security policy.

The Bell-LaPadula model led to much controversy when it was introduced, and McLean [23, 24] (according to Bishop [3]) argued that the value of the Basic Security Theorem was overrated because it had no practical meaning in real systems. Bell-LaPadula responded that what they had introduced was an axiomatic model for security and Bell [25] (also according to Bishop) emphasized the nature of fundamental modeling and aspects in theory of science. Bell pointed out that the McLean model and Bell-LaPadula model defined security differently. Consider this example with two different definitions of *prime number*.

EXAMPLE 2.3.2: A *prime number* is an integer $n > 1$ that has only 1 and itself as divisors.

EXAMPLE 2.3.3: A *prime number* is an integer $n > 0$ that has only 1 and itself as divisors.

Both definitions are valid (mathematically) and consistent with the laws of mathematics. The question is, is the integer 1 prime or not? First definition says no, but the other definition says yes. Neither is "right" or "wrong". This controversy raised questions about the very foundation of computer security.

2.3.7 Clark-Wilson Model

In systems where integrity is more important than confidentiality, it may be more interesting to have a security policy with focus on this aspect. For example, an inventory control system does not handle any confidential information but if the data would randomly change it would seriously affect the function of the system.

A model that focus on this aspect is the Clark-Wilson model [8] (according to [3]), a model based on transactions. The system is in a *consistent* state if a set of given consistency conditions hold. For example in an accounting system a consistency condition could be:

$$D + YB - W = TB$$

D is the amount of money deposited so far today, W is the withdrawn money so far today and YB is the total balance of all accounts at the end of yesterday. TB is the total amount of money in all accounts so far today.

A transaction may leave the data in an inconsistent state. Consider a depositor transferring money from one account to another. It requires two transactions, a withdrawal from the first account and an insertion to the other account. Each isolated transaction is inconsistent and does not satisfy the stated consistency condition. However, both transactions combined in that order satisfy consistency. This serie of transactions are called *well-formed transactions*.

Another aspect with integrity requirements is the integrity of the transactions. Who or what examines that the transactions are well-formed? If only one person performs all depositions, it is possible for that person to actually steal money from the bank. He could still perform a well-formed transaction, but to his own account. The consistency is maintained but not necessarily allowed by the bank policy. This is where the principle of separation of duty can be applied, by requiring that all depositions are performed by two different persons. In integrity policies there are three key principles: *separation of duty*, *separation of function* and *auditing*.

Integrity Principles

If more than one step is required for a critical operation, the principle of separation of duty states that two different persons should perform this operation. For example, if the operation is to install development code in a production system it is recommended to involve both the developer and a installer. The installer should not be anyone involved in the development. Part of this process is to validate that the code works in this new environment, and it is more likely to detect faults by a person that has not been a part of the development.

The principle of separation of function states that non-related operations or services should not be part of the same system. For example an MTA (mail transport agent) should not do anything else than deliver electronic email, making it easier to detect if it is not doing what it is intended to do. Another example is that developers do not develop code on the production system, because untested code could include errors and damage the production data.

Auditing is the process of analyzing all performed operations and who executed the operations. The purpose is to detect any unauthorized operations and other violating operations. It involves logging which is the base of auditing, and thus very important to protect. A valid question is how can the integrity of logs be maintained and who audits the auditor? A solution is to apply the principle of separation of duty because auditing is a highly critical operation.

Clark-Wilson Model

Clark-Wilson defines the set of items that should be integrity controlled as the set of *constrained data items* (CDI). Items which should not be controlled exist in the set of *unconstrained data items* (UDI). In our bank example the balance of all accounts would be CDIs. The requirement of consistency is met with a set of *integrity constraints* and the consistency condition in our bank example can be modeled as an integrity constraints in the Clark-Wilson model.

To conform to the integrity constraints, this model defines two sets of procedures: *integrity verification procedures* (IVP) and *transformation procedures* (TP). IVP checks that the data in a system is in a consistent state and a TP changes the state of the data in the system. The model is built of certification and enforcement rules, the first two certification rules follows.

2.3.1. Certification rule 1 (CR1): When any IVPs is run, it must ensure that all CDIs are in a valid state.

2.3.2. Certification rule 2 (CR2): For some associated set of CDIs, a TP must transform those CDIs in a valid state into a (possibly different) valid state.

In the bank example, an IVP controls that the accounts are balanced and depositing money (or withdrawal) is a TP. A set of CDIs is associated with a particular TP, called a *certified* relation, and let C be the certified relation. Then if a TP f operates on a CDI o, $(f, o) \in C$. This yields the following enforcement rule.

2.3.3. Enforcement rule 1 (ER1): The system must maintain the *certified* relations, and must ensure that only TPs certified to run on a CDI manipulate that CDI.

This is however not enough, an arbitrary person is not allowed to perform a specific TP. For example a janitor should not be allowed to balance an account. A user performing an operation must be added, and a set of triples $(user, \text{TP}, \{\text{CDI set}\})$ is defined. This relation capturing the association of users, TPs and CDIs, is called *allowed* and enforced by:

2.3.4. Enforcement rule 2 (ER2): The system must associate a user with each TP and set of CDIs. The TP may access those CDIs on behalf of the associated user. If the user is not associated with a particular TP and CDI, then the TP cannot access that CDI on behalf of that user.

The certification for the allowed relation must also be added.

2.3.5. Certification rule 3 (CR3): The *allowed* relations must meet the requirements imposed by the principle of separation of duty.

Correct identification of all users performing a TP must be ensured, thus another enforcement rule that states this must be added.

2.3.6. Enforcement rule 3 (ER3): The system must authenticate each user attempting to execute a TP.

No authentication is required when a user enters the system because the user is allowed to modify UDIs, but as soon as the user wants to modify a CDI object it must use a TP which requires authentication. Auditing in Clark-Wilson is satisfied by logging all TPs. The log in itself is a CDI and all TPs appends to the log. Yielding the following certification rule:

2.3.7. Certification rule 4 (CR4): All TPs must append enough information to reconstruct the operation to an append-only CDI.

2.3.8. Certification rule 5 (CR5): Any TP that takes as input a UDI may perform only valid transformations, or no transformations, for all possible values of the UDI. The transformation either rejects the UDI or transforms it into a CDI.

This certification rule states that information must not be certified when it enters the system. For example, a person depositing money in an Automated Teller Machine (ATM) could have by accident entered an incorrect amount of money deposited. This discrepancy will be detected by the personell when the ATM is opened and fixed before the deposit enters the account. In this case, the UDI is the deposited amount which the person entered, and before the transaction is performed the UDI is checked (any discrepancies are fixed) and transformed into a CDI.

Enforcement rules ER2 and ER3 state that a user is associated with each TP and all users executing a TP are authenticated. It does not prevent a user from creating a TP, and associate some entities and herself with that TP. If she would associate the TP with some UDI or any other unauthorized information, she could perform acts that violated integrity constraints. To prevent this, the last enforcement rule invokes the principle of separation of duty, which states that only a certifier is allowed to change the list of entities associated with a TP, and this certifier is not allowed to execute the TP which she has certified.

2.3.9. Enforcement rule 4 (ER4): Only the certifier of a TP may change the list of entities associated with that TP. No certifier of a TP, or of an entity associated with that TP, may ever have execute permission with respect to that entity.

A correct implemented system is designed to ensure that the enforcement rules of the Clark-Wilson model are followed. The certification rules often requires some assumption of what can be trusted and enforced outside the system implementation. What is unique with this model (compared with other integrity models) is that it makes a clear distinction between certification and enforcement [3]. A system should primarily focus on implementing mechanisms to ensure that the enforcement rules are satisfied.

2.4 Enforcing Security

When it is clearly defined what security for a system means (security policy), the next activity in security engineering is to implement tools or procedures to enforce the policy. These tools and procedures are called security mechanisms, and this section will describe some of the security mechanisms available today.

A common misinterpretation of security is that strong cryptography leads to stronger security. This is not entirely false, because it has some level of correlation, but it is important to understand that it does not solve the entire problem with security. Bruce Schneier[7], in the first sentence of the preface of his book "Secret and Lies" [27], says:

> I have written this book partly to correct a mistake.

The mistake, according to himself, is that in his book "Applied Cryptography" [26] he has written "It is insufficient to protect ourselves with laws; we need to protect ourselves with mathematics." which he then claims is not true. Cryptography can not do any of that. The

[7] Author of the book "Applied Cryptography" [26]

reason is not that cryptography weakened, but cryptography is a branch of mathematics and "mathematics is logical; people are erratic, capricious, and barely comprehensible", Schneier writes. He had discovered that the weak points had nothing to do with the mathematics but with hardware, buggy software and the people.

Encrypting passwords in a password file with 1024-bits one-way encryption algorithm makes it very difficult to decrypt the password file. But, it does not protect against the lazy user choosing a weak password, for example choosing the username as the password. To obtain access to a password protected system it is not necessary to break the password file, it is easier to guess the password. The old saying, "no chain is stronger than the weakest link" is highly applicable in security engineering.

2.4.1 Authentication Mechanisms

Password is one of the many mechanisms to authenticate a person. The goal of an authentication mechanism is to have a tool or procedure that guarantee the authenticity of someone. For example, a driver's license includes a photo of the driver which then can be compared with the person in question.

Authentication mechanisms can be grouped into three categories [28], with the weakest mechanism at the top.

- *Knowledge* - something you know.
- *Ownership* - something you have.
- *Biological* - something you are.

Knowledge

The weakest authentication mechanisms are the mechanisms which falls into this category. Knowledge based mechanisms use some secret information, only available to the authenticating person, to control the authenticity of this person. Example of this mechanism are password authentication and personal questions e.g. asking for mother's maiden name. There are psychological issues with these types of mechanisms [2]:

- Prevent disclosure of the secret information to a third-party.
- Difficulties when entering the secret information.
- Difficulties with remembering the secret information.

An attack known as *social engineering* is a specific type of a fraud with the aim to make a person deliberately or undeliberately disclose some secret information. For example, calling someone and by impersonating a bank clerk obtain private account information.

Another human issue is if the secret is complex it can be very difficult to enter the correct information. For example, if the password for an ATM is long and complex the user can by mistake enter the wrong password, and if the user only has three attempts (as a brute force protection) to enter a valid password for an ATM she could falsely be denied access to the machine, even though she is the "right" person.

A problem related to long and complex secrets is that the secret can be very difficult to remember. For example a long password (more than 20 letters) is hard to remember, and a common solution is to write down the password on a note. Even worse, if the note is placed in junction to the protected entity.

Ownership

The ownership category requires a user to have a specific physical entity, for example a physical key to a lock. However, exactly as with knowledge, it is sufficient to only have this entity and that it is the owner of the key who unlocks the lock is not guaranteed. It is common to combine this with something you know, for example to get money from an ATM it is required to have both a credit card and the password associated with this password.

Until recently, this has been a sufficient mechanism. A relative new attack against ATMs is to copy the credit card and with a camera record the password. The attack is performed by assembling a little card reader "on top" of the real reader and with a small camera the password is recorded when a user enters the password. This information is then sent to a nearby mobile unit, managed by the attacker. The attacker then creates a replica with the obtained card information and with the obtained password he / she could take money from the associated account at another ATM.

Biological

Security mechanisms in this category are also known as *biometrics* and are based on the physical characteristic of the authenticating user. For example, a fingerprint, voice pattern or a photograph of the person.

Biometrics can also be combined with an ownership mechanism and replace the knowledge based password part. A problem with biometrics is that some biometric data must be public (in comparison with passwords which can be kept as a secret), for example fingerprints can be considered to be public. An attacker could obtain someone else's fingerprint and bypass the biometric sensors to obtain illegal access to a resource. Waldmann et.al. [29] suggest a solution to this problem by protecting the verification data (digital representation of a fingerprint) transmitted between the biometric sensor and the ownership entity, for example a smart-card. This mechanism can be intrusive and in some cases a threat to the personal integrity, which could be an explanation to its limited use.

2.4.2 Access Control Mechanisms

An access control matrix is suitable for modeling access control but not necessarily a very efficient mechanism as it does not scale well. For example, an organization with 1000 employees and 10000 applications and documents would create a matrix with 10 million entries. There are basically two ways of implementing an access control matrix, either by columns (access control lists) or rows (capabilities). To obtain effective access control it must be combined with some kind of authentication of the subjects.

Access Control Lists

Storing the access control matrix a column at a time is also known as *access control lists* (see Figure 2.5 on the following page). An access control list contains a list of subjects and their access permissions for a specific object. Access control list is an advantage when the access control are discretionary, that is the users manage their own file security or the protection is data-oriented. It should not be used in an organization with many users (the access lists tends to grow) and the user population is constantly changing. When a subject (a user) is removed from the system, the subject must be removed from all objects in the system. Also, it can be very time consuming to find all objects a user are authorized to use.

	Subject 1
Subject 1	owner, control
Subject 2	
Subject 3	

Figure 2.5: Column-wise representation of an access control matrix is called an access control list

Example of an operating system that invokes access control lists is Unix[8] (and its variant Linux). Files have the access rights of `rwx`, rights to read, write and execute a file. These access rights can be set for the owner, a group and others in the system. For example (adapted from [2]):

```
drwxrwxrwx Alice Accounts
```

The first flag indicates that the file is a directory, the first group determines the owner-access, the second the group rights and last the access rights for others. Then the owner and the group of this directory is specified. In this example anyone has full access to this directory. In the next example, the owner (Alice) has both read and write permissions and the group (Accounts) has only read permissions.

```
-rw-r----- Alice Accounts
```

A more fine-grained control over this simple access control can be achieved with the POSIX Access Control Lists [30], available for Unix and recently also available for Linux. This is an extension of the access control mechanism presented above with a *minimal access control list* and an *extended access control list*. The extended ACL adds the possibility to mask access permissions, giving access control with higher granularity.

