Mitigating the Risk of Insider Threats When Sharing Credentials

by

Muntaha NourEddin Qasem Alawneh

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

Information Security Group
Royal Holloway, University of London
2012

Declaration

These doctoral studies were conducted under the supervision of Dr. Allan Tomlinson.

The work presented in this thesis is the result of original research carried out by myself, in collaboration with others, whilst enrolled in the Information Security Group as a candidate for the degree of Doctor of Philosophy. This work has not been submitted for any other degree or award in any other university or educational establishment.

Muntaha NourEddin Qasem Alawneh
May 2012

Abstract

This thesis extends DRM schemes which address the problem of unauthorized proprietary content sharing in home networks to address the problem of unauthorized confidential content sharing in organizations. In particular it focuses on how to achieve secure content sharing between employees in a group while limiting content leakage to unauthorized individuals outside the group. The thesis discusses the main organization types, process workflow and requirements. Our main interest is in organizations which consider content sharing between groups of employees as a fundamental requirement. Achieving secure content sharing requires a deep analysis and understanding of security threats affecting such a fundamental requirement. We study and analyze one of the major threats which affect secure content sharing, which is the threat of content leakage. In this thesis we focus on content leakage which happens when authorized employees share their credentials with others not authorized to access content, thus enabling unauthorized users to access confidential content. Leaking content in this way is what we refer to as content leakage throughout this thesis. We found that to limit the content leakage threat effectively we have to split it into two main categories: *internal leakage* and *external leakage*. In the thesis we define each category, discuss the intersection between the categories, and consider how they can be realized.

Next, we analyze and assess existing content protection schemes, which focus on content sharing and protection from authorized employees misusing their privileges. These mainly include Enterprise Rights Management (ERM) and Digital Rights Management (DRM) schemes. Based on the analysis we identify the weaknesses found in these schemes for mitigating the content leakage threat.

Following that we develop a framework, which we use to mitigate the content leakage threat. This framework is based on the authorized domain concept which was first proposed to address DRM threats. We extend the authorized domain concept so that it consists of a group of devices owned by an organization, whose employees need to share a pool of content amongst each other, e.g. a group of individuals working on a project. In other words, we group devices and content together in a controlled and secure environment. In this thesis, we propose two types of domains: the global domain and the dynamic domain that we use to address the identified content leakage threats. The proposed schemes allow secure content sharing between devices in a dynamic and global domain, and limit the leakage of content to devices outside the domain.

Next, we extend our study to cover secure information sharing not only within a single organization but also to cover this important requirement within collaborating organizations. We then describe and analyze how the content leakage threat can be realized between collaborating organizations. We propose a scheme to control content sharing and, simultaneously, to limit the effect of content leakage when an organization needs to collaborate with other organizations.

Acknowledgements

I would like to thank my supervisor Dr. Allan Tomlinson for his supervision, help, support, and encouragement. I am grateful to Prof. Keith Martin, Dr. Jason Crampton, and Prof. Chris Mitchell for their useful discussion and comments. I also wants to thank my examiners Dr. Andrew Simpson and Dr. Stefan Poslad for their valuable comments.

I would like to thank my parents, who have always encouraged and supported me. Finally, I want to thank my husband for his support and patience throughout the time I have worked on this thesis, as it would have been impossible, without his help, to have a family and engage in full time study at the same time.

Contents

List of Figures

Chapter 1

Introduction

Contents

This chapter gives an overview of the thesis. We provide the motivation for the research and describe the contributions of this thesis. We also present the overall structure of the thesis.

1.1 Motivation

Organizations have always faced challenges in protecting confidential information from being revealed to unauthorized parties. Such challenges are even harder to address when an organization needs such confidential content to be shared between its own employees to achieve a particular task. Threats from outside the organization are always a major worry; however, organizations cannot ignore the risk of authorized employees, or *'insiders'*, revealing confidential content to unauthorized parties. This is because insiders may have privileges and know where to obtain confidential content from within the organization. Thus the risk associated with insider threats is often greater than the risk of threats originating from outside the organization

[63, 70]. Insiders can use different ways to leak content to unauthorized parties; e.g. by sharing their credentials with an unauthorized party. Sharing credentials allows unauthorized users to access content using an authorized employee's credentials. This problem is what we refer to as **content leakage** throughout this thesis (what we mean by content is defined Section in 4.2.1).

1.1.1 Research Question

Our research question is *"Can the content leakage threat be limited when insiders share their credentials in organizations?"*

The problem of content leakage by insiders sharing their credentials is the primary motivation behind the research presented in this thesis. We start by studying this problem, and its impact. We then consider current content protection schemes to assess how they may mitigate this threat. During our research we found that this problem has been considered in different fields of research for different types of content. The area of research which focuses on addressing content leakage when users share the means to access content is Digital Rights Management (DRM). DRM technology was proposed to address the problem of proprietary content piracy for home networks [7, 8]. This problem starts when proprietary content with the means of accessing this content is shared with an uncontrolled number of unauthorized users.

Organizations often manage the sharing and protection of content by controlling what authorized employees are allowed to do with confidential content. This is typically achieved by defining access rights that restrict what authorized employees can or cannot do with content. Widely discussed access control schemes such as Discretionary Access Control (DAC) [38] and Mandatory Access Control (MAC) [69] protect content by enforcing access rights where content is stored. However, when content is copied to another device, the access rights are not copied with the content; this means that the access rights are no longer enforceable on other devices. Such schemes are good enough to satisfy content protection requirements where content is not transferred between users [70]. However, with conventional operating system access control systems, once content leaves the device where it is stored it becomes disassociated from its access rights and thus loses its protection.

The demand for distributing digital content between an organization's departments and the need for content to be shared between employees without affecting its protection motivates the need for schemes which are capable of protecting content even after it has been distributed or shared. Such schemes are analyzed and discussed in detail in this thesis, for example Enterprise Rights Management (ERM) schemes [11, 31, 52, 57]. ERM schemes attempt to expand the policy enforcement point to cover not only where content is stored, but also where it is subsequently sent and used. Although there are major differences between traditional access control schemes and ERM schemes, they both aim to protect content from unauthorized usage. Within any organization, traditional access control schemes and ERM schemes rely on an employee's credentials to authenticate users and provide them with proper authorization rights. But what if authorized employees share their credentials with an unauthorized party? Are these schemes still capable of protecting organizations' confidential content? During our research we found that neither traditional access control schemes nor ERM schemes address the problem of protecting content from authorized employees if they share their credentials with unauthorized parties.

All the above motivates us to consider the potential of applying DRM schemes within organizations to mitigate the content leakage threat. DRM schemes, as we discuss in Chapter 3, do not meet some of the organization requirements, e.g. dynamic structure and workflow. However, we found that we can modify DRM schemes so that they can be integrated in organizations to limit the leaking of confidential content when authorized users share their credentials.

Our proposed scheme can be applied to ERM schemes to limit content leakage by controlling access to content. This thesis is not concerned with fine-grained access rights such as the number of times content could be accessed, forwarded or printed. ERM schemes (see, for example, [11, 31, 52, 57]) focus on rights management which can be integrated with our proposed schemes to manage access rights for employees in the organization. We discuss this point in more detail in Chapter 3.

During our research we observed that there are wider applications suffering from similar content leakage problems. We therefore extended our proposed solution to address similar leakage problems within wider application domain: collaborating organizations.

1.2 Contributions and Publications

The proposals in this thesis are different from DRM. However, they build on part of the work proposed by a previous student's thesis in DRM and extend this to develop a framework for controlling access to content within enterprise organizations. This was accomplished by developing a new set of protocols and a key management scheme. We now briefly describe the differences and similarities between this thesis and the DRM work.

Initial DRM schemes were mainly focused on protecting content distributor rights by binding content to a specific device. However, such schemes ignored many consumer and copyright law requirements (e.g. enabling licence holders to use their content on their devices using a single licence file similar to the physical media distribution). To meet this requirement, DRM schemes integrate the authorized domain concept which was originally proposed in [41]. DRM with Authorized Domain schemes focused on satisfying consumer rights; however, there were still some gaps in satisfying the copyright requirements. This is because the domain was not bound to the domain owner (i.e. licence holder) and any user can join the domain to use the copyrighted content. The work of [6, 7, 8, 10] proposed four mechanisms to strongly bind the domain owner with the authorized domain and to manage the number of devices within the authorized domain.

The starting point for the work presented in this thesis is built on one of the previously proposed four mechanisms, namely the Authorized Domain concept, which uses a master control device. Our work develops the concept of the master control device and the authorized domain management. As stated earlier, DRM schemes were proposed for a different environment (i.e. home networks) and different types of content than ERM schemes. Thus we have two different environments with different properties which result in the need to change the way DRM with authorized domain functions so that it can satisfy the requirements for an organizational environment.

In this thesis we have extended the DRM authorized domain concept to the *dynamic domain concept*, we have changed the way the master controller is used so that it can manage the organization domain, and we update the way devices are bound to the domain. In other words we have used some of the DRM basic entities to propose a new scheme which reduces the impact of authorized users leaking the organization's content by sharing their credentials with unauthorized parties. A more detailed analysis and discussion about the boundaries between the two areas — where each starts and ends — is provided in Chapter 3 and Chapter 9.

This thesis includes the following novel contributions (illustrated in Figure 1.1).

1. It proposes content leakage categories (i.e. internal leakage and external leakage) and their definitions. This contribution is published in [16].

2. It proposes the dynamic domain concept to limit internal leakage. This contribution is published in [15].

3. It proposes the global domain concept to limit insider leakage. This contribution is published in [14].

4. It proposes the dynamic domain and global domain schemes workflow which is used to mitigate the content leakage threat when users share their credentials in organizations.

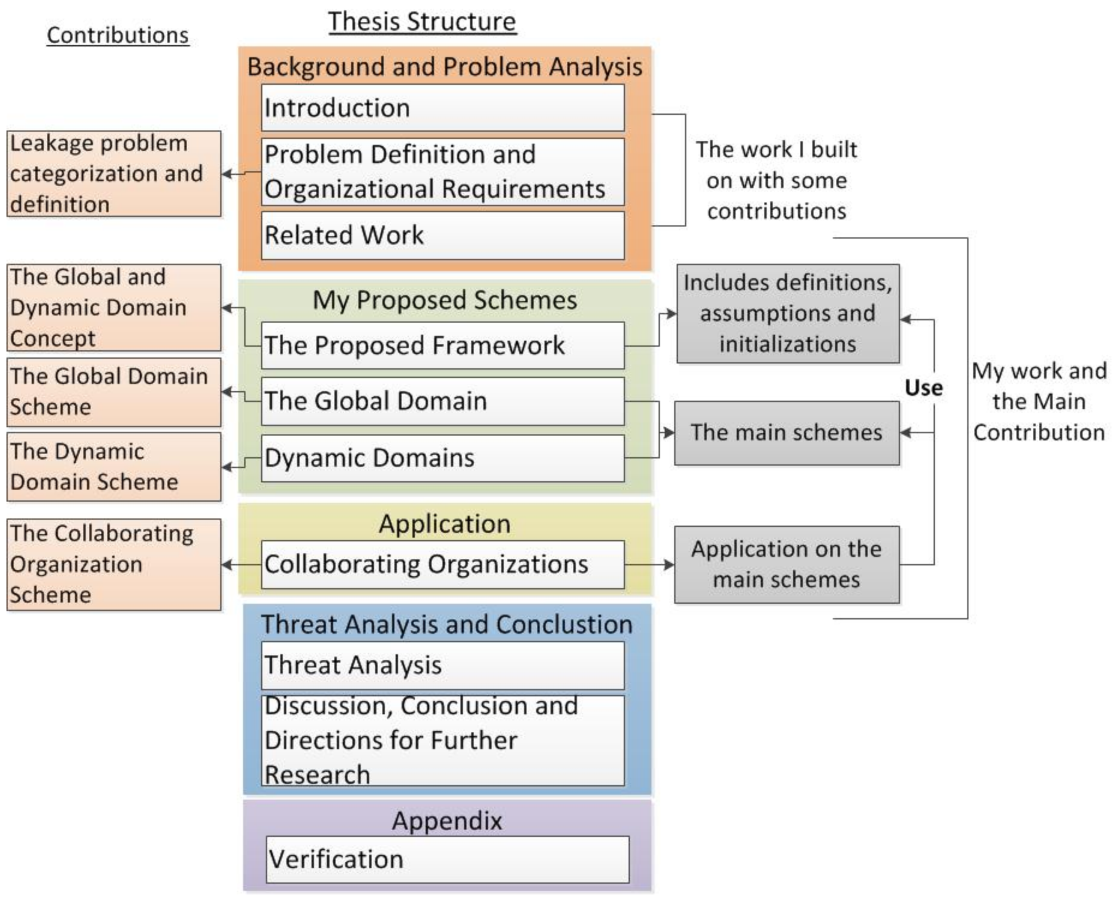

Contributions
Thesis Structure
Background and Problem Analysis
Introduction
Problem Definition and Organizational Requirements
Related Work
Leakage problem categorization and definition
The work I built on with some contributions
My Proposed Schemes
The Proposed Framework
The Global Domain
Dynamic Domains
The Global and Dynamic Domain Concept
The Global Domain Scheme
The Dynamic Domain Scheme
Includes definitions, assumptions and initializations
The main schemes
Use
My work and the Main Contribution
Application
Collaborating Organizations
The Collaborating Organization Scheme
Application on the main schemes
Threat Analysis and Conclustion
Threat Analysis
Discussion, Conclusion and Directions for Further Research
Appendix
Verification

Figure 1.1: Thesis Structure, Summary and Contributions

5. It extends the dynamic and global domain schemes to consider content sharing between collaborating organizations. We provide a scheme to manage the exchange of content between two collaborating organizations and to mitigate the effects of the resulting content leakage threat. This contribution is published in [14].

1.2.1 List of Publications

A number of publications have contributed to this thesis as follows:

1. **Muntaha Alawneh**, Imad M. Abbadi: Combining DRM with Trusted Computing for Effective Information Access Management. PTITS 2008 [13].

   I made the following contribution which is mainly covered in Chapters 2 and 3.

   I proposed the idea of using DRM in other application domains, e.g. to protect content in organizations and healthcare, and described the DRM model and how it could possibly be used to protect content.

2. **Muntaha Alawneh**, Imad M. Abbadi: Sharing but Protecting Content Against Internal Leakage for Organisations. DBSec 2008: 238-253 [15].

   I made the following contributions which are mainly covered in Chapter 6:

   I proposed the dynamic domain concept for protecting content in organization.

   I extended the DRM's master controller and the software applications' functions to be capable of managing the newly proposed dynamic domain. The detailed differences between the DRM master controller and the new scheme is covered in Chapter 3 and Chapter 9. However, the description of our master control and the software application functions are discussed in the chapters where I need to use them, i.e. Chapters 4, 6, 5, and 7.

   I also proposed the dynamic domain scheme workflow which is used to mitigate content leakage threat when users share their credentials in organizations.

3. **Muntaha Alawneh**, Imad M. Abbadi: Preventing information leakage between collaborating organisations. ICEC 2008: 38 [14].

   I made the following contributions which are covered in Chapter 7:

   I developed the dynamic domain concept to apply to collaborating organizations.

   I also extended the global domain concept to collaborating organizations to address the problem of insider leakage when users share their credentials.

   This work draws upon existing schemes which establish secure channels (e.g. VPN) by integrating Trusted Computing concepts to establish trust between organizations end points.

4. **Muntaha Alawneh**, Imad M. Abbadi: Analyzing Insiders and their Threats in Organizations. SPIoT 2011 [16].

   I made the following contribution which are included in Chapter 2.

I updated current definition of insider and extended it to differentiate between insiders and malicious insiders. It also defines internal and external leakage.

1.3 Organization of the Thesis

This thesis consists of four main parts: background and problem definition; proposed schemes; application; and threat analysis and conclusion (Figure 1.1 illustrates these and emphasizes our contribution).

The first part (i.e. background and problem definition) consists of two chapters: Chapter 2 defines the thesis problem and the system requirements, while Chapter 3 analyzes the most commonly discussed schemes for secure content sharing and the DRM related work which we build on. Chapter 3 also provides an overview of Trusted Computing technology since this forms the basis of our enhanced protection scheme.

The second part (i.e. the proposed schemes) consists of three chapters: Chapter 4, 5 and 6, which cover my own work as follows. Chapter 4 (the proposed framework) defines the terms, assumptions, and initialization protocols which are used in the proposed schemes. Chapter 5 (the global domain) covers the global domain scheme and Chapter 6 (dynamic domains) covers the dynamic domain scheme. The last two chapters use the definitions, assumptions, and initialization protocols provided in Chapter 4.

The third part (i.e. application) consists of Chapter 7 (collaborating organizations) which provides an application that *extends* the proposed dynamic and global domain schemes to cover content leakage threats that arise between collaborating organizations.

The fourth part (i.e. threat analysis and conclusion) consists of Chapters 8 and 9. Chapter 8 (threat analysis) provides a threat analysis of all the proposed schemes, Chapter 9 (Discussion, limitations, and further research) concludes the thesis, provides a comparison between our schemes and DRM schemes, discusses the thesis limitations, and provides future research directions.

The last part (i.e. the verification) consists of the Appendix. The appendix proposes, as a proof of concept, a prototype and its execution output for the initialization algorithms and the global domain algorithms discussed in this thesis.

Part I

Background and Problem Analysis

Chapter 2

Problem Definition and Organizational Requirements

Contents

In this chapter we start by defining insiders and insider threats. We then define what we mean by content leakage by insiders when they share credentials. This is the central problem of our research. Following that we discuss the main types of enterprise organizations and provide a generic model for such organizations. Based on this we identify the fundamental requirements for the type of organization we are interested in.

2.1 Introduction

Enterprises need to trust their employees to carry out the enterprise's business in a workflow that achieves their business objectives. Trusting internal employees, in itself, does not guarantee protection of the enterprise's confidential assets. Therefore a robust system should exist to organize the enterprise's internal assets and, at the same time, prevent content leaks. These leaks may be caused either accidentally or deliberately by insiders giving content to unauthorized entities. In this chapter we explore this problem in detail by first discussing what we mean by 'insiders' and 'content leakage' in the context of this research. At this stage content is considered to be any confidential corporate information; a more detailed definition is provided in Chapter 4.

We believe that finding a practical solution to this problem requires a careful understanding of different organization types and their requirements. Therefore in this chapter we describe the main types of enterprise organizations. We conclude that different organization types have common entities and fundamental requirements. Based on this, we provide a generic model for an enterprise organization that is composed of common entities and which uses a common workflow. In this we highlight the main organizational requirements that we are interested in.

2.2 Problem Definition

2.2.1 Insider Definition

Insider problems are often cited as one of the most serious security problems and the most difficult problem to deal with [19]. The most common notion of 'insiders' is that they are internal employees working for an organization. However, when we look at modern enterprise businesses we find that there are many employees working for an organization who are not direct employees of that organization. For example, recent technologies enable organizations from different countries to collaborate to achieve common goals; this is the case with grid and cloud computing. This collaboration might enable an employee from one organization to have partial access to a collaborating organization's data to achieve a specific task. As another example, many organizations outsource part of their activities (e.g. IT services), and employ contractors. Such examples illustrate that it is too simplistic to say that only direct employees of an organization are the 'insiders' [29]. Therefore, we find that we need a more detailed definition of an 'insider'.

The Oxford English Dictionary defines an insider as "a person who is within a society, organization, etc". Although this is a general definition and it is not supposed to be used as an information security term, and it stresses that an insider is an internal user.

Within the academic community we find a diversity of definitions of the term ‘insider’ [22]. The summary of the 2008 Dagstuhl Seminar on countering insider threats [19] proposes several definitions of an insider as follows.

1. Someone defined with respect to a resource, leading to degrees of ‘insiderness’.
2. Somebody with legitimate, past or present, access to resources.
3. A wholly or partially trusted subject.
4. A system user who can misuse privileges.
5. Someone with authorized access who might attempt unauthorized removal or sabotage of critical assets or who could aid outsiders in doing so.

While the above definitions may be adequate for a simple self-contained organization, they have some shortcomings when considering more complex modern enterprises. Moreover, there are more complicated cases for insiders which may not be clear in the previous definitions. For example, when an authorized employee shares his credentials with an unauthorized individual, the unauthorized individual may subsequently use the credentials to leak confidential content from the employee’s organization. In this case, it is not clear whether this is an ‘insider’ attack or an external ‘masquerade’ attack, and in this case who is considered to be the insider: the authorized employee or the unauthorized individual? The previous definitions of insiders need to be refined to cover this and similar cases.

Another, more general, definition of an insider is *“someone with access, privilege, or knowledge of information systems and services”*[22]. This definition considers both external and internal users who possess authorized access or knowledge as insiders. However this definition is perhaps too general, as it does not explicitly state the conditions under which the knowledge or information was obtained.

From all these definitions we may deduce that any person who has access to confidential enterprise content may be considered an ‘insider’. The question then arises as to the status of employees of an organization who do not normally have access to confidential content: should they still be considered as ‘insiders’ when assessing ‘insider threats’?

When an organization requires someone to work on confidential content, it would typically carry out a background check on the user. The extent of this will depend on how confidential the content is. The individual may then have to sign some statement agreeing to behave in a trustworthy manner. The organization would

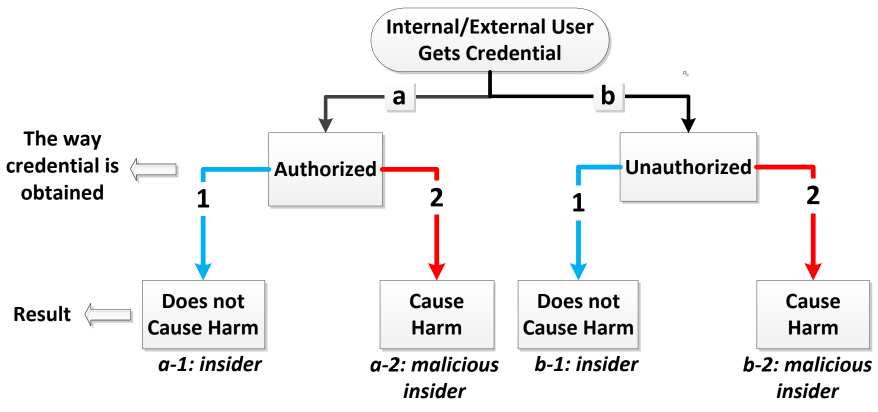


Figure 2.1: Factors Affecting Insiders Definition

then provide the user with credentials enabling them to access the content required to carry out their agreed duties. This is the point when an organization should consider the user as an insider. It is our contention that *having valid credentials* is an important requirement when considering whether a user is an insider or not. This requirement needs to be part of the insider definition.

To extend the previous definitions of an 'insider' we consider four main factors. Figure 2.1 illustrates these in a conceptual diagram, which has the following main elements.

1. A user's relation with the organization whilst performing an action on the confidential information or content. The user could be either internal or external to the organization. In the context of our thesis both internal and external users could be insiders.

2. The method used to obtain credentials. This could be either authorized or unauthorized. Authorized means that credentials are granted to the user in an authorized way. Unauthorized means that the user obtains the credentials either a) by mistake (e.g. overheard accidentally or sent by mistake) or b) deliberately (e.g. stolen from somewhere or obtained by social engineering).

3. The result of a user's access to the organization's confidential content. What we mean by this the consequences that are caused by the user when accessing content using the obtained credentials. This could either cause harm or not.

From Figure 2.1 we can see there are four possible cases to consider.

1. Case I — route from a – to – 1. In this case a user is (a) granted a credential in an authorized way and (1) when accessing content does not cause harm. An

example of this is internal employees who are granted credentials to perform their duties. This case applies when employees do their job as expected.

2. Case II — route from a – to – 2. In this case a user is (a) granted a credential in an unauthorized way and (2) uses it for a purpose other than the one for which it was originally intended which results in harm. For example, internal employees who are granted credentials to perform their daily activities, but when accessing content they intentionally or accidentally misuse their privileges, e.g. update someone's salary, delete an important file, or leak content to a competitor.

3. Case III — route from b – to – 1. In this case a user has (b) obtained a credential in an unauthorized way but (1) does not cause harm. For example, when a user obtains some credentials by mistake (e.g. sent by mistake) but he does not act on this knowledge.

4. Case IV — route from b – to – 2. In this case a user has (b) obtained a credential in an unauthorized way and (2) when accessing content, caused harm. For example, an unauthorized user obtains credentials from a friend and uses this to update someone's salary, delete an important file, or leak content to a competitor.

After analyzing the above cases we conclude that there are two different concepts: 'insider' and 'malicious insider'. In the following we propose our definition for both concepts.

Definition 2.2.1. An insider is a user who is granted a credential in an authorized way to access confidential corporate content for a specific purpose defined by the organization (does not cause harm), or a user who obtains a credential in an unauthorized way but does not use it to cause harm.

This definition explicitly identifies insiders in Figure 2.1 to include routes (a – to – 1, and b – to – 1), and explicitly excludes routes (a – to – 2 and b – to – 2).

Definition 2.2.2. A malicious insider is an internal or external user who uses credentials, obtained by either authorized or unauthorized means, to access confidential corporate content that results in harm to the organization. Such a misuse could be either accidental or deliberate.

This definition explicitly identifies malicious insiders in Figure 2.1 to include routes (a – to – 2 and b – to – 2), and explicitly excludes routes (a – to – 1 and b – to – 1).

Based on these definitions a 'malicious insider' could be one of the following.

1. An internal employee who possesses a valid credential to access confidential content. This access accidentally or deliberately results in harm to the organization.

2. An external user who possesses valid credentials to access confidential content. This access accidentally or deliberately results in harm to the organization. This case includes examples like contractors, third party vendors who have access to corporate internal information, and employees from collaborating organizations who have access to each other's organization's information.

3. An internal employee who obtained a valid credential, by unauthorized means, to access confidential content to cause harm.

4. An external user who obtained a valid credential, by unauthorized means, to access confidential content to cause harm. This case includes anyone who has gained access to internal resources by masquerading as an authorized internal employee.

2.2.2 Insider Threats

Having defined what we mean by an 'insider' and what cases it includes, we can now discuss the *insider threats* that we will focus on in this thesis. Brackney et al. define an insider threat as *"malevolent (or possibly inadvertent) actions by an already trusted person with access to confidential information and information systems"*[21]. Insiders' actions could affect information or service availability, integrity, or confidentiality. In the following we briefly define and provide examples for each case.

1. **Availability** is ensuring the content can be accessed any time when requested. An insider could severely disrupt service availability, for example, by deleting content or invalidating backup.

2. **Integrity** is invalidating the integrity of content, for example, by inserting wrong results, executing a process for granting an employee a bonus, etc.

3. **Confidentiality** is when an insider reveals confidential content to unauthorized parties, which we refer to it as content leakage throughout the thesis, e.g. retrieving content to a client device and then forwarding it (e.g. using email) to a competitor.

In this thesis we are mainly concerned about insider threats which affect content confidentiality — integrity and availability is a planned aspect of future work as discussed in Chapter 9.

There are several ways allowing malicious insiders to leak content from controlled resources, which are summarized as follows:

(a) Leaking confidential content by exploiting the **analog hole problem**.[1] This is defined as: *"Once digital information is converted to a human-perceptible (analog) form, it is a relatively simple matter to digitally recapture that analog reproduction in an unrestricted form, thereby fundamentally circumventing any and all restrictions placed on copyrighted digitally-distribited work.*[2] For example, the analog hole could be recording content, memorizing content, or conveying content to others via a phone call. In this thesis we do not consider the analog hole problem.

(b) Leaking confidential content by malicious insiders sending unprotected content using on-line media (e.g. via the Internet) or by storing and forwarding using physical media (e.g. printing text files or copying unprotected content to a USB memory stick).

    There are several areas of research which attempt to mitigate this particular insider threat [7, 8, 37]. This is typically done by protecting content in such a way that only encrypted content can be exchanged between devices and unprotected content cannot leave the device. Moreover, recent advances in trusted computing help in enforcing such control measures [65, 68]. In our proposed scheme we integrate recent research in DRM for addressing this point (see for example [8]).

(c) Leaking confidential content by malicious insiders when sharing the means of accessing the content (e.g. their credentials) with unauthorized individuals / parties inside or outside the organization enabling them to access protected resources. This is the main threat which our thesis focuses on.

2.2.3 Content Leakage Overview

We now define the content leakage threat more specifically, based on insider threat (c) above.

Definition 2.2.3. Content leakage is a threat to secure content sharing which is realized when malicious insiders share their credentials with unauthorized individuals enabling them to access content which they are not authorized to access.

We categorize content leakage into two main subcategories: internal and external leakage.

[1]http://en.wikipedia.org/wiki/Analog hole
[2]http://en.wikipedia.org/wiki/Analog hole

Definition 2.2.4. Internal leakage is a threat to secure content sharing which is realized when an authorized internal employee shares their credentials with unauthorized internal employee(s), enabling them to access confidential content.

Definition 2.2.5. External leakage is a threat to secure content sharing which is realized when an internal employee shares their credentials with an unauthorized third party outside the organization, enabling them to access confidential content.

The external leakage threat can be realized by an authorized employee who can access protected content and who wants to share it with unauthorized parties outside the organization. In a typical organization, authorized employees can access content from any device, and from any location, so long as they provide their credentials and the organization firewall does not prevent access.

Using firewalls to mitigate external leakage requires an organization's firewall to prevent employees from transferring content outside the organization. It also requires preventing employees from using portable media. This is not practical for many organizations and is discussed at length by the Jericho Forum.[3]

Moreover, recent ERM schemes consider allowing 'outsiders' to interact with enterprises as an essential requirement to allow modern business functions [52, 57]. If firewall rules allow the transfer of content outside an organization and allow external entities to communicate with the organization's authentication service, then an authorized employee can give their credentials to unauthorized third parties to access confidential content.

While the two types of content leakage defined above apply to a single organization, when organizations collaborate we find that addressing content leakage between collaborating organizations requires further analysis. This we refer to as *content leakage between collaborating organizations*, which is derived from the above two types of content leakage and is defined below.

In order to simplify the description of this type we call the organization who owns the content the 'source' organization, and the organization that receives content as the 'destination' organization.

Definition 2.2.6. Leakage between collaborating organizations is a threat to secure content sharing which is realized when an authorized employee in the source organization shares their credentials with an unauthorized party in the destination organization, enabling them to access the source organization's confidential content.

In practice many organizations need to share content with other organizations. In some organizations completing a business process requires accessing another organization's confidential content. Organizations have a 'duty of care' to protect their

[3]http://www.opengroup.org/jericho/

own confidential content and the confidential content they obtain from individuals or from other organizations. Thus, organizations need to find ways to prevent the transfer of their confidential content to unauthorized individuals and third parties and, at the same time, allow sharing of content with other collaborating organizations. In this situation, authorized employees in the collaborating organizations often access source organization content from any device so long as they provide proper authentication credentials. Consequently if an authorized employee in one collaborating organization shares his credentials with unauthorized users, this will enable the unauthorized users to access the source organization content, i.e. this results in content leakage between collaborating organizations.

2.2.4 Motivating Examples

Now we list some examples showing the importance of addressing content leakage problem in general. Jonathan Pollard,[4] who had high-level security clearance, was arrested for passing tens of thousands of pages of classified U.S. information (e.g. satellite photographs, weapon and systems data) to Israel. According to the 2008 Information security breaches survey *"16% of the worst security incidents were caused by insiders; 47% in large companies"*[30]. According to the 2007 CSI Annual Computer Crime and Security Survey *"Insider misuse of authorized privileges or abuse of network access has caused a great damage and loss to corporate information"* [63].

We now list some specific examples on the problem of insider threats when sharing credentials. According to the 2008 Insider Threat Study: Illicit Cyber Activity in the Government Sector, *"42% of incidents, insiders used an account other than their own in carrying out their malicious activities"* [3]. According to the 2008 Insider Threat Study: Illicit Cyber Activity in the Information Technology and Telecommunications Sector, *"23% of the insiders used shared accounts to carry out their activities"* [4].

As discussed above, content leakage caused by insiders sharing credentials is a considerable problem. We argue that more research is needed to bring greater attention to this problem in relation to other areas of access control. This is because, in comparison with outsiders, insiders often have better knowledge of internal systems, and know where to find and how to abuse their organization's confidential assets.

In our research we focus on limiting the effects of content leakage as defined in Section 2.2.3. More specifically, we focus on protecting confidential content even if an authorized user shares their credentials with others, inside or outside the organization, not authorized to access content.

[4]http://en.wikipedia.org/wiki/Jonathan_Pollard

To investigate this problem without considering organization types nor the dynamic nature of organizations would not result in a practical solution. Therefore in Section 2.3 we describe the main types of organization structures, and define the organization requirements which we consider within our proposed scheme.

2.2.5 Problem Analysis

When we look at our definition of an insider, and the insider leakage threats, it is clear that providing a solution which completely mitigates these threats is not going to be an easy task. This is because insiders have credentials which allow access to content typically without detection other than via audit logs. While audit logs may be useful after an incident has taken place, they do not prevent access in the first place. In practice most access control schemes manage access to content by checking authorization credentials. Thus, if access to content is controlled by credentials that the user can share with others, this does not mitigate the threat of leakage due to insiders sharing credentials.

Sharing credentials or the means of accessing content has caused many breaches in DRM. Consequently, researchers in DRM have attempted to provide several solutions to this problem. Their initial idea for mitigating this threat was to bind content with a specific device [5]. However binding content to a specific device has been publicly criticized since it does not satisfy many consumer and copyright law requirements. This led to the authorized domain concept in DRM schemes that allows content to be bound to several devices. Our paper [13] proposes the idea of using DRM in wider application domains, e.g. it discusses the possibility to use DRM schemes to protect content in organizations. In this thesis we discuss the possibility of using DRM schemes to address content leakage problem. We analyze this point in detail in Chapter 3, where we set the relation and exact boundaries between our work and DRM.

Many alternative solutions may be proposed to mitigate the threat of insider leakage in enterprise organizations. However, learning from the DRM experience, one important point to consider is the acceptance of the solution by organizations and users. Therefore, proposing any mitigation scheme to meet insider threats requires a good understanding of both the organization's requirements and the security requirements.

The problem we face when we attempt to identify a typical organization's requirements is that there are many types of organization. Each has its own set of requirements which might be different from other organizations depending on its structure and process workflow. Thus, we need to specify which type of organization we are interested in when mitigating the insider leakage problem. In the next section we start by studying the main types of organization and the different requirements for each type. We conclude that, although there are many different organization structures and workflows, most share a common base. Therefore, we are able to define a generic model for an organization that can be applied to different organization types.

2.3 Organization Types

Different structures exist for organizations depending on the nature of the organization's business [53]. In our research we are mainly interested in non-military organizations which use dynamic groups of employees and projects to fulfil their objectives. Most organizations also require the sharing of content and content between group members and, at the same time, require protection of the content from the threat of leakage. In the following we outline three main types of organizational structure: Functional Structure, Divisional Structure (also called Product Structure), or Matrix Structure.

Functional Structure. The functional structure type of organization, as illustrated in Figure 2.2, splits employees into role-specific groups. Employees join the groups based on their roles in the organization.

Divisional Structure. The divisional (or product) structure type of organization, as illustrated in Figure 2.3, splits employees into division-specific groups based on product line, customer market and geography. Each division has its own resources and, hence, can act independently.

Matrix Structure. The third type of organizational structure, as illustrated in Figure 2.4, is an overlay of a product structure on an existing functional structure. It splits employees into groups based on both their roles and product lines.