Capabilities

When the matrix is stored a row at a time it is known as *capabilities* (see Figure 2.6 on the next page), and the advantages and disadvantages are the opposite of access control lists. It is called capabilities because it models what a subject is capable of. A subject `Subject 1` is for example capable of reading `File 1` and writing `File 1`. Capabilities provide a more efficient runtime checking, because in many cases it is a subject that performs an operation, for example read access for a file can already be controlled before the file is found.

An operating system to implement capabilities is Windows 2000, with a mechanism that overrides or complement the ACL in NT4. An object-oriented database called an *Active Directory* manages all groups and stores the permissions for each group. The database is indexed and searching can be performed on any attribute. For example, in Windows 2000 there exists a special type of program running in the background called *services*. The policy that only administrators can start and stop services is enforced by creating a group that is

[8] POSIX file system object permission model.

	Subject 1	Subject 2	Subject 3	File 1	File 2	Process 1
Subject 1	owner, control	owner, control	call	owner, read, write		

Figure 2.6: With capabilities the matrix is stored a row at a time

capable of this operation. For a subject to be able to start or stop services, it is required that the subject is a member of this specific group.

According to de Vivo et.al. in an overview paper of capabilities [31], the approach of using capabilities as access control mechanism seems to scale very well and is a suitable security mechanism in distributed system.

2.4.3 Non-repudiation Mechanisms

A foundation in many of the described policies is the notion of trust. How can trust be enforced? In some sense, trust is an integrity property as well, because trusting a subject means that the subject does what it is intended to do (otherwise it would be no meaning in trusting that subject). As mentioned earlier, integrity violations can either be detected or prevented. This section will describe a mechanism to detect integrity violations with respect to trusted subjects.

A trusted subject is a subject entrusted with a critical operation, or an operation that could violate the security policy. For example declassification in Bell-LaPadula model or certifying a TP in Clark-Wilson model. A trusted subject can also be a user with extended privileges for instance the administrator of a computer system. Philosophically and socially, the value of trust has a price and with enough money anyone can be bought. For example, the administrator of the system could be "motivated" to perform an operation that violates the policy. This problem is not isolated to humans only, even when the trusted subject is a program with certified code, it is almost impossible to prove that the program will not perform a policy-violating operation. It could be either a bug or an error exploited by a virus (often in some kind of combination).

The concept of ensuring that a contract can not be denied by one of the parties involved is called *non-repudiation*, a concept to prohibit false-deniability. A concept also used to verify that the sender of a message is in fact also the one who claimed to have sent the message. This concept can be used to prevent a trusted subject to falsely deny a policy violating operation, which makes it more difficult to "get a way with it". To enforce false-deniability or non-repudiation, digital signatures can be used. If all operations are digitally signed by the subject performing the operation, a subject can not deny that the subject has performed that specific operation.

Digital Signatures

In the same way as with traditional signatures used in the physical world, a *digital signature* should confirm that only the legitimate signer can produce this signature. Compare with a bank check (not used so often any more) for a financial transaction. A check authorizes a financial transaction, a check can not be altered or reused and a third-party can confirm the authenticity of the check by analyzing the hand-written signature. There are two

requirements of a digital signature, it must be:

- *Unforgeable.* If a person s signs a message M with his signature $sig_s(M)$ it should be impossible for anyone else to produce the pair $[M, sig_s(M)]$.
- *Authentic.* If a person r receives the pair $[M, sig_s(M)]$ she must be able to check that the signature is really from s, that is only s could have created the pair.

Asymmetric encryption or public key cryptography can be used to provide digital signatures, where a signature is obtained by encrypting M with the private key. The receiver can check the authenticity of M with the corresponding public key.

Asymmetric Encryption

Diffie and Hellman (according to [28]) introduced a new type of cryptography system called *public key cryptography* or *asymmetric encryption.* In this system the user has a public and a private key. The public key can be published and distributed freely but the private key must be kept secret. This is possible since both keys are needed to complete the encryption/decryption process. One of the keys are the inverse of the operation of the other key.

Let k_{PRIV} be a private key and k_{PUB} a public key then the plain text P, encrypted with the public key k_{PUB}, is decrypted with the corresponding private key k_{PRIV}.

$$P = D(k_{PRIV}, E(k_{PUB}, P))$$

A user can decrypt with the private key what someone has encrypted with the corresponding public key. This public key can be distributed to all people which a person will receive encrypted messages from. The public key can not be used to decrypt a message encrypted with the public key, because of the relationship stated above. And, if a message is encrypted with the private key, only the corresponding public key can decrypt the message.

$$P = D(k_{PUB}, E(k_{PRIV}, P))$$

An implementation of this type of encryption is the RSA (Rivest-Shamir-Adelman [32]) cryptosystem, an algorithm introduced in 1978 using number theory and the degree of difficulty in prime number factorization. Let e be the encryption key, then any plaintext P is encrypted to the ciphered text C with:

$$C \equiv P^e \text{ modulo } N \tag{2.1}$$

A number $N = pq$ is chosen, where p and q are large prime numbers. It is easy to calculate N but factoring N to obtain p and q is very difficult. The public key e has no common factors with either $p - 1$ or $q - 1$ and the private key is the prime numbers p and q. Decryption is basically defined as:

$$P \equiv \sqrt[e]{C} \text{ modulo } N \tag{2.2}$$

A decryption key d can be obtained from (2.2) such that decryption can be written as:

$$P \equiv (P^e)^d \text{ modulo } N$$

This cryptography system can also be used with digital signatures to enforce non-repudiation. For example, after a file modification is performed the entire file F and subject

ID s_{id} are signed with the subject's private key giving the triple $[F, sig_s(F + s_{id}), s_{id}]$[9]. With the public key of s_{id} it can be verified that the last modifier was s_{id} by decryption of $sig_s(F + s_{id})$ and comparison with F and s_{id} in the triple. In RSA, digital signatures are accomplished with the formula:

$$sig_d(M) \equiv M^d \text{ modulo } N \tag{2.3}$$

The receiver has access to e and can verify the authenticity by taking the power e of $sig_d(M)$. It is verified by comparing the calculated M_{verify} with the original M

$$M_{verify} \equiv (sig_d(M))^e \text{ modulo } N \tag{2.4}$$

2.4.4 Integrity Mechanisms

Mechanisms to enforce the integrity of a system, mainly data integrity, are presented and discussed here. As stated before, integrity violations can either be detected, prevented or both. A mechanism to detect and prevent that a document (an email, contract, etc) is modified is with the usage of a *message digest*. A digest of a message is a fixed-size representation of the message, independent of the size of the message, sometimes referred to as a checksum. The pair $(M, dig(M))$ where $dig(M)$ is the message digest is distributed, and to check that M is unmodified $dig(M)$ is calculated and compared with the provided $dig(M)$. If the calculated digest mismatch with the provided digest, the message was in some way modified. A usual application is signing email (PGP [33, 34] or PEM [35]) where a digest of the message is computed and signed by the sender with a secret key. This is called an *all-or-nothing* flow, i.e. the entire flow (or in this case the message) is signed.

An algorithm that takes an input message of arbitrary length and produces a 128-bit value is the MD5 message digest algorithm [36]. It is designed to work fast on 32-bits computers and implemented compactly. A brief description of how the digest is calculated follows:

1. Append padding bits

   The message length (in bits) is extended to a length congruent to 448, modulo 512.

2. Append length

   A 64-bit representation of the length of the message before the padding bits were added is appended. The message now has a length that is an exact multiple of 512.

3. Initialize MD buffer

   The four-word buffer to compute the message digest is initiated.

4. Process message in 16-words blocks

   For a detailed algorithm description of this step, consult the RFC 1321.

5. Output

The MD5 digest value is calculated in a way that it is computationally infeasible to produce two messages with identical digest value. A stronger (in comparison with MD5) message digest algorithm is SHA (secure hash algorithm) that produces a 160-bit digest value from a message.

[9] The +-operation is string concatenation in this example.

An information provider of sensitive information (financial information or other authorized information) is required to be able to provide an authentic subset (answer to a query) of the information it is storing. To enforce the authenticity of these answers, it is not possible to use an all-or-nothing flow because it is very uncommon that the answer to a query is the entire database. Instead, an approach called *tree chaining* [37] can be used. Tree chaining uses a tree (Markle Hash Tree) where the root is the digest of the entire data and a subset of the data is represented by the children. To verify a subset (a child) only the path to the root needs to be verified.

A Merkle Hash Tree is shown in Figure 2.7 where the parent of a leaf h_3 and h_4 is h_{34}. The digest of the parent h_{34} is calculated by $h(h_3 \parallel h_4)$[10] where $h(\cdot)$ is a message digest function, for example MD5. A requirement is that the hash function is collision-resistant, which means that it should be very difficult to obtain two different messages with the same hash value. The value of h_4 can then be verified if the correct value of h_{34} is known to the client. This design makes it hard to forge the contents of the children.

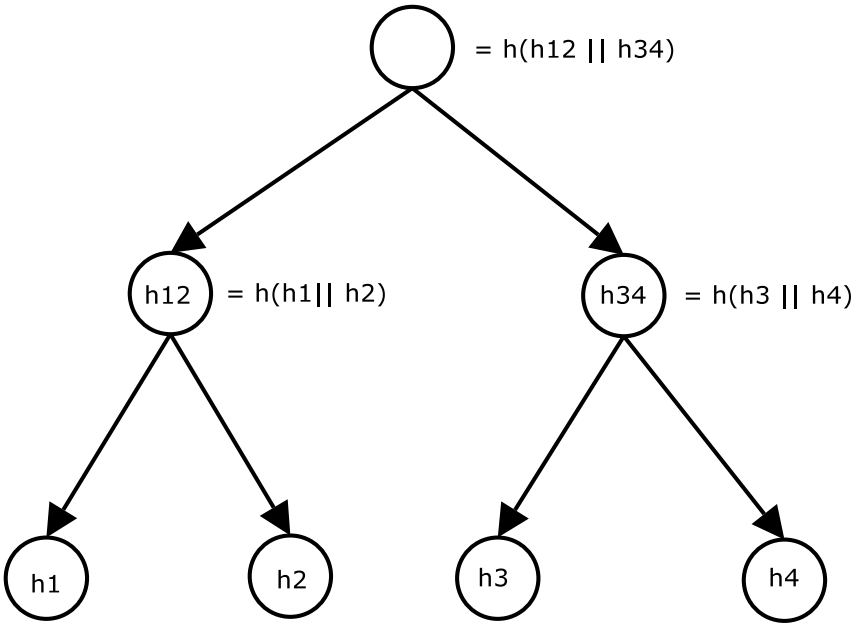

Figure 2.7: An example of a Merkle Hash Tree

Since it is a concatenation, a total order of the children is enforced, i.e the MD5 value of $h_3 \parallel h_4$ is not the same as $h_4 \parallel h_3$. Compare with the MD5 value of a short message `HELLO`, a value that of course should not be the same as the MD5 value of `HLELO`. Important to know is that the binary tree in Figure 2.7 can easily be extended to a greater branching factor.

[10]The symbol $\parallel$ denotes concatenation.

Chapter 3

Document Management Security Policy

This chapter describes the document management security policy framework for an arbitrary document management system. First the security aspects of the document management standard are discussed followed by the requirements for the security policy framework. Then, the framework is introduced followed by a validation of the framework with respect to the obtained requirements and an example of how this policy framework can specify security in an example document management system.

The next task of the thesis project, to develop a security policy for Infologic, is presented in the next chapter.

3.1 Security Aspects of SS-EN 82045-1

With respect to security, what must be considered for an electronic document management system to follow the standard are discussed in this section.

3.1.1 Information Security

In a larger organization, a common demand is to limit the information available to a specific group or to a person assigned a specific role. Information leakage is a threat to any information systems with restricted information and must be prevented.

Initially (according to the standard [4]), a document is assigned a security classification which is applied to all document versions. If the classification is changed, the classification of all document versions is also affected. All initiated and working documents are stored in a document storage or a vault and access to both vault and documents must be controlled. The vault controls the discretionary access[1] to the documents, for example to limit the access to only authorized subjects or organizations. The usage of security classifications implies that information can be restricted, thus confidentiality is of importance.

The security classification used in Bell-LaPadula model[2] can be applied in this system to restrict the flow of information. Access to a single document is controlled either with a discretionary access control or the contents of a document are restricted with a mandatory

[1] Discretionary Access Control is described in subsection 2.3.4
[2] See subsection 2.3.6

access control[3]. The security classification in Bell-LaPadula implies that a subject also is assigned a security clearance.

3.1.2 Aggregated Documents

Another aspect to the confidentiality property is the usage of aggregated documents, i.e. a composition of documents where each document has its own metadata. To be able to read the information in an aggregated document, a subject must have access to all aggregated documents.

3.1.3 Approval

An approved document version is not modifiable without creating a new revision. It should also be possible to trace who has approved a document. A threat to the approval process is that the approver neglects to perform the actual control, i.e. not reading the document, but still approves the document. Another threat is that an approver is impersonated, either by another person or the same person writing the document, and an invalid document is approved.

Approving a document can be performed by signing the document with some kind of digital signature. Instead of signing the entire document a digest (one-way hash value) of the contents can be signed. It is important that the digest is calculated with a cryptographically secure hash function. Also, it is essential that the document (or the digest) is signed with a secret (private key) only accessible to the approver, to ensure that an approver can not deny an approval and that an approver can not be impersonated (in section 2.4 are these mechanisms described in more detail).

A version controlled repository is responsible for that a modification to a document under version control results in a new document version. However, all changes must also be traceable and thereof the modification (or the new document version) is signed by the modifier. This is not the same as approving the revision. Approving a revision is similar to how approving a document is performed.

3.1.4 Archivation

A document can be archived in a long-term storage, where an archived document and its metadata must be protected from modification and deletion. The archived document can be compressed (to save storage space) and if required encrypted. The security classification of a document can not change once it is archived but the access permissions to the document can be changed. For example to be able to restrict access to archived documents to a limited group of subjects. When an archived document expires, the document and all copies of the document is deleted from the archive. A document can only be deleted by a subject with explicit permissions to do so.