From the above we see that all three types of organization arrange their employees into groups which interact with each other to achieve the organization's goals [53]. The nature of the grouping and the interaction between groups is based on the organization's process workflow.[5] According to Sandhu *et al.* [67] the relationship between groups within an organization could be either isolated or connected. *Isolated* groups are realized when group operations and employee membership have no impact

[5] *"Workflow may be seen as any abstraction of real work, segregated in workshare, work split or whatever types of ordering. For control purposes, workflow may be a view on real work under a chosen aspect, thus serving as a virtual representation of actual work. The flow being described often refers to a document that is being transferred from one step to another."[http://en.wikipedia.org/wiki/Workflow]*

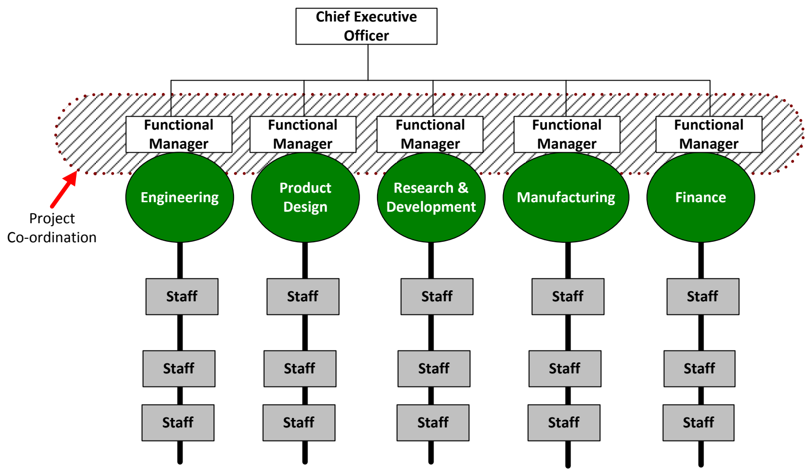


Figure 2.2: Example of a Functional Organization Structure

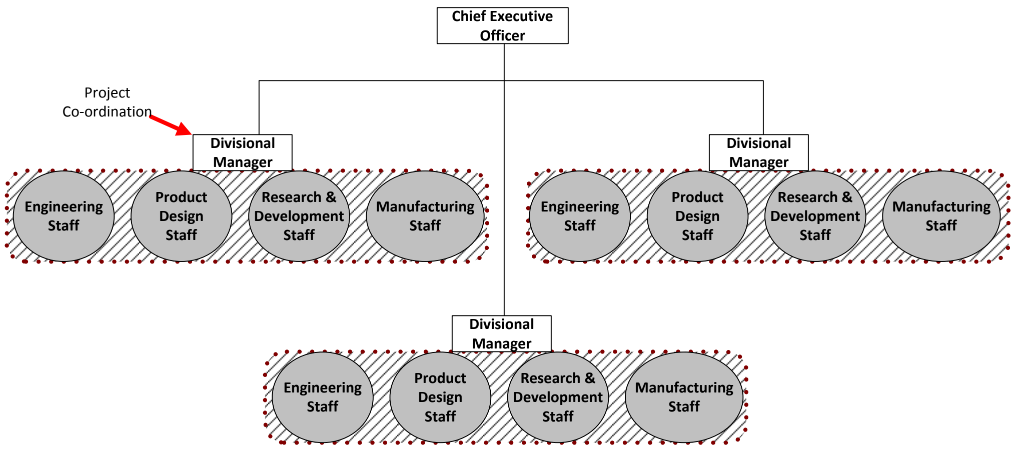


Figure 2.3: Example of a Divisional Organization Structure

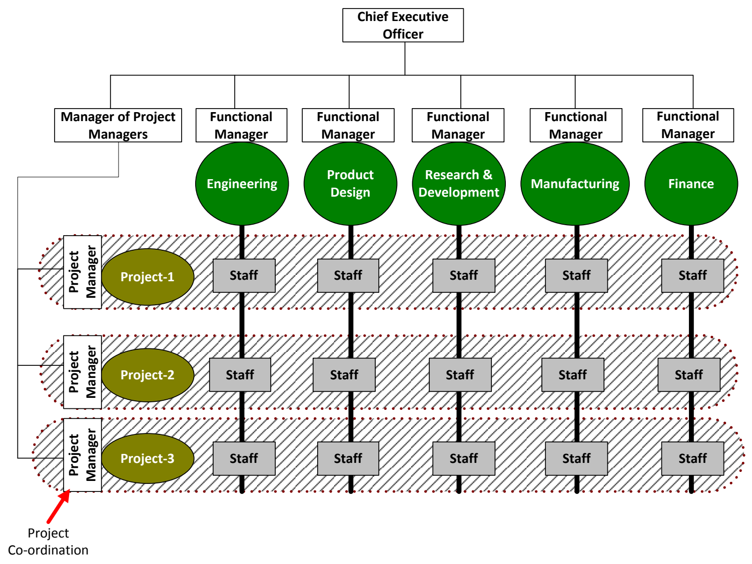


Figure 2.4: Example of a Matrix Organization Structure

on other groups. On the other hand, in *connected* groups some impact could arise, e.g. if groups are mutually exclusive.

Employees' access rights in either type of group can be of two types: differentiated or undifferentiated. *Undifferentiated* access rights are when all members of a group have the same access rights to the same item of content. However, with *differentiated* access rights, members of the same group have different access rights to the same item of content.

Moreover, groups in organizations are not static but dynamic. By dynamic we mean that a group can be expanded by adding members, can contract by removing members, can be combined with other groups to form a bigger group, and can split to form smaller groups. In this thesis we discuss *differentiated, isolated* and *differentiated, connected* groups.

Projects are initiated within an organization to perform specific business tasks. A project manager manages employees assigned to the project. Project management and assigning employees to projects is based on the organization's structure [33, 53]. A project typically has a dynamic group whose members have been selected from other groups in the organization. In this thesis we are mainly concerned with projects which require secure content sharing. In other words, employees within a project need to share the project's content to achieve the project goals, and, at the same time, the project's content should not be available to employees outside the group.

Based on the definitions of the different types of organizational structure, we make the following observations.

1. There is no specific organization type that can be applied to all organizations. Also, an organization could be composed of different types if it requires such a thing.

2. Employees in an organization are grouped according to the organization type and employee role. Interaction between employees in one group and employees in different groups is achieved by defining projects that transcend structural group boundaries.

3. Employees may be involved in multiple projects at the same time. In this case an employee might have multiple roles based on each project the employee is member of.

4. Although organizations have various structures, they still have entities in common. For example, all type of organizations have employees, system administrators, and project managers. Also, organizations categorize employees into groups, and initiate projects to fulfill their objectives.

The different types of organization share a common foundation. They all have groups, projects, and content that need to be shared between groups and project members. Moreover, this content needs to be protected from access by unauthorized users. Having such a shared foundation leads us to suggest the following general model that can be applied to any of the three main organization types.

2.4 General Model for an Organization

In this section we define a general model that includes the common entities from the previous three organization types. In our research we found that defining such a general model is a very important step, as it helps to define a set of common requirements for different organization types. Once we propose a scheme that covers such common requirements then this scheme can be implemented in any type of organization i.e. Functional, Divisional, and Matrix.

In our general model, we use the same grouping concept as discussed in the previous section in which employees are split into groups. Also, we use the project concept, where projects are initiated within our defined organization model to perform particular tasks. Employees assigned to a project are selected from one or more groups within the organization. The selection is based on the roles required to fulfill the project's objectives. Employees may also be assigned to work on multiple projects. Each employee assigned to a project may also have different rights, i.e. the project will have *differentiated rights*. Projects can be dynamic as described previously.

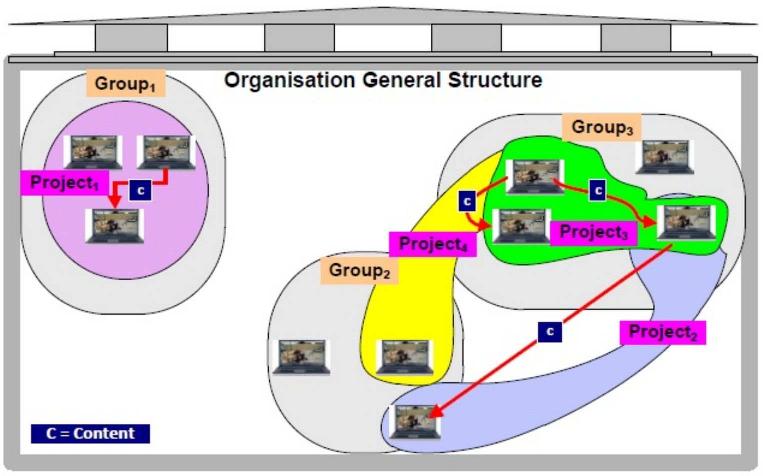


Figure 2.5: Generic Organization: Grouping, Project Initiation and Content Flow

Figure 2.5 shows our general model of an organization and illustrates the entities described above. It shows an organization which has three groups of employees (Group$_1$, Group$_2$ and Group$_3$). These groups may represent different functional groups in a functional organization structure or different divisions in a divisional organization structure. Four projects are then initiated in the organization: Project$_1$ undertakes tasks that require employees from Group$_1$. Members of Project$_2$ are selected from Group$_2$ and Group$_3$. Project$_3$ undertakes tasks that require only employees from Group$_3$. Project$_4$'s members are one employee from Group$_2$ and another employee from Group$_3$ who is also a member of Project$_3$.

We argue that to build a successful content protection solution for any type of organization as described in Section 2.3, we should take into consideration the fundamental requirements derived from our general model. In the remaining part of this chapter we summarize the main organizational requirements for our general model.

2.5 Requirements

Based on the generic organization model described in Section 2.4 we identify the following requirements that should be in any proposed scheme.

Requirement 1: **Support for organization groups and dynamic nature**. Any proposed scheme needs to meet the following conditions in order to meet Requirement 1:

1. Organization grouping structure. Organizations have groups and projects, which can be created at any time as needed. An employee might par-

ticipate in multiple groups and projects at the same time. *Any proposed scheme must be capable of creating and deleting projects and groups.*

2. Organization dynamic nature. Groups and projects within organizations have a dynamic nature. Groups and Projects can expand and contract. In addition, organizations may need to re-allocate group members. *Any proposed scheme must allow expansion and contraction of groups.* For example, if an organization requires changing its layout, say after one year, this might require devices reallocating. When a device is reallocated to be used by a new project which might require accessing different kind of content, it can join all dynamic domains where the content is bound.

Requirement 2: **Content Sharing**. This requirement is about providing the ability for authorized employees to exchange and access content. It is one of the fundamental requirements for the kind of organizations we discuss in this chapter. Members of a project or group typically need to share resources to achieve their goals. Without content sharing, it will be difficult for projects to achieve their objectives. *Any proposed scheme must allow secure content sharing between groups and projects.* For example, for a chief information officer (CIO) of an organization to be able to access all of an organization's shared but protected information, the CIO device needs join all of the organization dynamic domains. From this we can see how our proposed scheme is able to provide controlled content sharing.

Content sharing can be classified as of one two main types: centralized sharing and decentralized sharing.

1. **Centralized content sharing** provides the ability for authorized users to share and access content via a central server.
2. **Decentralized (or disseminated) content sharing** provides the ability for authorized users to exchange and access content between client devices without the need to store and retrieve it from a central server.

In this thesis we focus on decentralized content sharing.

Requirement 3. **Content protection.** In this thesis our main concern is protecting content confidentiality from insiders who share their credentials. This is the content leakage threat described by Definition 2.2.3. *Any scheme that is proposed to mitigate the content leakage threat must mitigate the two types of content leakage threats, which are the internal leakage threat and the external leakage threat.*

Requirement 4. **Rights management and enforcement**. The organization types described in Section 2.3 require that employees assigned to a project may not necessarily have unified access rights. Assigning employees access rights to the project's content would be subject to the roles required to achieve the project goals. Therefore, *any scheme proposed must provide: (a) a mechanism to manage employees' access rights; and (b) a mechanism to enforce access rights wherever content is transferred between client devices.*

We mainly focus on the first three requirements in our proposed scheme. We rely on existing ERM schemes to meet Requirement 4 as ERM could be integrated into our scheme.

In addition to the above requirements, there are other important sub requirements. These requirements include performance, interoperability, ease of use, and scalability. Such additional requirements should also be considered as far as possible in any proposed scheme. The extent to which any proposed scheme supports these additional requirement will affect how acceptable the scheme is to users.

2.6 Conclusion

In this chapter we have defined the problem and the type of organization this thesis is focusing on. We started by defining what we mean by an insider and then identified the main insider threats. Following that, we defined the content leakage problem and its two main types. We then specified the type of organization our scheme is proposed to apply to. This was used to define the fundamental requirements for the type of organization where our content protection scheme will be applied.

We are mainly focusing on mitigating the insider threat on content confidentiality when insiders share their credentials in our defined type of organization. This organization will have dynamic groups and initiate dynamic projects that need to share content between projects' and groups' members and, at the same time, mitigate the insider leakage threat. The groups may be either isolated or connected. Finally, this thesis considers the case when rights are differentiated.

Chapter 3

Related Work

Contents

This chapter discusses related work and provides an overview of Trusted Computing technology.

3.1 Introduction

In this chapter we review the most commonly discussed schemes for secure information sharing. We consider three research areas which are related to our work: (a) research that discusses insider threats, (b) research that discusses group-centric secure information sharing, and (c) research which extends content protection from server to client side.

Research which focuses on extending content protection from server to client (i.e. point (c)) is directly related to our research. This is because policy enforcement at the client side increases the possibility for authorized users to share their credentials with unauthorized users to leak content. In other words if there is no enforcement at the client side users do not really need to share their credentials once content is transferred to their devices as they can delete the content, copy it, save it, or even forward it to unauthorized users. Examples of such schemes which attempt to enforce rights at the client side are ERM and DRM schemes, which we discuss and analyze in this chapter. The other areas of research (i.e. points (a) and (b)) provide models to analyze the insider threat and to manage groups and access rights within organizations. Such models are not directly related to our research problem, i.e. the leakage problem we defined in Chapter 2. However, these models enrich our research with good examples and definitions. They also help us to understand the scope and context of our work and the importance of the problem. These are discussed at the end of this chapter.

In this chapter, Section 3.2 discusses traditional access control models. We then discuss ERM schemes in Section 3.3. After that, in Section 3.4 we discuss DRM schemes. Following this, we discuss in Section 3.5 other research areas which might support our schemes. We then discuss trusted computing, which is fundamental to our proposed scheme in Section 3.6. Finally, we conclude this chapter in Section 3.7.

3.2 Access Control

A general model of an access control mechanism is shown in Figure 3.1. In this model access rights are defined by an access control policy. The users in the organization are the principals, and subjects act on behalf of these principals. A subject can be an application or a user agent that runs on the principal's PC, and subjects may request access to any resources in the system. Access control schemes use two main concepts: (a) a policy decision point (PDP) that interprets the defined policy and decides on how the content should be used; and (b) a policy enforcement point (PEP) that enforces the decision made by the PDP. These decisions are based on an access control policy. The policy determines how subjects may access the resource

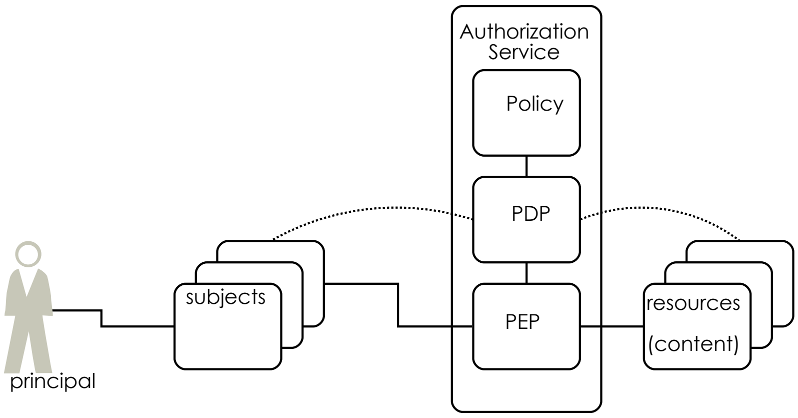


Figure 3.1: Access Control

depending on the rights held by the subject. This authorization service is also known as a reference monitor and is generally implemented as part of the trusted computing base (TCB) [49]. In the remaining part of this section we attempt to identify access control schemes which focus on secure information sharing for the types of organization described in Chapter 2.

We split our access control discussion based on where it protects content: (a) schemes which protect content at the server side only, and (b) schemes which extend content protection to the client side.

3.2.1 Server Side Access Control

Organizations use access control mechanisms to protect their content from being accessed by unauthorized individuals and to control what an authorized user can and cannot do with content [61]. We first define what we mean by a server and a client. A server is the device from where content/access rights are served to other clients; a client is the device/application that requests content/access rights from the server. We now provide a scenario in a typical client-server application. Users would typically retrieve content from a server using a client application that understands the context of the data and how to interact with a data management system running on the server. The data management system manages data storage and security, including access control. When the user (principal) wants to retrieve an item of content they instruct the client application (subject) to do that. The client application connects to the server, where the data management system runs. The server (and/or the data management system) first authenticates the user, typically using a username/password. If the authentication succeeds, a ‘session’ is established between the user workstation and the server. The client application sends a request via the established session to the data management system to retrieve the requested data. If the user is authorized, the data management system returns the requested data to the client application. The user can do whatever they want to the data from his workstation. However, if the user attempts to update the data on the server, the data management system checks if the user has update permission for the data. If so, changes on the data are permanently saved at the server. Otherwise, an error will be returned to the client application. Examples of server side access control are the Discretionary Access Control (DAC) [38] and Mandatory Access Control (MAC) [69] approaches.

The requirement of content protection that we identify in Section 2.5 has the following vulnerabilities. The above access control mechanisms do not enforce access rights (i.e. access control policy) on client devices. This is because the PDP and PEP run on the server where content is stored. In another words, there is no enforcement on the client side and so authorized users can transfer content to others without the need to share their credentials.

3.2.2 Server and Client Side Access Control

In this section we focus on schemes that protect content on both server and client sides, i.e. protecting content wherever it is stored and transferred. We split the schemes focusing on enforcing rights on the client side into two main categories, based on the way the subject gets access rights: static or dynamic. Static rights are those that are predefined by administrators or content creators and stored at the principal's devices. Dynamic rights, on the other hand, are centrally stored and managed access rights. Dynamic rights can be changed at any time with immediate effect on client devices when they access content associated with dynamic rights.

`Case 1:` Subject has embedded static rights. In this case, access control mechanisms in general assume the following: the subject is an agent on the principal's client device, and the authorization service runs on the principal's device using a specific application when accessing content. An example of case 1 is content-specific applications (e.g. Adobe Reader [12]) that are installed with predefined static rights forcing how the client device can use the content. Another example is customized applications provided by a third party to access a server side application. In this case the application could have predefined static access rights that can be enforced on all content accessed by such an application. This case does not consider the fundamental organizational requirements, which we identified in Chapter 2. For example, employees' access rights are predefined at a client workstation on all items of content the user is authorized to access, i.e. it does not provide dynamicity for rights enforcement and assumes unified access rights for all items of content.

`Case 2:` Subject obtains dynamic rights to access content from a centralized rights server. The main research area that focuses on this type for accessing content in an organization is Enterprise Rights Management (ERM), which emerged from Digital Rights Management (DRM).

In subsequent sections we discuss ERM and DRM as they both focus on enforcing access rights wherever content is stored and transferred.

3.3 Enterprise Rights Management

ERM schemes are proposed to protect an enterprise's content when exchanged between devices in organizations, for example, when an employee creates a document then sends it to his colleagues. The sender wants to ensure that receivers will use the content based on usage conditions which are defined by himself. To address such a requirement, ERM schemes provide the ability to allow the sender to define access rights which are bound to content and which are enforced on client devices.

Most of the current ERM schemes in use today have a common workflow; however, they vary in the implementation and the usage terminology. In this section we provide an abstract description of the common workflow for ERM schemes. This mainly covers Microsoft Windows Rights Management Services (RMS) [52], Oracle Information Rights Management (IRM) [57], Adobe Rights Management Enterprise Suite (ES) [11], and the EMC Documentum Information Rights Management (IRM) [31]. At the end of this section we provide an outline of the main differences between these schemes, and a security analysis of the common workflow.

The main entities in an ERM scheme would consist of principals, client devices and a rights management server, which are described as follows (see Figure 3.2).

A principal is an employee working in an organization. The principal can use only the organization's devices to access content, for which he is authorized.

A client device is where principals access content from. In ERM schemes each device is assumed to have subjects, objects and a trusted authorization service (referred to as a client ERM agent). A **subject** is an application-specific software agent that is used in two ways: (1) to create content and assign access rights to content; and (2) to access content based on the defined access rights associated with the content. An **object** can be of either two types: content or a rights objects. A **rights object** is a content-specific file which is created by an ERM agent and contains the content encryption/decryption key and content access rights.

The client ERM agent is the authorization service which implements the PDP and PEP. It does the following.

(a) Interacts with subjects on the device.

(b) Interacts with the rights management server.

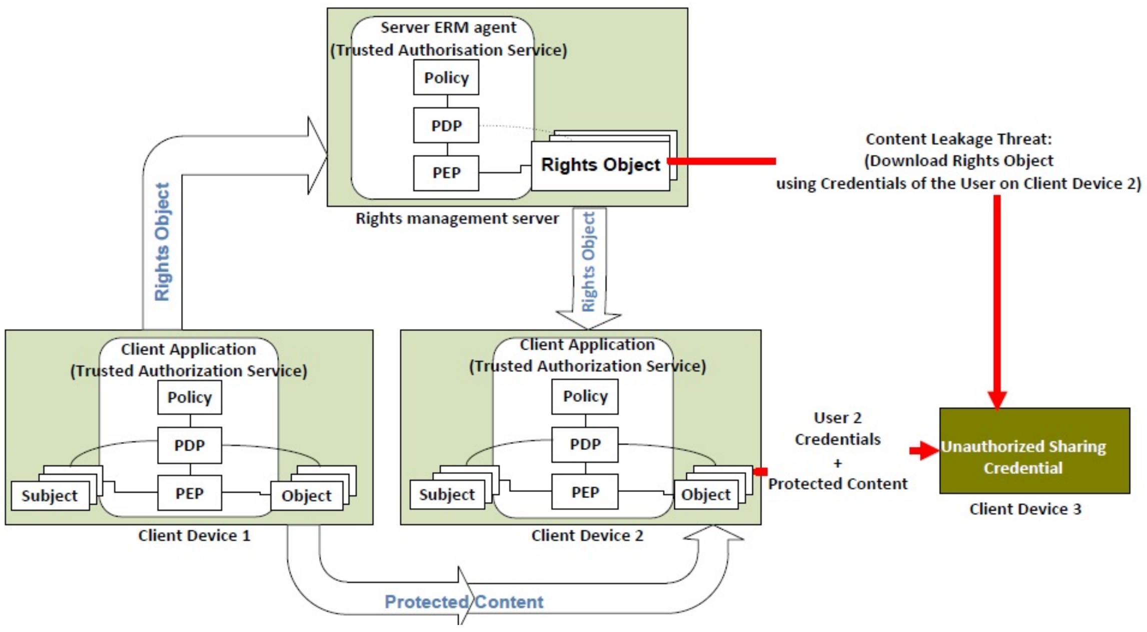


Figure 3.2: ERM Main Entities

(c) When the subject creates content it creates a content encryption key at the content creator's device. It associates the encryption key with content-specific access rights, and then sends the result in a form of an encrypted rights object to the rights management server. It also encrypts/decrypts content after it has been received from a sender and then discards the content-specific key. The ERM agent is designed to not reveal the content encryption key.

A rights management server is a centralized server responsible for securely storing and managing rights objects. The rights management server provides rights objects to the client ERM agent running on a principal's device, assuming that the principal has been successfully authenticated. ERM schemes are designed so that the rights management server consists of rights objects and a server ERM agent. This is explained as follows. An object is an access rights object. The **Server ERM agent** is an authorization service running on the rights management server. It is in charge of the following.

(a) Interacts with the client devices' ERM agent.

(b) Stores rights objects.

(c) Grants access rights for successfully authenticated users.

(d) Protects objects from being revealed or altered.

A typical ERM workflow consists of three main phases.

Phase I. Content and its specific rights object are created in the client device. In this, ERM assumes the following.

1. Subjects on the client device act on behalf of principals and they represent the content-specific applications. These subjects need to communicate with the authorization service (ERM agent) on the client device when encrypting/decrypting content. The subject in the client device securely creates content and it will not reveal the content unencrypted to any applications except the ERM agent (authorization service).

2. When content is created, the subject assigns content-specific access rights on how content should be used. The authorization service running on the client device receives the content and the access rights from the subject. In this stage the authorization service prevents content from being revealed unencrypted to others.

3. The authorization service creates an encryption key to protect the content, encrypts the content, and stores the encrypted content on the client device.

4. The authorization service running on the client device associates the content encryption key with content-specific access rights (received in step 2) to form a rights object. It then encrypts the rights object using the rights management server's public key. The authorization service prevents the content encryption key from being revealed unencrypted to others.

Phase II. The authorization service running on a client device sends the rights object to the rights management server. In this, ERM assumes the following.

1. The authorization service running on the rights management server securely receives, stores and manages content rights objects.

2. The PEP running in the rights management server manages access to rights objects according to decisions made by a PDP.

3. The rights management server should have a robust authentication mechanism.

Phase III. Protected content is transferred to another client device. The client device authorization service must obtain the content-specific rights object from the rights management server. In this, ERM assumes the following.

1. For a decision to be made, the rights management server authenticates the user (principal). If authentication succeeds, the rights management server creates a new rights object for that particular user containing the content decryption key, an expiry time, and the access rights granted to the principal for the content. The rights management server then transfers the rights object to the client device's authorization service.

2. The authorization service (i.e. the ERM agent running in the client device) ensures that the content decryption key is discarded after decrypting content.

3. The authorization service running on the client device enforces content access rights. More specifically, a PEP manages access to content according to decisions made by a PDP. These decisions are based on the subject's defined access rights, which are obtained from the rights management server.

3.3.1 Differences and Similarities Between ERM Schemes

After describing the abstract model for ERM schemes workflow, we now outline the main differences between the most popular commercial schemes.

(a) Each scheme uses its own terminology describing the above entities; for example, Microsoft Windows (RMS) [52] uses the term RMS server and Lockbox to describe the rights management server and the client device ERM agent respectively. Oracle IRM uses IRM server and Oracle IRM desktop to describe the same respectively.

(b) Adobe Rights Management ES [11] supports only PDF type of documents. Other schemes do not impose restrictions on the content type.

(c) All schemes require external users who need to access the organization content to be enrolled into the organization rights management server. In other words, organization security administrators need to create a user account for each external user. Moreover such an external user would need to connect to the organization rights management server to authenticate to the server and download content usage license.

3.3.2 ERM Abstract Model Analysis

ERM schemes attempt to meet organizational requirements for content dissemination as discussed in Section 2.5. In this section we assess how well ERM schemes address such requirements.

The ERM model described in the previous section successfully meets the sharing requirement as defined in Section 2.5. In ERM schemes, content can be created, stored and exchanged between client devices without the need to store the content using a centralized server. However, content protection is subject to the following vulnerabilities.

1. ERM assumes that the PDP and the PEP cannot be subverted, and that the integrity of the policy and the subjects' access rights are protected. Failure of the PDP or PEP will result in failure of the authorization service. If the authorization service can be subverted, for example by malware, then the threat of content leakage can be realized.

2. A second vulnerability is that of content leakage caused by authorized users sharing their credential. In a typical enterprise organization, users have a degree of freedom. Users may choose to abuse their access privileges, for example by revealing content or sharing credentials used to access content. If users abuse their access privileges then the threat of content leakage can be realized.

   Now, we discuss how these vulnerabilities can be exploited in the commercial schemes we mentioned above.

   (a) None of the commercial schemes address the internal leakage problem, as employees can access content from any device by having a valid credential. If an employee wants to leak a confidential project's content to another employee, the former can provide his credential to the latter enabling the latter to access the confidential content. Alternatively, if the employee does not like providing his credentials to others, he can download a rights object into another unauthorized employee's device enabling him to access the content until the downloaded license expires. In Figure 3.3, we illustrate how this could happen for Windows RMS. This applies equally to all other schemes.

   (b) None of the commercial schemes address the external leakage problem defined in Section 2.2.3. This is because in all these discussed schemes a user can access organization content from any device. The client device needs to install a publicly available client side software agent. For example with Windows RMS a device must have a valid Lockbox, which is included in Windows RMS SP1. Such a step does not require administrators' involvement or special authorization. Therefore, in all schemes

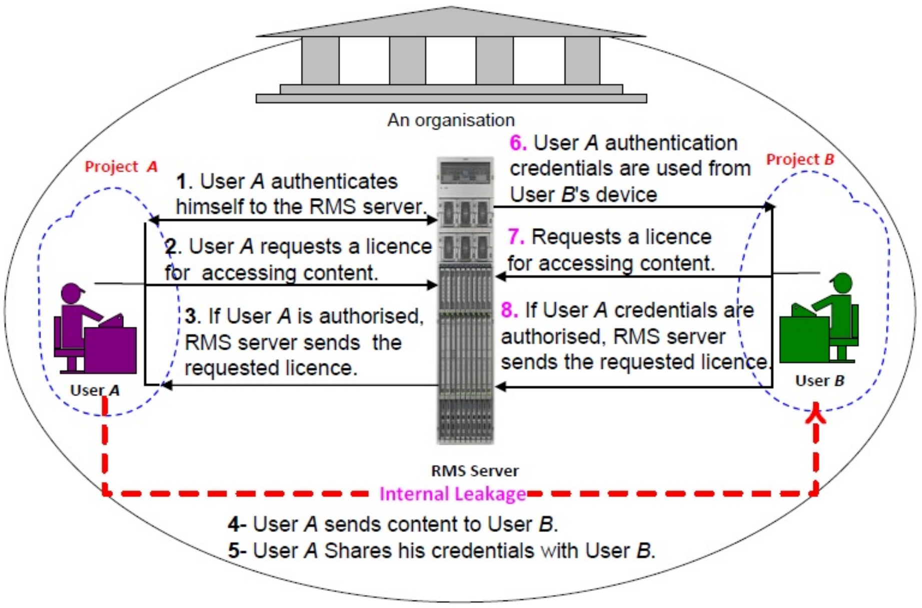

An organisation
Project A
Project B
1. User A authenticates himself to the RMS server.
2. User A requests a licence for accessing content.
3. If User A is authorised, RMS server sends the requested licence.
6. User A authentication credentials are used from User B's device
7. Requests a licence for accessing content.
8. If User A credentials are authorised, RMS server sends the requested licence.
User A
User B
RMS Server
Internal Leakage
4- User A sends content to User B.
5- User A Shares his credentials with User B.

Figure 3.3: Internal Leakage

including Windows RMS, a user with a valid credential can access protected content from outside the organization enabling any employees to download valid usage licenses associated with protected content on an unauthorized third party's device. This is illustrated in Figure 3.4 for Windows RMS, and applies equally to all other schemes.

(c) Some of the commercial schemes discussed in this section claim that they mitigate content leakage between collaborating organizations by requiring system administrators from all participating organizations to register their employees who need to access the collaborating organizations' content in all organizations' rights management servers. This mechanism does not address the leakage problem between collaborating organizations as it mainly binds accessing content to user authentication using username/password, which can be shared with others, i.e. it has the same drawbacks as the external leakage described above. This is illustrated in Figure 3.5 in the case of Windows RMS, and applies equally to all other schemes. In addition, the schemes' proposed policy has the following associated concerns: (1) it raises user account management problems for users from other organizations (i.e. managing password, rules and users joining and leaving all collaborating organizations who need to access shared content with the source organization); and (2) it raises privacy concerns, as in many cases, organizations do not want other organiza-

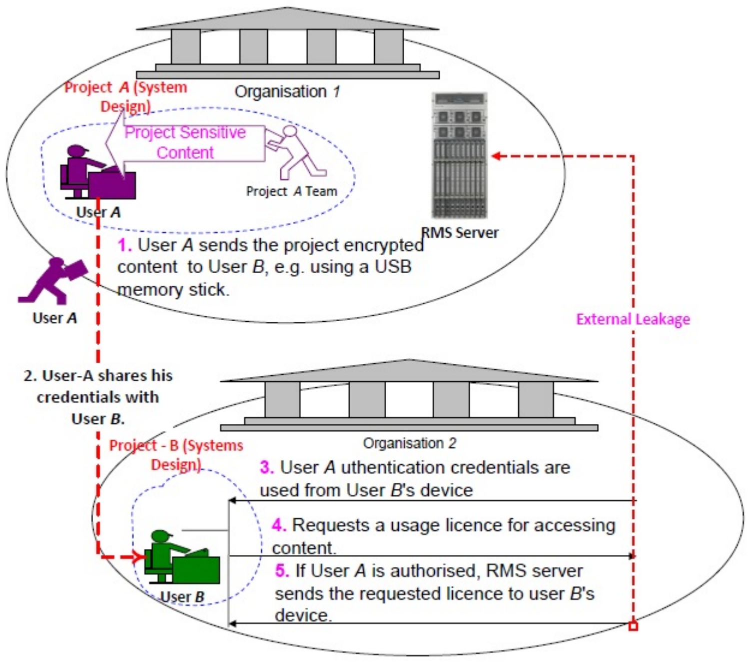

Organisation 1
Project A (System Design)
Project Sensitive Content
Project A Team
User A
RMS Server
1. User A sends the project encrypted content to User B, e.g. using a USB memory stick.
User A
External Leakage
2. User-A shares his credentials with User B.
Organisation 2
Project - B (Systems Design)
3. User A uthentication credentials are used from User B's device
4. Requests a usage licence for accessing content.
5. If User A is authorised, RMS server sends the requested licence to user B's device.
User B

Figure 3.4: External Leakage

tions to know such details about their internal system structure, e.g. employees who are member in a specific department, or who are working on a specific project or a task.

ERM provides schemes which achieve rights management and enforcement on the client side. However these schemes suffer from weaknesses when considering the type of organization we focus on. We outline these weaknesses in Chapter 9. Despite the weakness in rights management we can still use these schemes to provide rights management to our schemes at a latter stage (planned future work is to focus on this access rights in details).

3.4 Digital Rights Management

Digital Rights Management (DRM) is a term that is used to represent the technologies and standards that prevent illegal copying and allowing the imposition of fees, processing of payments, and protection of principal rights and profits [42]. In other words, DRM is proposed for client-side enforcement of access rights. The question we raise is whether DRM schemes "as is" can be used in organizations to address the insider content leakage problem and simultaneously meet the organization requirements defined in Section 2.5.

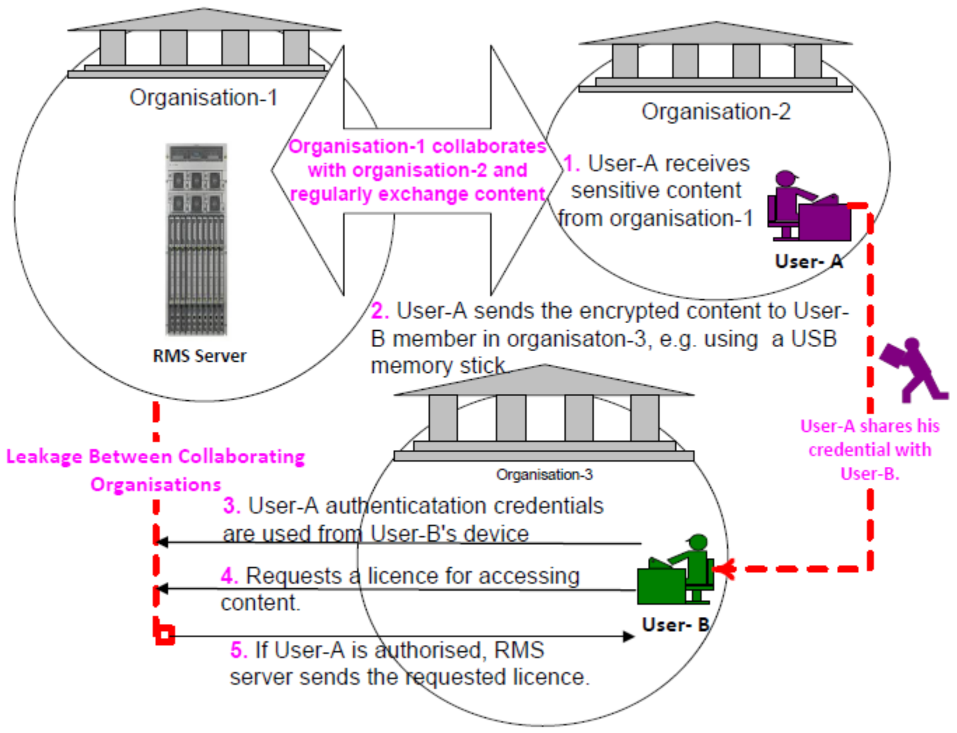

Organisation-1
RMS Server
Organisation-1 collaborates with organisation-2 and regularly exchange content
Organisation-2
1. User-A receives sensitive content from organisation-1
User- A
2. User-A sends the encrypted content to User-B member in organisaton-3, e.g. using a USB memory stick
User-A shares his credential with User-B.
Leakage Between Collaborating Organisations
Organisation-3
3. User-A authenticatation credentials are used from User-B's device
4. Requests a licence for accessing content.
User- B
5. If User-A is authorised, RMS server sends the requested licence.