3.1.5 Storage

It can be concluded that two types of storages are needed: an access controlled document storage (vault) and an access controlled redundant storage (archive). The threats to these storage spaces are discussed in this section.

[3]Mandatory Access Control is described in subsection 2.3.5

The vault stores all documents and both access to the vault and access to a specific document in the vault is controlled. Access to the documents can be defined with an access control matrix. The available access permissions are: read (`r`), write (`w`) and delete (`d`). Read-access permits that the document can be searched for and found in document listings, write-access permits that the document can be modified and delete-access permits deletion of the document. However, the vault does not enforce the information flow defined by the Bell-LaPadula security classifications. Allowed access to a document does not imply that the information can be read. This is a protection against the threat of a compromised vault, i.e. even if the access control of the vault is breached the documents in the vault are still protected.

The archive must handle everything that the vault can handle with some additions. A document can be archived for many years and the information must be intact, thus integrity is of importance. Also, a deletion should not be possible if the document is still "valid", i.e. expiration date has not passed. This must be added to the access control of the archive. The archive should also support some kind of compression to save storage space.

When a document is deleted from the archive all replicas (created for redundancy?) must also be deleted. Depending on the level of security (and economy) different measures can be taken to ensure that the document is really cleaned up, e.g. reformatting sectors of the hardware storage etc. A physical threat to the archive is malicious or accidental destruction of data, i.e. bit errors on hard drive or crashed disk. If it is not possible to protect against these threats, a lowest level of security requires that the error is at least detected. Calculating checksums or other mechanisms can detect these faults[4].

3.2 Security Requirements for SS-EN 82045-1

From this analysis of the standard, the following security requirements for the framework are suggested:

1. Access control and Information flow
   (a) Access rights to a document are separated from the access rights to the contents of a document.
   (b) The contents of a document have a security level classification.
   (c) A documents security level can change during the lifetime of a system.
   (d) A document can include other documents thus access to these documents must follow the same condition in (a).
2. Integrity
   (a) An approved document can not be altered.
   (b) An archived document can not be altered.
   (c) An archived document can not be deleted before it has expired.
   (d) The security level of a document can not be altered if the document is archived.
3. Traceability
   (a) All documents have a system-unique identity.

[4]For example Merkle Hash Trees (section 2.4.4)

(b) All documents have an originating creator.

(c) All approved documents have an approver identification.

Version control is deliberately omitted from these general requirements because it can be handled by the storage container. However, same restrictions to a document are applied to a document version. This limitation is merely a simplification.

To define a security policy for a document management system, a framework suitable for these types of requirements is developed. The framework must deal with access control, information flow control, integrity and traceability.

3.3 Document Management Policy Framework

The goal of this model is to provide a framework to specify a security policy for an arbitrary document management system that follows the SS-EN 82045-1 standard. Fundamental parts of the document management framework is a document, the authorized access to a document and the integrity of a document. A document can be in one of a predefined set of valid states.

All documents are CDIs (constrained data items) as defined in the Clark-Wilson model[5]. Access to documents are performed with authorized operations, which is an operation authorized with a set of rules. The rules specifies the non-secure states of the document management system and a rule-based specification language [38] is used. The language is closely related to a logical programming language. An operation modifies the document or the state of the document. Authorized operations can be compared with IVPs (integrity verification procedures) defined by Clark-Wilson. A secure system is a system where all operations are assigned a rule and no operations that violates the rules are performed.

Every document is associated with a security level and discretionary access to a document are controlled with an access control matrix. This is based on the Bell-LaPadula model of secure information flow (described in subsection 2.3.6). A subject is either a person or a program that wants to access the information in a document. A special type of subjects also exist, called trusted subjects. A trusted subject is a subject that can not falsely deny the performed operations.

3.3.1 Definitions

A complete list of all sets defined in this framework is available in appendix A on page 73.

Objects

The fundamental objects in a document management system are documents, persons and programs accessing the documents. Access to documents are performed with operations and operations that adheres to the policy are authorized operations. The objects in a document management system are a set S of subjects and a set D of documents. A document can be in one of the states in Σ_{doc}.

Definition 3.3.1. A *document* $d : (e, M_{data}, \sigma), d \in D$ is a tuple of the contents $e \in E$, a set of metadata elements $M_{data} \subseteq M = \{m_{id}, m_{author}, m_{approver}, m_{level}\}$ and the current state $\sigma \in \Sigma_{doc} = \{\sigma_{approved}, \sigma_{archived}, \sigma_{cancelled}\}$.

[5]Described in subsection 2.3.7

Every document is associated with a set of metadata M_{data} that describes the properties of a document, for example security level for a document or name of the author. A document tuple includes a contents tuple $e \in E$ (E is the set of all these tuples) which include both information (text, figures, tables etc) and a set of subdocuments to support aggregated documents, suggesting a recursive structure:

Definition 3.3.2. Contents $e : (\alpha, D_{sub}) \in E$, of a document $d \in D$ is a tuple of the information $\alpha \in A$, a set of another documents $D_{sub} \subseteq D$ and document $d \notin D_{sub}$. A is the set of all kinds of information.

This prevents that immediate recursive cycles exist, i.e. a document can not include itself. Still, indirect recursive cycles can exist. For example a document d can include another document that in turn include d (See Figure 3.1).

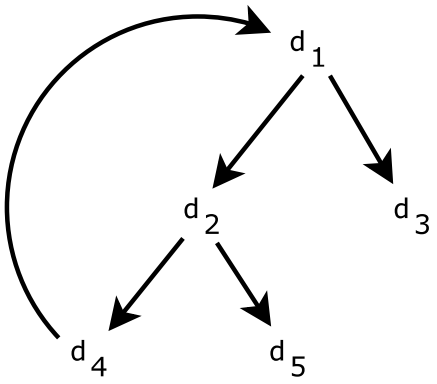

Figure 3.1: Example of a cyclic structure with the documents $d_1 = ((\alpha_1, \{d_2, d_3\}), \ldots)$, $d_2 = ((\alpha_2, \{d_4, d_5\}, \ldots)$, $d_3 = ((\alpha_3, \{\}), \ldots)$, $d_4 = ((\alpha_4, \{d_1\}), \ldots)$ and $d_5 = ((\alpha_5, \{\}), \ldots)$.

The classification system is based on the security levels introduced in the Bell-LaPadula model. A document is assigned a security level. A security level consists of a security classification or clearance and a set of categories. The security level of a document is stored as metadata to the document.

Definition 3.3.3. Let C be the set of classifications (or clearances), let K be the set of categories and let $L = C \times K$ be the set of security levels. For all documents $(e, m_{level}, \sigma) \in D$ there exists metadata $m_{level} \in M_{data} \subseteq M$, where $m_{level} \in L$.

Discretionary access to a document is defined in an access control matrix. A document is either readable (`r`), writable (`w`) or deletable (`d`).

Definition 3.3.4. The set T is the access rights to a document where $T = \{\mathtt{r}, \mathtt{w}, \mathtt{d}\}$.

The requirements of a system-unique identity and originator creator suggest that identification information and creator are stored in the metadata of a document, defined as:

Definition 3.3.5. For all documents $d : (e, m_{id}, \sigma) \in D$ there exists metadata $m_{id} \in M_{data} \subseteq M$ which is a unique identification of document d.

Definition 3.3.6. For all documents $d : (e, m_{author}, \sigma) \in D$ there exists metadata $m_{author} \in M_{data} \subseteq M$ which identifies the originating creator of document d.

It is suggested by the traceability property that an approved document must also store an identification of the approver. This property yields the definition below:

Definition 3.3.7. For all approved documents $d : (e, m_{approver}, \sigma) \in D$ there exists metadata $m_{approver} \in M_{data} \subseteq M$ which identifies the approver of a document.

System

A document management system is represented with a set of states and the security policy define which states are secure and which are non-secure. It has a sequence of operations, decisions, states and an initial state: $\Sigma(O, B, W, z_0)$. Let F be the set of tuples (f_s, f_d) where f_s associate with each subject a current security level and f_d current security level of a document.

Definition 3.3.8. A document management system $\Sigma(O, B, W, z_0)$ has a set of operations O, a set of decisions (outcomes) $B = \{\texttt{true}, \texttt{false}\}$, a set of actions $W \subseteq O \times B \times V \times V$ and an initial state z_0. V is the set of all states.

A state $v \in V$ is a tuple (p, a, f, d) where $p \in 2^{S \times D \times T}$ tells which subjects have permission to access which documents, and what these permissions are. The access control matrix for the current state is a, the current state of all documents $d \in 2^{E \times M \times \Sigma_{doc}}$ and $f \in F$ is the tuple that indicates the current security levels of a document or a subject. An *action* moves the system from one state to another state, and defined as follows:

Definition 3.3.9. $(o, b, v, v') \in O \times B \times V \times V$ is an *action* of $\Sigma(O, B, W, z_0)$ if and only if there is an $(x, y, z) \in \Sigma(O, B, W, z_0)$ and a $t \in N$ such that $(o, b, v, v') = (x_t, y_t, z_t, z_{t-1})$.

Thus, an action is an operation/decision pair o and b, and moves the system from the state $v' \in V$ to the (possibly different) next state $v \in V$.

N is the set of positive integers representing time and $X = O^N$, whose elements x are sequences of operations. $Y = B^N$ is the set where element y represent a sequence of decisions and $Z = V^N$ the set where z are a sequence of states. An interpretation of this is that for a $t \in N$ the system is in state $z_{t-1} \in V$, a subject performs an operation $x_t \in O$, a decision is made $y_t \in B$ and as a result the system is transferred into a (possibly new) state $z_t \in V$. The system can then be formalized as:

Definition 3.3.10. Let N be the set of positive integers, let $X = O^N$, let $Y = B^N$ and let $Z = V^N$. A document management system is $\Sigma(O, B, W, z_0) \subseteq X \times Y \times Z$ where z_0 is the initial state of the system. An *appearance* of $\Sigma(O, B, W, z_0)$ is (x, y, z), $(x, y, z) \in \Sigma(O, B, W, z_0)$ if and only if $(x_t, y_t, z_t, z_{t-1}) \in W$ for all $t \in N$.

Access to documents are performed with operations and it is the only way a document and its contents are modified, or transformed into another state. For example reading a document, modifying a document or approving a document is an operation. In this case, approving a document also changes the current state of the document.

An *operation* is a function $o : V \to V$ that transfers the system from a state to a (possibly new) state. Intuitively, it modifies the contents of a document, its metadata, changes the current state of the document or the state of the system. Formally it is defined as:

Definition 3.3.11. Let $O = \{o_1, \ldots, o_k\}$ be a set of operations. For states $v, v' \in V$, there is an unique integer $i, 1 \leq i \leq k$, such that $o_i(v) = v'$.

By assigning an operation with a rule or a condition, a special type of operation, called authorized operation, is added to the framework. This operation is equivalent to a certified relation, defined by Clark-Wilson, and the enforcement rules requires that an authorized operation is associated with a subject that performs the operation.

An *authorized operation* is a function $\rho : O \times S \times V \rightarrow B \times V$, i.e. an authorized operation takes an operation, a subject and a state; and then determines if the operation meets the condition of a rule (decision). Then it moves the system to a (possibly different) state.

Definition 3.3.12. Let $\omega = \{\rho_1, \ldots, \rho_m\}$[6] be a set of authorized operations. For operation $o \in O$, subject $s \in S$, decision $b \in B$, and states $v, v' \in V, (o, b, v, v') \in W(\omega)$ if and only if there is an unique integer $i, 1 \leq i \leq m$, such that $\rho_i(o, s, v) = (b, v')$.

An authorized operation is for example an authorized read operation, an operation that requires the reader to have both access to the document and have the correct security clearance to access the information.

A secure document management system is a system where only authorized operations on a document are performed. That is, no rules specified are violated and the following theorem is established.

Theorem 3.3.1. *Let $\Sigma(O, B, W, z_0)$ be a document management system with a secure initial state z_0. A* secure *document management system is a system, where every action $(o, b, v, v') \in W(\omega)$, i.e. only authorized operations are performed.*

The formal security policy consists of a set of authorized operations and the rules associated with the operations. Rules are specified with a rule-based policy specification language [38], closely related to a logical programming language.

3.3.2 Rule-Based Policy Specification Language

The basis of a rule-based specification language are functions, where a function defines a rule. As all operations, e.g. reading, writing and deletion of a document must be authorized, rules specify and enforce this authorization. These rules are defined with a clausal logic, and the rules are defined with a declarative language which binds logic with rules:

Definition 3.3.13. A *clause*, also known as a *rule*, takes the form:

$$H \leftarrow B$$

where H stands for rule head and B for rule body.

The body B defines the conditions that must be met for the authorization rule defined in the head H. This can be compared with an assertion. For example to allow Alice to access the document `Foo` the following clause or rule must be true:

$$access(\mathtt{r}, \mathtt{Alice}, \mathtt{Foo}) \quad \leftarrow \quad in(\mathtt{r}, permissions(\mathtt{Alice}, \mathtt{Foo}))$$

It is not limited to handle boolean functions only, e.g. the function called permissions in the example above returns a set. However, it is essential that the resulting function in the body is a boolean function. The body can be any valid logical expression, i.e. it is possible to combine several conditions. For example:

[6] ω is the Greek letter omega.

$$H \leftarrow c_1 \wedge c_2 \wedge \cdots \wedge c_n$$

A rule that specifies that Alice can only read the documents $d \in D$ which Bob also can read, can be written the following way:

$$\begin{aligned} access(\texttt{r}, \texttt{Alice}, d) \quad \leftarrow \quad & in(\texttt{r}, permissions(\texttt{Alice}, d)) \wedge \\ & in(\texttt{r}, permissions(\texttt{Bob}, d)) \end{aligned}$$

The framework provides a set of predefined rules, defined as functions. There are three types of functions: *specification functions*, *utility functions* and *authorization functions*. A specification function is a predefined function that is used to define a rule. The function *permissions*() is an example of a specification function. Utility functions are useful functions not specific for this framework.