Figure 3.5: Leakage between Collaborating Organizations

To answer this question we have to analyze DRM schemes and asses them. DRM schemes have evolved through three main generations. In the following we analyze these DRM generations, then, at the end of each DRM generation, we assess whether it can be applied directly into an organization's workflow.

3.4.1 DRM First Generation

In the first generation of DRM, content is bound to a specific device which would mainly require each device to have a specific hardcoded key, where content is encrypted using this key.[1] Thus, if encrypted content is distributed to others they cannot access it on their devices. To implement this model in organizations we would require the binding of downloaded access rights with user identity rather than payment. Implementing this model in an organization will fail to meet the organizations' requirements discussed in Section 2.5. This is because DRM first generation does not support organization grouping and dynamic structure. DRM first generation binds content to a specific device. Such binding does not satisfy the content sharing requirement as defined earlier.

[1]For example, Apple Fairplay [17]

3.4.2 DRM Second Generation

In the second generation of DRM content is associated with a specific rights object defining its access rights [56]. The rights object is defined by the content distributor based on payment. A device can download content associated with the rights object after paying the usage fees. The rights object is bound to a specific device. Protected content can be transferred between devices. Accessing protected content on a new device requires communicating with the content distributer, paying the usage fees, and downloading a new rights object bound to the new user device. Unlike DRM first generation, protected content can now be exchanged between devices. Satisfying all content sharing requirements would require that devices obtain rights objects from a centralized server. It also would require replacing payment with user authentication and authorization in order to obtain rights. Adding such modifications to a DRM scheme would make it the same as the ERM schemes discussed in Section 3.3

3.4.3 DRM Third Generation

The first two generations of DRM schemes have been widely criticized because they did not satisfy many consumer and copyright law requirements [5]. For example, such schemes do not allow proprietary content to be freely transferred between devices owned by a single individual using the same licence file. This led to the integration of the authorized domain concept into DRM third generation schemes. An authorized domain allows content to be shared between a set of devices using a single licence file. An authorized domain owner (i.e. authorized licence holder) creates a domain with a specific number of devices in the domain. Domain membership is controlled by a counter enforcing a limit on the number of devices that can be in a domain. The limit of this counter is defined by a trusted third party. Joining devices to a domain is conditional on the domain counter remaining below a predefined limit and the joining device not being member of another domain. The authorized domain has a domain key that is shared by all domain devices and is used to protect domain content. The content is sent to the domain encrypted with a content encryption key that is stored inside the licence file. Once the domain owner installs the licence file into a device which is a member of the domain, the content encryption key (as stored in the licence file) is decrypted, and then re-encrypted using the authorized domain key. As this key is only shared between domain devices it enables content sharing between such devices.

DRM authorized domain schemes were proposed to limit content proliferation when users share the means to access content on a personal network. The latest DRM third generation work done by Abbadi et al. [7, 8, 10] was specifically proposed to strongly bind the authorized domain with the domain owner. Abbadi's schemes suggest four mechanisms to achieve such binding as follows: using location-based services ([6]), using electronic payment systems ([8]), using a mobile phone ([10]), and using a master control device ([7]). Abbadi then integrates the Authorized Domain with his proposed strong authentication measures to satisfy copyright law requirements and simultaneously to allow licence holders to use content on all devices they have.

After we studied and analyzed the four DRM schemes we found that the scheme using the master control device is the only one that can be used in an organization environment. This is because mobile phones and electronic payment systems use personal equipments (i.e. mobile phone and payment card) for user authentication and authorized domain management which are inappropriate in organizational context. Location-based services restrict accessing content to specific location but not to specific users which will not solve the leakage problem. Also, it is not suitable for organization workflow as organizations might require employees to work from multiple locations. The master control device, on the other hand, uses a specific device for user authentication and domain management. This scheme, although designed for DRM, may be useful in a dressing the problem of content leakage in enterprise organizations. The concept of the master controller would need to be adapted and extended to move from managing the number of devices that is allowed to access content to which devices are allowed to access content. As such we focus on the DRM-authorized domains using master control device scheme in our analyses.

Research on DRM authorized domains using a master control device [7] focuses on personal networks which have different requirements, entities, and workflows in comparison with enterprise networks. For example, in personal networks there is one domain owner, while in organizations the domain owner concept does not exist at all. Domains are owned by the organization and managed by security administrators. Moreover, in DRM there are third parties who are called right issuers who create licenses for users; however, in an ERM environment there are right objects which are created in a different way depending on the type of organization. Therefore, using a DRM scheme in an enterprise network would face challenging problems related to organizational structure and process workflow.

To answer 'Can DRM authorized domain schemes using a master control device be implemented in organizations to limit content leakage when user share their credentials?' we need to assess the use of the DRM authorized domain scheme using master control device "as is" in organizations in relation to the organizational requirements of Section 2.5.

1. Support for dynamic groups. The DRM scheme does not meet the requirement

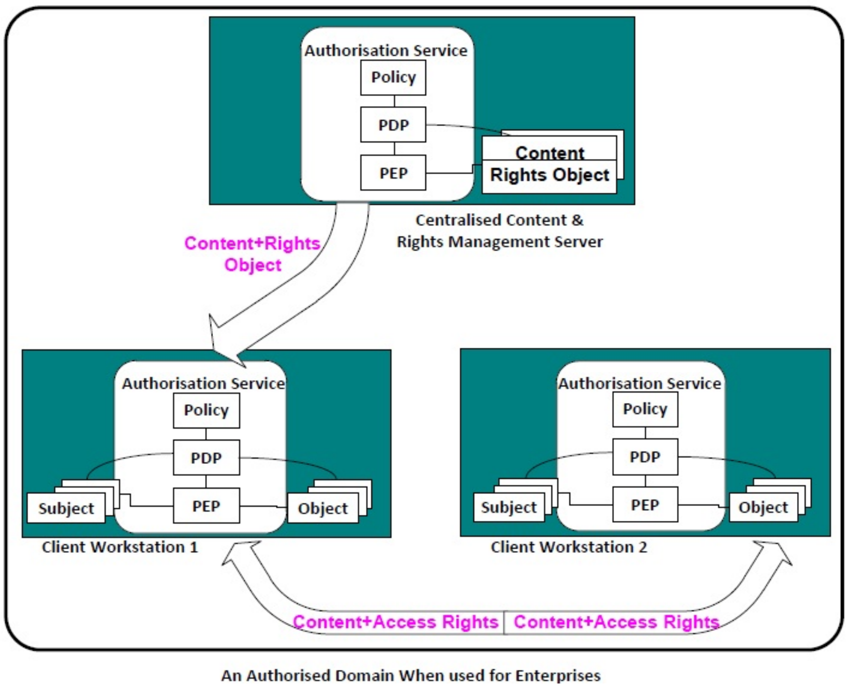


Figure 3.6: DRM with an Authorized Domain Model

for the generic organizational workflow of Section 2.4 for the following reasons:

(a) Organization grouping structure. (a) In our workflow an organization is split into groups and projects. Therefore, organizations require multiple domains and groups. DRM in a personal network constitutes only one specific domain. (b) In DRM, all authorized domains are completely isolated from each other; however, an organization's domains might need to interact based on a predefined policy. (c) In an organization a device might participate in multiple groups and projects at the same time; but, in DRM a device can only be in one domain.

(b) Organization dynamic nature. Unlike DRM, which has a fixed number of devices per domain, in organizations groups are not static but dynamic; for example, in an organization the workflow might change leading to groups' expansion/contraction.

2. Content Sharing. The DRM scheme satisfies the content sharing requirement.

3. Content Protection. The threat of content leakage when authorized users share their credentials can still be realized in the third generation of DRM. As we discussed earlier, adding devices to a domain is conditional on the domain

counter not exceeding a predefined limit. This means the transfer of content protection key to a device is based on the number of devices in a domain and not on which devices are allowed to join the domain. Hence, if the DRM scheme is used for an organization, then any device can join the organization domain and use the domain content even if this device does not belong to the organization or to an authorized employee.

4. Rights Management. The DRM scheme assumes that all users in a domain access each item of content using the same licence file (undifferentiated rights per item of content). However, in the defined generic organizational model, employees do not necessarily have unified access rights on project content (differentiated access rights per item of content). This, in turn, shows that there is a need to look for a proper access rights management scheme to be integrated with the DRM scheme if it is going to be used in organizations.

The work presented in this thesis is attempting to solve *similar* problem to that addressed by DRM, however, we are concerned with a completely different environment. We however build our schemes on the DRM authorized domain schemes using a master control device. In the schemes proposed in this thesis we extend the DRM master control device scheme so that the new scheme addresses the weaknesses in the extended DRM scheme when integrating within organizations. The new proposed schemes have borrowed some of the DRM scheme concepts but use them in different way to achieve our schemes functions which can then be applied to the organization environment. As at this stage we haven't explained our schemes yet, we leave the detailed comparison of the similarities and differences between our proposed scheme and the DRM master controller scheme to Chapter 9.

3.5 Other Schemes

There are other research areas which would be beneficial to our work as they extend our view to insider threats and dynamic domain management. They also can be integrated to our research to extend our work to address other features. These areas are briefly outlined in this section.

3.5.1 Trusted Virtualization

Machine virtualization is a hot topic which is based on concepts that were started in the 1950s. Originally it was proposed to partition expensive mainframes to share resources by hosting different applications shared by many users. The recent adoption of virtualization is to provide multiple execution environments on the same hardware platform. There are many advantages associated with the use of virtual machines, such as process isolation and resource consolidation; however, it is also associated with weaknesses which have been discussed widely (see, for example, [27, 36, 54, 72]). In our thesis we build our work at this stage on physical devices and do not consider the application of our scheme in a virtualization environment. The main reason is that machine virtualization is associated with many properties that require careful analysis when considering it in a solution. Specifically, the migration property requires careful analysis as it allows virtual machines to move between physical devices.

The work of [23, 37] discusses the usage of Trusted Virtual Domains (TVD) to provide process isolation that governs the interaction between independent tasks on a user workstation. We next use the TVD definition from a recent publication [24]: *"TVD is a coalition of virtual machines that trust each other, share a common security policy and enforce it independently of the particular platform they are running on. Moreover, the TVD infrastructure contains the VMM and the physical components on which the virtual machines rely to enforce the policy. In particular, the main features of TVDs and the TVD infrastructure are:*

- *Isolation of execution environments. The underlying VMM provides containment boundaries to compartments from different TVDs, allowing the execution of several different TVDs on the same physical platform.*
- *Trust relationships. A TVD policy defines which platforms (including VMM) and which virtual machines are allowed to join the TVD. For example, platforms and their virtualization layers as well as individual virtual machines can be identified via integrity measurements taken during their start-up.*
- *"Transparent policy enforcement. The Virtual Machine Monitor enforces the security policy independently of the compartments.*
- *"Secure communication channels. Virtual machines belonging to the same TVD are connected through a virtual network that can span over different platforms and that is strictly isolated by the virtual networks of other TVDs."*

Although both this thesis scheme and the TVD scheme have used the domain keyword, both schemes address different areas. For example, the TVD scheme does not discuss the insider threat problem and it does not discuss the content leakage when authorized users share their credentials. TVD focuses on controlling information flow amongst virtual machines. Our scheme, on the other hand, focuses on using physical devices to limit the content leakage threat and we do not consider virtual infrastructure at this stage. The TVD scheme could be integrated with ours to provide additional features, as in the case of process isolation between different applications running on different virtual machines.

3.5.2 Others

The work of [46, 67] presents a set of models for secure information sharing that supports ad-hoc patterns of sharing. The authors refer to such a mode of information sharing as group-centric secure information sharing (g-SIS), which mainly focuses on (what the authors refer to as) the containment challenge. This is to ensure that protected information is accessible on the recipient's computer only as permitted by the policy, including inability to make unprotected or less-protected copies. In g-SIS, users can access information by forming a group. Users join the group and information is made available to members of the group by adding the information to the group. The model discusses all possible cases for accessing information considering the timing of when users join and leave the group, and the time at when information is added or removed from a group. The model also discusses the cases that result in restricting information access when users join multiple groups under predefined conditions, e.g. groups that are mutually exclusive. In the simplest case, multiple groups can be isolated or independent in that membership in one group has no impact on what a user can do in another group, whereas with coupled or connected groups such impact can occur. Looking within a group, g-SIS distinguishes undifferentiated versus differentiated groups. In an undifferentiated group, user authorizations are undifferentiated once users are admitted into the group. Specifically, authorizations do not depend on attributes other than group membership. Combining these two characteristics, g-SIS would have four possible cases for g-SIS models: (isolated, undifferentiated), (isolated, differentiated), (connected, undifferentiated), and (connected differentiated). The g-SIS research undertaken so far has mainly focused on the (isolated, undifferentiated) and (isolated, differentiated) cases, and the model is still under development with further research ongoing in this area.

g-SIS is a generic high level model that does not propose schemes for model enforcement and implementation. Therefore, it assumes that an adequate level of assurance is available in the hardware and software components. The g-SIS does not discuss the insider problem and content leakage. These differentiate our research from the g-SIS. The g-SIS model is of great interest of us as we use it as guidance for future research when working on domain management, managing domain rights, and managing the interaction between domains.

The work in [28, 29] provides *"an analysis of the insider threat problem and formulates a set of requirements for next-generation access-control systems, whose realization might form part of an overall strategy to address the insider problem*

[28]." This work provides an addition to our research in understanding the wider picture of the insider threat in general and provides examples of insider threats. It also helps us to more accurately define the insider threat. This work does not focus on insider threats that can be caused when authorized employees share credentials.

Logging and auditing mechanism enable recording and monitoring system activities. Some schemes attempt to decrease the probability of the content leakage threat by proposing a method for monitoring the activities action on content. The work of Park et al. [59] *"provides scalable and reusable mechanisms to monitor insiders' behavior in organizations, applications, and operating systems based on insiders' current tasks"*. This is achieved by monitoring if an authorized user is performing an abnormal activity on content. The creation of logging that relates user actions, events or conditions is the standard practice for documenting activities that may be part of an attack or that may increase the risk of a future attack [29]. Although this method attempts to detect information leakage, it does not provide a mechanism for preventing content leakage. We believe that preventing information leakage should come before detecting a leakage. This is not to reduce the importance of detection, which should follow the prevention as there is nothing like a 100% secure system. Such a mechanism could be integrated with our proposed scheme to achieve other objectives.

PERMIS [25] is an authorization infrastructure enabling the distributed management of credentials across multiple domains (as in the case of grid computing). Their method controls credential and policy management across different security domains. This method again does not focus on content leakage once the content is in the hands of authorized users.

3.6 Trusted Computing

The proposed schemes in this thesis rely upon trusted computing concepts to securely manage the schemes' keys. This section presents the necessary background about trusted computing and its functions which are required to understand the schemes of the thesis. The material in this section is mainly derived from [34, 35, 65, 74, 75, 76].

TCG specifications require Trusted Platforms (TPs) to have the following functionalities [35].

1. An authenticated boot process.

2. Platform attestation to external entities.

3. Protected storage functionality.

4. A secure boot process.

5. Process isolation.

TCG identifies three "roots of trust", which are used to help in providing TP functionalities. These roots of trust are: (1) Root of Trust for Measurement (RTM); (2) Root of Trust for Storage (RTS); and (3) Root of Trust for Reporting (RTR).

The reminder of this chapter starts by discussing the core component in TCG specifications (i.e. the Trusted Platform Module). We then briefly discuss the three roots of trust. Next, we outline the five main TP components. Finally we discuss the main criticisms raised by researchers on using TPs.

3.6.1 Trusted Platform Module

The core component to establishing trust in an IT system based on TCG specifications is the Trusted Platform Module (TPM) [74, 75, 76]. TPM is generally implemented as a component which must be physically bound to a platform. A TPM must be tamper-evident, i.e. it must provide a limited degree of protection against physical attack.

A TPM incorporates various functional components and features including the following.

I. A cryptographic processor that supports the following operations: asymmetric key generation, asymmetric encryption, digital signing capabilities, hashing, and random number generation. The asymmetric keys that are generated by a TPM could be either migratable or non-migratable. Migratable keys can be transmitted to other TPs if authorized by both a selected trusted authority and the TPM owner. A non-migratable key, on the other hand, is bound to the TP that created it, and cannot be moved.

II. A SHA-1 engine.

III. Protected Storage Capabilities. Once a TPM has been assigned an owner, it generates a new Storage Root Key (SRK), which is an asymmetric key pair. Each TPM has an SRK, which is securely stored inside the TPM and never leaves it. Other TPM objects (key objects or data objects) are protected using keys that are ultimately protected by the SRK in a tree hierarchy. Each object protected by a TPM includes an optional secret 20 bytes of authorization data, which is known as *AuthData*. Proving the knowledge of the value of the AuthData associated with an object grants access to that object.

IV. Non-volatile and volatile memory. Non-volatile memory is needed to store data inside the TPM that should not be removed when the platform loses power or reboots such as the SRK.

V. Platform configuration registers (PCRs) are special purpose registers for only storing platform state. Each PCR is a 20-byte register securely present in a TPM; TCG specifications require that a TPM must contain a minimum of 16 PCR registers. PCR values are reset every time the platform restarts.

Each TPM is associated with a statistically unique asymmetric encryption key pair called an endorsement key pair (EK), which can be generated either internally or using an external key generator at the time of manufacturing. The EK is used only for encryption/decryption purposes. The EK is stored in the TPM at the time of production by the manufacturer. The private decryption endorsement key is known only to the TPM and never revealed outside the TPM. The EK is used when assigning TPM ownership.

3.6.2 TCG Roots of Trust

We now briefly discuss the three TCG roots of trust that we introduced in Section 3.6.

3.6.2.1 The Root of Trust for Measurement

The RTM is a computing engine capable of making reliable measurements of TP running components, with these reliable measurements is known as an integrity measurement.

Definition 3.6.1. Integrity Measurement is a cryptographic digest or hash of a platform component, e.g. a piece of software executing on a platform [50].

In order to ensure that an unlimited number of measurements can be stored in the limited number of PCRs in a TPM, the concept of integrity metric has been raised, which is defined as follows.

Definition 3.6.2. Integrity Metric is a hashed value of integrity measurements. It is calculated by concatenating a new integrity measurement with the existing content of a PCR, and hashing the resulting string. Following that, the resulting integrity metric replaces the old value of the PCR. This is known as “extending” the PCR value.

The RTM is controlled by a particular instruction set, which is known as the core root of trust for measurement (CRTM). On a PC, the CRTM may be contained within the BIOS, and is executed by the platform when it is acting as the RTM. The CRTM must be protected against software attack. The CRTM first measures the first piece of software to be executed during system boot. Next, it passes the measurement result to the RTS that records the result in the TPM PCRs, and then passes control to the next piece of software to be executed, which has a measurement agent (MA) embedded within it. This MA measures the next piece of software to be executed, passes the result to the RTS that records the result in the TPM PCRs, and passes control to the next piece of software to be executed, and so on. MAs are used to build up a chain of trust in the form of a series of integrity measurements. The results of integrity measurements made by the CRTM and MAs are known as measurement events; these are made up of two classes of data: measured values, which are representations of embedded data or program code, and measurement digests, which are hashes of the measured values. The measurement digests are stored in the TPM PCRs. The measurement values are stored in the stored measurement log (SML), which is stored outside the TPM.

3.6.2.2 The Root of Trust for Storage

The RTS is a collection of capabilities which must be trusted if storage of data in a platform is to be trusted [60]. The RTS uses TPM components to achieve its functions. These main functions are as follows.

- Maintaining the integrity measurements made by the RTM by generating the integrity metrics.
- Provides confidentiality and integrity protection to keys and data.

3.6.2.3 The Root of Trust for Reporting

The RTR is a collection of capabilities that must be trusted if reports of integrity metrics are to be trusted (platform attestation) [60]. The RTR works in conjunction with the RTM and the RTS for the implementation of platform attestation.

The RTR enables a TPM to reliably report information about its identity and the current state of the TPM host platform. This is achieved using a set of keys and certificates which are signed by a variety of third parties that must be trusted if the state of the platform is to be trusted. These certificates are as follows.

1. An endorsement credential contains the public EK belonging to a particular TPM. This credential is signed by a trusted platform module entity (TPME), which attests that a particular TPM is genuine. The TPME is likely to be the TPM manufacturer.

2. A conformance credential is signed by a conformance entity (CE) to attest the TP design, i.e. the design of the TPM and other trusted platform building blocks, when integrated into a particular design of platform, meets the TCG specifications.

3. A platform credential is signed by a platform entity (PE) to attest that a particular platform is an instantiation of a TP design, as described in specified conformance credentials. The PE may be the equipment manufacturer.

4. A validation certificate is signed by a validation entity (VE) to certify software components' integrity measurements. The integrity measurements correspond to a correctly functioning platform component (i.e. a piece of software). These validation certificates are used by a challenger wishing to evaluate the state of a challenged TP. The VE is typically the component supplier.

There are also an important set of keys which provide anonymity and simultaneously attest that a particular platform is genuine. These set of keys are called Attestation Identity Keys (AIKs). AIKs (which are signature key pairs) function as aliases for the EK; they are generated by the TPM, and the public part is included in a certificate known as an Identity Credential. The identity credential asserts that the (public part of the) AIK belongs to a TP with specified properties, without revealing which TP the key belongs to. The first generation of TPM (i.e. V1.1) requires a privacy certification Authority (Privacy CA) for certifying AIKs, i.e. signing the AIK Credential confirming that it belongs to a genuine TP.

Before generating an AIK credential, the privacy CA verifies a series of signed credentials belonging to the platform, including the endorsement credential, conformance credential and platform credential. AIKs are used to sign data generated inside the TPM, including the values of PCRs which hold measurements of platform state. AIKs can also be used to sign other keys.

3.6.3 TP Main Components

In this section we discuss the TP main components.

3.6.3.1 The Authenticated Boot Process

Establishing trust in a TP starts by having an initial trusted state. Achieving this requires the assurance of the trustworthiness of a platform whilst the platform starts up during the boot process, i.e. during the authenticated boot process. The authenticated boot process requires the interaction amongst two main TCG components, namely, the RTM and the RTS. We now illustrate this in a form of an example for currently available PCs.

1. The CRTM first measures itself and the rest of the BIOS. The result is then passed to the RTS.

2. The RTS hashes the CRTM output and stores it in PCR 0. The measurement values (i.e. prior to hashing) are stored in the stored measurement log (SML), which is stored outside the TPM. Control is then transferred to the POST BIOS.

3. The POST BIOS measures the host platform configuration, the option ROM code, and the OS loader. The results are then passed to the RTS.

4. The RTS hashes the POST BIOS output and stores these in PCR 1-5. The measurement values are stored in the SML. The RTS then passes the output to the POST BIOS. Control is then transferred to the OS loader.

5. The OS loader measures the OS.

6. At each stage the result of measuring is passed to the RTS, that hashes and then store it. Control is then passed again to the next components, exactly as discussed above, until the OS is loaded.

3.6.3.2 Protected Storage

The protected storage is a fundamental function provided by TPM, which mainly relies on the RTS component to ensure not only data confidentiality and integrity when stored on untrusted devices, but also to bind the usage of the protected data to a specific platform when its execution status in a specific predefined state. In this section we discuss this in detail.

As we discussed earlier each TPM has a specific asymmetric encryption key pair known as SRK. The private key is securely generated and stored inside the TPM and is never released outside it. SRK is the root of the TPM's protected object hierarchy, which is used to protect all objects underneath it. TPM's protected objects could be of either of two types: key object or data object, both of which are discussed below.

Key Object. A TPM incorporates a functional component that supports the protected storage capability, which is a cryptographic co-processor. Part of the cryptographic co-processor's function is generating asymmetric key pairs, where the private part of the key is associated with a data structure containing a set of constraints controlling key usage, for example, forcing the private key to be used only on a specific TPM (i.e. the private key is never exported unencrypted outside the TPM), and forcing the key to be used only when the platform on a specific predefined state.

Data Objects can be either data or symmetric keys that are used to protect bulk data using the platform main processor. A TPM protects a data object's confidentiality by encrypting it using a key at a higher layer of the hierarchy. It also implicitly protects a data object's integrity by associating 20 bytes authorization data (AuthData) with the data object before encryption. When a data object is decrypted, the AuthData is requested and then is compared with the recovered value. If both values do not match, then the decrypted key object will not be released to the caller. On the other hand, if the values match, then the value could be released based on the key storage type. More specifically, TP provides sealing functionality, which only enables the decryption of data objects using the same TPM that encrypted it, and only when the host platform is in a predefined state. This is achieved by associating three additional values with the encrypted data object: (1) tpmProof, which is a TPM-specific secret value forcing which TPM can successfully decrypts the data object; (2) digest at creation, which represents the state of the host platform when the data was sealed enabling the verifier to validate the state of the host platform to ensure that the data was not sealed by a rogue software; and (3) digest at release, which specifies required platform state for releasing the decrypted data object.

3.6.3.3 Platform Attestation

Establishing trust in a TP is based on the mechanisms used for measuring, storing, reporting and verifying platform integrity metrics, i.e. it relies on the following TP components: RTM, RTS, and RTR. Platform attestation is a method to show to a remote party (the verifier) the status and the running environment of a local platform at the time of attestation. The remote party needs to trust the attestator to reliably measure and report its configuration.

The TCG defines integrity management as *"the management of component-information throughout the supply chain to ensure their integrity (tamper-free state) and also to the management of the runtime integrity of the entire TP through the correct management of its components, both at load-time and at runtime"* [73]. As described in Section 3.6.2, TP measurements are performed using the RTM, which measures software components running on a TP. The RTS stores these measurements inside TPM-shielded locations. Next, the RTR mechanism allows TP measurements to be reliably communicated to an external entity in the form of an integrity report. The integrity report is signed using an AIK private key, and is sent with the appropriate identity credential. This enables a verifier to be sure that an integrity report is bound to a genuine TPM. The term *measurement* is used in various ways, as described below.

Loadtime measurements refer to integrity measurements of TP components made whilst the platform is booting-up.

Runtime measurements refer to integrity measurements of TP components that are generated during the operation of the platform, i.e. after the end of a boot-up sequence.

Reference measurements refer to a collection of digest values of TP components, each of which must be collected from the component manufacturer. This provides an authoritative source of component integrity information, which can be read by a verifier of the state of a TP.

Platform attestation works as follows when a requestor, for example, is seeking a service from a verifier.

1. The requestor sends a request to the verifier.
2. The verifier sends a challenge to the requestor, which includes a nonce, to perform an integrity measurement of the entire platform.
3. The requestor returns a platform integrity report to the verifier (using the RTR). The returned reports include the current platform state reflected in integrity metrics associated with the sent nonce. This is then signed using the platform AIK private key, and is associated with SML and the appropriate identity credential.
4. The verifier first needs to verify the TPM's signature and the AIK credential.
5. The verifier then needs to verify if it is safe to trust all or part of the software environment running on the platform. This is achieved by getting the reference measurements for each component of the requestor's platform from its manufacturer. The integrity metric provider, e.g. the hardware manufacturer or software vendor, makes these reference measurements accessible. In this way, the verifier knows both the current integrity-status of the component making-up the requestor's platform, as well as the source-authenticity of those components (as coming from the manufacturer).

6. The verifier needs to identify each component of the requestor's platform and compare the reported measurement against the expected reference measurement value (for each component). If the result is positive, the verifier can provide the requested service.

3.6.3.4 Isolated Execution Environment

An isolated execution environment requires the host platform to provide the following services [50].

1. Whilst a program being executed it should be protected from external interference, for example, by being accessed using Direct Memory Access.

2. Executed programs on the same machine can only communicate via a secure and controlled interprocess communication.

3. Executed programs on different machines (hardware or virtual machine) must communicate via a secure communication channel.

4. Executed programs must communicate with I/O devices via a secure communication channel.

Most proposed mechanisms for providing software isolation are mainly focused on Virtual Machine Monitor (VMM) technology to provide an isolated secure execution environment, and, also, on the use of new processor generation provided, for example, by AMD SVM and Intel's TXT initiative [39, 51, 55].

The virtual machine approach supports multiple operating systems from different vendors that, under the control of VMM, utilizes the hardware of a single machine. The hypervisor presents a virtual machine interface to the operating system and arbitrates requests from the operating system in the virtual machines. Thus, the hypervisor can support the illusion of multiple machines, each of which can run a different operating system image.

3.6.4 Challenges in TCG Specifications

The TCG specifications are large, complex and are based on certain assumptions. In addition to its complexity, building a system that can satisfy such assumptions using today's hardware devices and operating systems is a technical challenge, and is the subject of ongoing research [35, 65]. The following list, based on that given by Gallery and Mitchell [35] and by Sadeghi [65], summarizes these challenges.

- The DAA protocol adapted in TCG specifications is subject to anonymity attack by a malicious DAA issuer, as discussed by Rudolph [64].

- Smyth, Ryan and Chen [71] have pointed out a possible privacy vulnerability in the implementation of the DAA protocol, described as Corrupt Administrator Attacks. In this the verifier with the help of a corrupt DAA issuer can identify a trusted platform.

- The TCG specifications assume that platform configurations cannot be manipulated after the corresponding hash values have been computed and stored in the TPM's PCRs. Satisfying this assumption requires a secure operating system that is especially designed to consider this requirement. Currently available operating systems can easily be modified, e.g. by exploiting security bugs.

- The deployment and use of trusted computing based on the TCG specifications requires a fully functioning trusted computing PKI, which is currently unavailable.

- As discussed earlier in this chapter, a verifier can determine the trustworthiness of code from hash values (binary measurements of running code). Such a binary-based attestation mechanism has the following shortcomings: i) It reveals information about the platform's hardware and software configuration to a verifier, ii) It allows remote parties to exclude certain system configurations, iii) It requires the verifier to know all possible trusted configurations of all platforms, and iv) Most importantly, updates in firmware or software, or hardware migrations, result in changed hash values for the updated components. This, in turn, prevents access to data bound to the previous configuration.

  In principle, attestation should only determine whether a system/component configuration has a desired property. Several methods have been proposed to meet this requirement, such as property-based attestation [2, 47, 66], anonymous property-based attestation [26], and semantic remote attestation using language-based trusted virtual machines [40].

- The TCG specifications implicitly require the establishment of secure channels between hardware components. TPM chips integrated into currently available devices are connected to the I/O board with an unprotected interface that can be eavesdropped upon and manipulated [48]. Secure channels between hardware components can be established using cryptographic mechanisms supported by an appropriate PKI.

- Currently available trusted platforms come pre-equipped with a TPM chip; however they do not have isolation technology and CRTM. Therefore, platform state cannot be reliably measured. This undermines the effect of sealing and platform attestation techniques.

- As discussed in [35], current generation of TC has usability and conformance problems. For example, (a) the TC platform owner when enables a TPM must understand BIOS settings; (b) the TC platform owner is also required to set a TPM owner password; and (c) there are password management issues, as unique passwords may be associated with the TPM owner as well as with data and keys protected by a TPM.

Considering the above problems, it is not to say that TC is far from being realized in practical; great support for TC technology is emerging from the open source community, and from collaborative research projects (e.g. OpenTC[2]). Open source trusted virtualization layers are being developed by both the Xen and L4 communities [18]. Considering that, and, in addition, enterprises infrastructure are more advanced and more managed in comparison with home network environments, it is likely that the technology will first emerge in enterprises rather than in home networks.

3.6.5 TPM Commands

In this section we briefly define the main commands which we use throughout the thesis. These are extracted from [76].

$\mathrm{TPM}_{\mathrm{G}etRandom}$ returns the next requested bytes from the random number generator.

$\mathrm{TPM}_{\mathrm{C}reateWrapKey}$ both generates and creates a secure storage bundle for asymmetric keys. The newly created key can be locked to a specific PCR value by specifying a set of PCR registers.

$\mathrm{TPM}_{\mathrm{L}oadKey2}$ loads a key into the TPM for further use (before the TPM can use a key to wrap, unwrap, unbind, seal, unseal, sign or perform any other action, it needs to be present in the TPM). The TPM assigns the key handle. The TPM always locates a loaded key by use of the handle. The assumption is that the handle may change due to key management operations. It is the responsibility of upper level software to maintain the mapping between the handle and any label used by external software. The load command must maintain a record of whether any previous key in the key hierarchy was bound to a PCR using parentPCRStatus. The flag parentPCRStatus enables the possibility of checking that a platform passed through some particular state or states before finishing in the current state.

[2]http://www.opentc.net/

$\mathrm{TPM}_{\mathrm{S}eal}$ allows software to explicitly state the future "trusted" configuration that the platform must be in for the secret to be revealed. It also implicitly includes the relevant platform configuration (PCR-values) when the SEAL operation was performed. The $\mathrm{TPM}_{\mathrm{S}eal}$ uses the tpmProof value to BIND the sealed blob to an individual TPM.

$\mathrm{TPM}_{\mathrm{U}nSeal}$ operation will reveal $\mathrm{TPM}_{\mathrm{S}ealed}$ data only if it was encrypted on this platform and the current configuration (as defined by the named PCR contents) is the one named as qualified to decrypt it. Internally, $\mathrm{TPM}_{\mathrm{U}nSeal}$ accepts a data blob generated by a $\mathrm{TPM}_{\mathrm{S}eal}$ operation. $\mathrm{TPM}_{\mathrm{U}nSeal}$ decrypts the structure internally, checks the integrity of the resulting data, and checks that the PCR named has the value named during $\mathrm{TPM}_{\mathrm{S}eal}$. Additionally, the caller must supply appropriate AuthData for blob and for the key that was used to seal that data. If the integrity, platform configuration and authorization checks succeed, the sealed data is returned to the caller; otherwise, an error is generated.

$\mathrm{TPM}_{\mathrm{C}ertifyKey}$ operation allows one key to certify the public portion of another key.

$\mathrm{TPM}_{\mathrm{S}ign}$ operation signs data and returns the resulting digital signature.

3.7 Summary

In this chapter we have reviewed and assessed various content protection schemes which focus on secure information sharing and protection. Based on the analysis we have considered how suitable these schemes would be to address the content leakage threat defined in Chapter 2. We also considered the organizational requirements and how well those schemes meet the requirements defined in Section 2.5. We conclude that the means for accessing content needs to be strongly bound not only to the user who is authorized to access the content but also to the device(s) where the user is permitted to work. An organization's security administrators control the type of content each device can access. This is based on an organization's business requirement. In addition, most of the discussed solutions do not use trusted devices, and they mainly rely upon software-only techniques to protect their secrets. Software-only techniques cannot provide a high degree of protection for secret keys stored in a device. This and other problems raise the need for a trusted computing technology to be integrated which can enforce access control rights which could be used to help build a secure application.

This chapter also outlines the main functionality of trusted platforms based on the TCG specifications. The TPM, which is the core component of a TCG-compliant platform, can be conveniently integrated into consumer devices as it is not expensive, does not result in increased device size, and does not introduce new vulnerabilities into end user computing equipment. Moreover, in this chapter we outlined the mechanisms behind platform integrity reporting and validation. This chapter also outlines the main challenges underneath the TCG specifications.

At the end of the first part of this thesis we defined the content leakage problem, the organizational requirements, and we discuss why current mechanisms do not address this problem. In our discussion we explained why we consider the DRM with master control device to be a starting point for our work. In the next part we propose our schemes which attempt to address the content leakage problem. In doing this, we start by introducing our scheme framework which provides our scheme entities, definitions, assumptions, and initialization protocols which are going to be used for the following chapters. Following that, we propose the global and dynamic domain schemes.

Part II

Proposed Schemes

Chapter 4

The Proposed Framework

Contents

This chapter discusses the framework which is required to support the proposed schemes. It covers the common elements, assumptions and definitions which are required in subsequent chapters. It starts by describing the main entities and then introduces the global and dynamic domain concepts, which are discussed in detail in *subsequent chapters. Next, we describe the common initialization steps; specifically we discuss the master controller functionality and the applications that would need to be installed on devices used within the organization to implement the proposed scheme.*

4.1 Introduction

This chapter forms the base for the schemes proposed in this thesis. It introduces all the shared entities, definitions, assumptions and initialization steps. The proposed schemes' entities are repeated in each scheme (e.g. employee, device, and security administrators) and they have the same definitions, requirements, and assumptions in each scheme. In this chapter we have defined these common entities and we set the assumptions and the requirements that are needed in following chapters. The next two chapters, which are based on this chapter, propose the global and dynamic domain schemes. We start with the global domain chapter before the dynamic domain chapter as the global domain represents the parent domain which is composed of dynamic domains. In other words, all devices in an organization are member of the global domain; however, a dynamic domain is composed of selected devices from the global domain. After that, we describe the common initialization steps which are required in subsequent chapters; specifically, we discuss the master controller functionality and the master application that would need to be installed on the master controller and devices used within the organization to implement the proposed scheme.

4.2 The Framework Entities

In this section we define the main roles which are used in our scheme. Other implementation-specific roles will be introduced when needed.

4.2.1 Content

Definition 4.2.1. Content: we are mainly interested in digitized content which needs to be exchanged between client workstations. The nature of content could be text files, images, or executable files. The content could be created by a project's employees, created by a project manager, and/or created by others and then assigned to the project. Content needs to be exchanged between a predefined set of employees who are working on a specific project. In this workflow, content must be protected from being accessed by unauthorized employees. However, we do not address continuously streaming content such as pay TV, as it uses broadcasting principle which are not used by the types of organizations we are focusing on and, in addition, it is related to copyrighted content that DRM solutions focus on.

Assumption 4.2.2. *We assume that content could be either stored in a centralized server or at client devices.*

4.2.2 Employee

Definition 4.2.3. An employee is a user working for an organization to achieve a set of functions defined by the organization. Each employee has a set of authorization privileges based on his role. We require that each employee is assigned a device from where (s)he is authorized to access organization content. An employee is an insider and could also be a malicious insider.

4.2.3 Access Rights

Definition 4.2.4. Access rights are content-specific access rights specific to each employee who requires access to content. Rights could be any fine-grained access conditions which need to be enforced when an employee attempts accessing content. Example of access rights are read, write, execute, and forward.

Assumption 4.2.5. *We assume that members of the organization would be granted access rights on project's content within the system, and that a mechanism is available to manage access rights.*

4.2.4 Security Administrators

Definition 4.2.6. A security administrator is a member of an organization who has high privileges and who is responsible for implementing security policies and procedures. In the proposed scheme the security administrator is responsible for initializing the organization's master controller (defined later in Section 4.4.1), creating domains, and adding/removing devices to/from the dynamic and global domain, as described in Chapter 5. As the security administrators are the main parties who implement the scheme this might overload them. To ease their job an organization might allocate sufficient security administrators to manage the scheme components. Different components might be managed by different administrators for security reasons and to not overload the allocated administrators.

Definition 4.2.7. Authentication Credential: The organization grants security administrators authentication credentials enabling them to access the system. The nature of this credential could be something you know (e.g. password), something you have (e.g. smart card) or something you are (e.g. biometric). We require M out of N security administrators to be authenticated before performing an action, where M is less than N. This provides a dual control by ensuring that more than one security administrator comes to agreement before authorizing an action.

Security administrators are insiders who could also be malicious insiders. Each organization should have its own procedures to verify security administrators' trustworthiness. An example of such procedures are as follows.

1. Following hiring best practices, such as reference checks, background investigation, criminal history, employment termination, and gap periods.
2. Requiring M out of N security administrators to be authenticated before performing an action, where M is less than N. This provides a dual control by ensuring that more than one security administrator come to agreement before authorizing an action.
3. Ensuring that only legitimate security administrators are accessing the scheme confidential devices by using CCTV, which records/tracks devices when moved inside/outside an organization.

4.2.5 Organization Devices

Definition 4.2.8. A device is a personal computer or a server platform that is owned by the organization, and which is capable to of achieving our requirements and assumptions (as described in the following).

In our proposed scheme we require each device to have the following.

1. Software-only techniques do not provide an adequate foundation for building a high-assurance trusted platform [65, 70]. Hence, we require that each devices' hardware is enhanced with trusted computing technology described in Chapter 3. Trusted computing technology can be used to enforce access control policies in such a way that an employee cannot bypass these policies [65].
2. Each device within the organization is required to run a client application, which is trusted to implement the scheme correctly, as discussed in Section 4.5. Each device can verify that the client application is running correctly on another device, using remote attestation discussed in Chapter 3.
3. Each device is required to securely generate for each domain a specific content protection key, which is used to protect the domain content's confidentiality and integrity whilst stored in the device. In this we assume that the encryption algorithm provides authenticated encryption. Authenticated encryption

techniques (for example, OCB 2.0 or Key Wrap [45]) can be used to provide data confidentiality, data integrity, and data origin authentication services. A mechanism of this type typically involves either a combination of a MAC algorithm and a symmetric encryption scheme, or uses an encryption algorithm in a special way so that it provides both integrity and confidentiality protection [45].

4. We also require each device within the organization to have a certified Attestation Identity Key (AIK). We also require that each device within the organization to have a copy of the master controller AIK certificate.

Definition 4.2.9. A device content protection key k_C is a symmetric key that is securely generated and stored by the TPM of the device. Each device generates a specific k_C for each domain it joins. The key k_C is used to protect content whilst stored in the device. k_C is not available in the clear even to the device's authorized user and it cannot be copied between devices. Chapter 5 describes the key generation process when a device joins the global domain and Chapter 6 describes the key generation process when a device joins a dynamic domain.

Assumption 4.2.10. *We assume that each device is used only by a specific employee.*

Assumption 4.2.11. *We assume that devices' hardware is enhanced with trusted computing technology described in Chapter 3.*

Assumption 4.2.12. *We assume that each device has an authentication mechanism that ensures only authorized employees can have access to their assigned devices i.e. authentication/authorization techniques should be integrated with the proposed scheme to verify whether the employee accessing a device is an authorized employee or not; see, for example, [32, 58, 59, 68].*

4.2.6 Certification Authority

A scheme-specific certification authority (CA) could be distributed across multiple locations and needs to be trusted by internal organization domain devices, and across collaborating organizational devices in the case of collaborating organizations. The CA is required to maintain and disseminate a revocation list. In the case of a single organization, the CA could be the organization itself; in the case of a collaborating organizations the CA could a member organization which is trusted by all other member organizations.

4.3 Global and Dynamic Domain Concept

Definition 4.3.1. A domain, for the purposes of this thesis, is defined to be a set of organization devices sharing a secret key. The shared key is used to protect content confidentiality and integrity whilst in transit between the domain devices. Each device within the domain possesses a copy of the shared key that enables only such devices to access the domain assigned content.

In our scheme we use two types of domains: the first we refer to as the **global domain** and the second, which is a subset of the global domain, we refer to as the **dynamic domain**. The following is a brief description of each domain type.

4.3.1 Global Domain

Definition 4.3.2. A global domain is an organization-specific domain, which consists of all devices owned by the organization. The global domain has a unique identifier i_G, a unique shared symmetric key k_G, and a public key list (PKL_g) composed of the public keys of all devices in the domain. Each device in the organization would have a copy of the global domain key k_G. The key k_G is used to protect content in transit that needs to be shared between all devices within an organization, and it is also used to prove a device's ownership to an organization. Each device is required to securely generate a global domain symmetric key k_C, which is used to protect the global domain content when stored in the device. The detail of the global domain proposed workflow are provided in Chapter 5.

Definition 4.3.3. The global domain credentials include the global domain unique identifier i_G, the global domain unique symmetric key k_G, and the public key list (PKL_g) which is composed of the public keys of all devices in the domain. The global domain credentials are discussed in detail in Chapter 5.

4.3.2 Dynamic Domain

Definition 4.3.4. A dynamic domain represents the group of devices that need to share a pool of content. The dynamic domain devices are part of the global domain. Each dynamic domain has a unique identifier i_D, a shared unique key k_D, and a specific PKL_d composed of all devices in the dynamic domain. k_D is shared by all authorized devices in a dynamic domain and is used to protect the dynamic domain content whilst in transit. This key is only available to devices that are members of the domain. Thus only such devices can access the pool of content bound to the domain. As in the case of a global domain, each device is required to securely generate for each dynamic domain a symmetric key k_C, which is used

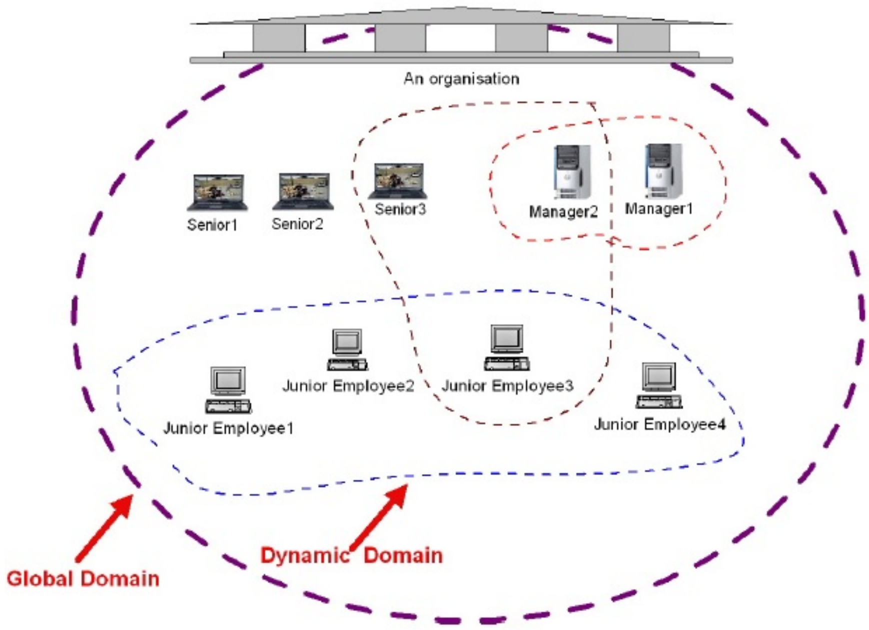


Figure 4.1: Global and Dynamic Domains

to protect the dynamic domain content when stored in the device (see Definition 4.2.9). The dynamic domain is discussed in detail in Chapter 6.

A device can join multiple dynamic domains to access content bound to these domains. By referring to our example in Section 2.4 (where an organization has a group of employees that requires its devices to access a specific pool of content and it does not want that pool of content to leak to other groups) the organization can create a dynamic domain consisting of all devices used by this group of employees, and bind the pool of content to this dynamic domain. Authorized employees, who use devices that are members of a specific dynamic domain, can access the protected content bound to that domain. On the other hand, employees cannot access the protected content from devices that are not members of this domain even if they have a copy of the protected content. Devices that are not members of the domain do not possess a copy of k_D, and hence cannot decrypt the domain-specific content.

Definition 4.3.5. The dynamic domain credentials: each dynamic domain possesses the following credentials: the dynamic domain key (k_D), its identifier (i_D), and public keys for all devices member in the dynamic domain (PKL$_d$). The dynamic domain credentials are discussed in detail in Chapter 6.

Definition 4.3.6. System credentials include the administrators credentials, the global domain credentials, and the dynamic domain credentials.

4.4 Master Controller

The framework requires each organization to have a master controller as defined below.

Definition 4.4.1. **A master controller** is an organization-specific trusted device. The master controller is a dedicated server device (e.g. has high performance capabilities) which is in charge of managing the proposed scheme. The security administrators use the master controller to manage the membership of devices in the global and dynamic domain. Moreover, the master controller enforces organizational policies for domain membership. It runs a trusted master application that is in charge of performing the master controller's main functions as explained in Section 4.5. It is the most important entity that needs to be protected.

We require that the master controller is located in a physically secure environment ensuring that it cannot be stolen and that it is bound to work at a specific location (e.g. bound with a desk). This can be achieved, for example, by binding its execution to a specific location as has been discussed in [9]. We also require that the master controller is monitored by CCTV, and cannot be moved out of the organization's premises. In our schemes the master controller has to have auditing and logging facilities.

The master controller functions are scheme-specific. The global domain master controller functions are included in the dynamic domain master controller functions. Analogously, the dynamic domain master controller functions (which covers the global domain) are included in the collaborating organization master controller functions. We now list the common master controller functions (the details of this are described in Chapters 5 and 6 for the global and dynamic domains respectively).

1. It authenticates the security administrators and securely protects their credentials (where authentication credential is defined in Definition 4.2.7).

2. It creates and manages global and dynamic domains. This includes the following. (a.) Securely generating and storing the global and dynamic domain credentials; (b.) Challenges each device whilst joining a domain to attest to its execution environment status. The master controller then verifies the attestation to ensure that the device is trusted to securely store domain shared keys, securely generates and stores for each domain a content encryption key, protect organization content and execute as expected; (c.) Upon successful attestation, adding devices to a domain by releasing the domain-specific keys (i.e. k_G and k_D); and (d.) Challenges a device which needs to be removed from a domain to attest to its execution environment status. The master controller

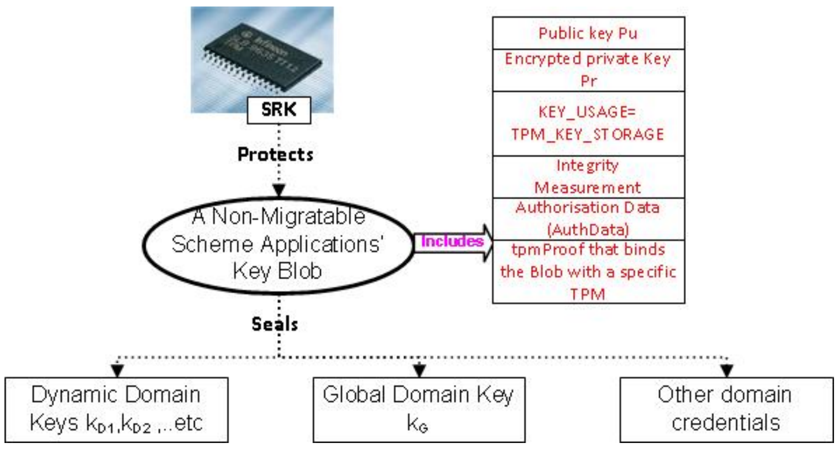

SRK
Protects
A Non-Migratable Scheme Applications' Key Blob
Includes
Public key Pu
Encrypted private Key Pr
KEY_USAGE= TPM_KEY_STORAGE
Integrity Measurement
Authorisation Data (AuthData)
tpmProof that binds the Blob with a specific TPM
Seals
Dynamic Domain Keys k_{D1},k_{D2} ,..etc
Global Domain Key k_G
Other domain credentials

Figure 4.2: Key Hierarchy

then verifies the attestation to ensure that the device is trusted to remove the domain key. Upon successful attestation removing a device from a domain by instructing the device to remove the domain credentials from its storage.

3. It manages domain content by assigning each pool of content to the domain which it belongs to. It then protects the domain content with the corresponding domain key.

4.5 Our Scheme Required Applications

The proposed schemes require software applications to run on clients and servers. We now provide the definition of the scheme applications.

Definition 4.5.1. The Scheme applications act as a reference monitor that are required to implement the proposed scheme. These applications would need to be installed into organization devices including the organization master controller. In our schemes there are two types of application which organizations must implement. The first type (referred to as master application) is used by the master controller for implementing its functions; the second type (referred to as client application) is used by client devices when interacting with the master controller for joining a domain, for domain management, for contentcreation, and for binding the content to a domain. The master application must provide the functions that the organization requires. For example, if the organization has a dynamic domain, then the master application should be capable of supporting dynamic domain functions.

Figure 4.2 illustrates the key hierarchy which is used by the scheme applications.

- tpmProof is *"a random number that each TPM maintains to validate blobs in the SEAL and other processes. The default value is manufacturer-specific."* [75] This is mainly used to bind a key blob with a specific TPM, which makes the key blob non-migratable, as explained below.
- KEY_USAGE is assigned the value TPM_KEY_STORAGE, which indicates the encrypted key blob is a storage key [75]. Usage is explained below.
- AuthData is the authorization session digest for inputs and keyHandle. This means the AuthData value when sealing the blob must match the provided AuthData for unsealing the blob [75].

In this figure a TPM using the SRK seals a private key Pr of an asymmetric key pair (Pr, Pu) to the scheme applications in a form of a non-migratable key blob. The sealing assures that the non-migratable key blob can only be accessed inside the device TPM by the scheme applications when it works as expected. These are assured as follows.

> The value of the tpmProof inside the non-migratable key blob is known only to a specific TPM. Therefore, other TPMs cannot access the key encrypted in the blob.
>
> The integrity measurement (digest at release) stored inside the blob is compared against the platform integrity measurement when unsealing the blob. Only if values match is access to the unsealed blob revealed to the requestor application.

Content protection keys (i.e. k_C, k_D and k_G) are sealed using Pr, i.e. only the trusted scheme applications can decrypt and access content protection keys.

Assumption 4.5.2. *We assume that the scheme application software is designed in such a way that it will not reveal the content protection keys k_D and k_C in the clear, it does not transfer content protection keys to others, and it does not transfer confidential content unprotected to others. TCG-compliant hardware using the sealing mechanism ensures that the only means to access protected content is through the trusted application.*

The client application is responsible for ensuring that access to protected content is granted only to specific requestor applications running on authorized devices when their execution environment is trusted. The specific requestor applications requiring access to content would need to communicate with the device-specific client application to get access to clear text content. In this case, the client application first verifies the requestor application is trusted, and, if so, it decrypts content and then releases the content to the application. This is described in Section 4.5.3. In the next two subsections we discuss the two main types of applications.

4.5.1 Master Application Initialization

This section describes the protocol of initializing the master application. The main objective of this protocol is to prepare the master controller to implement the proposed scheme and manage domain membership. This includes the following:

(1) The security administrators install the master application on the master controller device. The master application installation includes generating a non-migratable key pair (Pr, Pu) to protect system credentials.

(2) The master application retrieves the security administrators' credentials (as defined in Definition 4.2.7), then securely stores them to be used whenever the administrators' needs to authenticate themselves to the master controller.

The first time the security administrators run the master application on the master controller, it performs the following initialization steps (as described by algorithm 1). The objective of this algorithm is to initialize the master application. The master application executes and sends a request to the master controller TPM to generate a non-migratable key pair, which is used to protect system credentials (as defined by Definition 4.3.6). The TPM then generates this key and seals it to be used by the master application when the platform execution status is trusted.

The master controller then needs to ensure that only the security administrators can use the master application. For this, the master application running on the master controller instructs security administrators to provide their authentication credentials (e.g. password/PIN), as described by Algorithm 2. The objective of this algorithm is to enroll security administrators into the master controller. The master application then instructs the TPM to store the authentication credentials of the organization's security administrators associated with its trusted execution environment state (i.e. the integrity measurement, which is stored in the TPM's PCR as described in Chapter 3) in the master controller protected storage. We mean 'sealing data' is storing data in a protected storage so that the data can only be accessed by the trusted master application, as described in Section 3. The authentication credential is used to authenticate security administrators before using the master application; see Algorithm 3. The objective of this algorithm is to verify the security administrators. This algorithm is used every time security administrators want to create, expand, shrink or change a global domain and/or dynamic domains, as explained later in this chapter.

Given the definitions and the assumptions above, the protocol is described by Algorithms 1, 2, and 3. The relationships between these algorithms and their relationship to the overall scheme is illustrated in Figure 5.1 and 6.1. The following are the notations used in the provided protocols.

- M is the master application running on the master controller.
- TPM_M is the TPM on the master controller.
- S_M is the platform state at release as stored in the PCR inside the TPM_M.
- (Pu_M, Pr_M) is a non-migratable key pair such that the private part of the key Pr_M is bound to TPM_M, and to the platform state S_M.
- The following protocol functions are defined in Section 3.6.5: $\text{TPM}_{\text{C}reateWrapKey}$, $\text{TPM}_{\text{L}oadKey2}$, $\text{TPM}_{\text{S}eal}$, and $\text{TPM}_{\text{U}nseal}$.

Algorithm 1 Master Application Initialization (steps need to be applied in order)

1. $M \rightarrow \text{TPM}_M$: $\text{TPM}_{\text{C}reateWrapKey}$.
2. TPM_M: generates a non-migratable key pair (Pu_M, Pr_M).

   Pr_M is bound to TPM_M, and to the required platform state S_M at release, as stored in the PCR inside the TPM_M.
3. $\text{TPM}_M \rightarrow M$: TPM_KEY12$_M$[$\text{Pu}_M$, Encrypted Pr_M, TPM_KEY_STORAGE, tpmProof=TPM_M (NON-MIGRATABLE), S_M, Auth_data]

Algorithm 2 Administrators Registration (steps need to be applied in order)

1. $M \rightarrow$ Administrators: Request security administrator's authentication credentials.
2. $M \rightarrow \text{TPM}_M$: $\text{TPM}_{\text{L}oadKey2}$($\text{Pr}_M$).

   Loads the private key Pr_M in the TPM trusted environment, after verifying the current PCR value matches the one associated with Pr_M (i.e. S_M). If the PCR value does not match S_M, M returns an appropriate error message.
3. $M \rightarrow \text{TPM}_M$: $\text{TPM}_{\text{S}eal}$(Authentication_Credential). Where Authentication_Credential is defined in Definition 4.2.7.

Algorithm 3 Administrator Verification (steps need to be applied in order)

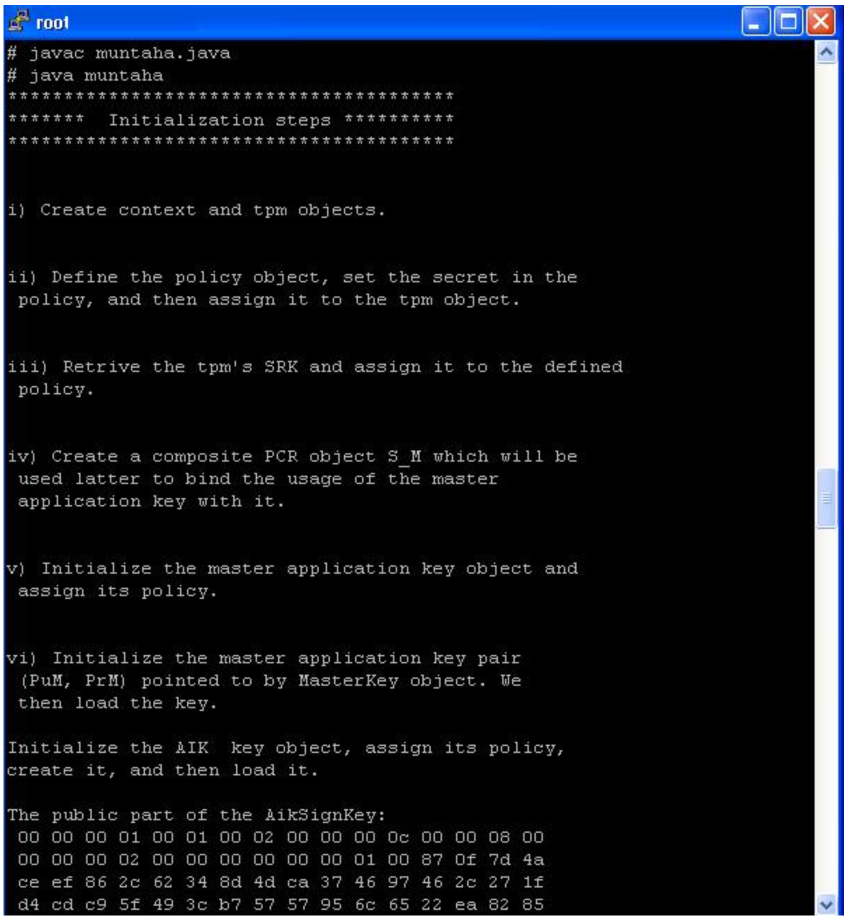
```
1. $M \rightarrow$ Administrators: Request for authentication credentials.

2. $M \rightarrow \text{TPM}_M$: $\text{TPM}_{\text{L}oadKey2}(\text{Pr}_M)$. TPM on $M$ loads the private key $\text{Pr}_M$ in
   the TPM trusted environment, after verifying the current PCR value matches
   the one associated with $\text{Pr}_M$ (i.e. $S_M$). If the PCR value does not match $S_M$,
   $M$ master application returns an appropriate error message.

3. $M \rightarrow \text{TPM}_M$: $\text{TPM}_{\text{U}nseal}$(Authentication_Credential).

4. $\text{TPM}_M$: Decrypts the string Authentication_Credential and passes the result
   to $M$.

5. $M$: Verifies the administrators using the recovered authentication credentials.
   If authentication fails, $M$ returns an appropriate error message.
```

If either of the above algorithm fails, an appropriate error message will be returned which the administrators would need to address before re-executing the algorithm.

Proof of concept code was implemented to verify the protocols; the details of the hosting environment and the code itself are given in Appendix A. The execution output is illustrated in screen-shots 4.3, 4.4, 4.5, and 4.6 for algorithms 1, 2, and 3.

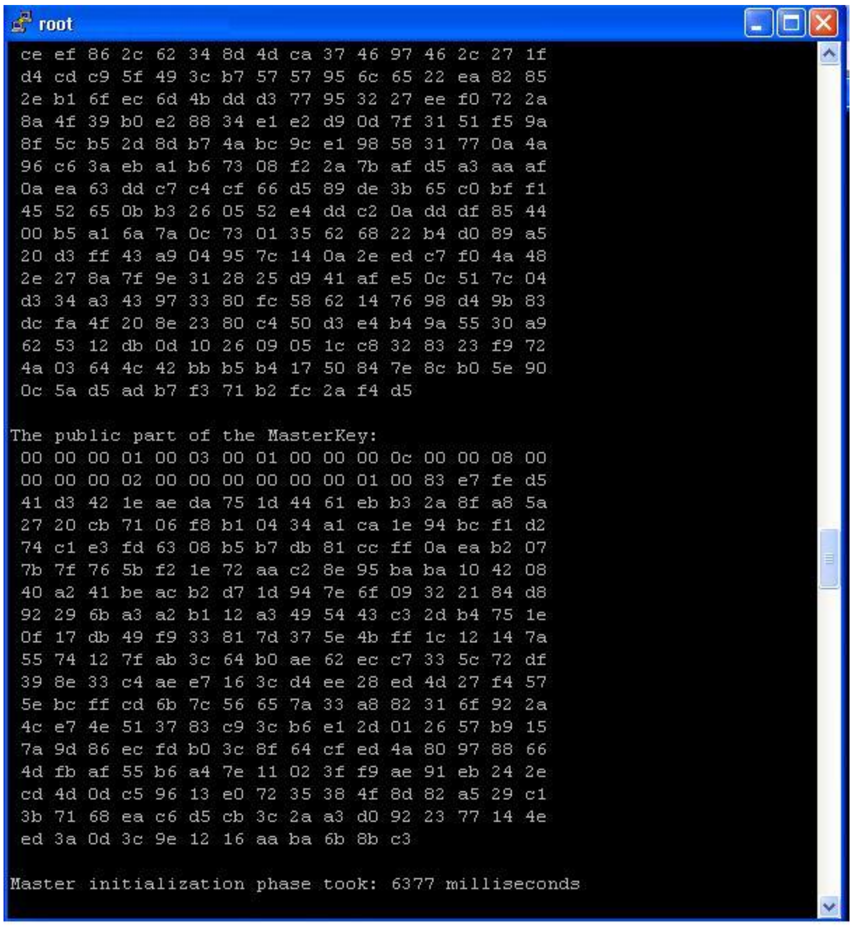
Figures 4.3 and 4.4 show the master initialization process as described by Algorithm 1 and shows the non migratable keys Pu, Pr, and AIK being generated. The values of the keys Pu and AIK are printed out after the generation steps. The process took just over 6 seconds which would vary based on the capabilities of the used device.

Figures 4.5 and 4.6 show the administrator's registration and the administrator's verification described by Algorithms 2 and 3. As illustrated in the screen-shots our proposed scheme requires more than one administrator to be registered. This provides a dual control by ensuring that more than one security administrator comes to agreement before authorizing an action. In the provided screen-shots every time three administrators are authenticated before the application is enabled. The administrators registration took nearly 24 seconds, and the administrator verification took just nearly 14.5 seconds. Note that the administrator registration required much longer time as there are delays caused when waiting for administrator's input. Also, these times would vary based on the running environment.



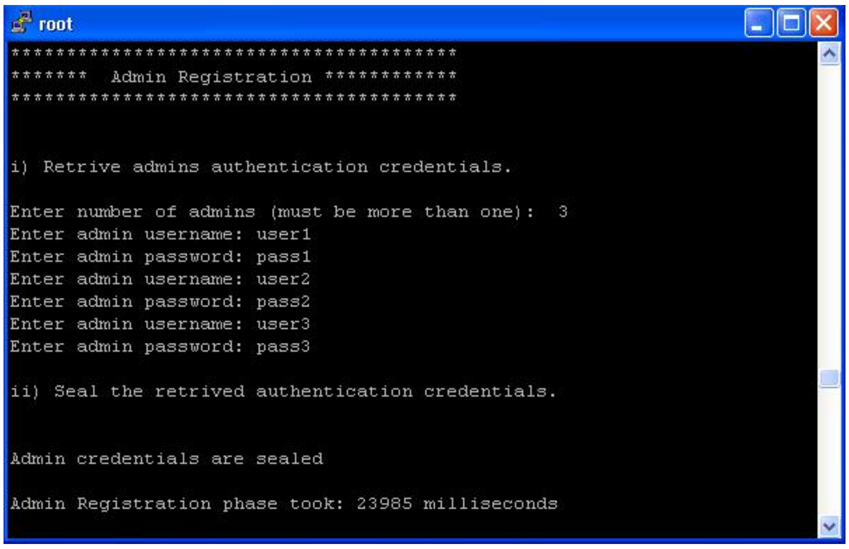
```
# javac muntaha.java
# java muntaha
*****************************************
*******  Initialization steps **********
*****************************************


i) Create context and tpm objects.


ii) Define the policy object, set the secret in the
 policy, and then assign it to the tpm object.


iii) Retrive the tpm's SRK and assign it to the defined
 policy.


iv) Create a composite PCR object S_M which will be
 used latter to bind the usage of the master
 application key with it.


v) Initialize the master application key object and
 assign its policy.


vi) Initialize the master application key pair
 (PuM, PrM) pointed to by MasterKey object. We
 then load the key.

Initialize the AIK  key object, assign its policy,
create it, and then load it.

The public part of the AikSignKey:
 00 00 00 01 00 01 00 02 00 00 00 0c 00 00 08 00
 00 00 00 02 00 00 00 00 00 00 01 00 87 0f 7d 4a
 ce ef 86 2c 62 34 8d 4d ca 37 46 97 46 2c 27 1f
 d4 cd c9 5f 49 3c b7 57 57 95 6c 65 22 ea 82 85
```

Figure 4.3: Master Initialization (a)



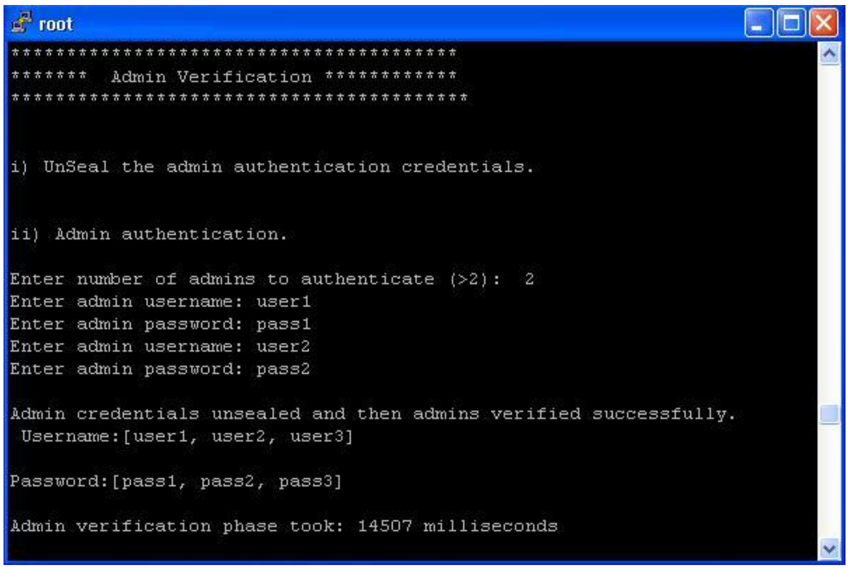
```
 ce ef 86 2c 62 34 8d 4d ca 37 46 97 46 2c 27 1f
 d4 cd c9 5f 49 3c b7 57 57 95 6c 65 22 ea 82 85
 2e b1 6f ec 6d 4b dd d3 77 95 32 27 ee f0 72 2a
 8a 4f 39 b0 e2 88 34 e1 e2 d9 0d 7f 31 51 f5 9a
 8f 5c b5 2d 8d b7 4a bc 9c e1 98 58 31 77 0a 4a
 96 c6 3a eb a1 b6 73 08 f2 2a 7b af d5 a3 aa af
 0a ea 63 dd c7 c4 cf 66 d5 89 de 3b 65 c0 bf f1
 45 52 65 0b b3 26 05 52 e4 dd c2 0a dd df 85 44
 00 b5 a1 6a 7a 0c 73 01 35 62 68 22 b4 d0 89 a5
 20 d3 ff 43 a9 04 95 7c 14 0a 2e ed c7 f0 4a 48
 2e 27 8a 7f 9e 31 28 25 d9 41 af e5 0c 51 7c 04
 d3 34 a3 43 97 33 80 fc 58 62 14 76 98 d4 9b 83
 dc fa 4f 20 8e 23 80 c4 50 d3 e4 b4 9a 55 30 a9
 62 53 12 db 0d 10 26 09 05 1c c8 32 83 23 f9 72
 4a 03 64 4c 42 bb b5 b4 17 50 84 7e 8c b0 5e 90
 0c 5a d5 ad b7 f3 71 b2 fc 2a f4 d5

The public part of the MasterKey:
 00 00 00 01 00 03 00 01 00 00 00 0c 00 00 08 00
 00 00 00 02 00 00 00 00 00 00 01 00 83 e7 fe d5
 41 d3 42 1e ae da 75 1d 44 61 eb b3 2a 8f a8 5a
 27 20 cb 71 06 f8 b1 04 34 a1 ca 1e 94 bc f1 d2
 74 c1 e3 fd 63 08 b5 b7 db 81 cc ff 0a ea b2 07
 7b 7f 76 5b f2 1e 72 aa c2 8e 95 ba ba 10 42 08
 40 a2 41 be ac b2 d7 1d 94 7e 6f 09 32 21 84 d8
 92 29 6b a3 a2 b1 12 a3 49 54 43 c3 2d b4 75 1e
 0f 17 db 49 f9 33 81 7d 37 5e 4b ff 1c 12 14 7a
 55 74 12 7f ab 3c 64 b0 ae 62 ec c7 33 5c 72 df
 39 8e 33 c4 ae e7 16 3c d4 ee 28 ed 4d 27 f4 57
 5e bc ff cd 6b 7c 56 65 7a 33 a8 82 31 6f 92 2a
 4c e7 4e 51 37 83 c9 3c b6 e1 2d 01 26 57 b9 15
 7a 9d 86 ec fd b0 3c 8f 64 cf ed 4a 80 97 88 66
 4d fb af 55 b6 a4 7e 11 02 3f f9 ae 91 eb 24 2e
 cd 4d 0d c5 96 13 e0 72 35 38 4f 8d 82 a5 29 c1
 3b 71 68 ea c6 d5 cb 3c 2a a3 d0 92 23 77 14 4e
 ed 3a 0d 3c 9e 12 16 aa ba 6b 8b c3

Master initialization phase took: 6377 milliseconds
```

Figure 4.4: Master Initialization (b)



```
****************************************
*******  Admin Registration ************
****************************************


i) Retrive admins authentication credentials.

Enter number of admins (must be more than one):  3
Enter admin username: user1
Enter admin password: pass1
Enter admin username: user2
Enter admin password: pass2
Enter admin username: user3
Enter admin password: pass3

ii) Seal the retrived authentication credentials.


Admin credentials are sealed

Admin Registration phase took: 23985 milliseconds
```

Figure 4.5: Administration Registration



```
root
****************************************
*******  Admin Verification ************
*****************************************


i) UnSeal the admin authentication credentials.


ii) Admin authentication.

Enter number of admins to authenticate (>2):  2
Enter admin username: user1
Enter admin password: pass1
Enter admin username: user2
Enter admin password: pass2

Admin credentials unsealed and then admins verified successfully.
 Username:[user1, user2, user3]

Password:[pass1, pass2, pass3]

Admin verification phase took: 14507 milliseconds
```

Figure 4.6: Admin Verification

4.5.2 Client Application Initialization

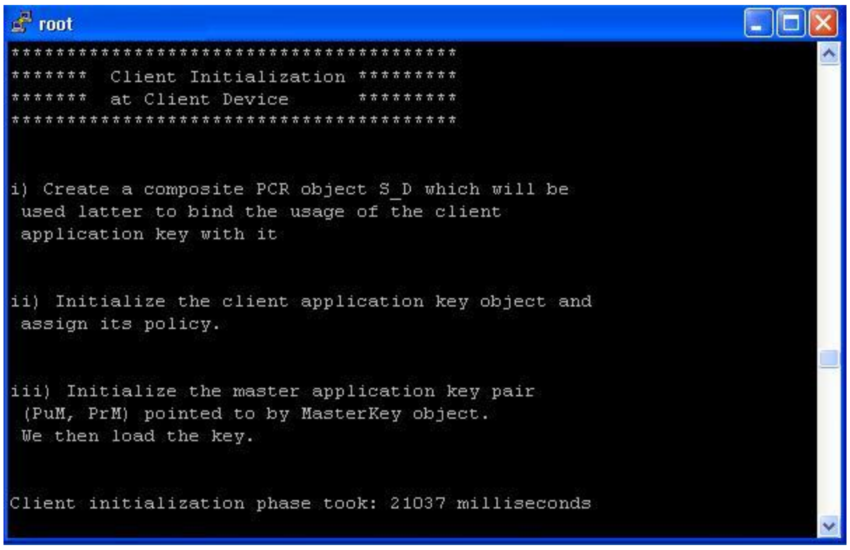
This section describes the protocol for initializing the client application on devices. The goal of this protocol is to prepare a client device to join a domain. The protocol is described by Algorithm 4. The objective of this algorithm is to install at each device a copy of the client application, which generates a non-migratable key to protect system credentials (as defined in Definition 4.3.6) on the client side.