Authorization functions represent the predefined set of rules in this framework which can be associated with operations to create authorized operations. There are two types of authorization functions: *basic authorization rules* and *derived authorization rules*. Basic rules is only based on specification functions or utility functions. When basic rules are combined to form a rule, it is called a derived rule. If necessary, the framework can be extended by adding new derived rules which are specified in the security policy. All predefined authorization functions are presented in Table 3.4 on page 41.

3.3.3 Specification Functions

Some of the predefined specification functions are explained in more depth in this section. Table 3.1 on the facing page has the complete list of specification functions and all utility functions are shown in Table 3.2 on page 40.

Because the framework represents documents and a requirement for authorized operations is that the operation is associated with a subject (e.g. a person), it must ensure that identification of the subject is correct. It suggests a specification function that returns true if a subject s is correctly identified:

$$authenticated : S \rightarrow \{\texttt{true}, \texttt{false}\}$$

A subject can also be a program, suggesting that the authentication mechanisms may vary depending on type of subject.

Access control is defined with an access control matrix, e.g. example of a matrix is shown in Table 3.3 on page 40. The following function retrieves an entry from the matrix, where the types of permissions are $T = \{\texttt{r}, \texttt{w}, \texttt{d}\}$:

$$permissions : S \times D \rightarrow 2^T$$

The codomain is the power set of T, for example $permissions(\texttt{Alice}, \texttt{Bar})$ of the example matrix (Table 3.3) give $\{\texttt{r}, \texttt{d}\}$.

$$permissions(s, d) = \quad T, \quad (s, d, T) \in \{\text{access control matrix}\}$$

To determine whether a security level (c, k) dominates another security level (c', k'), the following specification function can be used:

$$dom : (C \times K) \times (C \times K) \rightarrow \{\texttt{true}, \texttt{false}\}$$

Table 3.1: Specification Functions

Functions	$B = \{\texttt{true}, \texttt{false}\}$	Semantics
$M(d)$	$D \rightarrow M$	Returns the metadata of a document d.
state(d)	$D \rightarrow \Sigma_{doc}$	Returns the current state of a document d.
dom(l, l')	$(C \times K) \times (C \times K) \rightarrow B$	Returns true if level l dominates l'
level(p)	$S \cup D \rightarrow C \times K$	Returns the level of principal p.
permissions(s, d)	$S \times D \rightarrow 2^T$	Return the entry for subject s and document d in the access control matrix.
trusted(s)	$S \rightarrow B$	Return true if the subject s is a trusted subject.
authenticated(s)	$S \rightarrow B$	Return true if subject s is authenticated correctly.
expires(t)		Returns true if the duration t expires, that is t is less than the value of current time.
Document Functions		
subdocuments(d)	$D \rightarrow D$	Return all documents included in document d.
duration(d)	$m_{duration} \in M(d)$	Returns the duration constraint assigned to the document d.
creator(d)	$m_{author} \in M(d)$	Returns the subject who initiated the document d
cancelled(d)	$\sigma_{cancelled} \in state(d)$	Returns true if the document d has been cancelled.
approved(d)	$\sigma_{approved} \in state(d)$	Returns true if the document d is approved.
archived(d)	$\sigma_{archived} \in state(d)$	Returns true if the document d is archived.

The function is defined below and based on the *dom* relation defined in the Bell-LaPadula model:

$$dom((c,k),(c',k')) = \begin{cases} \texttt{true}, & \text{if } c' \leq c \wedge k' \subseteq k \\ \texttt{false}, & \text{otherwise} \end{cases}$$

3.3.4 Basic Authorization Rules

The specification functions are used to form the basic authorization rules provided by this framework. The first rule enforce discretionary access control and is defined as:

Rule 3.3.1. The discretionary access control for a subject s to a document d, is defined

Table 3.2: Utility Functions

Functions	Semantics
in(x, y)	Returns true if x is a member of y.
equals(x, y)	Return true if $x = y$.
forall(a, A, f)	Return true if function f on all elements a in A is true.

Table 3.3: Example of an access control matrix

	Foo	Bar	Doc1	Doc2
Alice	`rwd`	`r-d`	`r--`	`rw-`
Bob	`rwd`	`r-d`	`r--`	`rw-`
Trudy	`-wd`	`--d`	`r--`	`rwd`

with the following rule. The variable r defines the type of access rights ($r \in \{\mathtt{r}, \mathtt{w}, \mathtt{d}\}$):

$$\begin{aligned} access(r, s, d) \quad \leftarrow \quad & authenticated(s) \wedge \\ & in(r, permissions(s, d)) \end{aligned}$$

Second and third rule enforce parts of the NRU and NWD properties defined by Bell-LaPadula.

Rule 3.3.2. The simple security condition (no reads up) of Bell-LaPadula model for a subject s and a document d is defined as:

$$\begin{aligned} read\_up(s, d) \quad \leftarrow \quad & authenticated(s) \wedge \\ & dom(level(s), level(d)) \wedge \\ & forall(e, subdocuments(d), read\_up(s, e)) \end{aligned}$$

Rule 3.3.3. The star property (no writes down) of Bell-LaPadula model for a subject s and a document d is defined as:

$$\begin{aligned} write\_down(s, d) \quad \leftarrow \quad & authenticated(s) \wedge \\ & dom(level(d), level(s)) \wedge \\ & forall(e, subdocuments(d), write\_down(s, e)) \end{aligned}$$

3.3.5 Derived Authorization Rules

Rule 3.3.4. Whether a subject s can read the document d or not is specified with the rule:

$$\begin{aligned} can\_read(s, d) \quad \leftarrow \quad & access(\mathtt{r}, s, d) \wedge \\ & read\_up(s, d) \wedge \\ & \neg cancelled(d) \end{aligned}$$

Table 3.4: Authorization Functions

Basic Functions	Derived Functions	Semantics
access(t, s, d)		Defined by rule 3.3.1
read_up(s, d)		Defined by rule 3.3.2
write_down(s, d)		Defined by rule 3.3.3
	can_read(s, d)	Defined by rule 3.3.4
	can_modify(s, d)	Defined by rule 3.3.5
	can_delete(s, d)	Defined by rule 3.3.6
	can_approve(s, d)	Defined by rule 3.3.7
	can_reclassify(s, d, l)	Defined by rule 3.3.8

Rule 3.3.5. A subject s can modify a document d if the following is true:

$$\begin{aligned} can\_modify(s,d) \quad \leftarrow \quad & access(\mathtt{w},s,d) \wedge \\ & write\_down(s,d) \wedge \\ & \neg approved(d) \wedge \\ & \neg archived(d) \end{aligned}$$

Rule 3.3.6. The document d can be removed by subject s only if one of the following holds:

$$\begin{aligned} can\_delete(s,d) \quad \leftarrow \quad & access(\mathtt{d},s,d) \wedge \\ & archived(d) \wedge expires(duration(d)) \\ can\_delete(s,d) \quad \leftarrow \quad & access(\mathtt{d},s,d) \wedge \\ & \neg archived(d) \wedge approved(d) \wedge cancelled(d) \end{aligned}$$

Rule 3.3.7. A subject s can only approve a document d if the following is true:

$$\begin{aligned} can\_approve(s,d) \quad \leftarrow \quad & can\_read(s,d) \wedge \\ & \neg approved(d) \end{aligned}$$

Rule 3.3.8. A document d can be reclassified with level l by the subject s if and only if the following conditions hold[7]:

$$\begin{aligned} can\_reclassify(s,d,l) \quad \leftarrow \quad & access(\mathtt{r},s,d) \wedge \\ & trusted(s) \wedge \\ & \neg archived(d) \end{aligned}$$

3.3.6 Policy Specification

This framework can then be used to specify a security policy, which defines what is allowed and not allowed in a specific document management system. The structure of a security policy is not defined but should include at least the following parts:

[7]Level l is not used at the moment, but saved for future use.

- **Information Flow**: All security levels and conditions for the discretionary access control are specified here.
- **Operations**: This is the central part of the policy, where all available operations associated with their authorization rules are defined.
- **Derived Rules**: This section is optional. All added rules, derived from the framework are defined here. By deriving new rules the framework can be extended.

3.4 Security Policy Validation

First the framework is examined in light of the security requirements and a discussion of limitations of this framework are held. Then the authorized rules are derived and validated.

3.4.1 Access Control and Information Flow

First requirement stated that:

> Access rights to a document are separated from the access rights to the contents of a document.

This requirement is satisfied by adapting the Bell-LaPadula model of secure information flow. The access rights to a document are specified in an access control matrix (discretionary access) but the information is restricted to only subjects with a correct clearance. By using the Bell-LaPadula model, the second requirement is also satisfied.

> The contents of a document have a security level classification

A classification that gives access to information on a need-to-know basis. This is achieved by requiring all documents to have the m_{level} metadata. The information flow model is enforced with the *can_read* and the can_modify rule, of which a deeper discussion is held later in this section.

A consequence of the next requirement is that the framework must handle the *declassification problem*.

> A documents security level can change during the lifetime of a system.

The solution is to have a set of trusted subjects that remove all sensitive information from a higher level object before it is down-classified to a lower level, enforced by the $can\_reclassify$ rule which require the reclassifying subject to be trusted. A trusted user implement the principle of weak tranquility[8]. Last requirement stated the need for aggregated documents:

> A document can include other documents thus access to these documents must follow the same condition in (a)[9].

Satisfied by requiring all subdocuments of a document to meet the same information rules as the main document, i.e. a subject can only read a document if it also can read all subdocuments.

[8] Defined in subsection 2.3.6

[9] With (a) being the first requirement.

3.4.2 Integrity

The version control is intentionally not included in this framework. In this framework, a document version is equivalent with a document, i.e. no distinction between a document and a document version is made. A succeeding (or replacing) version of a document is simply a copy of a document with some modifications.

An approved document can not be altered.

A consequence of this requirement is that an approved document can never be altered (enforced by the can_modify rule), which at first look seems as a great limitation. A suggested solution is that a modification to an approved document (called a revision) is performed by creating a new document version, i.e. a copy of the document with the approved flag removed. Approval of the revision is performed exactly the same way as a document approval is performed. The next requirements deal with integrity of archived documents.

An archived document can not be altered.

An archived document can not be deleted before it has expired.

Both the can_modify rule and the can_delete rule enforce the above requirements. The last requirement below is enforced with the $can\_reclassify$ rule which require that a document to be reclassified is not archived.

The security level of a document can not be altered if the document is archived.

For consistency it is required (but not specified) that an approver have read access to the document to be approved and that the document is not approved. This is enforced by the $can\_approve$ rule. All rules above require a subject, thus the enforcement rules of Clark-Wilson model are satisfied.

3.4.3 Traceability

The defined metadata variables, together with the specification functions, enforce the specified traceability properties.

All documents have a system-unique identity.

All documents have an originating creator.

All approved documents have an approver identification.

They infer no specific rules but the information provided can be used by (possibly authorized) operations in the system.

3.4.4 Rules

To show that the rules, provided by this framework, really enforce the above discussed properties all rules are fully derived.

The *can_read* Rule

A derivation (3.1) of the rule give the following:

$$
\begin{array}{rcl}
can\_read(s,d) & \leftarrow & access(\mathtt{r},s,d) \;\wedge \\
 & & read\_up(s,d) \;\wedge \\
 & & \neg cancelled(d) \\
 & & \Downarrow \\
can\_read(s,d) & \leftarrow & \underbrace{authenticated(s) \wedge in(\mathtt{r}, permissions(s,d))}_{access()} \;\wedge \\
 & & read\_up(s,d) \;\wedge \\
 & & \neg cancelled(d) \\
 & & \Downarrow \\
can\_read(s,d) & \leftarrow & authenticated(s) \wedge in(\mathtt{r}, permissions(s,d)) \;\wedge \\
 & & authenticated(s) \;\wedge \\
 & & dom(level(s), level(d)) \;\wedge \\
 & & forall(e, subdocuments(d), read\_up(s,e)) \;\wedge \\
 & & \neg cancelled(d) \\
 & & \Downarrow \\
can\_read(s,d) & \leftarrow & authenticated(s) \wedge in(\mathtt{r}, permissions(s,d)) \;\wedge \\
 & & dom(level(s), level(d)) \;\wedge \\
 & & \neg cancelled(d) \;\wedge \\
 & & read\_up(s,e_1) \wedge \ldots \wedge read\_up(s,e_n) \\
 & & \Downarrow \\
can\_read(s,d) & \leftarrow & authenticated(s) \wedge in(\mathtt{r}, permissions(s,d)) \;\wedge \\
 & & dom(level(s), level(d)) \;\wedge \\
 & & \neg cancelled(d) \;\wedge \\
 & & dom(level(s), level(e_1)) \wedge \ldots \wedge dom(level(s), level(e_n)) \;\wedge \\
 & & \neg cancelled(e_1) \wedge \ldots \wedge \neg cancelled(e_n) \;\wedge \\
 & & forall(e_{1_1}, subdocuments(e_1), read\_up(s,e_{1_1})) \;\wedge \\
 & & \ldots \;\wedge \\
 & & forall(e_{n_1}, subdocuments(e_n), read\_up(s,e_{n_1})) \\
 & & \Downarrow \\
 & & \vdots
\end{array}
\tag{3.1}
$$

An authorized operation that uses this rule must supply a subject to the operation, a requirement to satisfy the enforcement rule (ER2) of Clark-Wilson model which specifies that all operations are associated with a subject. Derivation in (3.1) also show that enforcement rule (ER3) is satisfied with the specification function *authenticated*.

Discretionary read access is required, but as the derivation shows, only for the main document. If a subject has the right to read a document, it can access all subdocuments as well. However, the information flow model controls the subject's access to the contents of these subdocuments. Not only the information in a subdocument are controlled, but also the information in a subdocument's subdocument are controlled.

The NRU property in Bell-LaPadula model is satisfied by the *dom* specification function and as the derivation shows a subject s must dominate a document d and all its subdocuments.