The following are the notations used in the provided protocol.

- D is the client application running on a client device.
- TPM_D is the TPM on a client device.
- S_D is the platform state as stored in the PCR inside the TPM_D.
- (Pu_D, Pr_D) is a non-migratable key pair such that the private part of the key Pr_D is bound to TPM_D, and to the platform state S_D.

Algorithm 4 Client Application Initialization (steps need to be applied in order)

1. $D \rightarrow \text{TPM}_D$: $\text{TPM}_{CreateWrapKey}$.
2. TPM_D: generates a non-migratable key pair (Pu_D, Pr_D).
3. $\text{TPM}_D \rightarrow D$: TPM_KEY12$_D$[$\text{Pu}_D$, Encrypted Pr_D, TPM_KEY_STORAGE, tpmProof=TPM_D (NON-MIGRATABLE), S_D, Auth_data]



```
root
****************************************
*******  Client Initialization *********
*******  at Client Device      *********
****************************************


i) Create a composite PCR object S_D which will be
 used latter to bind the usage of the client
 application key with it


ii) Initialize the client application key object and
 assign its policy.


iii) Initialize the master application key pair
 (PuM, PrM) pointed to by MasterKey object.
 We then load the key.


Client initialization phase took: 21037 milliseconds
```

Figure 4.7: Client Initialization

We have provided in Appendix A a possible implementation of the above algorithm. The execution output of the client initialization algorithm is also illustrated in Figure 4.7. This took 21 seconds on the hardware mentioned previously.

4.5.3 Requstor Application Interaction with the Client Application

In this section we briefly outline how applications can access protected content. When an application requests access to a specific item of content it should communicate with the client application running on the user device. The client application first checks to which domain the item of content belongs. Content is stored protected using a specific domain's content protection key k_C. k_C is stored protected using the client application specific key (Pr as in Figure 4.2), which is protected by the SRK. The SRK is protected inside the device-specific TPM. Both SRK and Pr are not revealed outside the TPM.

The following is the sequence of steps that is performed by the client application when an requstor application wants to access a specific item of content (see Figure 4.8).

1. The client application on the device sends a request to the device-specific TPM to load its own key Pr. The TPM decrypts the client application key using the SRK, and then verifies the client application execution status is as stored inside the key blob. This is done as follows:

   Client application$\rightarrow$ TPM$_D$: TPM$_{\text{L}oadKey2}$(Pr);

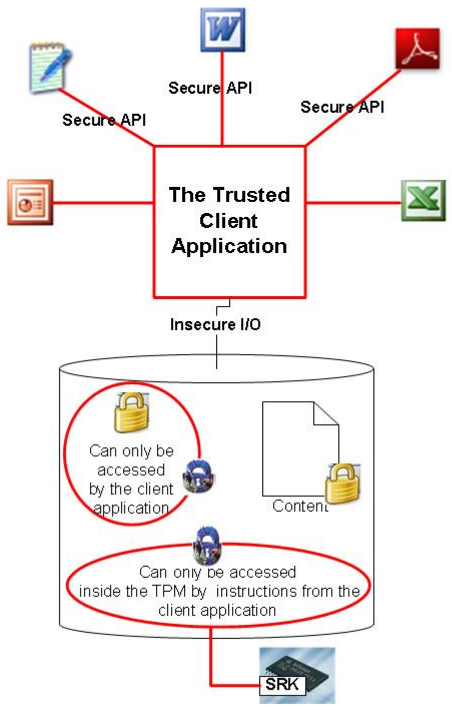

Secure API
Secure API
Secure API
The Trusted Client Application
Insecure I/O
Can only be accessed by the client application
Content
Can only be accessed inside the TPM by instructions from the client application
SRK

Figure 4.8: Requstor Application Interaction with the Client Application

loads the private key Pr in the TPM trusted environment, after verifying the current PCR value matches the PCR value at release associated with Pr.

2. The client application instructs the TPM to decrypt domain key k_C using the client application key. This is done as follows:

   Client application$\rightarrow$ TPM$_D$: TPM$_{\mathrm{U}nseal}(k_C)$.

3. If the client application execution status is as stored inside the key blob the TPM then decrypts k_C. It then releases k_C to the client application.

   TPM$_D$ $\rightarrow$ Client application: k_C

4. The client application instructs the TPM to unload its key. This is done as follows:

   Client application$\rightarrow$ TPM$_D$: TPM$_{\mathrm{E}victKey}$

5. When an application requests access to a protected document, it sends a request to the client application.

6. The client application verifies the trustworthiness of the requestor application. If it is trusted, the client application decrypts the content using k_C and sends it to the requestor application. This is done as follows: (a.) the client application extracts the PCR value of the running application; (b.) the client application also gets the authoritative information, i.e. the Reference Measurements, for the running application from its manufacturer (e.g. software vendor, makes these Reference Measurements accessible in some way); and (c.) the client application now knows both the current integrity-status of the application, as well as the source-authenticity of this (as coming from the manufacturer). The client application then compares the reported measurement against the expected reference measurement value. If the result is positive, the client application can provide the content to the requestor application.

4.6 Assumptions

In this section we re-list our assumptions as follows:

1. Assumption 4.2.2: we assume that content could be either stored in a centralized server or at client devices.

2. Assumption 4.2.5: we assume that members of the organization would be granted access rights on a project's content within the system, and that a mechanism is available to manage the access rights.

3. Assumption 4.2.10: we assume that each device is only used by a specific employee.

4. Assumption 4.2.11: we assume that devices' hardware is enhanced with trusted computing technology described in Chapter 3.

5. Assumption 4.2.12: we assume that each device has an authentication mechanism that ensures only authorized employees can have access to their assigned devices i.e. authentication/authorization techniques should be integrated with the proposed scheme to verify whether the employee accessing a device is an authorized employee or not; see, for example,[32, 58, 59, 68].

6. Assumption 4.5.2: we assume that the scheme application is designed in such a way that it will not reveal the content protection keys k_D and k_C in the clear, it does not transfer content protection keys to others, and it does not transfer confidential content unprotected to others. TCG-compliant hardware using the sealing mechanism ensures that the only means to access protected content is through the trusted client application.

4.7 Summary

This chapter describes the common entities and workflow that are used as a base for subsequent chapters. It starts by defining the main roles which are used in our scheme. This is followed by defining the global and dynamic domain concepts, which are the most important concepts that we used throughout the remaining of this thesis to reduce the effect of information leakage and, simultaneously, limit the effect of content leakage. Next we defined the master controller and its main functions. Finally, we defined the scheme applications and their functions for implementing the core components of the proposed scheme.

Chapter 5

The Global Domain

Contents

In this chapter we describe the scheme which presents the global domain concept. This scheme involves managing a global domain using an organization-specific master controller. We then discuss how the proposed scheme prevents external leakage (as defined by Definition 2.2.5) whilst allowing authorized users to access organization's confidential information from inside or outside an organization's premises.

5.1 Introduction

In this chapter we propose a scheme for the global domain, introduced in Chapter 4. The proposed scheme mainly focuses on protecting content in such a way that if it is transferred, it cannot be accessed except on devices authorized by the organization's security administrators. In other words, accessing content is restricted to devices owned by the same organization and authorized by security administrators. Therefore, authorized users cannot disclose protected information to others not authorized to access such information.

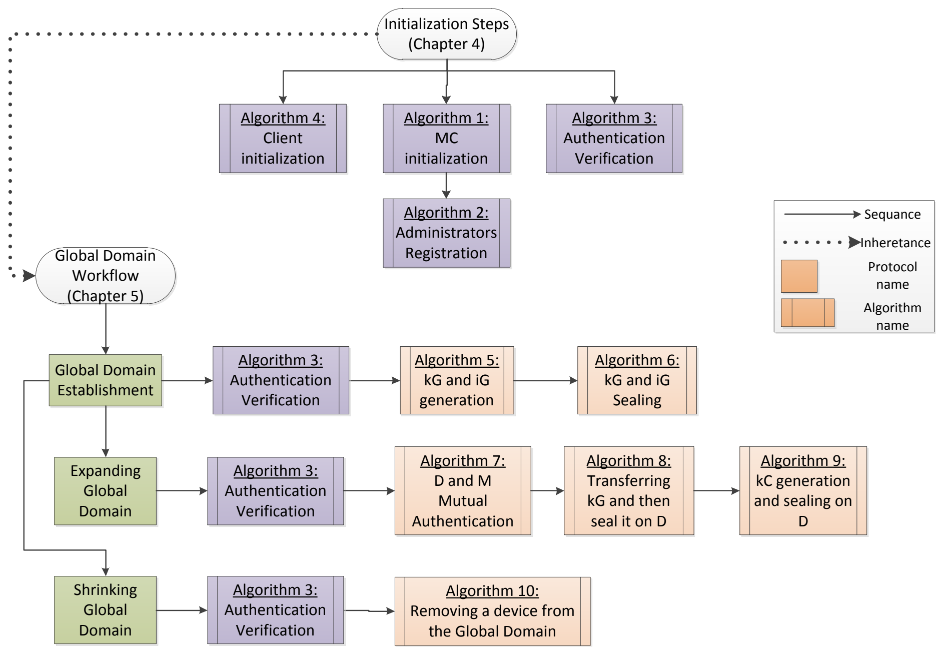

Initialization Steps (Chapter 4)
Algorithm 4: Client initialization
Algorithm 1: MC initialization
Algorithm 3: Authentication Verification
Algorithm 2: Administrators Registration
Sequance
Inheretance
Protocol name
Algorithm name
Global Domain Workflow (Chapter 5)
Global Domain Establishment
Algorithm 3: Authentication Verification
Algorithm 5: kG and iG generation
Algorithm 6: kG and iG Sealing
Expanding Global Domain
Algorithm 3: Authentication Verification
Algorithm 7: D and M Mutual Authentication
Algorithm 8: Transferring kG and then seal it on D
Algorithm 9: kC generation and sealing on D
Shrinking Global Domain
Algorithm 3: Authentication Verification
Algorithm 10: Removing a device from the Global Domain

Figure 5.1: Global Domain Algorithms Sequence

This chapter is organized as follows. Section 5.2 describes the proposed solution and the process workflow. Section 5.3 describes how the proposed scheme controls domain membership. Section 5.4 describes the importance of the proposed scheme in the context of a scenario. Section 5.5 provides conclusions.

5.2 System Workflow

The global domain was introduced in Chapter 4 . In this section we provide details of the proposed workflow. Figure 5.1 illustrates the sequence of algorithms which are required in this chapter. The figure illustrates three main domain workflow protocols, i.e. global domain establishment, expanding the global domain, and shrinking the global domain. For each protocol, the figure illustrates the algorithms associated with the protocol and their order of execution. For example, the global domain establishment protocol involves executing Algorithms 3, 5, and then 6 in order. The arrows in the figure indicate the sequence/dependencies of events. For example, both expanding the global domain and shrinking the global domain depends on the global domain establishment. A failure in any algorithm would require a rollback of all dependent algorithms. In such cases the system administrators would need to check the reasons for failures before proceeding to re-run the protocol.

In this section we require that the scheme applications have already been installed on devices, exactly as described in Section 4.5. This includes installing both master application and the client application, which interact with their corresponding devices' TPM to generate a non-migratable key pair which can be only used by the scheme applications. This key pair is used to protect global domain credentials including the global domain unique identifier i_G, the global domain unique symmetric key k_G, and the public key list of the global domain (PKL_g). In the following we define each credential.

Definition 5.2.1. A global domain public key list ($\mathbf{PKL}_g$) is a list that is composed of the public keys of all devices in the organization. The PKL_g is securely stored in the master controller and is managed by security administrators using the master application.

Public key list management (PKL_g) is a data structure whose integrity and confidentiality need to be protected. We protect PKL_g confidentiality to stop attackers from understanding the organization domain's structure. PKL_g is managed by security administrators via the master application. Security administrators follow organization requirements and policy when deciding on devices that should be in the domain, i.e. added to PKL_g. When a client device requests to join a domain, the master controller asks the joining device to attest to the trustworthiness of the client device, as discussed in Section 5.2.2. If the client device is trusted then the master controller should ensure that the public key of the client device is in the PKL_g. Each public key in PKL_g represents a client device specific application public key. The client device is designed to generate the key as a non-migratable key, and to seal this key to when a client device requests "joining work" with the client application when it runs as expected, as discussed in Chapter 4. By this we can be assured that the public key cannot be sent to other devices enabling such devices to pretend as one of the domain devices.

Definition 5.2.2. A global domain identifier i_G is a random number that is securely generated and stored by the TPM of the master controller. i_G is used to identify the global domain whenever a device needs to join the global domain or encrypt content with the global domain key, as described in Section 5.2.1.

Definition 5.2.3. A global domain key k_G is a symmetric key that is securely generated and stored by the TPM of the master controller as described in Section 5.2.1. k_G is not available in the clear even to the domain security administrators and it cannot be directly copied between devices. The key k_G is only transfered from the master controller to a device when a device joins the global domain. The key

k_G is used to protect content which needs to be shared between the global domain devices (i.e. all devices within an organization) **whilst in transit**, and it is also used to prove a device's ownership to an organization.

A device proves membership of a global domain by proving possession of k_G; this is because the device can only obtain k_G from the master controller: k_G is transferred from the master controller to all devices in the organization after verifying the client application running on devices performs as expected, as discussed in Section 4.5. This means the client application will not reveal k_G in the clear, and will not transfer k_G to others. Devices that are not owned by the organization will fail to have a copy of k_G, unless security administrators have agreed for these devices to become part of the global domain and then add them to the PKL_g. This is because k_G can only be copied from the master controller to joining devices after successfully authenticating security administrators and checking that the device public key is in the PKL_g. Each device generates a unique k_C which is used to protect global domain content while being stored on the client device, as defined in Definition 4.2.9. The key k_C generation process is described in Section 5.2.1.

5.2.1 Global Domain Establishment

The master controller defined in Section 4.4 is responsible for setting up the global domain. The master controller initiates the global domain creation process using the following protocol. The objective of this protocol is to initialize global domain credentials at the master controller, which include the global domain key, identifier and the PKL_g. These are protected by the master controller, which manages domain membership.

The following notations are used in the protocol.

- M is the master application running at the master controller.
- TPM_M is the TPM on the master controller.
- S_M is the platform state at release as stored in the PCR inside the TPM_M.
- (Pu_M, Pr_M) is non-migratable key pair such that the private part of the key Pr_M is bound to TPM_M, and to the platform state S_M.
- The functions $\text{TPM}_{\text{C}reateWrapKey}$, $\text{TPM}_{\text{L}oadKey2}$, $\text{TPM}_{\text{S}eal}$, $\text{TPM}_{\text{S}ign}$ and $\text{TPM}_{\text{U}nseal}$ are defined in Section 3.6.5.

The establishment of a global domain proceeds as follows.

1. The security administrators instruct the master controller to create the organization's global domain.

2. The master controller authenticates security administrators as described in Algorithm 3.

3. If the authentication succeeds, M instructs security administrators to provide the public keys of all devices in the organization in the form of a public key list (PKL$_g$).

4. The master application interacts with the TPM to securely generate a global domain-specific secret key k_G and a global domain specific identifier i_G as follows.

Algorithm 5 k_G and i_G generation (steps need to be applied in order)

```
(a) $M \rightarrow \text{TPM}_M$: $\text{TPM}_{\text{G}etRandom}$;
    Generates a random number to be used as a global domain key $k_G$
(b) $\text{TPM}_M \rightarrow M$: $k_G$
(c) $M \rightarrow \text{TPM}_M$: $\text{TPM}_{\text{G}etRandom}$;
    Generates a random number to be used as a global domain identifier $i_G$
(d) $\text{TPM}_M \rightarrow M$: $i_G$
```

If Algorithm 5 fails the credentials will not be created, an appropriate error message will appear, and algorithm 6 will not run.

5. The global domain credentials k_G, i_G, and PKL$_g$ are stored in the master controller protected storage, and sealed to the master application so that only the master application can access these credentials when its execution status is trusted. This is achieved as follows.

Algorithm 6 k_G and i_G sealing (steps need to be applied in order)

(a) $M \rightarrow \text{TPM}_M$: $\text{TPM}_{\text{L}oadKey2}(\text{Pr}_M)$;

M loads the private key Pr_M in the TPM trusted environment, after verifying the current PCR value matches the one associated with Pr_M (i.e. S_M). If the PCR value does not match S_M, M returns an appropriate error message.

(b) $M \rightarrow \text{TPM}_M$: $\text{TPM}_{\text{S}eal}(k_G||i_G||\text{PKL}_g)$.

TPM_M securely stores the string $k_G||i_G||\text{PKL}_g$ using the platform protected storage, such that they can only be decrypted on the current platform by M, and only if the platform runs as expected (when the platform PCR values matches the ones associated with Pr_M, i.e. S_M).

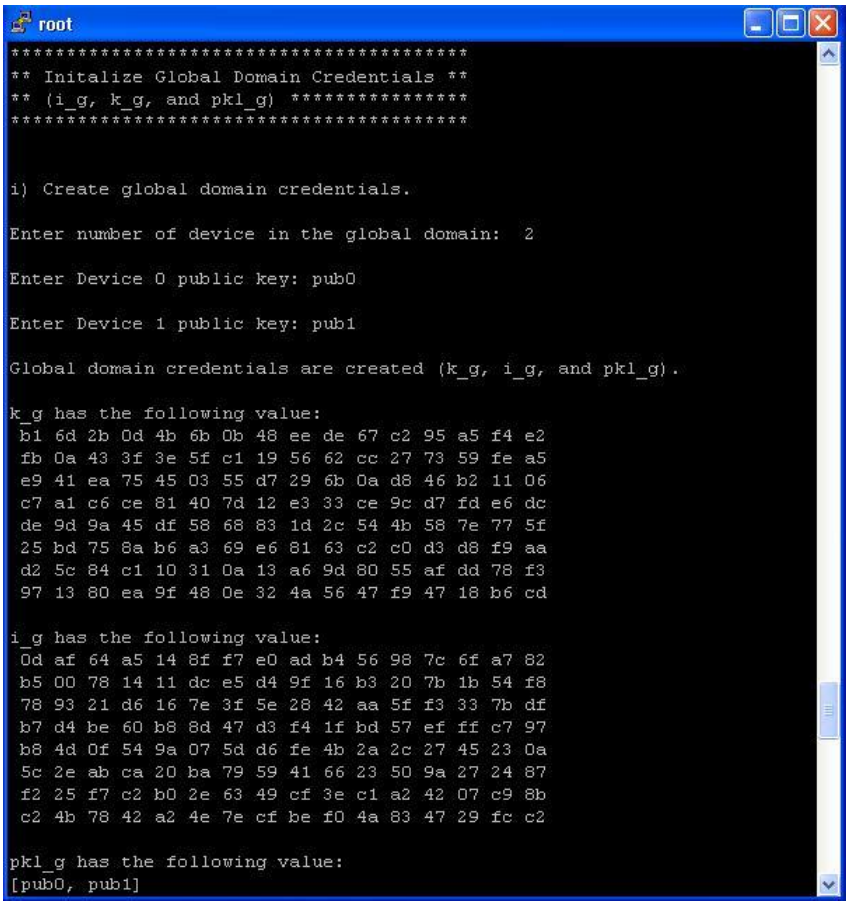
Algorithms 5 and 6 were implemented in Java code which is given in Appendix A. Figures 5.2 and 5.3 show screen-shots of running algorithms. These algorithms were executed on the environment as described in Appendix A and they took around 10 seconds. In these screen-shots we generate the global domain credentials (i_i, k_g, PKL_g). We then print their values and finally we seal them to the device.

5.2.2 Expanding Global Domain

This section describes the process of expanding a global domain by adding a device to an organization-specific global domain. We require that a global domain has been established in the master controller, as described in the previous section. We also require that the client application has already been installed on the device as described in Algorithm 4.

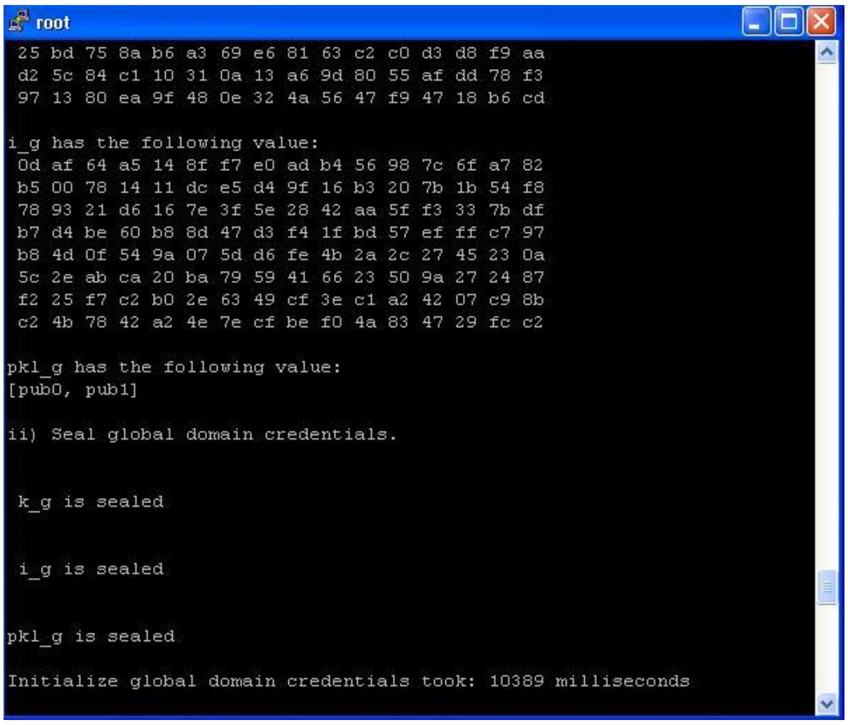
The following notation is used in the protocol.

- D is the client application running on a client device.
- M is the master application running on the master controller.
- TPM_D is the TPM on a client device.
- TPM_M is the TPM on the master controller.
- S_D is the platform state at release as stored in the PCR inside the TPM_D.
- S_M is the platform state at release as stored in the PCR inside the TPM_M.
- (Pu_D, Pr_D) is a non-migratable key pair such that the private part of the key Pr_D is bound to TPM_D and to the platform state S_D.



```
*******************************************
** Initalize Global Domain Credentials **
** (i_g, k_g, and pkl_g) ****************
*******************************************


i) Create global domain credentials.

Enter number of device in the global domain:  2

Enter Device 0 public key: pub0

Enter Device 1 public key: pub1

Global domain credentials are created (k_g, i_g, and pkl_g).

k_g has the following value:
 b1 6d 2b 0d 4b 6b 0b 48 ee de 67 c2 95 a5 f4 e2
 fb 0a 43 3f 3e 5f c1 19 56 62 cc 27 73 59 fe a5
 e9 41 ea 75 45 03 55 d7 29 6b 0a d8 46 b2 11 06
 c7 a1 c6 ce 81 40 7d 12 e3 33 ce 9c d7 fd e6 dc
 de 9d 9a 45 df 58 68 83 1d 2c 54 4b 58 7e 77 5f
 25 bd 75 8a b6 a3 69 e6 81 63 c2 c0 d3 d8 f9 aa
 d2 5c 84 c1 10 31 0a 13 a6 9d 80 55 af dd 78 f3
 97 13 80 ea 9f 48 0e 32 4a 56 47 f9 47 18 b6 cd

i_g has the following value:
 0d af 64 a5 14 8f f7 e0 ad b4 56 98 7c 6f a7 82
 b5 00 78 14 11 dc e5 d4 9f 16 b3 20 7b 1b 54 f8
 78 93 21 d6 16 7e 3f 5e 28 42 aa 5f f3 33 7b df
 b7 d4 be 60 b8 8d 47 d3 f4 1f bd 57 ef ff c7 97
 b8 4d 0f 54 9a 07 5d d6 fe 4b 2a 2c 27 45 23 0a
 5c 2e ab ca 20 ba 79 59 41 66 23 50 9a 27 24 87
 f2 25 f7 c2 b0 2e 63 49 cf 3e c1 a2 42 07 c9 8b
 c2 4b 78 42 a2 4e 7e cf be f0 4a 83 47 29 fc c2

pkl_g has the following value:
[pub0, pub1]
```

Figure 5.2: Initialize Global Domain (a)



```
root
 25 bd 75 8a b6 a3 69 e6 81 63 c2 c0 d3 d8 f9 aa
 d2 5c 84 c1 10 31 0a 13 a6 9d 80 55 af dd 78 f3
 97 13 80 ea 9f 48 0e 32 4a 56 47 f9 47 18 b6 cd

i_g has the following value:
 0d af 64 a5 14 8f f7 e0 ad b4 56 98 7c 6f a7 82
 b5 00 78 14 11 dc e5 d4 9f 16 b3 20 7b 1b 54 f8
 78 93 21 d6 16 7e 3f 5e 28 42 aa 5f f3 33 7b df
 b7 d4 be 60 b8 8d 47 d3 f4 1f bd 57 ef ff c7 97
 b8 4d 0f 54 9a 07 5d d6 fe 4b 2a 2c 27 45 23 0a
 5c 2e ab ca 20 ba 79 59 41 66 23 50 9a 27 24 87
 f2 25 f7 c2 b0 2e 63 49 cf 3e c1 a2 42 07 c9 8b
 c2 4b 78 42 a2 4e 7e cf be f0 4a 83 47 29 fc c2

pkl_g has the following value:
[pub0, pub1]

ii) Seal global domain credentials.


 k_g is sealed


 i_g is sealed


pkl_g is sealed

Initialize global domain credentials took: 10389 milliseconds
```

Figure 5.3: Initialize Global Domain (b)

- $(\mathrm{Pu}_M, \mathrm{Pr}_M)$ is a non-migratable key pair such that the private part of the key Pr_M is bound to TPM_M and to the platform state S_M.
- Cert_M is the master controller's AIK certificate.
- Cert_D is the joining device's AIK certificate.
- A_M is an identifier for the master controller included in Cert_M.
- A_D is an identifier for a device included in Cert_D.
- Pr_{MAIK} is the private key of the master controller AIK.
- Pr_{DAIK} is the private key of the device AIK.
- N_1 is a randomly generated nonce.
- N_2 is a randomly generated nonce.
- $e_{Pu_D}(Y)$ denotes the asymmetric encryption of data Y using key Pu_D, and where we assume that the encryption primitive in use provides non-malleability, as described in [44].
- SHA1 is a one way hash function.

- The protocol functions $\text{TPM}_{\text{C}reateWrapKey}$, $\text{TPM}_{\text{L}oadKey2}$, $\text{TPM}_{\text{S}eal}$, $\text{TPM}_{\text{S}ign}$ and $\text{TPM}_{\text{U}nseal}$ are defined in Section 3.6.5.

The client device sends a join domain request to the master controller to install the domain specific key k_G. This request includes the global domain specific identifier i_G and is achieved as follows.

1. $D \rightarrow M$: Join_Global_Domain

Three algorithms are then initiated to add the device to the global domain, which are discussed below.

The first algorithm involves the master controller and the joining device to mutually authenticate each other conforming to the three-pass mutual authentication protocol [43]. The objective of this algorithm is to establish a trusted communication channel between the master controller and the client device. The master controller sends an attestation request to the joining device to prove its trustworthiness then the device sends the attestation outcome to the master controller. These steps are achieved using the following algorithm.

Algorithm 7 D and M mutual authentication (steps need to be applied in order)

1. $M \rightarrow \text{TPM}_M$: $\text{TPM}_{\text{G}etRandom}$.
2. $\text{TPM}_M \rightarrow M$: Generates a random number to be used as a nonce N_1.
3. $M \rightarrow \text{TPM}_M$: $\text{TPM}_{\text{L}oadKey2}(\text{Pr}_{MAIK})$;

   M loads the master controller AIK in the TPM trusted environment, after verifying the current PCR value matches the one associated with Pr_{MAIK}.
4. $M \rightarrow \text{TPM}_M$: $\text{TPM}_{\text{S}ign}(\text{N}_1)$.
5. $\text{TPM}_M \rightarrow M \rightarrow D$: $N_1||\text{Cert}_M||\text{Sign}_M(N_1)$.
6. D verifies Cert_M, extracts the signature verification key of M from Cert_M, and checks that it has not been revoked, e.g. by querying an OCSP service. D then verifies message signature. If the verifications fail, D returns an appropriate error message.
7. $D \rightarrow \text{TPM}_D$: $\text{TPM}_{\text{G}etRandom}$.
8. $\text{TPM}_D \rightarrow D$: Generates a random number N_2 that is used as a nonce.
9. $D \rightarrow \text{TPM}_D$: $\text{TPM}_{\text{L}oadKey2}(\text{Pr}_{DAIK})$;

   D loads the private key Pr_{DAIK} in the TPM trusted environment, after verifying the current PCR value matches the one associated with Pr_{DAIK}.
10. $D \rightarrow \text{TPM}_D$: $\text{TPM}_{\text{C}ertifyKey}(\text{SHA1}(N_2||N_1||A_M||i_G),\text{Pu}_D)$. TPM_D attests to its execution status by generating a certificate for the key Pu_D.
11. $\text{TPM}_D \rightarrow D$: $N_2||N_1||A_M|_D||S_D||i_G||\text{Sign}_D(N_2||N_1||A_M||i_G||Pu_D||S_D)$.
12. $D \rightarrow M$: $N_2||N_1||A_M||Pu_D||S_D||i_G||\text{Cert}_D||\text{Sign}_D(N_2||N_1||A_M||i_G||Pu_D||S_D)$.
13. M verifies Cert_D, extracts the signature verification key of D from the certificate, and checks that it has not been revoked, e.g. by querying an OCSP service. M then verifies message signature, message freshness by verifying the value of N_1, and then verifies it is the intended recipient by checking the value of A_M. M determines if D is executing as expected by comparing the platform state given in S_D with the predicted platform integrity metric. If these validations fail, then M returns an appropriate error message.

If Algorithm 7 fails the secure channel will not be established between the master controller and the device. In addition, an appropriate error message will be returned which the administrators would need to address before executing the algorithm again.

If the joining device execution environment is trusted, the master controller checks if the device's public key is included in the public key list for the global domain. If so, it securely releases the global domain specific key k_G to the device as follows.

The second algorithm for adding a device into the global domain starts upon successful completion of the above algorithm. The objective of this is to securely transfer the domain key k_G to the joining device D. The key are sealed on D, so that it is only released to the client application when its execution environment is as expected.

Algorithm 8 Transferring k_G and then seal it on D (steps need to be applied in order)

1. $M \rightarrow \text{TPM}_M$: $\text{TPM}_{\text{L}oadKey2}(\text{Pr}_M)$.

   The TPM on M loads the private key Pr_M into the TPM trusted environment, after verifying the current PCR value matches the one associated with Pr_M (i.e. S_M). If the PCR value does not match S_M, the master application returns an appropriate error message.

2. $M \rightarrow \text{TPM}_M$: $\text{TPM}_{\text{U}nseal}(k_G||i_G||\text{PKL}_g)$.

3. $\text{TPM}_M \rightarrow M$: decrypts the string $k_G||i_G||\text{PKL}_g$ and passes the result to M.

4. M verifies i_G matches the recovered global domain identifier and Pu_D is included in the PKL_g. If so M encrypts k_G using the key Pu_D as follows $e_{Pu_D}(k_G)$.

5. $M \rightarrow \text{TPM}_M$: $\text{TPM}_{\text{C}ertifyKey}(\text{SHA1}(N_2||A_D||e_{Pu_D}(k_G)),\text{Pu}_M)$.

6. $\text{TPM}_M \rightarrow M$: attests to its execution status by generating a certificate for the key Pu_M, and sends the result to M.

7. $M \rightarrow D$: $N_2||A_D||Pu_M||S_M||e_{Pu_D}(k_G)||\text{Sign}_M(\text{N}_2||A_D||e_{Pu_D}(k_G)||\text{Pu}_M||S_M)$.

8. The device D verifies message signature, it is the intended recipient by checking the value of A_D, and verifies message freshness by checking the value of N_1. If verifications succeed, D stores the string $e_{Pu_D}(k_G)$ in its storage.

If Algorithm 8 fails the global domain key will not be transfered, an appropriate error message will be returned which the administrators would need to address before re-executing the algorithm.

The third algorithm for adding a device into the global domain starts upon successful completion of the above algorithm. The objective of the protocol is to securely generate content encryption key k_C for the domain. The keys are sealed on D, so that they are only released to client application when its execution environment is as expected.

Algorithm 9 k_C generation and sealing it on D (steps need to be applied in order)

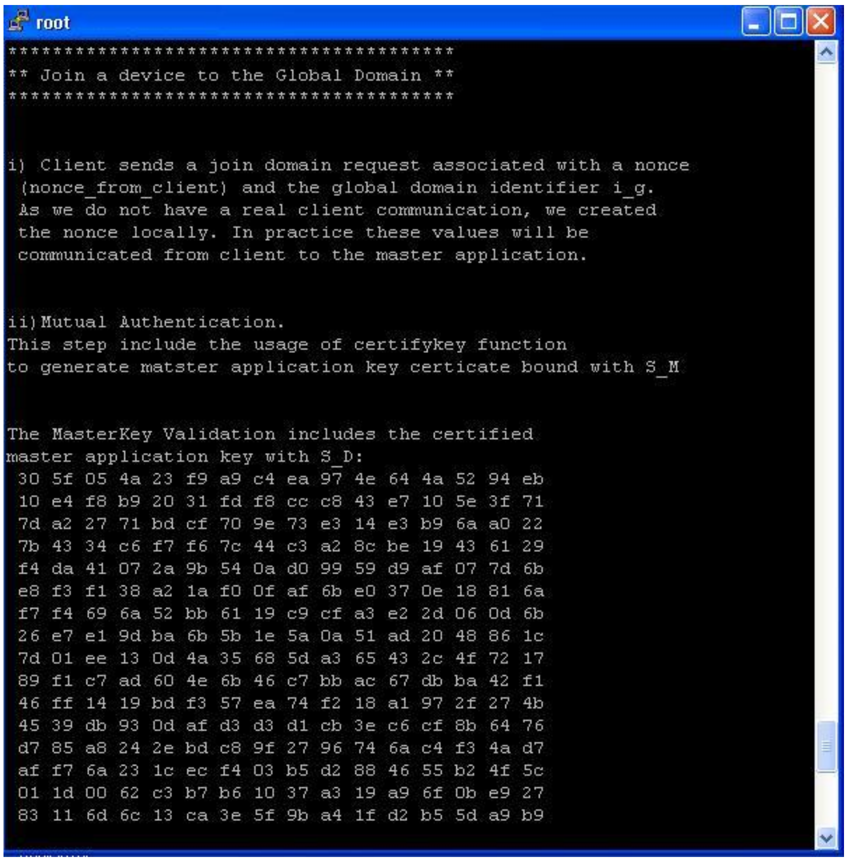
The device D securely generates k_C, and then backs up k_C at the master controller. This is done as follows.

1. $D \rightarrow \text{TPM}_D$: $\text{TPM}_{\text{G}etRandom}$.
2. $\text{TPM}_D \rightarrow D$: Generates a random number to be used as a key k_C.
3. D then seals k_C to the client application exactly as it does for k_G. Because k_G and k_C are encrypted using Pu_D it means the keys are stored using the platform protected storage mechanism and sealed to TPM_D, such that they only can be decrypted when the platform PCR values matches the ones associated with Pr_D (i.e. S_D).
4. $D \rightarrow M$: D sends a Backup_Request to the master controller M to keep a copy of C. The request is as follows $e_{Pu_M}(k_C)$.
5. M stores $e_{Pu_M}(k_C)$ in its protected storage.

If Algorithm 9 fails the device key k_C will not be created and sealed, and an appropriate error message will be returned which the administrators would need to address before re-executing the algorithm.

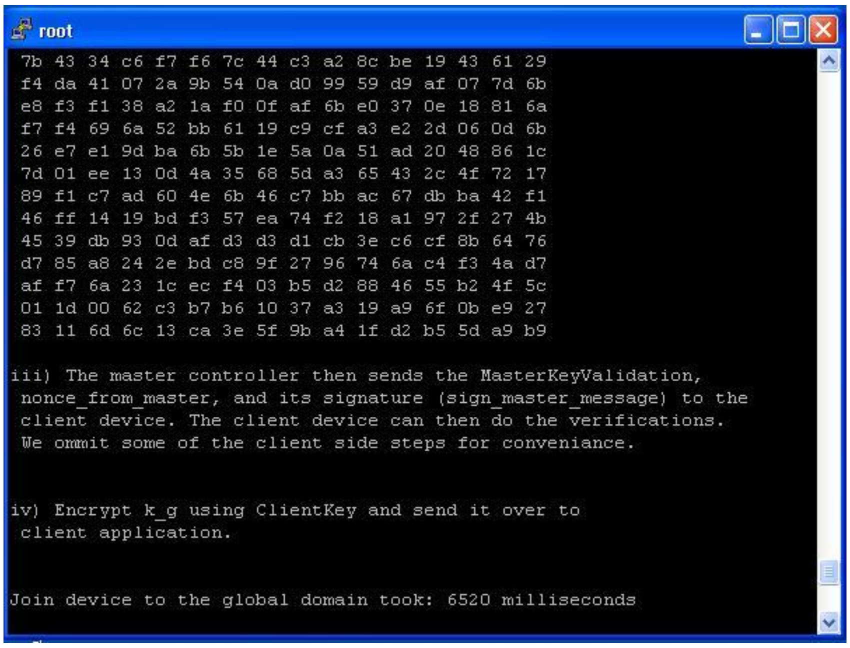
Algorithms 7 and 8 were implemented in Java code which is given in Appendix A. Figures 5.4 and 5.5 illustrate screen-shots of the running algorithms. Executing these algorithms on the environment as described in Appendix A took just over 6.5 seconds. Algorithm 9 was also implemented in Java code which is given in Appendix A. Figures 5.6 shows a screen-shot of the running algorithm. Executing these algorithms on the environment as described in Appendix A took just over 0.8 seconds.

At the successful completion of the above protocol the joining device D and the master controller establish a secure communication channel that is used to transfer the global domain key to the device. This path provides the assurance to the master controller about the device current state, and also forces the use of the key when



```
root
*****************************************
** Join a device to the Global Domain **
*****************************************


i) Client sends a join domain request associated with a nonce
 (nonce_from_client) and the global domain identifier i_g.
 As we do not have a real client communication, we created
 the nonce locally. In practice these values will be
 communicated from client to the master application.


ii)Mutual Authentication.
This step include the usage of certifykey function
to generate matster application key certicate bound with S_M


The MasterKey Validation includes the certified
master application key with S_D:
 30 5f 05 4a 23 f9 a9 c4 ea 97 4e 64 4a 52 94 eb
 10 e4 f8 b9 20 31 fd f8 cc c8 43 e7 10 5e 3f 71
 7d a2 27 71 bd cf 70 9e 73 e3 14 e3 b9 6a a0 22
 7b 43 34 c6 f7 f6 7c 44 c3 a2 8c be 19 43 61 29
 f4 da 41 07 2a 9b 54 0a d0 99 59 d9 af 07 7d 6b
 e8 f3 f1 38 a2 1a f0 0f af 6b e0 37 0e 18 81 6a
 f7 f4 69 6a 52 bb 61 19 c9 cf a3 e2 2d 06 0d 6b
 26 e7 e1 9d ba 6b 5b 1e 5a 0a 51 ad 20 48 86 1c
 7d 01 ee 13 0d 4a 35 68 5d a3 65 43 2c 4f 72 17
 89 f1 c7 ad 60 4e 6b 46 c7 bb ac 67 db ba 42 f1
 46 ff 14 19 bd f3 57 ea 74 f2 18 a1 97 2f 27 4b
 45 39 db 93 0d af d3 d3 d1 cb 3e c6 cf 8b 64 76
 d7 85 a8 24 2e bd c8 9f 27 96 74 6a c4 f3 4a d7
 af f7 6a 23 1c ec f4 03 b5 d2 88 46 55 b2 4f 5c
 01 1d 00 62 c3 b7 b6 10 37 a3 19 a9 6f 0b e9 27
 83 11 6d 6c 13 ca 3e 5f 9b a4 1f d2 b5 5d a9 b9
```

Figure 5.4: Join Global Domain (a)



```
root
 7b 43 34 c6 f7 f6 7c 44 c3 a2 8c be 19 43 61 29
 f4 da 41 07 2a 9b 54 0a d0 99 59 d9 af 07 7d 6b
 e8 f3 f1 38 a2 1a f0 0f af 6b e0 37 0e 18 81 6a
 f7 f4 69 6a 52 bb 61 19 c9 cf a3 e2 2d 06 0d 6b
 26 e7 e1 9d ba 6b 5b 1e 5a 0a 51 ad 20 48 86 1c
 7d 01 ee 13 0d 4a 35 68 5d a3 65 43 2c 4f 72 17
 89 f1 c7 ad 60 4e 6b 46 c7 bb ac 67 db ba 42 f1
 46 ff 14 19 bd f3 57 ea 74 f2 18 a1 97 2f 27 4b
 45 39 db 93 0d af d3 d3 d1 cb 3e c6 cf 8b 64 76
 d7 85 a8 24 2e bd c8 9f 27 96 74 6a c4 f3 4a d7
 af f7 6a 23 1c ec f4 03 b5 d2 88 46 55 b2 4f 5c
 01 1d 00 62 c3 b7 b6 10 37 a3 19 a9 6f 0b e9 27
 83 11 6d 6c 13 ca 3e 5f 9b a4 1f d2 b5 5d a9 b9

iii) The master controller then sends the MasterKeyValidation,
 nonce_from_master, and its signature (sign_master_message) to the
 client device. The client device can then do the verifications.
 We ommit some of the client side steps for conveniance.


iv) Encrypt k_g using ClientKey and send it over to
 client application.


Join device to the global domain took: 6520 milliseconds
```

Figure 5.5: Join Global Domain (b)

the device is on a specific state. The device D is now part of the global domain, as it possesses a copy of the keys k_G and k_C and its public key matches the one stored in the master controller.

All devices in the organization need to perform the previous steps to become members of the global domain. Member devices of the global domain can access the global-domain associated pools of content, and hence such pools of content are now shared by all organization devices.

5.2.3 Shrinking Global Domain

There are two cases in which the device may need to be removed from the organization, which are as follows. (1) The device has a problem that might affect the security of the keys stored in the device. In this case, the device needs to be removed and the keys stored on the device need to be revoked. The key revocation procedure and the steps which must be followed are discussed in Section 6.3.4. (2) The device has no problem and it is still trusted to securely remove the keys from its storage. This case applies, for example, when the global domain needs to be shrunk, or a new device is replacing an existing device. In such cases, the organization should be given the flexibility to update its global domain. The way to remove a device from the global domain is as follows.

1. The organization's security administrators instruct the master controller to remove a device from the domain.

2. M authenticates security administrators as described in Algorithm 3 and instructs them to provide the device public key.

3. M and D mutually authenticate each other's trustworthiness, exactly as described in Algorithm 7 to validate the device is trusted to remove the key Pr_D from its storage.

4. The master controller instructs the leaving device to delete Pr_D from its storage as described in Algorithm 10. The objective of this algorithm is to remove a device from the global domain.

5. The master controller then removes this device's public key from the public key list of the global domain.

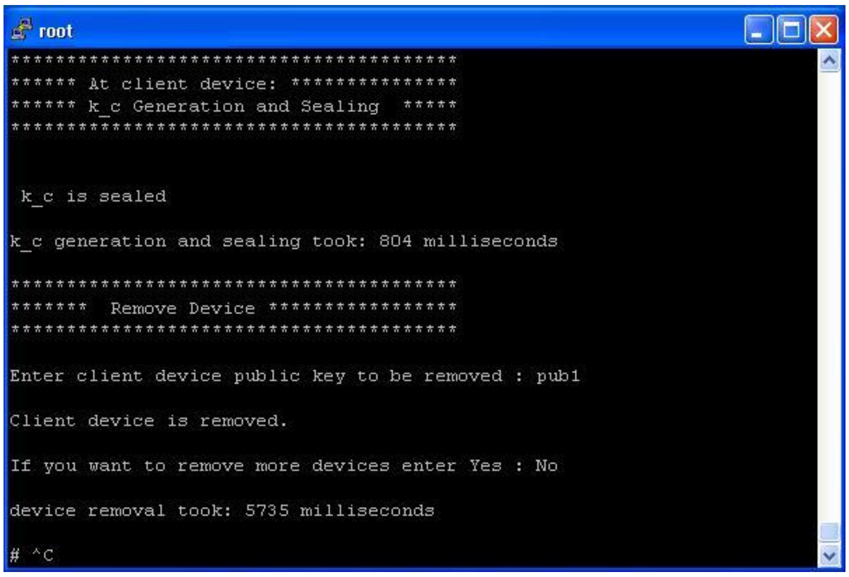
Algorithm 10 Removing a device from a global domain (steps need to be applied in order)

```
1. $M \rightarrow \text{TPM}_M$: $\text{TPM}_{\text{L}oadKey2}(\text{Pr}_M)$.
   The TPM on $M$ loads the private key $\text{Pr}_M$ in the TPM trusted environment,
   after verifying the current PCR value matches the one associated with $\text{Pr}_M$
   (i.e. $S_M$). If the PCR value does not match $S_M$, the master application returns
   an appropriate error message.
2. $M \rightarrow \text{TPM}_M$: $\text{TPM}_{\text{U}nseal}(k_G||i_G||\text{PKL}_g)$.
3. $\text{TPM}_M \rightarrow M$: decrypts the string $k_G||i_G||\text{PKL}_g$ and passes the result to $M$.
4. $M$ instructs $D$ to delete $\text{Pr}_D$.
5. $M \rightarrow D$: Leave_Global_Domain$||A_D||$Sign$_M$(Leave_Global_Domain$||A_D$).
6. $D$: verifies message signature and that it is the intended recipient by checking
   the value of $A_D$.
   If verifications succeed, $D$ removes the private key ($\text{Pr}_D$), which disables the
   device from accessing the global domain and all other dynamic domain credentials.
   $D \rightarrow M$ Remove_Succeeds.
```

6. Once M receives the Remove_Succeeds message it removes Pu_D from the PKL_g.

On the other hand, if the execution status of the device is not trusted or the device is not available (as in the case of device hardware failure), each organization should define their policies on how to handle such cases. For example, an organization might have a revocation list for this purpose, where non-responsive removed devices are added to this list. Other devices before interacting amongst each other should regularly check the revocation list when dealing with other devices' membership of a domain. This will exclude such devices from receiving new content. The key revocation procedure is explained in detail in Section 6.3.4.

Algorithm 10 was implemented in Java code which is given in Appendix A. Figures 5.6 shows a screen-shot of the running algorithm. Executing this algorithms on the environment as described in Appendix A took just over 5.7 seconds.



```
*****************************************
****** At client device: ***************
****** k_c Generation and Sealing  *****
*****************************************


 k_c is sealed

k_c generation and sealing took: 804 milliseconds

*****************************************
*******  Remove Device *****************
*****************************************

Enter client device public key to be removed : pub1

Client device is removed.

If you want to remove more devices enter Yes : No

device removal took: 5735 milliseconds

# ^C
```

Figure 5.6: K_C Initialization and Removing Devices

5.3 How Global Domain Protects Content

As we described earlier, encrypting content alone does not necessary mean it is protected. Robust content protection requires ensuring that the environment where content is accessed, stored and transferred is secure and protected. In this section, we cover the roles of the main entities in the proposed system which help in achieving content protection. The main point in this scheme is ensuring that content can be accessed using only authorized devices; this is achieved as follows.

1. **Creating a global domain.** In the proposed solution we create a global domain consisting of all devices in an organization. This domain has a domain-specific key k_G and a content protection key k_C. k_G is shared by all devices in the organization and is used to encrypt content, whilst in transit, which is required to be shared between all organizational devices. k_C is a device-specific key that is used to encrypt content whilst stored on a device. Only devices that possess the key k_G can decrypt the protected content after receiving the content from other devices. A device must join the organization global domain to receive the key k_G. Only an **authorized device** can join the global domain.

2. **Authorized Devices.** A device is considered an authorized device after satisfying the following conditions.

   (a) The device public key is listed in the public key list for the global domain.

   (b) It has been verified that the device is trusted, as described in Section 5.2.2 (i.e. the client application running on the device performs as expected, e.g. does not reveal k_G/k_C in the clear, does not transfer k_G/k_C to others, and does not transfer confidential content unprotected to others).

   After a device satisfies the above conditions of being an authorized device it can join the organization domain, and receive a copy of **the shared domain key** k_G.

3. **The keys k_G and k_C.** The successful operation of this scheme requires controlling, managing and securely protecting the keys k_G and k_C. This is ensured as follows.

   (a) The key k_G can only be transferred from the trusted master controller to an authorized device, and it cannot be copied between devices.

   (b) After an authorized device receives the key k_G, it securely generates k_C and stores both k_G and k_C in its protected storage sealed to a specific trusted environment configuration state, so that only the client application can access these keys. This is to ensure that these keys can only be released to the client application if the running execution status of the machine matches the one associated with the stored key.

   When a device wants to share content so that it is accessible to all organization devices it must encrypt content using the key k_G. **Protected content** (encrypted using k_G) can be freely moved to other devices; however, only devices that possess a copy of the key k_G, i.e. devices member in the same organization, can decrypt and then access the content.

4. **Protected content.** Content requiring protection must be stored protected inside devices using key k_C and must not leave organization devices unprotected. This is achieved by the client application, which is designed to store content that requires protection encrypted using k_C. When a device wants to transfer content to another device, the client application decrypts the content and re-encrypts it using the shared key k_G, as discussed in Chapter 4. Thus devices which receive such protected content cannot access such protected content without having the key k_G. This in turn means that these devices have joined the domain and have been verified as authorized device.

5.4 Scenarios

In this section we provide different scenarios for information leakage and how they are addressed by the proposed scheme. It is very important to make it clear that in our proposed scheme we do not address information leakage that results from the following cases: an authorized user renders content on an authorized device with the physical presence of another unauthorized user; and an authorized user memorizes/writes/records content and then transfers it to others.

Let us assume that an organization has a domain, consisting of all devices that require to access confidential content i.e. a global domain. The global domain membership is controlled by a domain-specific master-controller. The organization has a large confidential content C, which is protected against leakage using the global domain, as described in this chapter. Andrew who works for the same organization is authorized to access C on a device provided by his organization at work. Simon who works for another competitive organization and knows Andrew well, needs to get a copy of C. Andrew transfers C (encrypted using k_G) via the Internet or copies it (encrypted using k_C) into a CD-ROM, and then gives it to Simon, e.g. using the Internet. Simon could not access C because his device is not a member of Andrew's organization domain, and so he does not have a copy of the domain key. Andrew tried to transfer the key via the Internet or copy it into a CD-ROM then give it to Simon, but the client application running on Andrew's device does not allow the domain key to be transferred or coped. So the only way for Simon to have the key is to join the global domain to have a copy of the key k_G.

Andrew brings Simon's device to his work and attempts to add it into the organization domain. The master controller refused that, as Simon's device's public key is not in the domain public key list. After this, Andrew thought why not add Simon's device's public key to the domain public key list? This idea did not work because for any changes in the domain credentials, the master controller requires the security administrators' credentials and Andrew does not know the security administrators' authentication credentials. Andrew's final resort is to print the content and hand it over to Simon. The client application running on Andrew's device is configured not to allow printing for content C. Eventually Simon has failed to access C.

Sue is a maintenance engineer from an external company. Whilst she was fixing a problem in Andrew's device she directly copied a protected version of C (via secret and illegal means) from Andrew's hard-drive into her USB memory stick. Later on, Sue executed the client application on her device to access C; however, the client application was not capable of accessing C, as Sue's device does not have the device-specific content protection key k_C, which is used to encrypt C at Andrew's device.

Faye is a manager that needs to work from home and access the organization's content. The organization can provide Faye a laptop that is a member in the organization global domain. Alternatively, Faye can bring her private laptop to the organization premises in a legitimate way (i.e. with the knowledge of the security officers and security administrators), and then security administrators can add her laptop to the organization domain. Thereby, Faye's laptop gets a copy of the organization's content protection key k_G, and hence can access content from her laptop at home.

5.5 Summary

This chapter proposes a solution for the protection of confidential digital content from being leaked to unauthorized parties. The basis of the solution is grouping an organization's devices into a specific global domain. Domain membership is controlled by a master controller, which only allows authorized devices to join this domain after authenticating security administrators and verifying devices are trusted. Content can be accessed only by authorized employees using authorized devices that are members of the domain, and if an unethical employee attempts leaking such content to third parties, the content will not be accessible. This is because a device which does not belong to the organization does not have a copy of the content decryption key (the global domain key k_G). In other words, the global domain provides the environment to transfer confidential content to devices that are members of a specific organization and, simultaneously, ensures content is protected against being illegally accessed by other devices, i.e. the usage of the global domain covers requirement number 2 (Content sharing) and 3 (Content protection).

Chapter 6

Dynamic Domains

Contents

In this chapter we describe a scheme framework of the dynamic domains concept introduced in Chapter 4. This chapter proposes a mechanism for internal employees to share and simultaneously guard information assets from one another. This scheme involves managing dynamic domains using an organization-specific master controller. The master controller itself is controlled by the organization's security administrators. We then discuss how the proposed scheme prevents internal leakage whilst allowing authorized employees to access confidential content from inside or outside organizational premises.

6.1 Introduction

In this chapter we propose the dynamic domain scheme introduced in Chapter 4. The dynamic domain helps in mitigating the internal leakage, as defined in Definition 2.2.4. When an authorized employee sends content to others they will not be capable of accessing content except on devices that are members of the predefined group; thus internal leakage is prevented when using such a technique.

As outlined in Chapter 4, a dynamic domain consists of devices that must be members of the global domain. Each dynamic domain has a unique identifier, i_D, a unique shared symmetric key, k_D, and, for each device, a unique domain symmetric key, k_C. k_D is used to protect the dynamic domain-specific pool of content whilst in transit; it can only be accessed by the dynamic domain-specific set of devices. This key is only available inside the domain, thus only domain devices can access the pool of content bound to the domain. k_C is used to protect the dynamic domain-specific pool of content whilst stored in a device and can only be accessed by that specific device. The dynamic domain creation process is performed by the organization's authorized security administrators, who choose devices that need to be bound to a dynamic domain based on the organization's requirements.

Definition 6.1.1. A dynamic domain identifier i_D is a unique random number that is securely generated and stored by the TPM of the master controller. i_D is used to identify the dynamic domain whenever a device needs to join the dynamic domain or encrypt content with the dynamic domain key. This is described in Section 6.2.

Definition 6.1.2. A dynamic domain key k_D is a symmetric key that is securely generated and stored by the TPM of the master controller as described in Section 6.2. k_D is not available in the clear even to the domain security administrators and it can not be copied between devices. The key k_D is only transferred from the master controller to a device when a device needs to join the dynamic domain. The key k_D is used to protect the dynamic domain content *whilst in transit.*

Definition 6.1.3. A dynamic domain public key list (PKL$_d$) is a list that is composed of the public keys of the group of devices that perform the dynamic domain. The PKL$_d$ is a subset from the global domain PKL$_g$ defined in Section 5.2.1. The PKL$_d$ is securely stored in the master controller and is managed by security administrators using the master application.

PKL$_g$ management follows the same process as described in Section 5.2.

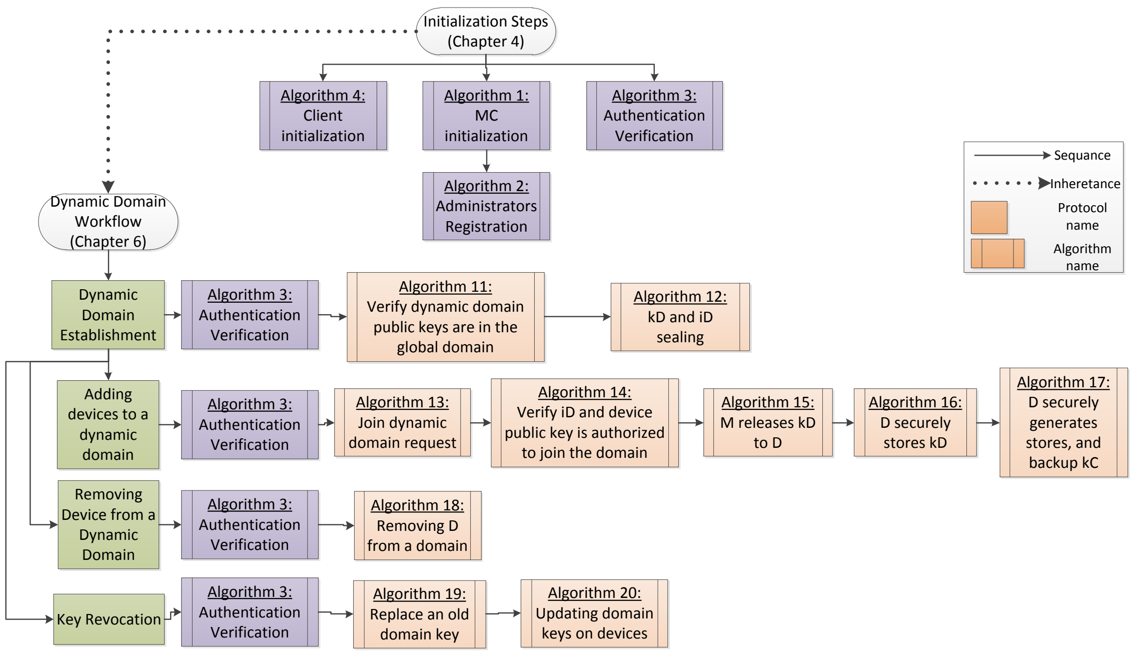

Initialization Steps (Chapter 4)
Algorithm 4: Client initialization
Algorithm 1: MC initialization
Algorithm 3: Authentication Verification
Algorithm 2: Administrators Registration
Dynamic Domain Workflow (Chapter 6)
Dynamic Domain Establishment
Algorithm 3: Authentication Verification
Algorithm 11: Verify dynamic domain public keys are in the global domain
Algorithm 12: kD and iD sealing
Adding devices to a dynamic domain
Algorithm 13: Join dynamic domain request
Algorithm 14: Verify iD and device public key is authorized to join the domain
Algorithm 15: M releases kD to D
Algorithm 16: D securely stores kD
Algorithm 17: D securely generates stores, and backup kC
Removing Device from a Dynamic Domain
Algorithm 18: Removing D from a domain
Key Revocation
Algorithm 19: Replace an old domain key
Algorithm 20: Updating domain keys on devices
Sequance
Inheretance
Protocol name
Algorithm name

Figure 6.1: Dynamic Domain Algorithms Sequence

6.2 System Workflow

Figure 6.1 illustrates the sequence of algorithms which are required in this chapter. The figure illustrates four main domain workflow protocols, i.e. dynamic domain establishment, adding devices to a dynamic domain, removing devices from a dynamic domain, and key revocation. For each protocol the figure illustrates the algorithms associated with the protocol and their order of execution. For example, the dynamic domain establishment protocol involves executing Algorithms 3, 11, and then 12 in order. The arrows in the figure indicate the sequence/dependencies of events. A failure in any algorithm would require a rollback of all dependent algorithms. In such a case the system administrators would need to check the reasons of failures before proceeding and re-running the protocol. The following sections discuss the workflow algorithms.

6.2.1 Dynamic Domain Establishment

In this section we require that the master application running on the master controller has been initialized and the global domain has been established, as discussed in Chapters 4 and 5, respectively (Figure 6.1 illustrates the sequence of algorithms required in this chapter). As we discussed in Chapter 4 the master application includes the dynamic domain additional functions. The additional functions are mainly related to validating that a dynamic domain is a subset of the global domain. Whenever an organization wishes to share a pool of content with a group of employees in such way that the content can only be accessed by devices member in the group, it needs to create a dynamic domain consisting of devices used by the employees. The process of creating a dynamic domain is as follows.

The organization decides how many devices need to access a specific type of content. The organization also decides which devices will access this type of content. This should be based on organizational needs. For example, a dynamic domain could consist of devices used by managers' levels, devices used by accounts department, etc. For a device that is chosen to be in a dynamic domain, its public key should be included in the dynamic domain public key list.

The security administrators instruct the master controller to create a new dynamic domain. The master controller then authenticates the organization security administrators, e.g. using a password, as described in Algorithm 3.

If authentication succeeds, the master controller instructs the security administrators to provide the public keys of devices that will be in the dynamic domain PKL_d.

The master controller then verifies that the provided PKL_d are included in the global domain public key list PKL_g, which is defined in Chapter 5. This is to ensure that the joining devices are owned by the organization. The protocol for this is as follows.

The following notation is used in the provided protocol.

- M is the master application.
- TPM_M is the TPM on the master controller.
- S_M is the platform state at release as stored in the PCR inside TPM_M.
- $(\mathrm{Pu}_M, \mathrm{Pr}_M)$ is a non-migratable key pair such that the private part of the key Pr_M is bound to TPM_M, and to the platform state S_M.
- PKL_d is the dynamic domain public key list.
- i_D is the dynamic domain identifier.
- k_D is the dynamic domain symmetric key.

Algorithm 11 Verify dynamic domains' public keys are in global domain (steps need to be applied in order)

1. $M \to \text{TPM}_M$: $\text{TPM}_{\text{U}nseal}(k_G||i_G||\text{PKL}_g)$.
2. $\text{TPM}_M \to M$: decrypts the string $k_G||i_G||\text{PKL}_g$ and passes the result to M.
3. M verifies that PKL_d entries are included in PKL_g.

If the verification succeeds, the master controller securely generates a shared dynamic domain specific symmetric key k_D and a dynamic domain specific identifier i_D. k_D and i_D are associated with the public key list PKL_d and then stored in the master controller protected storage and sealed to a trusted execution environment. The algorithm for these steps is as follows.

Algorithm 12 k_D and i_D Sealing (steps need to be applied in order)

1. $M \to \text{TPM}_M$: $\text{TPM}_{\text{G}etRandom}$;
2. $\text{TPM}_M \to M$: generates a random number to be used as a shared dynamic domain key k_D and sends it back to M.
3. $M \to \text{TPM}_M$: $\text{TPM}_{\text{G}etRandom}$;

   The TPM on M (TPM_M) generates a random number to be used as a dynamic domain identifier i_D
4. $\text{TPM}_M \to M$: i_D
5. $M \to \text{TPM}_M$: $\text{TPM}_{\text{L}oadKey2}(\text{Pr}_M)$;

   M loads the private key Pr_M into the TPM trusted environment, after verifying the current PCR value matches the one associated with Pr_M (i.e. S_M). If the PCR value does not match S_M, M returns an appropriate error message.
6. $M \to \text{TPM}_M$: $\text{TPM}_{\text{S}eal}(k_D||i_D||\text{PKL}_d)$.

   TPM_M securely stores the string $k_D||i_D||\text{PKL}_d$ using the platform protected storage mechanism bound to TPM_M, such that they can only be decrypted on the current platform by M, and only if the platform runs as expected (when the platform PCR values matches the ones associated with Pr_M, i.e. S_M).

Proof of concept code was not produced for the dynamic domain protocols. This is because it is similar (with some minor changes) to those provided in the global domain.

6.2.2 Adding Devices to a Dynamic Domain

This section describes the process for adding a device into a dynamic domain.

The following notation is used in the provided protocol.

- D is the client application.
- M is the master application.
- TPM_D is the TPM on a client device.
- TPM_M is the TPM on the master controller.
- S_D is the platform state at release as stored in the PCR inside the TPM_D.
- S_M is the platform state at release as stored in the PCR inside the TPM_M.
- (Pu_D, Pr_D)is a non-migratable key pairs such that the private part of the key Pr_D is bound to TPM_D and to the platform state S_D.
- (Pu_M, Pr_M) is a non-migratable key pairs such that the private part of the key Pr_M is bound to TPM_M and to the platform state S_M.
- Cert_M is the master controller AIK certificate.
- Cert_D is the joining device AIK certificate.
- A_M is an identifier for the master controller included in Cert_M.
- A_D is an identifier for a device included in Cert_D.
- Pr_{MAIK} is the private key of the master controller AIK.
- Pr_{DAIK} is the private key of the device AIK.
- N_1 is a randomly generated nonce.
- N_2 is a randomly generated nonce.
- $e_{Pu_D}(Y)$ denotes the asymmetric encryption of data Y using key Pu_D, and where we assume that the encryption primitive in use provides non-malleability, as described in [44].
- SHA1 is a one way hash function.

To add a device to a dynamic domain the client application sends a join domain request to the master application to install the dynamic domain specific key. This request includes the dynamic domain specific identifier i_D, which is achieved as follows.

Algorithm 13 join dynamic domain request (steps need to be applied in order)

1. $D \rightarrow \text{TPM}_D$: $\text{TPM}_{\text{L}oadKey2}(\text{Pr}_D)$.

   TPM on D loads the private key Pr_D in the TPM trusted environment, after verifying the current PCR value matches the one associated with Pr_D (i.e. S_D). If the PCR value does not match S_D, the client application exits with an appropriate error message.

2. $D \rightarrow \text{TPM}_D$: $\text{TPM}_{\text{G}etRandom}$.

   TPM_D generates a random number N_1 that is used as a nonce, and returns the result back to D.

3. $D \rightarrow M$: Join_Dynamic_Domain $(N_1||\text{Cert}_D||A_M||i_D)||\text{Sign}_D(N_1||A_M||i_D)$.

4. M verifies Cert_D, extracts the signature verification key of D from the certificate, and checks that it has not been revoked, e.g. by querying an OCSP service. M then verifies message signature, and verifies it is the intended recipient by checking the value of A_M.

The master controller and the joining device mutually authenticate each other and attests to each other's trustworthiness in the same way as discussed in Section 5.2.2.

Upon successful authentication, the master controller checks if i_D represents a valid domain and if the device's public key is included in the dynamic domain PKL_d, as follows:

Algorithm 14 Verifies i_D and device's public key is authorized to join the domain (steps need to be applied in order)

1. $M \rightarrow \text{TPM}_M$: $\text{TPM}_{\text{U}nseal}(k_D||i_D||\text{PKL}_d)$.

2. $\text{TPM}_M \rightarrow M$: decrypts the string $k_D||i_D||\text{PKL}_d$ and passes the result to M.

3. M verifies i_D matches the recovered dynamic domain identifier and Pu_D is included in the PKL_d.

If the result is positive, the master controller releases the dynamic domain specific key k_D to the device encrypted using the device's public key.

Algorithm 15 M releases k_D to D (steps need to be applied in order)

```
$M \rightarrow D$: $e_{Pu_D}(k_D)$;

$M$ encrypts $k_D$ using the key $\mathrm{Pu}_D$ and sends it to $D$.
```

The device stores the dynamic domain key in its protected storage.

Algorithm 16 D securely stored k_D (steps need to be applied in order)

```
1. $D$ stores the string $e_{Pu_D}(k_D)$, such that it only can be decrypted when the
   platform PCR values matches the ones associated with $\mathrm{Pr}_D$ (i.e. $S_D$).
```

The device securely generates a symmetric key k_C, which is used to encrypted the dynamic domain content whilst in storage. Then the device stores k_C in its protected storage. Finally, the device stores a copy of k_C, as a backup, at the master controller.

Algorithm 17 D securely generates, stores, and backup k_C (steps need to be applied in order)

```
1. $D \rightarrow \mathrm{TPM}_D$: $\mathrm{TPM}_{\mathrm{G}etRandom}$;

2. $\mathrm{TPM}_D \rightarrow D$: generates a random number to be used as a symmetric key $k_C$
   for encrypting the domain content in the device. $D$ encrypts $k_C$ using the key
   $\mathrm{Pu}_D$ as follows $e_{Pu_D}(k_C)$. $D$ stores the string $e_{Pu_D}(k_C)$ using the platform
   protected storage mechanism and bound to $\mathrm{TPM}_D$. This is done as follows.

3. $D \rightarrow \mathrm{TPM}_D$: $\mathrm{TPM}_{\mathrm{S}eal}(e_{Pu_D}(k_C))$.

4. $D \rightarrow M$: $D$ sends a Backup_Request to the master controller $M$ to keep a
   copy of $C$. The request is $e_{Pu_M}(k_C)$.

5. $M$ stores $e_{Pu_M}(k_C)$ in its protected storage.
```

This device is now part of the dynamic domain, as it possesses a copy of the dynamic domain key and its public key is included in PKL_d as stored in the master controller.

All devices in the PKL_d should follow the above steps to join the dynamic domain. All member devices of the domain can access the encrypted pool of content associated with that domain. All these devices have a copy of the shared dynamic domain-specific key k_D. Therefore, these devices can access the domain-specific content as protected using the key k_D.

As we stated earlier proof of concept code was not produced for the dynamic domain protocols. This is because it is similar (with some minor changes) to those provided in the global domain.