The *can_modify* Rule

This rule is derived (3.2) in a similar way as previous rule.

$$
\begin{array}{rcll}
can\_modify(s,d) & \leftarrow & access(\mathtt{w},s,d) & \wedge \\
 & & write\_down(s,d) & \wedge \\
 & & \neg approved(d) \wedge \neg archived(d) & \\
 & & \Downarrow & \\
can\_modify(s,d) & \leftarrow & authenticated(s) \wedge in(\mathtt{w}, permissions(s,d)) & \wedge \\
 & & dom(level(d), level(s)) & \wedge \\
 & & \neg approved(d) \wedge \neg archived(d) & \wedge \\
 & & dom(level(e_1), level(s)) \wedge \ldots \wedge dom(level(e_n), level(s)) & \wedge \\
 & & \neg approved(e_1) \wedge \ldots \wedge \neg approved(e_n) & \wedge \\
 & & \neg archived(e_1) \wedge \ldots \wedge \neg archived(e_n) & \wedge \\
 & & forall(e_{1_1}, subdocuments(e_1), write\_down(s, e_{1_1})) & \wedge \\
 & & \ldots & \wedge \\
 & & forall(e_{n_1}, subdocuments(e_n), write\_down(s, e_{n_1})) & \\
 & & \Downarrow & \\
 & & \vdots &
\end{array}
\tag{3.2}
$$

It is clear that no modifications can be made to an approved or archived document. This might seem as a big limitation, but as discussed earlier, it is solved by creating a copy of the document which is not approved, i.e. an approved version of a document can never be modified but a new version (revision) can.

Similar to the previous rule, this rule also requires that the operation using this rule is performed by a subject. A rule to satisfy the enforcement rules of Clark-Wilson model.

To prevent information leakage to lower security levels, the NWD property of Bell-LaPadula model is enforced by the specification function *write_down*. The derivation (3.2) shows that all documents (including subdocuments) must dominate the subject.

The *can_delete* Rule

The rule can be rewritten as followed:

$$
\begin{array}{rcll}
can\_delete(s,d) & \leftarrow & access(\mathtt{w},s,d) & \wedge \\
 & & [(archived(d) \wedge expires(duration(d))) \vee & \\
 & & (\neg archived(d) \wedge approved(d) \wedge cancelled(d))] & \\
 & & \Downarrow & \\
can\_delete(s,d) & \leftarrow & authenticated(s) \wedge in(\mathtt{d}, permissions(s,d)) & \wedge \\
 & & [(archived(d) \wedge expires(duration(d))) \vee & \\
 & & (\neg archived(d) \wedge approved(d) \wedge cancelled(d))] &
\end{array}
\tag{3.3}
$$

The derivation shows that a prerequisite is that a subject has the rights to delete a document. If the document is archived it must have expired before it can be deleted or else it must be both approved and cancelled.

The *can_approve* Rule

A straight-forward derivation (3.4) gives:

$$
\begin{array}{lcll}
can\_approve(s,d) & \leftarrow & authenticated(s) \wedge in(\mathtt{r}, permissions(s,d)) & \wedge \\
 & & dom(level(s), level(d)) & \wedge \\
 & & \neg cancelled(d) & \wedge \\
 & & \neg approved(d) & \wedge \\
 & & dom(level(s), level(e_1)) \wedge \ldots \wedge dom(level(s), level(e_n)) & \wedge \\
 & & \neg cancelled(e_1) \wedge \ldots \wedge \neg cancelled(e_n) & \wedge \\
 & & forall(e_{1_1}, subdocuments(e_1), read\_up(s, e_{1_1})) & \wedge \\
 & & \ldots & \wedge \\
 & & forall(e_{n_1}, subdocuments(e_n), read\_up(s, e_{n_1})) & \\
 & & \Downarrow & \\
 & & \vdots &
\end{array}
\tag{3.4}
$$

It is very similar to the *can_read* rule with the additional requirement that the document is not already approved. For consistency an already approved document can not be approved.

The *can_reclassify* Rule

The principle of weak tranquility, introduced by Bell-LaPadula, deals with the declassification problem. A derivation gives:

$$
\begin{array}{lcll}
can\_reclassify(s,d,l) & \leftarrow & authenticated(s) \wedge in(\mathtt{r}, permissions(s,d)) & \wedge \\
 & & trusted(s) & \wedge \\
 & & \neg archived(d) &
\end{array}
\tag{3.5}
$$

A reclassification can only be performed by a trusted subject which must have discretionary read access to the document.

3.5 Security Policy Example

To understand how this framework can be used, an example of a simple document management system is presented in this section. The system is first described and then a security policy for that system is presented.

A common demand is documentation of meetings. In many organizations it is a requirement and part of the decision process. It could be a board meeting, a project meeting or an annual meeting of a non-profit organization. To handle this process electronically, a document management system is developed. The goal of this system is to manage creation, approval and archivation of all meeting protocols. A non-functional requirement is that it should follow the SS-EN 82045 standard.

To keep it simple, a protocol is either *non-public* or *public*. Information in a non-public protocol should only be readable by the members of the organization and a public protocol can be read by everyone. Figure 3.2 on the next page describes the order of these security classifications.

An organization can be organized into several departments and to add some level of confinement we also add categories to a protocol. A single category represents a department. The security level of a protocol now consists of a security classification and a category. For example, a non-public protocol d for the economy department can have the level:

Figure 3.2: The two security classifications are arranged with the highest classification at the top

$$level(d) = (\texttt{NONPUBLIC}, \{\texttt{ECON}\})$$

A subject s that can read this protocol could have the following level:

$$level(s) = (\texttt{NONPUBLIC}, \{\texttt{ECON}, \texttt{HR}\})$$

But the subject s below is not allowed to access the information:

$$level(s) = (\texttt{NONPUBLIC}, \{\texttt{HR}, \texttt{DEVEL}\})$$

Another simplification is that no discretionary access control exists, thus all protocols are accessible by everyone. However, the classifications still prevent unauthorized access to the information. In practice this could mean that all files are stored in a world-accessible repository or filesystem. This means that all entries in the access control matrix have the following entry $\{\texttt{r}, \texttt{w}, \texttt{d}\}$ and the access function always evaluates to true, because:

$$in(t, permissions(s, d)) \Rightarrow in(t, \{\texttt{r}, \texttt{w}, \texttt{d}\})$$

To prevent meeting protocols to be falsely reclassified, e.g. released to the public, the principle of separation of privilege is used, i.e. two trusted subjects are required for reclassification. How a trusted subject is chosen is not part of this document management system, and not part of the security policy.

It is also not necessary to handle document versions because there only exists one version of a meeting protocol. All errors are corrected before the protocol is adjusted (approved) and it should not be allowed to modify an adjusted protocol. All these requirements are specified by the following security policy for a document management system to handle meeting protocols.

3.5.1 Meeting Protocol Management Policy

Information Flow

Protocols are assigned a security level. A level consists of a security classification and a category.

Information Flow 3.5.1. Let C be the set of security classifications. The set C is a totally ordered set under the relation $<$, $\{\texttt{NONPUBLIC}, \texttt{PUBLIC}\}$ where $\texttt{PUBLIC} < \texttt{NONPUBLIC}$.

Information Flow 3.5.2. Let Θ be the set of departments in the organization, then the set K of categories is the power set of Θ, $K = 2^{\Theta}$.

For example if the organization consists of the three departments: economy (ECON), development (DEVEL) and human relations (HR) available categories are: {}, {ECON}, {DEVEL}, {HR}, {ECON, DEVEL}, {ECON, HR}, {DEVEL, HR} and {ECON, DEVEL, HR}.

The access permissions for all subjects $s_i \in S, i = 1, \ldots, k$ and protocols $d_j \in D, j = 1, \ldots, n$ are defined with an access control matrix shown in Table 3.5 on the following page.

Table 3.5: Access control matrix in a meeting protocol management system

	d_1	d_2	...	d_n
s_1	rwd	rwd	...	rwd
⋮	⋮	⋮	⋮	
s_k	rwd	rwd	...	rwd

Table 3.6: Operations in a meeting protocol management system

Operation		Description
initiate(s, d)		Initiate protocol d with classification of subject s and protocol author is set to s. Defined by operation 3.5.1
Authorized Operation	Rule	
modify(s, d)	$can\_modify(s, d)$	Operation 3.5.2
read(s, d)	$can\_read(s, d)$	Operation 3.5.3
adjust(s_1, s_2, d)	$der\_can\_approve(s_1, s_2, d)$	Operation 3.5.4
reclassify(s_1, s_2, d, l)	$der\_can\_reclassify(s_1, s_2, d, l)$	Operation 3.5.5

Operations

A complete list of operations and associated rules are specified in Table 3.6 and the operation semantics are defined below:

Operation 3.5.1. A subject s can with the operation $o_{initiate}$ create a new meeting protocol. Document d is assigned the same security level $level(d) = level(s)$ as the subject, metadata m_{author} is set to s and the document state is set to $\sigma = \sigma_{initiated}$.

Operation 3.5.2. To modify a meeting protocol d, a subject s uses the operation o_{modify}. A modification is any change to the contents of a document.

Operation 3.5.3. Reading a protocol d by subject s, is performed with the operation o_{read}. This means that the subject s access the contents of a protocol without modifying it.

Operation 3.5.4. Approving a protocol d is performed by the subjects s_1 and s_2 with the operation o_{adjust}. The document state $\sigma = \sigma_{approved}$.

Operation 3.5.5. Reclassifying a protocol d to security level l is performed by the subjects s_1 and s_2 with the operation $o_{reclassify}$. The operation sets $level(d) = l$.

Derived Rules

Rule 3.5.1. A protocol d can be approved by the subjects s_1 and s_2 if and only if the following rule is satisfied:

$$\begin{aligned} der\_can\_approve(s_1, s_2, d) \quad \leftarrow \quad & can\_approve(s_1, d) \wedge \\ & can\_approve(s_2, d) \wedge \\ & \neg equals(s_1, s_2) \end{aligned}$$

Rule 3.5.2. A protocol d can be reclassified with level l by the trusted subjects s_1 and s_2 if and only if the following rule is satisfied:

$$\begin{aligned} der\_can\_reclassify(s_1, s_2, d, l) \quad \leftarrow \quad & can\_reclassify(s_1, d, l) \wedge \\ & can\_reclassify(s_2, d, l) \wedge \\ & \neg equals(s_1, s_2) \end{aligned}$$

Chapter 4

Security Policy for Infologic

At the end of the last chapter was an example of how a security policy for an artificial document management system could look like. Now, let us focus on the real task at hand, using this framework to develop the security policy for Infologic. We begin with a detailed description of Infologic followed by a threat analysis.

4.1 Detailed Description of Infologic

Zert Infologic is a program to facilitate the development of product manuals in multiple languages. Contents of a document are separated from the layout with advanced stylesheets that specify how information are presented. A document can be published in various formats, for example PDF, HTML or Microsoft Word.

The process of developing a product manual begins with defining the document structure in an initial language called source language. A document is an aggregation of phrases, figures and tables. All phrases and documents are stored in a content repository and since a document is an aggregation of phrases they can be reused in another document. This means that if a translation is created for a phrase and an identical phrase is used in another document, the translation is reused.

Next step is to publish the document, the step where a stylesheet is applied to the document. The stylesheet specifies how the information in a document is presented. The last step is to convert the document to a distributable format, e.g. a PDF document or an HTML page.

Either a phrase or an entire document can be exported for translation, i.e. extracted from the content repository. Translation can be performed by an external agency or internally at the company using this document management system. Then, a translation linked to a phrase or a document, can be imported and stored in the repository. The information is structured as a tree.

4.1.1 Document Structure

A product manual is an aggregation of phrases, figures and tables. A phrase (or paragraph) is also a document (subdocument) that can contain other subdocuments. In the remainder of this paper, a document is a subdocument (phrase, paragraph, figure, table, etc).

Figure 4.1 on the following page, shows an example of a document structure where the root node contain a unique identifier for this document. This document has only two

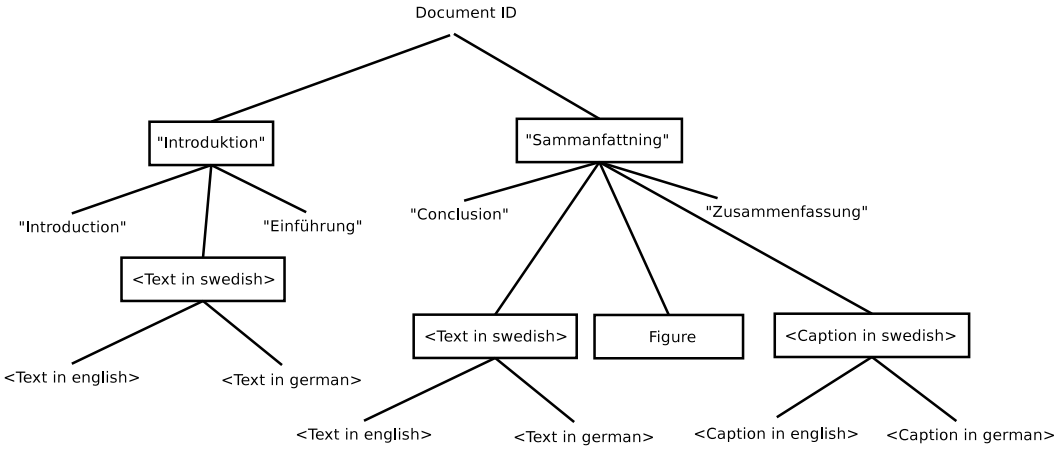

Figure 4.1: Example of a document and its structure in Infologic

sections, an introduction and a conclusion. The introduction contains one paragraph, the conclusion contains one paragraph and a figure with associated caption. The source language in this example is Swedish and squares represent these nodes. Translations are represented as child nodes to the source language node. As the example shows, the section title is in itself a phrase that can be translated. No distinction between a document and a subdocument (phrase, figure or table) has to be made, and all subdocuments also have a unique identification. All documents are stored in a repository that have the structure illustrated in Figure 4.2.