6.2.3 Removing a Devices from a Dynamic Domain

The organization's security administrators instruct the master controller M to remove a device from the dynamic domain, as identified by domain identifier i_D. M authenticates security administrators as described in Algorithm 3. Then M and D mutually authenticate each other's trustworthiness, exactly as described in Section 5.2.2. Upon successful authentication, the following algorithm is executed.

Algorithm 18 Removing D from a domain (steps need to be applied in order)

1. $M \rightarrow \text{TPM}_M$: $\text{TPM}_{\text{L}oadKey2}(\text{Pr}_M)$. TPM on M loads the private key Pr_M in the TPM trusted environment, after verifying the current PCR value matches the one associated with Pr_M (i.e. S_M). If the PCR value does not match S_M, the master application returns an appropriate error message.

2. $M \rightarrow \text{TPM}_M$: $\text{TPM}_{\text{U}nseal}(k_D||i_D||\text{PKL}_d)$.

3. $\text{TPM}_M \rightarrow M$: decrypts the string $k_D||i_D||\text{PKL}_d$ and passes the result to M.

4. M instructs D to remove the dynamic domain key from itself.

5. $M \rightarrow D$: $\text{Leave\_Dynamic\_Domain}||i_D||A_D||\text{Sign}_M(\text{Leave\_Dynamic\_Domain}||i_D||A_D)$.

6. D verifies message signature and it is the intended recipient by checking the value of A_D. If verifications succeed, D removes the dynamic domain credentials as identified by id i_D from its protected storage. D then sends Remove_Succeed message to M.

7. Once M receives Remove_Success message it removes Pu_D from the PKL_d.

6.3 Domain Management

We believe that in order for a solution to be accepted and be widely used, it should adapt with organizations' dynamic nature; for example, an organization might need to change its strategy, layout, business workflow, and/or replace its own devices. This requires that the solution can adapt to employee and device changes which we discuss in this section, i.e. replacing, adding and removing employees' devices from/to a dynamic domain.

6.3.1 Domain Expansion

An organization can expand a dynamic domain, for example, when adding more employees to perform a new business requirement or to help existing employees if business expands. In this case, the master controller instructs security administrators to provide the public keys of the new devices. The master controller securely updates the public key list, and finally it allows the new devices to join the domain as described in Section 6.2.2.

6.3.2 Domain Shrinking

An organization might need to remove certain devices from a dynamic domain for several reasons, such as changes in business requirements (e.g. the employees using these devices are no longer working on a project constituting the domain or the project work is completed and the organization does not want those employees to keep accessing the project content). In such a case the organization should be given the flexibility to remove domain devices as discussed in Section 6.2.3.

6.3.3 Device Changes

An organization might need to change certain devices from a dynamic domain for several reasons, such as changes in business requirements, e.g. an organization wants to replace its devices with the ones of the latest technology. In such a case the organization should be given the flexibility to update domain devices. This could be done by removing unwanted devices and adding replacement devices as discussed in Sections 6.2.3 and 6.2.2 respectively.

6.3.4 Key Revocation

In this section we describe the main steps that security administrators could follow to revoke a domain key. Our solution protects domain credentials using TPM functions. TPM is tamper evident and so it is not easy for the protected keys to get hacked in normal circumstances. Some organizations might decide to be extra cautious and revoke a domain key for every suspicious event. For this, it is the organization policy that decides when to revoke a domain key. Hacking a dynamic domain specific key only affects the dynamic domain-specific pool of content. Security administrators could revoke the dynamic domain key and generate a new domain key, which can be done as follows.

1. The security administrators instruct the master controller to change the key for a specific dynamic domain.

2. The master controller then authenticates the organization's security administrators.

3. If authentication succeeds, the master controller generates a new domain-specific key, and then replaces the old copy of the domain key with the new domain key in its protected storage. The protocol for this is as follows.

Algorithm 19 Replace an old domain key (steps need to be applied in order)

(a) $M \rightarrow \text{TPM}_M$: $\text{TPM}_{\text{G}etRandom}$;

(b) $\text{TPM}_M \rightarrow M$: generates a random number to be used as a replacement dynamic domain key k_{D2} and sends it back to M.

(c) $M \rightarrow \text{TPM}_M$: $\text{TPM}_{\text{L}oadKey2}(\text{Pr}_M)$; loads the private key Pr_M in the TPM trusted environment, after verifying the current PCR value matches the one associated with Pr_M (i.e. S_M). If the PCR value does not match S_M, M returns an appropriate error message.

(d) $M \rightarrow \text{TPM}_M$: $\text{TPM}_{\text{U}nseal}(k_D||i_D||\text{PKL}_d)$.

(e) $\text{TPM}_M \rightarrow M$: decrypts the string $k_D||i_D||\text{PKL}_d$ and passes the result to M.

(f) $M \rightarrow \text{TPM}_M$: $\text{TPM}_{\text{S}eal}(k_{D2}||i_D||\text{PKL}_d)$. TPM_M securely replaces the old key with the new key in the format $k_{D2}||i_D||\text{PKL}_d$ and using the platform protected storage mechanism bound to TPM_M, such that they can only be decrypted on the current platform by M, and only if the platform runs as expected (when the platform PCR values matches the ones associated with Pr_M, i.e. S_M).

4. The master controller reinstalls this key on domain devices; the master controller identifies devices using their public keys, which are securely stored inside the master controller PKL_d for the revoked domain key, as described in the previous step protocol. For each device, the master controller releases the new value of the domain key encrypted using the device's public key. The device, once it receives the key, replaces the domain key with the new value in its protected storage and binds it to the same execution environment used for the old key, as it has already been verified as trusted; see Section 6.2.2. The protocol for this steps is as follows (we do not include the detailed steps of establishing mutual authentication channels, as we discuss it in detail in previous protocol steps).

Algorithm 20 Updating domain key on devices (steps need to be applied in order)

```
(a) The master controller sends a Replace_Domain key request to each device in
    PKL$_d$. The request includes $i_D$ and $K_{D2}$.
(b) The device replaces the stored $k_D$ in its protected storage with $k_{D2}$, as follows.
      i. $D \to$ TPM$_D$: TPM$_{\text{L}oadKey2}$(Pr$_D$). TPM on $D$ loads the private key
         Pr$_D$ in the TPM trusted environment, after verifying the current PCR
         value matches the one associated with Pr$_D$ (i.e. $S_D$). If the PCR value
         does not match $S_D$, the client application exits with an appropriate error
         message.
     ii. $D \to$ TPM$_D$: TPM$_{\text{S}eal}$($e_{Pu_D}(k_{D2})$).
    iii. $D$ seals the string $e_{Pu_D}(k_{D2})$ and bound to TPM$_D$, such that it only can
         be decrypted when the platform PCR values matches the ones associated
         with Pr$_D$ (i.e. $S_D$).
```

Now, if the whole device is hacked, then security administrators could assume that intruders can possibly get access to all keys inside the device. In this case, and based on the organizational policy, security administrators could revoke all domain keys the device is a member of by following the above procedure for each domain key.

6.4 Key Refreshment

In this section we discuss domain key refreshment. The domain keys could be associated with a timer or a counter specifying when the key must get re-validated. For example, an organization might regularly update k_D/k_G, respectively. This can be done using either a push or a pull mechanism. A pull mechanism means each device must regularly check with the master controller for key updates. A push mechanism, on the other hand, means the master controller, "when change any of the keys", pushes the new value to all devices member in the domain.

This procedure is useful in many cases, for example, if a device leaves an organization without communicating with its master controller, as in the case of hacked devices. In this case, the device can only use the content for a short period (controlled using an associated counter or a time-stamp). In Section 6.3.4 we provide the protocol for updating a domain key using a push mechanism.

6.5 Binding Content to a Domain

As we have described in this thesis, there are different kinds of organizations, with each having its own requirements and process workflow. Such requirements and process workflow determine who would create content, and how content should be bound to a domain. Usually departments in organizations create their own content by a group of employees in the organization. These employees might be in one department or split across defferent departments. For simplicity we consider a single case, which could be easily altered to adapt to work for other kinds of organizations.

We now assume that an organization has defined a group of devices that need to be in a domain to share a specific pool of content. Security administrators instruct the master controller to create a dynamic domain for this group, as described in Sections 6.2.1 and 6.2.2. Once a domain is established and devices have been assigned to the domain, employees using these devices can add content to the domain by using a content-specific application that interacts with the client application to add content to the domain. The client application is used to encrypt content and bind it to a specific dynamic domain. Authorized employees (who are allowed to access the trusted client application) have the ability to create content and assign it to the domain.

We now describe the process for binding content to a domain in the context of a particular scenario. Assume an organization needs to work on a new project. This project requires the sharing of a specific pool of content. Employees working on this project need to share the pool of content in such a way the content is protected against internal leakage. In this case, the organization's security administrators create a dynamic domain identified by an identifier i_D. This dynamic domain consists of all devices that need to share the pool of content specific for this project. Content can be added to the project by either employees who are members of the domain or by authorized employees outside the domain. Authorized employees who are not members of the project domain transfer the created content associated with the domain identifier i_D to the master controller. The master controller identifies the dynamic domain using i_D, and then encrypts the received content with the dynamic domain-specific key. The encrypted content could be either sent to a list of employees or stored in a dynamic domain-specific location.

Employees who are members of the domain can assign content directly to the domain, as their devices possess the domain key. In either way, each domain could be associated with a usage rights policy controlling this process.

Next, each member device in a dynamic domain can download the protected content belonging to this domain, typically from a dynamic domain-specific location or receive it from another device. In this case, only member devices in the same domain, i.e. those that hold a copy of the dynamic domain-specific key k_D, can decrypt and then access the dynamic domain content. As we described earlier, different departments/groups in an organization, sometimes require sharing also but have to protect information. Our solution considers this requirement by allowing devices which need to share content with other departments or other dynamic domains to be able to join multiple dynamic domains. Therefore, a single device could join, for example, three domains and so have three dynamic domain-specific keys enabling it to access these dynamic domains content.

If someone copied such content he/she will not be able to access it except on devices holding the content-specific dynamic domain key, i.e. member devices in the content-specific dynamic domain.

6.6 Summary

In this chapter we propose a solution for protecting content against internal leakage in organizations. The proposed solution uses dynamic domains, consisting of devices owned by an organization. Devices can be dynamically reallocated between dynamic domains based on the organization needs. This protects content against leakage and simultaneously allows content to be shared amongst devices in the same domain. The dynamic domain provides organizations with the flexibility to form groups and projects, transfer confidential content between the devices member of such groups and projects, and ensure a project/group content is protected against being illegally accessed by devices that are not members of the project/group devices. In other words, the usage of the dynamic domain covers requirement numbers 1, 2 and 3.

Part III

Application

Chapter 7

Collaborating Organizations

Contents

In this chapter we extend our solution for preventing insider and internal leakage to cover content leakage within collaborating organizations. This involves proposing a scheme facilitating information workflow management and, simultaneously, protecting information that would need to be shared within organizations for achieving their objectives. We then discuss how the proposed scheme prevents content leakage whilst allowing controlled content flow within collaborating organizations.

7.1 Introduction

Many organizations need to share information with other organizations; for example, in some organizations completing a business process requires accessing another organization's confidential information. An example of this is the passport agency in the UK, which issues British passports to citizens. In order for the passport agency to provide this service, it must validate citizens' private information that is submitted in application forms. This is achieved by collaborating with other organizations such as the Home Office and the Metropolitan Police Service to check that the provided information in the passport application is correct. Also, these organizations check that there is no restriction for issuing a passport for a particular citizen. Another common example is embassies, which are located in different countries around the world. Embassies are required to share information with many government organizations located in the country they represent, enabling embassies to provide the right service to their citizens, and, also, the right service to citizens of the country they serve in.

Users who provide their private information to an organization and agree that the organization will keep their private information assume that their records are protected based on the Data Protection Act [1]. Also, they expect that their records will not be leaked accidentally or intentionally to third parties, and it only can be transferred to other organizations with their prior consent. Organizations have a 'duty of care' to protect their own confidential information and the confidential information they obtain from individuals or from other organizations. Thus, organizations need to find possible ways for preventing the transfer of their confidential information to unauthorized individuals and third parties, while simultaneously allowing sharing of content with other organizations.

7.2 Protect Content Between Collaborating Organizations

When an organization wants to transfer part of its confidential information to another organization, such confidential information needs to be protected at three stages. The first is protecting content whilst being transferred between collaborating organizations. The second is protecting content in the destination organization, in such a way it cannot be leaked outside it. The last, and the most complex to achieve, is restricting confidential information to be accessed by a group of users or a specific department in a destination organization. In the last case, the system should ensure that the confidential information can be accessed only by the intended users who are members of a group/department which is specified by the source organization, and if anyone attempts leaking such information to another department/group, the information will not be accessible. For example, if organization$_1$ transfers confidential information to the accounts department in organization$_2$, then only members of the accounts department should be able to access this information.

For the first case, there are many solutions addressing this point, such as using Virtual Private Networks (VPNs) [20], which typically define organizations' boundaries. In the proposed scheme we use a similar technique to the one used by VPN; however, in the proposed solution we integrate trusted computing concepts into it for ensuring that endpoints of communication are running (see Section 7.3 for details).

After we analyzed the second and the third cases, we found that proposing a solution begins with organizing the internal information system for collaborating organizations so that they have a systematic information flow. If collaborating organizations implement an organized information system and process workflow, then this would ease the communication amongst them and help in ensuring that content is transferred to the right department and the right user. In addition, having an organized internal information system increases the mutual level of trust across collaborative organizations. This, in turn, reduces the likelihood of content leakage. On the other hand, the lack of an organized information system or a systematic information flow increases the likelihood of content leakage. After this is achieved (i.e. organizing the internal information flow), a secure system should be proposed to help in protecting content from being leaked to unauthorized users inside or outside the collaborating organization. In our proposed scheme, we use the global domain and the dynamic domain concept to limit content leakage between collaborating organizations. We use the global domain concept to protect content from getting leaked to unauthorized users outside the collaborating organization, and the dynamic domain content to protect content from getting leaked to unauthorized users inside the collaborating organization (see Section 7.3 for details).

7.3 Process Workflow

This section describes the process workflow for secure information sharing between collaborating organizations and its corresponding protocols.

In this section we require that each collaborating organization has a specific master controller, which is initialized as discussed in Section 4.5.1. We also require that each organization has defined its own global and dynamic domains as discussed in Chapters 5 and 6. As we discussed in Chapter 4 the master application includes the collaborating organizations' additional functions. The additional functions are mainly related to the trusted channel establishment and policy setup.

7.3.1 Trusted Channel Establishment and Policy Setup

In real life, employees working in an organization collaborating with other organizations should at least have a basic understanding of the general structure and the functionality of departments of other organizations collaborating with their organization. This is required to enable users to understand where information should be sent and bound in the corresponding organization. This is achieved when the collaborating organizations' master controllers exchange a customized list of each other's departments/groups. Whenever either organization changes its layout it would need to re-push this list, via its master controller, to the corresponding organization's master controller. This list needs to be accessible by devices which need to send content to a collaborating organization.

When organizations, say organization$_1$ (O_1) and organization$_2$ (O_2), collaborate and wish to exchange confidential content, certain initialization steps need to be done. In this subsection we explain our proposed scheme for initiating a secure communication channel. This is to setup necessary policies and rules, and to establish a trusted channel that is used for protecting the transferred content.

The following notation is used in the provided protocol for initiating a secure communication channel.

- O_1 and O_2 are organization$_1$ and organization$_2$, respectively.
- M_1 and M_2 are the master controllers for O_1 and O_2, respectively.
- TPM$_{M_1}$ and TPM$_{M_2}$ are TPMs on O_1 and O_2 master controllers, respectively.
- S_{M_1} and S_{M_2} are the platform states as stored in the PCR inside TPM$_{M_1}$ and TPM$_{M_2}$, respectively.
- (Pu$_{M_1}$, Pr$_{M_1}$) is a non-migratable key pair such that the private part of the key Pr$_{M_1}$ is bound to TPM$_{M_1}$, and to the platform state S_{M_1}.
- (Pu$_{M_2}$, Pr$_{M_2}$) is a non-migratable key pair such that the private part of the key Pr$_{M_2}$ is bound to TPM$_{M_2}$, and to the platform state S_{M_2}.
- Cert$_{M_1}$ is the master controller AIK certificate for O$_1$.
- Cert$_{M_2}$ is the master controller AIK certificate for O$_2$.
- A_{M_1} is an identifier for O$_1$'s master controller included in Cert$_{M_1}$.
- A_{M_2} is an identifier for O$_2$'s master controller included in Cert$_{M_2}$.
- Pr$_{M1AIK}$ is the private key of O$_1$'s master controller AIK.
- Pr$_{M2AIK}$ is the private key of O$_2$'s master controller AIK.
- SHA1 is a one way hash function.

1. M$_1$ and M$_2$ authenticate their security administrators, exactly as described in Algorithm 3.
2. If authentications succeed, both M_1 and M_2 instruct their security administrators to enter their corresponding collaborating organization's master controller public key. The security administrators of O_1 provide M_1 with the public key

of M_2, and similarly security administrators of O_2 provide M_2 with the public key of M_1. Once the keys are entered, each organization-specific master controller stores the corresponding organization public key in its protected storage. This is to ensure that the keys cannot be tampered with. The assumption here is that the public keys are not manipulated when transferred from O_1 to O_2.

3. The master application running on either organization-specific master controller, say O_1, sends a create trusted channel request to the master controller of O_2. Then both master controllers mutually authenticate each other conforming to the three-pass mutual authentication protocol described in [43]. During this mutual authentication both master controllers attest to the execution environment of each other and validates their trustworthiness. The protocol for this is as follows.

   I. $M_1 \rightarrow \text{TPM}_{M_1}$: $\text{TPM}_{\text{G}etRandom}$.
   TPM_{M_1}: generates a random number to be used as a nonce N_1.

   II. $\text{TPM}_{M_1} \rightarrow M_1$: N_1.

   III. $M_1 \rightarrow M_2$: Establish_Secure_Channel request including N_1.

   IV. $M_2 \rightarrow \text{TPM}_{M_2}$: $\text{TPM}_{\text{G}etRandom}$.
   TPM_{M_2}: generates a random number to be used as a nonce N_2.

   V. $\text{TPM}_{M_2} \rightarrow M_2$: N_2.

   VI. $M_2 \rightarrow \text{TPM}_{M_2}$: $\text{TPM}_{\text{L}oadKey2}(\text{Pr}_{M2AIK})$; loads the private key Pr_{M2AIK} in the TPM trusted environment, after verifying the current PCR value matches the one associated with Pr_{M2AIK}.

   VII. $M_2 \rightarrow \text{TPM}_{M_2}$: $\text{TPM}_{\text{C}ertifyKey}(\text{SHA1}(N_2||N_1||A_{M_1}),\text{Pu}_{M_2})$.

   VIII. $\text{TPM}_{M_2} \rightarrow M_2$: generates a certificate for the key Pu_{M_2} and then sends the following string to M_2: $N_2||N_1||A_{M_1}||\text{Pu}_{M_2}||S_{M_2}||\text{Sign}_{M_2}(N_2||N_1||A_{M_1}||\text{Pu}_{M_2}||S_{M_2})$.

   IX. $M_2 \rightarrow M_1$: $N_2||N_1||A_{M_1}||\text{Pu}_{M_2}||S_{M_2}||\text{Cert}_{M_2}||\text{Sign}_{M_2}(N_2||N_1||A_{M_1}||\text{Pu}_{M_2}||S_{M_2})$.

   X. M_1 verifies Cert_{M_2}, extracts the signature verification key of M_2 from Cert_{M_2}, and checks that it has not been revoked, e.g. by querying an OCSP service. M_1 then verifies message signature, verifies M_2 is trusted by checking the value of S_{M_2}, and Pu_{M_2} matches the one defined by M_1 security administrators. It also verifies the value of N_1 and it is the intended recipient by validating the value of A_{M_1}. If verifications fail, the master application running on M_1 returns an appropriate error message.

XI. $M_1 \rightarrow \text{TPM}_{M_1}$: $\text{TPM}_{\text{L}oadKey2}(\text{Pr}_{M1AIK})$; loads the private key Pr_{M1AIK} in the TPM trusted environment, after verifying the current PCR value matches the one associated with Pr_{M1AIK}.

XII. $M_1 \rightarrow \text{TPM}_{M_1}$: $\text{TPM}_{\text{C}ertifyKey}(\text{SHA1}(N_2||A_{M_2}),\text{Pu}_{M_1})$.

XIII. $\text{TPM}_{M_1} \rightarrow M_1$:

Generates a certificate for the key Pu_{M_1}, and then sends the following string to M_1: $N_2||A_{M_2}||\text{Pu}_{M_1}||S_{M_1}||\text{Sign}_{M_1}(N_2||A_{M_2}||\text{Pu}_{M_1}||S_{M_1})$.

XIV. $M_1 \rightarrow M_2$: $N_2||A_{M_2}||\text{Cert}_{M_1}||\text{Pu}_{M_1}||S_{M_1}||\text{Sign}_{M_1}(N_2||A_{M_2}||\text{Pu}_{M_1}||S_{M_1})$. M_2 verifies Cert_{M_1}, extracts the signature verification key of M_1 from Cert_{M_1}, and checks that it has not been revoked, e.g. by querying an OCSP service. M_2 then verifies the freshness of the message by checking the value of N_2, verifies its intended recipient by checking the value of A_{M_2}, and verifies that Pu_{M_1} matches the one defined by M_2 security administrators. M_2 also verifies M_1 is trusted by checking the value of S_{M_1}. If so, M_2 verifies message signature. If the verifications fail, M_2 returns an appropriate error message.

4. If the mutual authentication succeeds and the execution environment of both master controllers are trusted, then both master controllers set up a secure authenticated channel with each other using standard techniques [62], whereby both master controllers agree on a specific secure key k_O to be used for protecting messages exchanged between each other. This key could be established, for example, based on Diffie-Hellman key exchange protocol [62] or by exchanging a common key encrypted using the corresponding master controller public encryption key.

   Definition 7.3.1. The master controllers trusted channel key k_O is a symmetric key that is securely generated by the TPM of either master controller in the collaborating organizations. Both master controllers require to agree on the key k_O as described in the following section. The key k_O is used to protect messages exchanged between master controllers. k_O is part of the system credentials defined in Section 4.3.6, which means both master controllers store it in their protected storage, and bind it to be used by the master application when the host platform state is trusted exactly as described in Section 4.5.

The protocol for generating the key k_O and store it is as follows.

I. $M_2 \rightarrow$ TPM$_{M_2}$: TPM$_{\text{G}etRandom}$. TPM$_{M_2}$ generates a random number to be used as a secret shared key k_O, and returns it back to M_2.

II. $M_2 \rightarrow$ TPM$_{M_2}$: TPM$_{\text{L}oadKey2}$(Pr$_{M_2}$). Loads the private key Pr$_{M_2}$ in the TPM trusted environment, after verifying the current PCR value matches the one associated with Pr$_{M_2}$ (i.e. S_{M_2}). If the PCR value does not match S_{M_2}, the master application returns an appropriate error message.

III. $M_2 \rightarrow M_1$: $N_1 || e_{Pu_{M_1}}(k_O) ||$Sign$_{M_2}(N1 || e_{Pu_{M_1}}(k_O))$.

IV. M_1 verifies message signature and verifies freshness by checking the value of N_1.

If so, M_1 decrypts the string $e_{Pu_{M_1}}(k_O)$ and securely associates k_O with Policy and stores the result using the platform protected storage mechanism, as follows.

5. Both M$_1$ and M$_2$ instruct their security administrators to enter their organization policy governing their interaction. Examples of policies in O_1's master controller could be: *"an email should be generated and sent to the duty manger whenever content is transferred from a specific department to any department in O_2"; "content sent from the finance department must always be bound to the finance department in O_2"; "the maximum number of devices that can receive content from senior accountants must not exceed 5 devices and must always be bound to senior accountant in O_2"; and "content can be transferred to O_2 between 8:00am-6:00pm"*.

6. Once the policy is defined, each organization-specific master controller stores this policy in its protected storage and seals it to a trusted environment execution state. This is to ensure that the policy cannot be tampered with, and only revealed to the master application when the execution status of the machine matches the one associated with the stored policy, i.e. when the master controller is in a trusted state. The protocol for this is as follows.

I. $M_1 store e_{Pu_{M_1}}(Policy || k_O)$.

II. $M_2 store e_{Pu_{M_2}}(Policy || k_O)$. .

7. By now, a trusted channel is established between the organizations' master controllers and a security policy is defined for each organization in its specific master controller. The organization-to-organization specific key k_O is used for encrypting/decrypting exchanged traffic between two specific organizations. Also, master controllers enforce the defined security policy. Now, all devices that are member of both organizations can exchange messages across each other subject to a predefined security policy, which is enforced by each organization master controller. As discussed in Section 4.5, the master controller TPM ensures that accessing and using k_O, and accessing and enforcing the security policy, are only available to the master application running on the organization-specific master controller only if the master controller is in a particular predefined trusted state.

7.3.2 Exchanging Content within Collaborating Organizations

In this section we describe the process for transferring confidential content between collaborating organizations. For simplicity, in this section we describe the process in a series of three scenarios: the first scenario requires binding content to a group of devices constituting an existing dynamic domain in the destination organization; the second scenario requires binding content to a group of devices that do not form a domain; and the last is for binding content so that it can be accessed by all devices in the destination organization.

We require that a trusted channel is established and the policy is defined for the collaborating organizations, as described in Section 7.3.1. We also require that both global and dynamic domains are established within collaborating organizations, as described in Chapters 5 and 6.

The first scenario, which is illustrated in Figure 7.1, assumes a user Andrew working on device$_1$ in O_1 wants to transfer content to O_2, so that the content can only be accessed by 'account department' (Andrew could obtain the list of O_2 layout from a specific location in his organization). Andrew wants to ensure that the content will not leak to other departments/groups of users in O_2. The process proceeds as follows.

1. Device$_1$ sends a transfer content request to the O_1-specific master controller. This request includes the confidential content associated with meta data enabling the master controller to know where content should be sent and to which domain it should be bound. The meta data, for example, could be the destination organization name/identifier and the destination organization's department/group (in our scenario this corresponds to the group of account

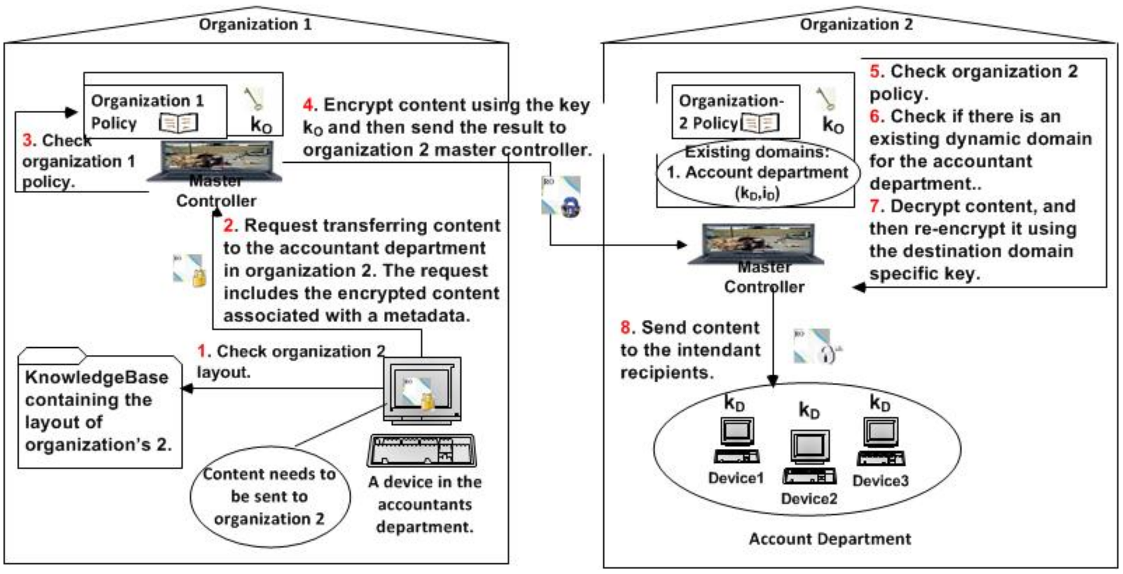

Organization 1
Organization 1 Policy
k_O
3. Check organization 1 policy.
4. Encrypt content using the key k_O and then send the result to organization 2 master controller.
Master Controller
2. Request transferring content to the accountant department in organization 2. The request includes the encrypted content associated with a metadata.
1. Check organization 2 layout.
KnowledgeBase containing the layout of organization's 2.
Content needs to be sent to organization 2
A device in the accountants department.
Organization 2
Organization-2 Policy
k_O
Existing domains:
1. Account department (k_D,i_D)
5. Check organization 2 policy.
6. Check if there is an existing dynamic domain for the accountant department..
7. Decrypt content, and then re-encrypt it using the destination domain specific key.
Master Controller
8. Send content to the intendant recipients.
k_D
k_D
k_D
Device1
Device2
Device3
Account Department

Figure 7.1: Exchanging Content Scenario

department in O_2).

The following is a list of three possible techniques for protecting the information exchanged from device$_1$ to the O_1-specific master controller (these are based on how the content is protected in device$_1$).

(a) The first is encrypting content using the O_1-specific global domain key k_G, assuming that the content is bound to O_1's global domain.

(b) The second case applies if the content is bound to a specific dynamic domain. Content then should be encrypted using the dynamic domain specific key k_D. For example, if Andrew is an accountant in O_1, then his device must be a member in the account domain, and hence possesses a copy of the account domain-specific key. All content bound to the mangers domain can only be transferred encrypted to other devices. Therefore, Andrew can only send content to the master controller encrypted using the account domain-specific key. The master controller, as we explained earlier, possesses a copy of all dynamic domains keys of a specific organization, and hence it can decrypt content.

(c) The last case is encrypting the content using the master controller public key or an agreed symmetric key between the master controller and the device, assuming that content in this case is not bound to any domain.

The protocol to transfer content is as follows.

I. $D_1 \rightarrow \text{TPM}_{D_1}$: $\text{TPM}_{\text{G}etRandom}$;

Generates a random number to be used as a nonce N_1.

II. $D_1 \rightarrow \text{TPM}_{D_1}$: $\text{TPM}_{\text{L}oadKey2}(\text{Pr}_{D1AIK})$;

Loads D_1 AIK in the TPM trusted environment, after verifying the current PCR value matches the one associated with Pr_{D1AIK}.

III. $D_1 \rightarrow \text{TPM}_{D_1}$: $\text{TPM}_{\text{S}ign}(N_1||A_{M_1})$.

IV. $D_1 \rightarrow M_1$: Transfers Content request $(N_1||A_{M_1}||\text{Cert}_{D_1}||\text{Sign}_{D_1}(N_1||A_{M_1}))$.

V. M_1 verifies Cert_{D_1}, extracts the signature verification key of D_1 from the certificate, and checks that it has not been revoked, e.g. by querying an OCSP service. M_1 then verifies the message signature and verifies it is the intended recipient by checking the value of A_{M_1}. If verifications fail M_1 returns an appropriate error message.

VI. $M_1 \rightarrow \text{TPM}_{M_1}$: $\text{TPM}_{\text{G}etRandom}$;

Generates a random number to be used as a nonce N_2.

VII. $M_1 \rightarrow \text{TPM}_{M_1}$: $\text{TPM}_{\text{L}oadKey2}(\text{Pr}_{M1AIK})$;

Loads the master controller AIK in the TPM trusted environment, after verifying the current PCR value matches the one associated with Pr_{M1AIK}.

VIII. $M_1 \rightarrow \text{TPM}_{M_1}$: $\text{TPM}_{\text{S}ign}(N_1||N_2||A_{D_1})$.

IX. $M_1 \rightarrow D_1$: $N_1||N_2||A_{D_1}||\text{Cert}_{M_1}||\text{Sign}_{M_1}(N_1||N_2||A_{D_1})$

X. D_1 verifies Cert_{M_1}, extracts the signature verification key of M_1 from the certificate, and checks that it has not been revoked, e.g. by querying an OCSP service. D_1 then verifies the message signature, verifies message freshness by checking the value of N_1, and verifies it is the intended recipient by checking the value of A_{D_1}. If verification succeeds D_1 sends content C to M_1 associated with sender and receiver details, as follows.

XI. $D_1 \rightarrow \text{TPM}_{D_1}$: $\text{TPM}_{\text{S}ign}(N_2||e_{k_{D_1}}(C)||i_{D_1}||\text{ACCOUNT})$; where ACCOUNT is the domain identifier for at O_2.

XII. $D_1 \rightarrow M_1$: $N_2||e_{k_{D_1}}(C)||i_{D_1}||\text{ACCOUNT}||\text{Sign}_{D_1}(N_2||e_{k_{D_1}}(C)||i_{D_1}||\text{ACCOUNT})$.

XIII. M_1 verifies the message signature and verifies message freshness by checking the value of N_2. If verifications succeed M_1 needs to check if D_1 is authorized to do this action and if any specific policy needs to be followed. This is described in the next step.

2. The O_1 master controller M_1 checks the policy ensuring existing rules do not restrict the transfer of content from device$_1$ to the account group in O_2. For example, a policy could state that device$_1$, which is used by Andrew, is not authorized to transfer content to O_2, or device$_1$ can only transfer content to the account department in O_2. In such cases the transfer process is rejected with an appropriate error message. Also, the policy could state that transactions should be recorded in a specific log file, and/or an email is sent to the duty manager in O_1 informing him about the transfer request. The log information could, for example, state the sender name, device IP address, recipient details, date and time of sending content, etc. The logging information must be protected from tampering, for example, by storing the log-files in a protected storage so that they can only be accessed by the master application after verifying its execution status as expected. Also, users activities associated with accessing content could be monitored to check for abnormal events, as has been extensively discussed by Park et al. [59]. The protocol for this step is as follows.

   I. $M_1 \rightarrow \text{TPM}_{M_1}$: $\text{TPM}_{\text{U}nseal}(\text{Policy}||\text{Pu}_{M_2}||k_O)$.

   TPM_{M_1} verifies M_1 master application is trusted. If so it decrypts the string $\text{Policy}||\text{Pu}_{M_2}||k_O$ and provide it to the master application running on M_1. M_1 enforces the policy rules which are related to this action. If the verification succeed M_1 proceeds to next step.

3. If there is no restriction on transferring the content, M_1 loads the trusted channel specific key k_O. As explained in Section 7.3.1, this key is sealed so that only the master application can access this key when the platform state is as expected. The master application encrypts content using k_O, and then transfers the encrypted content to O_2-specific master controller. The protocol for this step is as follows.

   I. M_1 loads K_{D_1} as follows.

   $M_1 \rightarrow \text{TPM}_{M_1}$: $\text{TPM}_{\text{U}nseal}(k_{D_1}||i_{D_1}||\text{PKL}_{D_1})$.

   M_1 verifies that Pu_{D_1} is a member in dynamic domain i_{D_1} by checking PKL_{D_1}. It then decrypts C using k_{D_1}, and then re-encrypts it using k_O as $e_{k_O}(C)$. M_1 now can start the process for transferring the encrypted content to the destination organization. M_1 verifies Cert_{M_2} that is stored in M_1, extracts the signature verification key of M_2 from the certificate, and checks that it has not been revoked, e.g. by querying an OCSP service.

II. $M_1 \rightarrow \mathrm{TPM}_{M_1}$: $\mathrm{TPM}_{\mathrm{G}etRandom}$;

The TPM on M_1 generates a random number to be used as a nonce N_3 and sends it back to M_1

III. $M_1 \rightarrow M_2$: $N_3||A_{M_2}||\mathrm{Cert}_{M_1}||\mathrm{Sign}_{M_1}(N_3||A_{M_2})$

IV. M_2 verifies Cert_{M_1} matches the one stored in M_2 storage, extracts the signature verification key of M_1 from the certificate, and checks that it has not been revoked. M_2 then verifies message signature and verifies it is the intended recipient by checking the value of A_{M_2}. If verifications succeed M_2 generate a random number as follows.