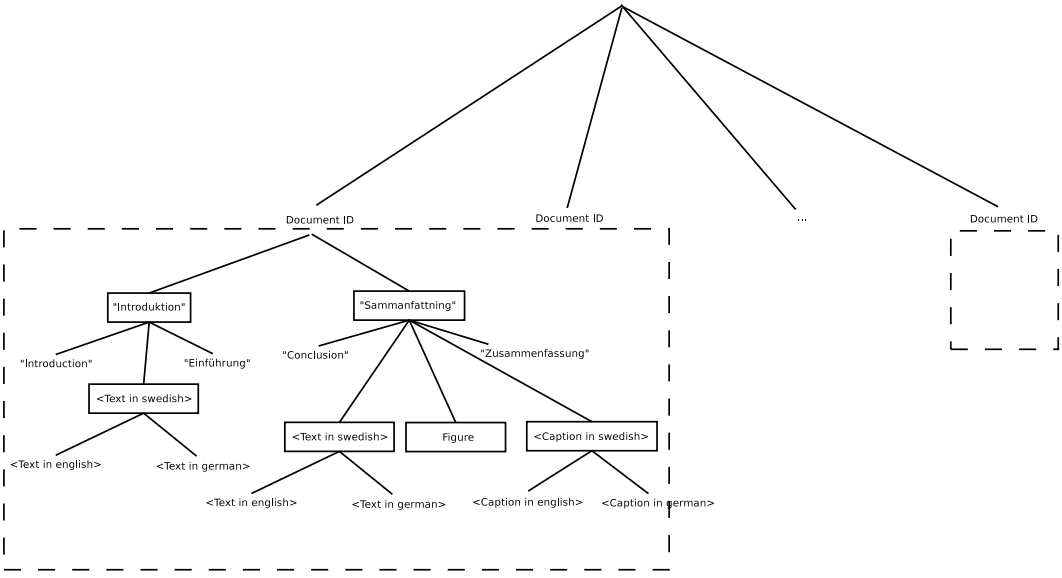

Figure 4.2: Example of how documents are stored in a repository in Infologic

A fundamental requirement is that the document management system must be secure, but to satisfy this it must first be defined what a secure document management system is.

Specifying what is allowed and not allowed in this document management system requires that threats to the system are analyzed first.

4.2 Threat Analysis

To understand the threats to this document management system, it must be placed in a context first. A threat analysis only present the threats and not any solutions. In theory, this system is not limited to handle only product manuals but can also handle other documents that may be available in more than one language. This analysis is limited to a product manual development context.

This document management system will be mainly used internally by the company that develops the product. However, it is common that entire manuals are exported to a translation agency for translation. This means that the agency has access to a complete document in its source language (source manual) yielding the following threat:

> A translator (translation agency) can intentionally or by mistake modify the source manual.

If the product is for example a medical equipment, faulty instructions could cause human injury. Thus, this threat raises an important issue that must be dealt with. Another treath with similar consequences is the following:

> A translator (translation agency) can intentionally or by mistake provide an incorrect translation.

Because a translation is linked to its source language and link information is stored in the translation, a threat is that the translator modifies the link information with the result of translating a totally different phrase (a threat to link integrity). This is analogue to the threat specified above since incorrect link information is the same as an incorrect translation. An incorrect translation or that the wrong phrase is translated can be very difficult to detect.

If a repository is corrupted by a hard drive crash, a destructing virus or in another way corrupted, it is a possibility that link information may be affected, with the consequence of an incorrect translation. This is another threat to the link integrity.

> A corrupted repository could cause incorrect link information resulting in incorrect translations.

A corrupt repository can also cause damage to entire manuals causing information loss. Another way of losing information is that a manual is deleted (by mistake or deliberately).

> A product manual can intentionally or unintentionally be deleted from the repository.

This does not mean that manuals should be unremovable, but some restrictions to what can be removed must be applied. Also, someone can deliberately remove the contents of a manual, that is remove all text in a phrase or insert blanks. In this case, information is also lost even when the manual still exists, in other words not deleted.

> The contents of a product manual can be overwritten or removed.

All manuals are not by default write protected but, as with removable manuals, restrictions to which manuals can be modified are applied.

A large company can have several departments each developing a product independently of the others, and how information can flow among them are restricted. If all departments share the same repository a threat to the restricted information flow exists.

> Information can leak to a department if the repository is shared between departments.

Another similar threat is that information can leak to a competitor or someone else outside the organization. This is a threat because a product manual in its development state could reveal technical details about the product.

> Information can leak to the public or to a competeting organization.

Either the repository, where the manuals are stored, is compromised or an "evil" translation agency could sell the information to a competitor. It could also be an employee that leaks the information to the public or a competitor.

These threats can be grouped into two categories: integrity and information flow (confidentiality).

4.2.1 Integrity

These are the threats to the integrity of the product manuals.

1. A translator (translation agency) can intentionally or by mistake modify the source manual.
2. A translator (translation agency) can intentionally or by mistake provide an incorrect translation.
3. A corrupted repository could cause incorrect link information resulting in incorrect translations.
4. A product manual can intentionally or unintentionally be deleted from the repository.
5. The contents of a product manual can be overwritten or removed.

4.2.2 Information Flow

The threats to the restricted information flow are the following:

1. Information can leak to a department if the repository is shared between departments.
2. Information can leak to the public or to a competeting organization.

A threat is referred to as $I(3)$ for threat 3 of the integrity category and $C(1)$ refers to threat 1 of the information flow category. These threats lay a foundation to what should be allowed and not allowed in a secure document management system for product manuals. This is formalized and specified in the developed security policy.

4.3 Infologic Security Policy

Based on these type of threats, a suitable model to define security should focus on integrity and confidentiality. A model that focus on both integrity and confidentiality is the document management framework introduced in this thesis, providing us with a framework suitable to specify the security policy. The policy only specifies what is allowed and not allowed, and does not discuss enforcing issues. The security mechanisms to enforce the policy are discussed later in this document.

The document management framework provided a formal definition[1] of the objects in a general document management system, and these definitions define the objects in this system. The framework defines that a document is an entity that has information, metadata and subdocuments. Information is generally defined as the contents of a document, which means that for this system, information is either a phrase / paragraph, a table or a figure. A translation is a subdocument to the source language document. Figure 4.3 shows how the document in Figure 4.1 on page 52 is described with the document management framework.

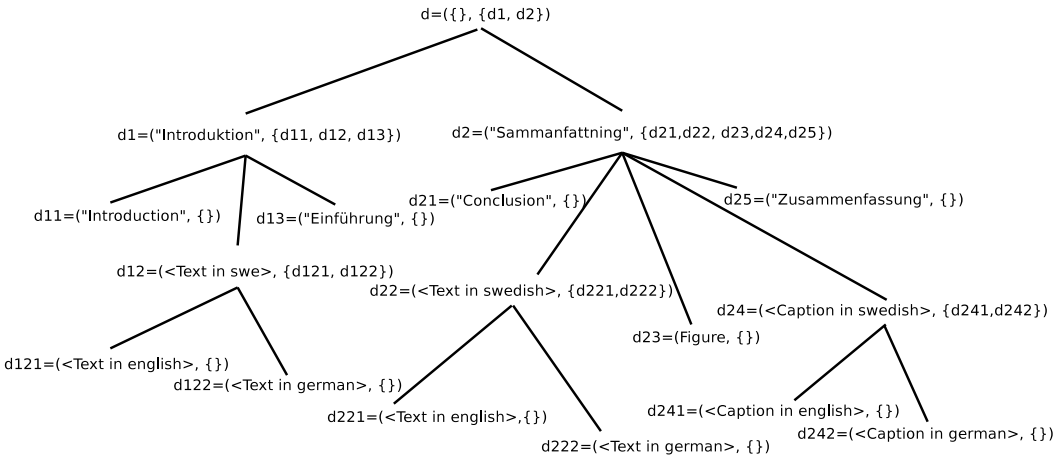

Figure 4.3: A document expressed with the document management framework

Both threats $C(1)$ and $C(2)$ require that the flow of information is restricted, and the document management framework requires that security classifications are developed. Another requirement added by the framework is that all subjects are authenticated.

To prevent information leakage it must first be defined what type of information flow are allowed. The framework classifies all information with a security level and all users are assigned a security level. Product manuals (until published) are stored in a repository in Infologic. Access to the repository requires authentication and access to a single document is specified with an access control matrix. Subjects in this matrix are the writers of product manuals and objects are the manuals.

Information can not leave Infologic without passing an information barrier[2], for example documents exported for translation must pass the barrier first (Figure 4.4 on the following page).

The barrier is built of three types of trusted subjects: a *translation-proxy*, an *approver* and a *publisher*. Figure 4.5 on the next page show a more detailed picture of how information flow

[1] See subsection 3.3.1

[2] The barrier is only a conceptual barrier and not a physical entity.

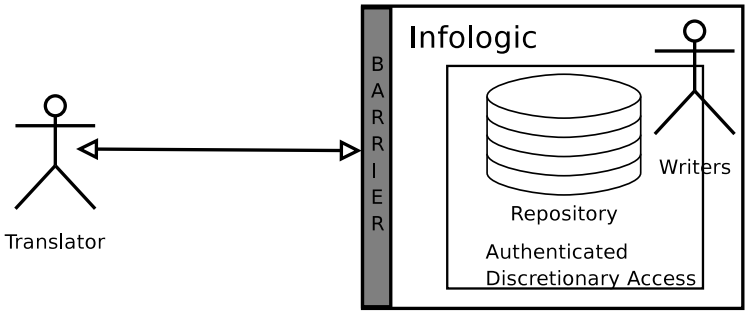

Figure 4.4: Overview of information flow in Infologic

from Infologic to a translator or the public. A product manual exported for translation first passes the translation-proxy and is then passed to the translator. On import, the translator pass the translation to the translation-proxy and then forward it to the repository. Approval of a manual is performed by a trusted subject in the barrier and information never leave the system, in other words the information is bounced against the information barrier. Releasing a product manual (publishing) is performed by the publisher subject.

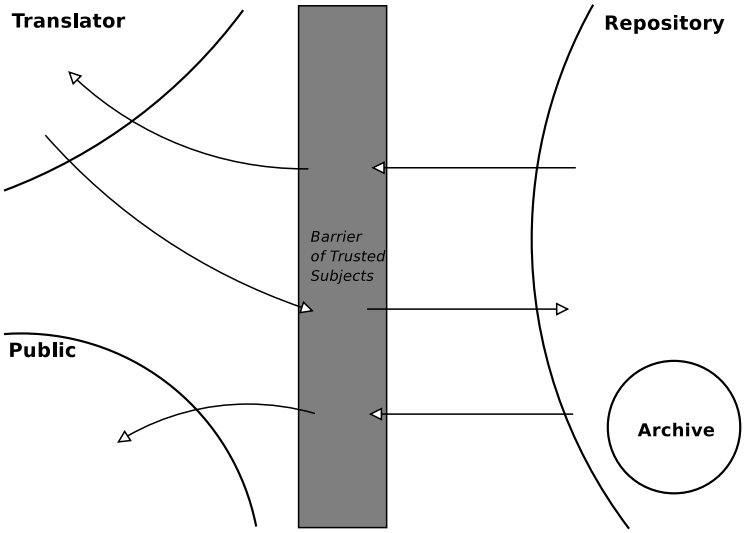

Figure 4.5: A more detailed figure of how information flow from Infologic

The translation-proxy is responsible for reclassification and validation. It can be done by either a human or a trusted program or a combination of both. The manual is classified in such way that the translator can read this source language manual but not being able to modify it.

The approver is responsible for approving a product manual and this action is performed by a human. An approved manual is flagged as approved and reclassified to assure that the manual can not be modified by any writer in the system.

The publisher is the last trusted subject of the information barrier. It is responsible for releasing the product manual to the public. Basically this is a declassification of the infor-

Table 4.1: Principals (subjects and objects) in Infologic. Italic indicates trusted subjects

Subjects	**Objects**
	Document
	Exported Document
	Translation
Writer	
Translator	
Public	
Translation-proxy	
Approver	
Publisher	

Table 4.2: Example of an access control matrix in Infologic

	d1	d12	d21	d421
Alice (w)	`rwd`	`r-d`	`r--`	`rw-`
Bob (w)	`rwd`	`r-d`	`r--`	`rw-`
Trudy (w)	`-wd`	`--d`	`r--`	`rwd`

mation in a product manual so the public can read it. It also validates that the publishing manual is approved, preventing that unapproved (unfinished) documents are released to the public (including competitors).

4.3.1 Information Flow

This design suggests the principals defined in Table 4.1. All objects (product manuals) and subjects are assigned a security level. A level consists of a security classification and a category. Writers' access to the manuals in the repository are specified with an access control matrix. Table 4.2 shows an example of such matrix.

The security classifications are H (high) and L (low) where high is a higher classification than low. The public is assigned the `L`-classification, and all unpublished manuals have the `H`-classification thus the public can only read published manuals. The system has initially only two categories: `TRANS` (translation) and `REPO` (repository). If a shared repository is used, that is the repository is shared with other departments, a repository category `REPO_DEP` for each department can be added. For simplicity we have only one repository category `REPO` here. Table 4.3 shows available classifications and Table 4.4 all category sets. Keep in mind that a subject must have discretionary access to a document first.
The subjects are assigned a security level shown in Table 4.5 on page 59. Unpublished manuals and its aggregated subdocuments in the repository are assigned the security level (H, { `REPO` }) to prevent that Public (L, {}) can read these documents, because Public does not dominate the manual as $H > L$ and { `REPO` } is not a subset of { }.

On export of a document, it is first reclassified by the trusted subject called translation-proxy. It removes the category classification, for example a document with (H, { `REPO` }) is reclassified as (H, { }) and then sent to a translator. A translator is assigned security

Table 4.3: Security classifications ordered by the highest classification at the top

Classification
H (high)
L (low)

Table 4.4: Category combinations in Infologic

Category sets
{ TRANS, REPO }
{ TRANS }
{ REPO }
{ }

level (H, { TRANS }) that dominates the exported document as $H \leq H$ and $\{\} \subseteq$ { TRANS }, and this specifies that a translator is not allowed to modify an exported document. The no-write-down property defines that writing to an object at lower security level is not allowed but the no-read-up property is satisfied thus a translator can read the exported document. The exported document is translated and the translation is assigned the security level (H, { TRANS }). The translation is passed to the translation-proxy and reclassified to (H, { REPO }). A translation can not be directly imported since a writer (H, { REPO }) would not be able to read the translation as { TRANS } is not a subset of { REPO } and it also specifies that the public is not allowed to read these documents. Figure 4.6 illustrates this exporting and importing routine.

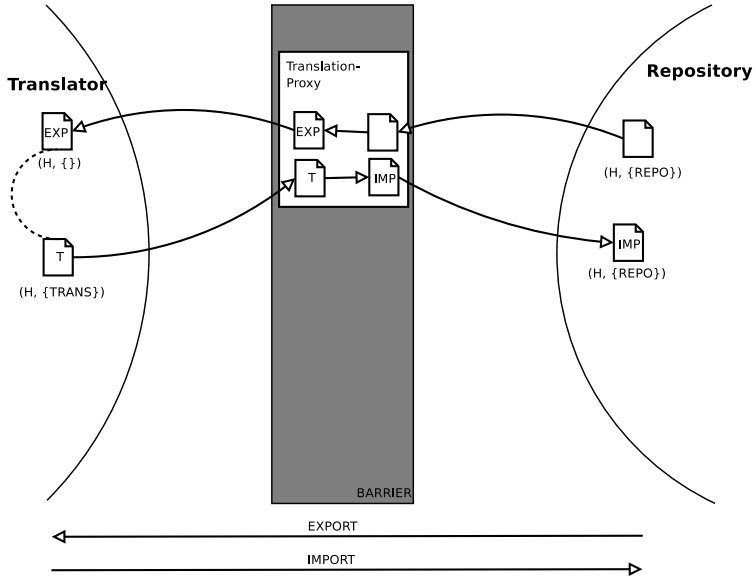

Figure 4.6: Exporting and importing documents

Table 4.5: The security levels of the principals in Infologic

Subjects	**Objects**	**Security Level**
	Document	(H, { `REPO` })
	Exported Document	(H, { })
	Translation	(H, { `TRANS` })
Writer		(H, { `REPO` })
Translator		(H, { `TRANS` })
Public		(L, { })
Translation-proxy		(H, { `REPO`, `TRANS` })
Approver		(H, { `REPO` })
Publisher		(L, { `REPO` })

Approval of a document is performed by a trusted approver subject. It is assigned the level (H, { `REPO` }) to be able to read the documents. An approved document is reclassified to level (L, { `REPO` }), and due to the no-write-down property, this specifies that an approved document can not be modified. However, a writer can still read the document as it dominates the approved document. Still, a public subject can not read these documents because they do not dominate the document as { `REPO` } is not a subset of { }.

The trusted publisher subject is assigned the level (L, { `REPO` }) which specifies that a publisher can only read approved documents. A publisher removes all categories from the document to be published and because the document already has L-classification the result of this declassification is (L, { }). It is now possible for the public to read the document[3]. Figure 4.7 illustrates this procedure.

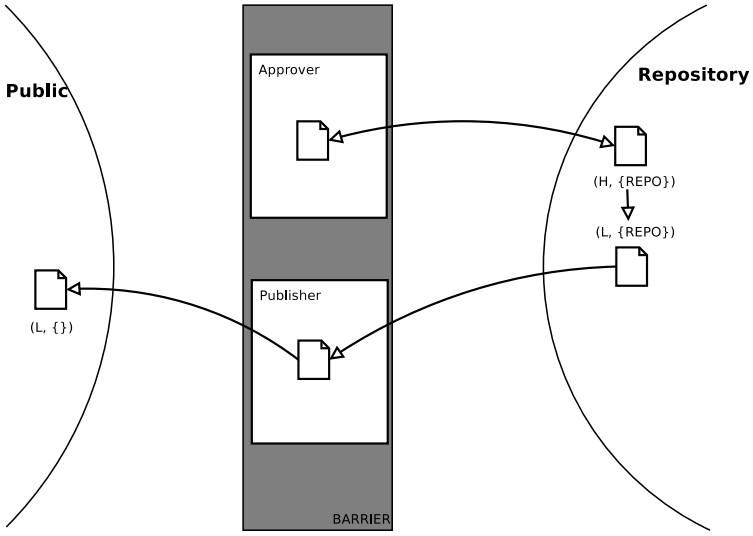

Figure 4.7: Approval and publishing procedures

[3]A problem is that the policy does not disallow that the public can modify the document. In practice, this is very difficult to prevent and thereof not specified by this policy. Later in this document a mechanism to detect these modifications are discussed.

This part of the policy specified the principals in Infologic (Table 4.5 on the page before), their security classifications and categories (Table 4.3 on page 58 and Table 4.4 on page 58) and the discretionary access, to show how information is allowed to flow in Infologic. Next part is to specify the operations and the rules for these operations.

4.3.2 Operations

The semantics of the operations in Infologic are informally described and the rules to perform an operation are defined here. This is the fundamental part of the policy, i.e. the part that specify what must be satisfied for all operations to maintain a secure system, based on the framework introduced and defined in section 3.3.

Table 4.6: Operations

Operation	Rule	Description
create(s, d, d_{parent})	$der\_can\_create(s, d, d_{parent})$	Create a new document (Operation 4.3.1 and rule 4.3.1)
read(s, d)	$can\_read(s, d)$	Read a document (Operation 4.3.2)
modify(s, d)	$can\_modify(s, d)$	Modify a document (Operation 4.3.3)
copy(s, d_{src}, d_{dest})	$der\_can\_copy(s, d_{src}, d_{dest})$	Copy a document (Operation 4.3.4 and rule 4.3.2)
delete(s, d)	$can\_delete(s, d)$	Delete a document (Operation 4.3.5)
archive(s, d, t)	$der\_can\_archive(s, d)$	Archive a document (Operation 4.3.6 and rule 4.3.3)
export(s, d)	$der\_can\_export(s, d)$	(Operation 4.3.7 and rule 4.3.4)
import(s, d, d_{parent})	$der\_can\_import(s, d, d_{parent})$	(Operation 4.3.8 and rule 4.3.5)
approve(s, d)	$der\_can\_approve(s, d)$	(Operation 4.3.9 and rule 4.3.6)
publish(s, d)	$der\_can\_publish(s, d)$	(Operation 4.3.10 and rule 4.3.7)

A complete list of operations and associated rules is found in Table 4.6 and a detailed description of each operation is presented below:

Operation 4.3.1. A subject s can with the operation o_{create}, create a new document d that is stored as a subdocument to d_{parent}. Document d is assigned the same security level as s, i.e. $level(d) = level(s)$. Also, $m_{author} = s$ which means that the author of the document is set to d and the state of d is set as initiated. The document d is added to the access control matrix.

Operation 4.3.2. Reading a document d by subject s, is performed with the operation o_{read}. This means that the subject s access the contents of a document (information) without modifying it.

Operation 4.3.3. To modify a document d, a subject s uses the operation o_{modify}. A modification is any change to the contents of a document.

Operation 4.3.4. Copying a document d_{src} to d_{dest} is performed with the operation o_{copy}. The operation is performed by a subject s. The contents of d_{src} is copied and stored in d_{dest} which requires that d_{dest} must have been created first.

Operation 4.3.5. The operation o_{delete} deletes a document d and is performed by subject s. A deletion is all types of document removal from the repository.

Operation 4.3.6. A document can be archived to ensure durability and to prevent deletion. Archiving a document d has a consequence that the document d can not be included in another document. It is performed by a subject s and operation $o_{archive}$. The state of d is set as archived and a date of expiration t is set.

Operation 4.3.7. The operation o_{export} removes all security categories from a document d, i.e. downgrading d to a lower security level. The security classification is intact and the operation is performed by a subject s.

Operation 4.3.8. Importing a document d to the repository is performed by subject s with the operation o_{import}. On import, the document d is reclassified as $(H, \{\texttt{REPO}\})$ and added to the access control matrix.

Operation 4.3.9. Approving a document d is performed with the operation $o_{approve}$, an operation that declassify d from H (high) to L (low) and the document state is set as approved.

Operation 4.3.10. The operation $o_{publish}$ removes all categories and sets the lowest security classification on the document d to be published.

A secure document management system is a system where only authorized operations are performed. An authorized operation is an operation associated with a rule provided by the framework or a derived rule. A derived rule is an extension to the framework.

4.3.3 Derived Rules

All rules that are not included in the framework are defined here with the rule-based policy specification language described in subsection 3.3.2.

Rule 4.3.1. A document d can be created by subject s and stored as subdocument of d_{parent} if and only if the following rule holds:

$$\begin{aligned} der\_can\_create(s, d, d_{parent}) \leftarrow\ & can\_read(s, d_{parent}) \wedge \\ & can\_modify(s, d_{parent}) \wedge \\ & \neg in(d, subdocuments(d_{parent})) \end{aligned}$$

Creating a document requires that; a subject must have read access both to the parent document and its contents, the parent document is not archived, cancelled or approved and the document is not already a subdocument to the parent document.

Rule 4.3.2. The contents of d_{src} can be copied into d_{dest} by subject s if and only if the following rule is satisfied:

$$\begin{aligned} der\_can\_copy(s, d_{src}, d_{dest}) \leftarrow\ & can\_read(s, d_{src}) \wedge \\ & can\_modify(s, d_{dest}) \end{aligned}$$

A subject copying a document must have read access to the source document and write permissions to the destination document. Source document have not been cancelled and destination document is not approved or archived. With the specified information flow system, it is clear that only a writer can copy a document at the same security level which is $(H, \{\texttt{REPO}\})$.

Rule 4.3.3. A subject s can archive a document d if and only if:

$$\begin{aligned} der\_can\_archive(s,d) \quad \leftarrow \quad & can\_read(s,d) \quad \wedge \\ & \neg archived(d) \quad \wedge \\ & approved(d) \end{aligned}$$

Rule 4.3.4. A subject s can export a document d if and only if the following holds:

$$\begin{aligned} der\_can\_export(s,d) \quad \leftarrow \quad & can\_reclassify(s,d,(H,\{\})) \quad \wedge \\ & can\_read(s,d) \end{aligned}$$

Only a trusted subject can export a document and the subject must also have read access and the document must not be archived or cancelled.

Rule 4.3.5. A subject s can import a document d and store as subdocument to d_{parent} if and only if the rule below is satisfied:

$$\begin{aligned} der\_can\_import(s,d,d_{parent}) \quad \leftarrow \quad & can\_reclassify(s,d,(H,\{\texttt{REPO}\})) \quad \wedge \\ & can\_modify(s,d_{parent}) \quad \wedge \\ & can\_read(s,d) \end{aligned}$$

Similar to the rule of export, an import requires that it is performed by a trusted subject that can reclassify a document. The document to be imported must not be cancelled and the document which it is imported to is neither approved or archived.

Rule 4.3.6. A subject s can approve a document d if and only if the following rule is satisfied:

$$\begin{aligned} der\_can\_approve(s,d) \quad \leftarrow \quad & can\_reclassify(s,d,(L,\{REPO\})) \quad \wedge \\ & can\_approve(s,d) \end{aligned}$$

Rule 4.3.7. A document d can be published by trusted subject s if and only if the following rule is satisfied:

$$\begin{aligned} der\_can\_publish(s,d) \quad \leftarrow \quad & can\_reclassify(s,d,(L,\{\})) \quad \wedge \\ & can\_read(s,d) \end{aligned}$$

Approving and publishing a document must also be performed by a trusted subject that have access permissions to the document.

In a secure system, only authorized operations are performed. These rules are part of the authorized operation and none of these rules are allowed to be broken. Now that it is specified what is allowed or not, what mechanisms can be used to enforce this policy are discussed.

4.4 Infologic Security Implementation

Security implementation is to enforce the developed security policy. Enforcing a security policy is achieved by some kind of security mechanisms. The suggested mechanisms which can be used to enforce the policy for Infologic are discussed in this section.

4.4.1 Authentication Mechanisms

The policy for Infologic specifies that all operations have a rule that can be derived to the authentication rule, which also is a requirement specified by the Clark Wilson model. This means that some sort of authentication mechanism is required, a mechanism that ensures that a subject is authentic.

For this system it is sufficient for the document writers to authenticate with passwords. To add a little more level of protection, it is required that a password includes both numbers and characters and should be at least 8 characters long.

A trusted subject is also authenticated with a password but in addition to only having a correct password, it must have a valid signing key which is also password protected. To act as a trusted subject it would be required to have both knowledge and entity (the private key). The notion of trust is enforced by requiring that this private key is certified by at least two other subjects, that is the key actually belongs to this trusted subject. The system will not accept a trusted subject without a certified private key.

4.4.2 Access Control Mechanisms

Various techniques exist to implement an access control matrix, and in this application an appropriate solution is to use access control lists[4], that is all documents have a list of users and their access. This is a discretionary access, since the owner of the document specifies this list. The policy states that there are three types of access rights: read, write and delete where each subject in the access list for a document has a combination of these rights.

The mandatory access control is enforced by the security classifications. The goal with having these classifications was to have two walls of protection. If the access control list mechanism is breached (or used in a wrong way) it would still be difficult for a non-authorized subject to read the document. The policy stated that there are only two levels: high (H) and low (L) and to prevent L-classified subjects to read H-classified documents, the H-class documents are encrypted and can only be decrypted by H-classified subjects. This also means that all documents in the repository should be encrypted.

The other part of the mandatory access control, the categories, are enforced by the controlling repository. In implementation, this mechanism does not have to differ so much from the ACL implementation, but a significant difference in usage is that a subject can not control the category of a document. If this mechanism is breached the only protection left is the encryption for H-classed documents.

4.4.3 Non-repudiation Mechanisms

To enforce the barrier of trusted subjects, it must first be discussed what level of trust must be accomplished. What is a trusted subject in this application? When the subject is a program, a trusted program is a program that contains certified code and is (with some method) guaranteed to do only what it is supposed to do. If the subject is a user,

[4]See subsection 2.4.2 about access control lists.

a mechanism to prevent false-deniability can be used. This means that the user can still do "evil" things, but he / she can never falsely deny what has been done, called non-repudiation[5].

In Infologic, the barrier consists of a set of trusted subjects. The subjects are authenticated users and the trust is based on non-repudiation. No documents can be transfered from the repository or approved without a certification from a trusted user. This person, performing one of these operations, cannot deny the action and is responsible for it.

A trusted user has a signature that is applied to all operations. This signature acts like a fingerprint that can be traced. The signature is applied to the operations that requires a trusted user. For example, exporting a document can only be done by a trusted user. This user approves the export by signing the exported document with a secret only available to he or she[6]. This operation is a declassification, and actually violates the information flow restriction, which makes it important that this operation can be traced. Import is performed in the same way.

An approved document is a document in which the contents is validated and approved by a subject. The subject approving a document is also a trusted subject, and in the same way as with export and import, this subject signs the document. The security level of this document specifies that it is not allowed to be modified, since it has a lower security classification than the subjects having access to this document. This can be enforced with the access control mechanisms but as a positive side-effect of the signature, a modification is also detected because signing a document is calculating a unique hash value of the contents which is encrypted with the private key of the signer.

The same is applied to a published document. This document is also signed by a trusted user, and this signature makes it possible to detect if the published document has been modified and redistributed. The authenticity of a published document is checked by recalculating the hash value of the document and compare it with the hash value included in the document. The included hash value is protected with asymmetric encryption[7] and can be decrypted by the signers public key. If the values mismatch, the document has been modified and is not authentic.

4.4.4 Repository Integrity Mechanisms

As documents are an aggregation of documents (subdocuments) it is better to use a tree chaining technique[8]. In tree chaining a Merkle Hash Tree is used where the root is the digest of the entire data and a subset of the data is represented by the children. To verify a subset (a child) only the path to the root needs to be verified.

Thus, the root document of a product manual in the repository contains a digest of the entire product manual. To verify that a phrase or a paragraph (a subdocument) is intact, the path to the root document only needs to be verified. This mechanism detects any anomalies of the storage, caused by bit-decay or malicious viruses.

Archived documents should be stored in a redundant storage, which guarantees that the documents are intact by storing multiple consistent copies of the documents. This can be implemented at hard-drive level with for example a RAID-system. If this technique is used it is no actual need to separate the storage for archived documents or other documents. However, to use the principle of separation of functions it is recommended that the archived

[5] See subsection 2.4.3 about non-repudiation.
[6] See subsection 2.4.3 about document signing with private keys.
[7] See subsection 2.4.3 about asymmetric encryption and authenticity certification.
[8] See subsection 2.4.4.

documents are written to write-only media such as tapes or compact discs.

4.5 Implementation Requirements for Infologic

The suggested requirements (with respect to security) for Infologic are grouped by the MoSCoW-criterias in Dynamic Systems Development Method[9].

4.5.1 Must

- All documents have a unique identification.
- All documents have an owner.
- All documents are assigned a security category.
- All subdocuments have a digest version.
- Every document has a list of subjects with their access permissions (ACL).
- A document is either high-classified or low-classified.
- All high-classified documents are encrypted.
- All high-classified subjects have the decryption key for high-classified documents.
- All subjects are authenticated with a password.
- A trusted subject is a program with certified code or a user that has a private key protected with a password.
- All operations performed by a trusted user are digitally signed with the user's private key.
- The digest value of an exported document is digitally signed by the trusted user (performing the export).
- The digest value of an imported document is digitally signed by the trusted user (performing the import).
- The digest value of an approved document is digitally signed by the trusted user (approving the document).
- The digest value of a published document is digitally signed by the publisher.

4.5.2 Should

- The system controls that a password is at least 8 characters long mixed with numbers and letters.
- The private key of a trusted subject is certified.

[9] `http://www.dsdm.org/tour/`

4.5.3 Could

- A Merkle Hash Tree is implemented for data integrity of the repository (or a single document).
- Archived documents are replicated for redundancy.

4.5.4 Would

- The private key of a trusted subject is certified by at least two other trusted subjects.
- Archived documents are written to write-only media for example compact discs or tape storages.

Chapter 5

Conclusion

The SS-EN82045-1 standard [4] defines a document structure and the procedures that an electronic document management system should use to assure a certain level of quality. This thesis presented an extendable framework for security policy specifications of SS-EN82045-1 compliant document management systems and as a reference implementation, a security policy for a product called Zert Infologic was specified with this framework.

The framework defines an object structure based on the document structure defined in SS-EN82045-1. A framework based on the Bell-LaPadula model of secure information flow [5, 6] and Clark-Wilson model of integrity ([8] according to [3]). A formal security policy defines what is allowed and not allowed by specifying the set of secure and non-secure states in a system. The proposed framework uses operations and rules to specify the secure states, and these rules are written in a rule based policy specification language [38]. Included in the framework are a set of predefined rules (specified as functions) called authorization functions. These provided rules ensure that the SS-EN82045-1 requirements are satisfied and these rules can also be combined to define new rules called derived rules.

The fundamental part of a document management system is the documents and the operations working with these documents. The framework defines that a secure document management system is a system with only authorized operations and where none of the associated rules are broken. An authorized operation is an operation with a rule associated to the operation. The confidentiality aspect of security is handled with the classification system defined by the Bell-LaPadula model and multi-level security is achieved. The integrity aspect is managed with the Clark-Wilson model, a model defining a set of constrained data items (CDIs), integrity verification procedures (IVPs) and transformation procedures (TPs). In this framework all documents are CDIs, an operation is a TP and the rules are the IVPs of the system.

Zert Infologic is a product to facilitate the management of product manuals in multiple languages. A document management system designed to follow the SS-EN82045-1 standard. Security is an important demand by customers and the company developing this product wanted to know what tools or procedures they had to implement to ensure a certain level of security. The method used to answer this question was inspired by the security engineering life cycle: *threat model* → *security policy* → *security implementation* ([2] and [3]). A threat analysis was first performed, followed by a definition of security in Zert Infologic and based on the policy what security mechanisms to implement were suggested.

To define security in Zert Infologic a security policy was specified, a security policy specified with the presented framework. A fundamental principle of security in Zert Infologic

is the notion of trust by requiring a special type of subjects to perform critical operations. These subjects are called trusted subjects. A trusted subject is a program, a tool or a person where all operations can be traced. Traceability and non-repudiation are the principles used to ensure the notion of trust. These principles were specified in the security policy by defining a set of operations and rules. The rules were based on the rules provided by the framework.

5.1 Future work

The framework does not deal with the security aspect called availability, that is the ability to access information or a resource. The framework could be extended by adding a set of predefined rules that can be used to specify a level of availability. To be able to do this it must first be analyzed what the general aspects of availability are. Enforcing availability in distributed systems or other networked applications is a very complex task and deliberately not discussed in this thesis.

Another limitation is that the object structure defined by the framework does not prohibit indirect recursive structures. It must first be determined whether recursive structures do occur or not and if not, a definition that prevents these type of structures can be added.

References

[1] Doan, T., Demurjian, S., Ting, T.C., Ketterl, A.: MAC and UML for secure software design. In: FMSE '04: Proceedings of the 2004 ACM workshop on Formal methods in security engineering, ACM Press (2004) 75–85

[2] Anderson, R.J.: Security Engineering: a guide to building dependable distributed systems. First edn. Wiley Computer Publishing (2001)

[3] Bishop, M.: Computer Security - Art and Science. First edn. Addison Wesley (2003)

[4] Svenska Elektriska Kommisionen: SS-EN 82045-1 Document Management - Part 1: Principles and Methods (2002) Can be obtained from http://www.sis.se/.

[5] Bell, D.E., La Padula, L.J.: Secure computer systems: Mathematical foundations. Technical Report ESD-TR-73-278, Electronic Systems Division, Air Force Systems Command, Hanscom AFB, Bedford, MA (1973)

[6] Bell, D.E., La Padula, L.J.: Secure computer systems: Unified exposition and multics interpretation. Technical Report MTR-2547, MITRE Corporation, Bedford, MA (1976)

[7] Biba, K.: Integrity considerations for secure computer systems. Technical Report MTR-3153, MITRE Corporation, Bedford, MA (1977)

[8] Clark, D., Wilson, D.: A comparison of commercial and military security policies. In: Proceedings of 1987 IEEE Symposium on Security and Privacy. (1987) 184–194

[9] Brewer, D.F.C., Nash, M.J.: The chinese wall security policy. In: IEEE Symposium on Security and Privacy. (1989) 206–214

[10] Devanbu, P., Gertz, M., Martel, C., Stubblebine, S.G.: Authentic third-party data publication. In: Proceedings of the 14th IFIP 11.3 Working Conference in Database Security. (2000) 101–112

[11] Devanbu, P., Gertz, M., Kwong, A., Martel, C., Nuckolls, G., Stubblebne, S.G.: Flexible authentication of xml documents. In: Proceedings of the 8th ACM conference on Computer and Communications Security, ACM Press (2001) 136–145

[12] Liu, V., Caelli, W., Foo, E., Russell, S.: Visually sealed and digitally signed documents. In: Proceedings of the 27th conference on Australasian computer science, Australian Computer Society, Inc. (2004) 287–294

[13] Grimaldi, R.P.: Discrete and Combinatorial Mathematics. Fourth edn. Addison Wesley (1999)

[14] Poulsen, K.: U.S. Air Traffic Control Found Vulnerable. Internet-only newsletter (2004) Retrieved from http://www.securityfocus.com/news/9729.

[15] Poulsen, K.: South Pole 'cyberterrorist' hack wasn't the first. Internet-only newsletter (2004) Retrieved from http://www.securityfocus.com/news/9356.

[16] Bishop, M.: Introduction to Computer Security. First edn. Addison Wesley (2004)

[17] de Vivo, M., de Vivo, G.O., Isern, G.: Internet security attacks at the basic levels. SIGOPS Oper. Syst. Rev. **32** (1998) 4–15

[18] Lampson, B.W.: Protection. In: Proceedings of the Fifth Princeton Symposium on Information Sciences and Systems. (1971) 437–443 reprinted in SIGOPS Oper. Syst. Rev **8**, (1974) 18–24.

[19] Harrison, M.A., Ruzzo, W.L., Ullman, J.D.: Protection in operating systems. Commun. ACM **19** (1976) 461–471

[20] Denning, D.E.: A lattice model of secure information flow. Commun. ACM **19** (1976) 236–243

[21] Volpano, D., Irvine, C.: Secure flow typing. Computers and Security **16** (1997) 137–144

[22] Denning, D.E., Denning, P.J.: Certification of programs for secure information flow. Commun. ACM **20** (1977) 504–513

[23] McLean, J.: A comment on the 'basic security theorem' of bell and lapadula. Inf. Process. Lett. **20** (1985) 67–70

[24] McLean, J.: Reasoning about security models. In: Proceedings of the 1987 IEEE Symposium on Security and Privacy. (1987) 123–131

[25] Bell, D.: Concerning 'modeling' of computer security. In: Proceedings of the 1988 IEEE Symposium on Security and Privacy. (1988) 8–13

[26] Schneier, B.: Applied Cryptography. Second edn. John Wiley and Sons Inc. (1996)

[27] Schneier, B.: Secret and Lies. First edn. John Wiley and Sons Inc. (2000)

[28] Pfleeger, C.P., Pfleeger, S.L.: Security in Computing. Third edn. Prentice Hall (2003)

[29] Waldmann, U., Scheuermann, D., Eckert, C.: Protected transmission of biometric user authentication data for oncard-matching. In: SAC '04: Proceedings of the 2004 ACM symposium on Applied computing, ACM Press (2004) 425–430

[30] Grunbacher, A.: POSIX access control lists on linux. Internet (2004) Retrieved 2004-12-21 from http://www.suse.de/~agruen/acl/linux-acls/online/.

[31] de Vivo, M., de Vivo, G.O., Gonzalez, L.: A brief essay on capabilities. SIGPLAN Not. **30** (1995) 29–36

[32] Rivest, R.L., Shamir, A., Adleman, L.: A method for obtaining digital signatures and public-key cryptosystems. Commun. ACM **21** (1978) 120–126

[33] Atkins, D., Stallings, W., Zimmermann, P.: RFC 1991: PGP message exchange formats (1996) Status: INFORMATIONAL.

[34] Henry, K.: Getting started with pgp. Crossroads **6** (2000) 8

[35] Kent, S.T.: Internet privacy enhanced mail. Commun. ACM **36** (1993) 48–60

[36] Rivest, R.: RFC 1321: The MD5 message-digest algorithm (1992) Status: INFORMATIONAL.

[37] Merkle, R.C.: A certified digital signature. In: Proceedings on Advances in cryptology, Springer-Verlag New York, Inc. (1989) 218–238

[38] Abiteboul, S., Grumbach, S.: A rule-based language with functions and sets. ACM Trans. Database Syst. **16** (1991) 1–30

Appendix A

Set Definitions

- A : The set of information.
- B : The set of possible outcomes.
- C : The set of classifications.
- D : The set of all documents.
- E : The set of all document contents.
- F : The set of all tuples that associates a principal with a classification.
- K : The set of categories.
- L : The set of security levels, $L = C \times K$.
- M : The set of all types of metadata.
- N : The set of positive integers representing time.
- O : The set of operations.
- S : The set of all subjects.
- T : The set of access rights, $T = \{\mathtt{r}, \mathtt{w}, \mathtt{d}\}$
- V : The set of all possible system states.
- W : The set of actions.
- X : The set whose elements x are sequences of operations.
- Y : The set whose elements y are sequences of decisions.
- Z : The set whose elements z are sequences of states.
- Σ_{doc} : All possible states of a document.