V. $M_2 \rightarrow \mathrm{TPM}_{M_2}$: $\mathrm{TPM}_{\mathrm{G}etRandom}$;

The TPM on M_2 generates a random number to be used as a nonce N_4 and sends it back to M_1 as follows

VI. $M_2 \rightarrow M_1$: $N_3||N_4||A_{M_1}||\mathrm{Sign}_{M_2}(N_3||N_4||A_{M_1})$

VII. M_1 verifies message signature, message freshness by checking the value of N_3, and verifies it is the intended recipient by checking the value of A_{M_1}. If verifications succeed M_1 sends encrypted C to M_2 associated with sender and receiver details, as follows.

VIII. $M_1 \rightarrow M_2$: $N_4||e_{k_O}(C)||\mathrm{ACCOUNT}||\mathrm{Sign}_{M_1}(N_4||e_{k_O}(C)||\mathrm{ACCOUNT})$.

4. M_2 checks the message signature and verifies message freshness by checking the value of N_4. If verifications succeed, M_2 loads the key k_O. It then decrypts C and verifies that the targeted group (i.e. account department in our scenario), constitutes an existing domain. If so, M_2 enforces the organization policy on the transferred content, exactly as described above for M_1 case. Then the master controller loads the dynamic domain-specific key, and re-encrypts the content using this key. The protocol for this is as follows.

I. M_2 loads K_{D_2} as follows.

$M_2 \rightarrow \mathrm{TPM}_{M_2}$: $\mathrm{TPM}_{\mathrm{U}nseal}(k_{D2}||i_{D2}||\mathrm{PKL}_{D2})$. M_2 encrypts C using k_{D_2} as $e_{k_{D_2}}(C)$.

5. Next, M_2 stores this content in a centralized location or sends it directly to device members in the targeted domain (i.e. the account domain in our scenario). The way this is organized is based on the organization defined policy that is stored and enforced by the master controller. This organization policy in turn would be based on who will receive the content and what operations would be performed on the content. For example, protected content could be stored in a central database management system, where authorized users from a specific set of devices regularly check for newly arriving content, for example, by using a specific software application. The software application, in turn, processes the content in a controlled manner. Alternatively, protected content could be sent directly, for example, via email to the intended recipients.

6. Only authorized users from devices that are member of the account domain in O_2 can access the protected content. This is ensured as the content is encrypted using the account's domain-specific key. This key is only available in device members of the account's domain, and hence only those devices can decrypt and access content bound to this dynamic domain. Therefore, if anyone copied the content they will not be able to access it except on devices possessing the content-specific dynamic domain key, i.e. devices that are member of the domain where the content is bound.

For the second scenario, Andrew who works on device$_1$ in O_1, wants to transfer content to O_2, so that the content can only be accessed by a set of users in O_2, who do not appear to constitute a specific group of users or an existing department as per the list provided by O_2. Also, Andrew wants to ensure that the content will not leak to other users in O_2. This scenario runs in a similar way to the first scenario described above; however, it requires Andrew to ask his own organization's security administrators to send a request to the security administrators of O_2 to establish a dynamic domain for this group of users that are required to share the confidential information. Once O_2's security administrators receive this request they initially verify that these users do not form an existing domain, then they check O_2's policy and make sure that there are no restrictions on this group of users to perform a new dynamic domain. If so, the security administrators creates a new dynamic domain and then add devices that are used by the specified users to the new domain. The way this is implemented is described in Chapter 6. The master controller of O_2 then updates its domain list and pushes it to O_1's master controller, which, in turn, stores it, as described in Section 7.3.1, point (8). Now, Andrew can send content to the newly established domain in O_2, exactly as described in the first scenario.

For the third and last scenario, Andrew, who works on device$_1$ in O_1, wants to transfer content to O_2. Andrew requires the content to be accessed by all devices in O_2; however, content must not leak to devices that are not member of O_2, i.e. if the content is transferred deliberately or accidentally to devices outside O_2 then the content should not be accessible. This scenario runs in a similar way to the first scenario described above. However, this scenario differs in the following: device$_1$ instructs the master controller in O_1 to send the content so that it is bound to O_2's global domain. Also, the master controller in O_2 decrypts the protected content received from O_1's master controller and then re-encrypts it using O_2-specific global domain key. All devices in O_2 possess a copy of the global domain key, and hence can access this content.

7.4 Discussion and Conclusion

Our solution attempts to protect content which needs to be shared between collaborating organizations. For achieving content protection between any collaborating organizations we need two important elements: the first is mutual trust between pairs of organizations that need to collaborate, and the second is having a secure system. For the first element, in practical life collaborating organizations mutually trust each other for protecting shared information, e.g. a collaborating organization policy makers will not deliberately authorize leaking information. Without having this level of trust, organization cannot rely on secure system alone to prevent content leakage. For the second element, collaborating organizations require a secure system which allows these collaborating organizations to specify their own security policies for shared content, and which allows enforcing these policies, and so preventing unethical employees from leaking organizations' content.

In this chapter we propose a system for protecting confidential information against leakage whilst being shared by collaborating organizations. As discussed in Section 7.2, when an organization wants to transfer part of its confidential information to another organization, the confidential information must be protected at three stages. The first is protecting content whilst being transferred between collaborating organizations. The second is protecting content in the destination organization, in such a way it cannot be leaked outside it. The third case is protecting content when it is restricted to be accessed by a group of users or a specific department in the destination organization. For the latter, the system should ensure that the confidential information only can be accessed by the intended users in a group/department, which are specified by the source organization.

In our proposed scheme we addressed the first problem by integrating the trusted computing features into the current VPN schemes to ensure that endpoints of communication are running as expected.

We address the leakage problem in the second case by creating a global domain in the destination organization; the global domain allows content to be shared between all devices member of the destination organization, and simultaneously it protects content from being leaked outside the destination organization. This is ensured as only member devices in the destination organization possess a copy of the global domain key. This key is not available in the clear even to users working on their devices or to security administrators, so it cannot be transferred from the organization's devices to devices outside the organization. The client application in organizational devices will not release the content unprotected to any other device, even to devices member of the same global domain. Therefore, if an employee in the destination organization attempts leaking content outside an organization they will fail as he is not capable of transferring unprotected content to others. Even if they transferred protected content to a device outside the organization, this device will not be able to access the content, as it does not possess a copy of the organization domain's key.

For the third case (i.e. protecting shared content to only be accessible by a specific group in the destination organization) the proposed solution is based on dynamic domains. Each dynamic domain consists of a specific group of devices. Each device possesses a copy of the dynamic domain-specific key that is used to protect content bound to that domain. Therefore, other devices cannot access the dynamic domain-specific content, as they do not possess a copy of this key. This is ensured in the same way as described for the first problem.

Part IV

Threat Analysis and Conclusion

Chapter 8

Threat Analysis

Contents

In this section we analyze the threats, services, and mechanisms for all the schemes discussed in this thesis, i.e. the global domain mechanism, the dynamic domain mechanism, and the collaborating organization scheme discussed in Chapters 5–7.

8.1 Global Domain Scheme

In this section we consider the threats, services, and mechanisms that apply to the global domain scheme discussed in Chapter 5. We focus on the threats, services and mechanisms that apply to digital content and the global domain credentials.

We split the system into six main cases to simplify the security analysis, as illustrated in Figure 8.1. Each case includes all the threats which might break the security of the system. After each case we explain how the threats can be mitigated. Some of the threats are addressed using the security services which can be provided using either trusted platform functionality, as discussed in Chapter 3, or using standard cryptographic mechanisms. Other threats are addressed using our provided proposed scheme.

I. **Case 1:** Security administrators when interacting with the master controller raise two main threats.

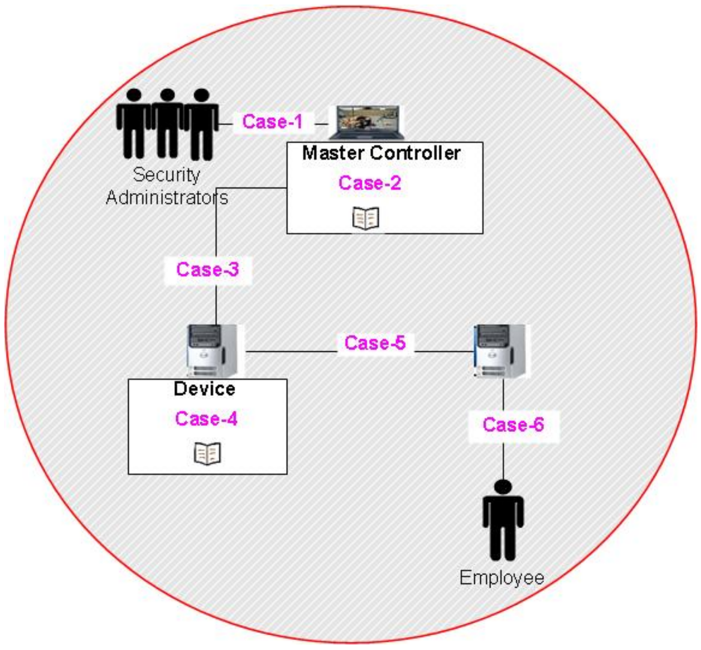


Figure 8.1: External Leakage Analysis

1. One or more security administrators violate their privileges in either of two ways: (a) Sharing their credentials with others who are not authorized to use the system to enable unauthorized parties to add unauthorized devices to a domain; or (b) One or more security administrators authorize adding unauthorized devices to a domain.
2. An unauthorized party steals security administrators authentication credentials to add an unauthorized device into a domain.

Security services and mechanisms provided to cover threats in Case 1 are as follows:

1. Security administrators authorization violation threat. The effect of this can be mitigated by combining different measures as discussed in Chapter 4, for example: (a) requiring that N out of M administrators successfully authenticate themselves directly to the master controller for request authorization; (b) using logging and auditing mechanisms that could detect abnormalities in the system; and (c) using the policy of separation of duty, for example, prevent security administrators from accessing log files, which are routinely examined by auditors.
2. An unauthorized party attempts to steal a security administrator's authentication credentials. i.e. someone steals the administrators' smart

card or tries to watch the keyboard when they enter the credentials. This can be mitigated by using strong authentication measures which involve a combination of "something the security administrator has" e.g. a smart card; "something the security administrator is", e.g. biometric verification; and/or "something the security administrator knows", e.g. a password or PIN.

II. **Case 2:** The organization master controller raises the following threats when processing and storing system credentials.

System credentials include only the administrators' credentials and the global domain credentials defined in Chapter 4. The threats are as follows.

1. Unauthorized manipulation of system credentials during use in the master controller.
2. Unauthorized manipulation of system credentials whilst stored in the master controller.

Security services and mechanisms provided to cover threats in Case 2 are as follows:

1. Confidentiality and integrity protection of system credentials during execution in a master controller. Providing this service requires process isolation techniques, in which applications run in isolation, free from being observed or compromised by other processes running in the same protected partition, or by software running in any insecure partition, as discussed in Chapter 3.
2. Confidentiality and integrity of system credentials whilst stored in the master controller. Providing this service requires protected storage, as discussed in Chapter 3.

III. **Case 3:** The interaction between a device joining a domain and the master controller raises the following threats to the global domain key k_G whilst in transit.

1. Unauthorized reading or alteration of k_G whilst in transit.
2. The master controller wittingly/unwittingly sending k_G to a malicious entity.
3. A device wittingly/unwittingly receiving k_G from a malicious entity.

4. Replay of communications between the master controller and a device.

Security services and mechanisms provided to cover threats in Case 3 are as follows:

1. Confidentiality and integrity of k_G whilst in transit. As discussed in Section 5.2.2, this service is provided by the use of asymmetric encryption and we assume that the encryption primitive in use provides non-malleability.

2. Entity authentication of a device to an organization master controller. This service involves a protocol exchange between the device and the master controller, as discussed in Section 5.2.2. It is initiated when the master controller and the joining device mutually authenticate to each other. This mutual authentication attests to the scheme applications execution status and whether the platform is trusted, as discussed in Section 5.2.2. By this the master application can only communicate with a trusted entity, and so cannot unwittingly send k_G to a malicious entity. Similarly, the client application, if it is not operating properly, cannot get k_G and so it cannot wittingly send k_G to a malicious entity.

3. Entity authentication of a master controller to a device. Same as point (2) above.

4. Prevention of replay of communications between a master controller and a device. This is provided by the inclusion of nonces in protocol messages (see Section 5.2.2).

IV. **Case 4:** Domain devices raise the following threats to the processing and storage of k_G and content.

1. Unauthorized reading or alteration of k_G during use in the device.

2. Unauthorized reading or alteration of k_G whilst stored in the device.

3. Unauthorized reading or alteration of content during use in the device.

4. Unauthorized reading or alteration of content whilst stored in the device.

Security services and mechanisms provided to cover threats in Case 4 are as follows:

1. Confidentiality and integrity of k_G during execution on a device is provided as discussed in Case 2, point (1) above.

2. Confidentiality and integrity of k_G whilst stored in a device is provided exactly as discussed in Case 2, point (2) above.

3. Confidentiality and integrity of domain content during execution on a device is provided partially as discussed in Case 2, point (1) above. In addition to that, the content-specific application needs to be implemented not to release unprotected content.

4. Confidentiality and integrity of domain content whilst stored on a device. Content is encrypted using the secret key k_C. The symmetric encryption technique in use is assumed to provide authenticated encryption [45]. The encryption key k_C is bound to the device's trusted environment, as discussed in Section 5.2.

V. **Case 5:** Interaction between domain member devices when exchanging content raises the following threats.

1. Unauthorized reading or alteration of content while in transit.

2. Transfer of content to an unauthorized entity.

3. A device receiving content from a malicious entity.

4. Replay of communications between devices.

Security services and mechanisms provided to cover threats in case - 5 are as follows:

1. Confidentiality and integrity protection of content whilst in transit. This service is provided by encrypting content using an authenticated encryption technique [45].

2. Entity authentication of the destination device whilst transferring content. A source device does not need to validate destination device. This is because when in transit content is always encrypted using the key k_D, which is known only to domain devices. k_D is a domain-specific key that is revealed only to the client application in a trusted environment configuration state. If content is leaked to an unauthorised party it cannot be accessed as it does not possess the key k_D.

3. Entity authentication of the source device whilst receiving content. A destination device does not need to validate the source device. This is because content is always sent encrypted using the key k_D, which is known only to domain devices. As above, k_D is a domain-specific that is revealed only to the client application in a trusted environment configuration state, i.e. a malicious entity does not have a copy of k_D and so it cannot send content encrypted using this key.

4. Prevention of replay of communications between devices is provided by nonces exactly as discussed in Case 3, point(4) above.

VI. **Case 6:** Organization-authorized employees raise the following main threats.

1. An authorized user may make content accessible to an unauthorized party (content leakage). This can be performed using different mechanisms such as:

   (a) Transferring digital content in the clear using physical media (e.g. USB memory stick) or communication media (e.g. the Internet) to unauthorized users.

   (b) Transferring encrypted digital content and the means for accessing it (e.g. encryption key) to unauthorized users.

   (c) An employee share his credentials with others who are not authorized to access content enabling them to have access to the content or to get the means to access the content in an unauthorized way.

2. An authorized user attempts to add an unauthorized device to the global domain.

3. An unauthorized user attempts to steal employee's authentication credentials enabling unauthorized access to content.

Security services and mechanisms provided to cover threats in case 6 are as follows:

1. Prevent content being accessible by unauthorized entities is provided by our proposed scheme as follows.

   (a) Prevent an authorized user from transferring content in the clear. When being transferred, content is always encrypted using the key

k_G which is only available to the trusted client application. The client application decrypts content and reveals it to trusted requestor applications. As we assumed in Section 4.5.2 the client application and content-specific applications are designed to not transfer unencrypted content to other entities. We rely on trusted computing functions described in Chapter 3 to make sure that the c;oemt application and the requestor applications will behave as expected.

(b) Prevent an authorized user from transferring content encryption key k_G to unauthorized users. The key k_G is only available to the trusted client application and it is not available to the users/employees. This is achieved as follows: the key k_G is sealed with the client application integrity measurements. Trusted computing functions described in Chapter 5 will make sure that the key is released only to the client application when it behaves as expected. As we assumed in Section 4.5.2 the client application is designed to not transfer content protection key to other entities.

(c) An employee shares his credentials with others who are not authorized to access content. This is to enable them to get a copy of the domain key then access content in an unauthorized way. To address this problem we bind the key with the device's public key. Therefore, even if users share their credentials they cannot access content except on a specific set of devices associated with a single employee.

2. Prevent adding unauthorized device to the global domain and prevent transferring k_G to unauthorized entities. This is achieved by different ways as discussed in previous points. For example, only the master controller can release k_G to joining devices after ensuring the device is a trusted authorized device. This requires verifying the devices status is trusted. Each domain has a predefined public key list (PKL) which specifies which devices can join the domain and have a copy of the domain key. Adding devices to a domain, is controlled by security administrators. The master controller needs to check that the device public key is included in the PKL, as defined by the security administrator's before it goes into the steps for releasing the key for devices request joining the domain.

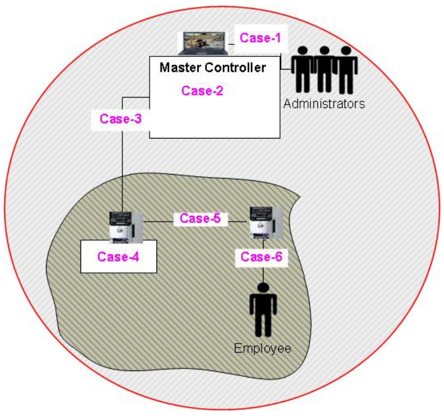


Figure 8.2: Internal Leakage Analysis

3. An unauthorized user attempts to steal employees authentication credentials enabling unauthorized access to content is mitigated exactly as discussed in Case 3, Case 1, point(2) above.

8.2 Dynamic Domain Scheme

In this section we consider the threats, services, and mechanisms that apply to the dynamic domain scheme discussed in Chapter 6. We mainly focus on the threats, services and mechanisms that apply to digital content and domain credentials. A dynamic domain credential consists of public keys of all devices that are members of the domain number of devices in the domain, the dynamic domain key and its identifier.

We split the system into eight main components to simplify the security analysis, as illustrated in Figure 8.2. The first six cases are similar to the same cases described in the previous section. Therefore, in this section we will only cover the last two cases.

VII. **Case 7:** An employee's device that is a member of multiple domains raises the following threats.

    1. An authorized employee member of multiple domains can cause an unauthorized data-flow between these domains. For example, an employee who is a member of domains D_1 and D_2 can read D_1 content and save it as D_2 content. This would enable employees who are member of D_2 but are not members in D_1 to access D_1's content in unauthorized way.

Security services and mechanisms provided to cover threats in Case 7 are as follows:

1. Controlling data-flow between domains subject to each domain-specific policy. Providing this service requires process isolation techniques, as discussed in Chapter 3.

VIII. **Case 8:** An employee who is a member of a domain when interacting with another employee member of another domain raises the following threats.

1. An authorized employee who is a member of one domain transfers unprotected content to another employee who is a member of a different domain.
2. An authorized employee who is a member of one domain transfers protected content and the means for accessing it to another employee who is a member of a different domain.

Security services and mechanisms provided to cover threats in Case 8 are as follows:

1. Protecting content whilst used by a device. Content is encrypted using the dynamic domain key k_D. The encryption key k_D can only be released to the trusted client application when the device execution environment is trusted, as discussed in Section 4.5. The trusted client application does not allow transferring unprotected content to others. This prevents authorized employees transferring unprotected content to others.
2. Binding access to content to specific devices. The proposed scheme binds access to content to specific devices. Therefore, an authorized user cannot transfer the means to access content to others.

8.3 Collaborating Organization Scheme

In this section we consider the threats, services, and mechanisms that apply to the collaborating organization scheme discussed in Chapter 7. We mainly focus on the threats, services and mechanisms that apply to digital content.

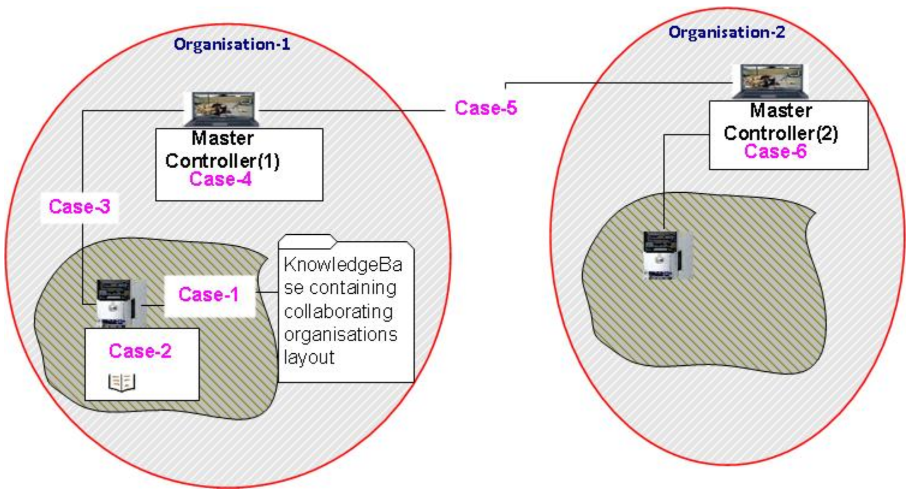


Figure 8.3: Collaborating Organization Analysis

The threats, services and mechanisms that apply to domain credentials are provided exactly as discussed in the previous schemes.

We split the system into six main components to simplify the scheme security analysis, as illustrated in Figure 8.3. In this section we do not cover the cases, which are discussed in the previous two schemes.

I. **Case 1:** A source organization device raises the following threats when interacting with the KnowledgeBase.

   1. A source organization device communicates with a malicious KnowledgeBase.
   2. Unauthorized alteration of collaborating organization layout whilst in transit from the KnowledgeBase to the device.

   Security services and mechanisms provided to cover threats in Case 1 are as follows:

   1. Entity authentication of KnowledgeBase to a device. This is provided as described in Section 8.1, Case 3, point (2).
   2. Integrity protection of KnowledgeBase reply whilst in transit. This service is provided by using digital signatures.

II. **Case 2:** A source organization device raises the following threats when processing content metadata.

1. Unauthorized reading or alteration of metadata during use in the device.

Security services and mechanisms provided to cover threats in Case 2 are as follows:

1. Confidentiality and integrity of content metadata during execution on a device. This is provided as described in Section 8.1, Case 2, point (1).

III. **Case 3:** A source device raises the following threats when sending content associated with metadata to the master controller.

1. Unauthorized reading or alteration of content and/or metadata on transit.
2. A device unwittingly sending content and/or metadata to a malicious entity.
3. The master controller unwittingly receiving content and/or metadata from a malicious entity.
4. Replay of communications between the master controller and a device.

Security services and mechanisms provided to mitigate threats in Case 3 are as follows:

1. Confidentiality and integrity protection of content and/or metadata on transit. This is provided exactly as described in Section 8.1, Case 5, point (1).
2. Entity authentication of the master controller to the device. A source device does not need to validate the master controller identity and trustworthiness. This is because communication is sent encrypted using the key k_D, which is shared between member domain devices and the master controller. Also, k_D is bound to trusted environment configuration state, i.e. k_D is revealed only to the client application in a trusted environment configuration state.
3. Entity authentication of a device to the master controller, is provided as described in the previous point.

4. Prevention of replay of communications between a master controller and a device. This is provided exactly as described in Section 8.1, Case 3, point (4).

IV. **Case 4:** The source master controller raises the following threats when processing content and associated metadata.

1. Unauthorized reading or alteration of content and metadata during use in the master controller.

Security services and mechanisms provided to cover threats in Case 4 are as follows:

1. Confidentiality and integrity protection of content and metadata during use in the master controller is provided exactly as discussed in Section 8.1, Case 4, point (3).

V. **Case 5:** The interaction between the source organization master controller M_S and destination organization master controller M_D raise the following threats.

1. Unauthorized reading or alteration of content and/or metadata on transit.
2. M_S unwittingly sending content and/or metadata to a malicious entity.
3. M_D unwittingly receiving content and/or metadata from a malicious entity.
4. Replay of communications between M_S and M_D.

Security services and mechanisms provided to cover threats in Case 5 are as follows:

1. Confidentiality and integrity protection of content and metadata on transit is provided as described in Section 8.1, Case 5, point (1).
2. Entity authentication of M_D to M_S. M_S does not need to validate M_D identity or trustworthiness. This is because communication is sent encrypted using the key k_O, which is shared between M_S and M_D. Also, k_O is bound to trusted environment configuration state, i.e. k_O is revealed only to the client application in a trusted environment configuration state.

3. Entity authentication of M_S to M_D is provided exactly as described in the previous point.

4. Prevention of replay of communications between M_S and M_D is provided as discussed in Section 8.1, Case 3, point (4).

VI. **Case 6:** The destination master controller raises the following threats when processing content and associated metadata including binding content to the destination domain.

1. Unauthorized reading or alteration of content and metadata during use in the master controller.

2. The master controller unwittingly sending content to a malicious entity

Security services and mechanisms provided to cover threats in Case 6 are as follows:

1. Confidentiality and integrity protection of content and metadata during use on M_D is provided as described in Section 8.1, Case 2, point (1).

2. Entity authentication of the receiving domain's devices to M_D. M_D does not need to verify destination domain devices identity and trustworthiness. This is because M_D encrypts content using the recipient domain key, which is available only to member domain devices when their execution environment is as expected.

8.4 Summary

In this chapter we have provided a detailed informal threat analysis of the schemes, which were discussed throughout the thesis. We show how our schemes mitigate the risk of data leakage from insiders in enterprises. This is achieved based on the way we designed the system using combination of standard cryptographic mechanisms and trusted computing technology.

Chapter 9

Discussion, Limitations, and Further Research

Contents

This chapter starts with a comparison between the DRM scheme and the scheme proposed in this thesis. Next it provides an analysis of our scheme and discusses the limitations. Finally, it gives some directions for further research.

9.1 Comparison with DRM

As discussed in Chapter 3 the work presented in this thesis is built upon part of the DRM work proposed by Abbadi et al. [7, 8, 10]. Figure 9.1 illustrates the boundaries between the work presented here and the DRM work. This figure starts from the bottom by stating the problem that DRM and our scheme focuses on. DRM schemes prevent content proliferation when users

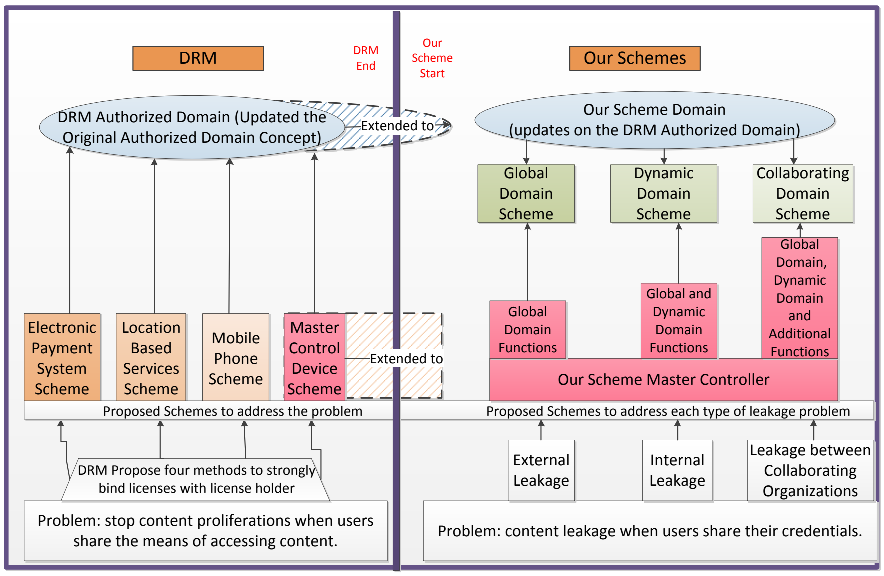

DRM
DRM End
Our Scheme Start
Our Schemes
DRM Authorized Domain (Updated the Original Authorized Domain Concept)
Extended to
Our Scheme Domain (updates on the DRM Authorized Domain)
Global Domain Scheme
Dynamic Domain Scheme
Collaborating Domain Scheme
Global Domain Functions
Global and Dynamic Domain Functions
Global Domain, Dynamic Domain and Additional Functions
Our Scheme Master Controller
Electronic Payment System Scheme
Location Based Services Scheme
Mobile Phone Scheme
Master Control Device Scheme
Extended to
Proposed Schemes to address the problem
Proposed Schemes to address each type of leakage problem
DRM Propose four methods to strongly bind licenses with license holder
External Leakage
Internal Leakage
Leakage between Collaborating Organizations
Problem: stop content proliferations when users share the means of accessing content.
Problem: content leakage when users share their credentials.

Figure 9.1: DRM and ERM Boundaries

share the means of accessing content, while our scheme focuses on limiting content leakage when users share their credentials. This figure also illustrates the four methods proposed by DRM to strongly bind licence holders with the authorized domain. These are electronic payment system, location-based services, mobile phone, and master control device. Figure 9.1 illustrates how our scheme extends one DRM scheme (i.e. the master control device) as it is more suitable for organizational context. This is extended to the master controller which is used in our schemes. The master controller provides the global domain scheme, the dynamic domain scheme, and the collaborating organization scheme functions. The figure also illustrates that we extend the authorized domain concept to a new updated authorized domain. This is used in the dynamic domain, global domain and collaborating domain schemes proposed in this thesis.

As indicated above, our schemes extended two DRM proposed concepts: the master controller and the authorized domain; however, the schemes proposed in this thesis change the way these components work. Now we will discuss the main differences between our schemes and the DRM with master control device scheme. We start by discussing the similarities and differences of requirements between DRM master control device and the master controller, we then discuss the similarities and differences of functions between DRM master control device and the master controller for each proposed scheme in this thesis. Finally, we explain the differences in the Authorized Domain concept.

9.1.1 Comparison based on Requirements

We now set the similarities and differences between the master control device requirements as discussed in [7] and the proposed scheme master controller requirements discussed in Chapter 4.

The master controller in our scheme is similar to the DRM master control device in [7] in the following ways: i) both schemes require the master controller to be a trusted device, and ii) both require a trusted software agent to implement the master controller functions.

The master controller in our scheme is different to the DRM master control device in the following ways: i) in the DRM scheme the master control device does not need to be a dedicated device — it could be any device (e.g. a mobile phone); our scheme, on the other hand, requires the master controller to be a dedicated high performance server device, ii) in the DRM scheme the master control device can be moved anywhere with the domain owner; the master controller in our scheme, on the other hand, cannot leave the organization premises and is bound to work at a specific physical location and it must be monitored by CCTV. In addition, in our schemes the master controller has to have auditing and logging capabilities.

9.1.2 Comparison based on Functions

We now compare the DRM master control device functions (as discussed in [7]) against our scheme master controller device functions. The master controller in both schemes (i.e. global domain and DRM) use similar function names, but with different implementations.

9.1.2.1 DRM Master Control Device Functions

The main function of the master control device in DRM [7] is to bind proprietary content's license with the license holder as follows:

- Bind the usage of the master control device functions to be used by a single user (i.e. the license holder). Two methods of authentication were discussed password/PIN and/or biometric authentication.
- Bind devices of the license holder to the authorized domain. This is achieved by creating an authorized domain with a domain key that is securely generated and transmitted from the master controller to joining

devices after satisfying four main conditions: successfully authenticating the domain owner; ensuring that the joining device is within a physical proximity to the master control device; validating the number of devices do not exceed a predefined limit; and validating the number of devices that have ever joined the domain do not exceed a predefined limit.

- It adds, removes and revokes devices in the domain.
- It communicates with a trusted third party to retrieve and update the certificate defining domain counter limits.

9.1.2.2 Global Domain Master Controller Functions

The global domain master controller functions proposed in this thesis include the common functions which are repeated in the other schemes (i.e. the dynamic and the collaborating schemes). The rest of the schemes have additional functions that are explained later in this chapter in each scheme's master controller functions. The global domain master controller functions are the following:

i) *Create domain function.* Although both this scheme and the DRM [7] scheme create a domain-specific symmetric key, they have a different implementation of this function. In DRM the create domain function associates the symmetric key with the domain owner authentication credentials and with two counters controlling domain limits. Our scheme creates domain function, and associates the symmetric key with the security administrators' authentication credentials and the domain-specific public key list.

ii) *Join domain function.* Although both this scheme and the DRM [7] scheme send the domain-specific symmetric key to joining devices after performing validations, they both have different validation requirements. In DRM the validation includes checking that the two domain counters are within the limits, checking physical proximity, and authenticating the domain owner. In our scheme the validation includes checking that the device public key is included in the domain-specific public key list, and it also requires authenticating security administrators. In other words, DRM binds content with the number of devices while our schemes bind content with the identity of devices. This is one of the most important differences between both schemes as it contributes in addressing the content leakage problem.

iii) *Remove device function.* Both this scheme and the DRM [7] scheme would need to validate the status of the device to be trusted and then instruct the device to remove the domain-specific key from protected storage. In the DRM scheme removing a device would also require decrementing the counter of number of devices in the domain. On the other hand, removing a device from the global domain requires removing the device public key from the global domain public key list. The global domain also requires removing the device from all dynamic domains the device is part of, which we discuss it latter in this section.

The master controller in the global domain has additional functions in comparison with the DRM scheme as follows:

i) It binds content to the domain by encrypting content with a domain specific key and it would then either store the encrypted content in a centralized server or distribute it to all member devices. On the other hand, in the DRM scheme the rights issuer binds content with any domain trusted device, which in turn binds the content to its specific domain.

ii) In the work presented in this thesis the master controller manages and securely stores the global domain public key list. On the other hand, the DRM scheme does not have a public key list to be managed.

iii) In the work presented in this thesis the master controller provides domain expand and shrink functions. On the other hand, in the DRM scheme the domain is static and not dynamic.

9.1.2.3 Dynamic Domain Master Controller Functions

The similarities, differences and additional functions which we discuss for the global domain master controller also apply to the dynamic domain master controller functions. However, the master controller in the dynamic domain also has additional functions which are not required for DRM [7].

i) *For the create domain function.* In addition to the differences and similarities of this function in the global domain scheme, we require an additional validation step in the dynamic domain scheme which checks that the dynamic domain is subset of the global domain. This is done by checking that the dynamic domain public key list is included in the global domain public key list.

ii) *For the expand domain function.* When adding a device to a dynamic domain the expand function checks that the device public key exists in the global domain public key list.

iii) *For the remove domain function.* Removing a device could require only removing it from a specific dynamic domain. However, this could also require removing it from multiple dynamic domains, or removing it from all dynamic domains and the global domain. This would depend on the reason for the removal, and would be decided by the security administrators.

iv) Manage multiple numbers of dynamic domains.

9.1.2.4 Collaborating Organization Master Controller Functions

The similarities, differences and additional functions which we discuss for the global and the dynamic domains' master controller applies to the collaborating organization master controller when managing the collaborating organizations' global and dynamic domains. However, the master controller in the collaborating organization domain (as discussed in Chapter 7) has additional functions as follows.

i) The master controller must initialize a trusted secure channel between master controllers of collaborating organizations.

ii) The master controller manages the policy of collaborating organizations.

iii) The master controller manages the processes of secure exchanging of content between collaborating organizations.

DRM, on the other hand, does not have the collaborating domain concept, i.e. one of the foundation of DRM is to stop content sharing between different domains.

9.1.3 DRM Authorized Domain vs. Proposed Scheme Domains

DRM requires each person to have their own specific authorized domain which includes all devices that need to access the proprietary content. Our schemes, on the other hand, require the organization to have two types of domains: the global domain and the dynamic domain. The organization would have a single global domain containing all devices that belong to the organization and multiple dynamic domains. A dynamic domain is a subgroup of the global domain which means that its devices must be members of the global domain. The DRM authorized domain is static, i.e. the number of devices is fixed and the relationship between devices and domains is many-to-one. Our scheme domains, on the other hand, are dynamic where the relationship between devices and domains are many-to-many and we do not worry about number of devices in a domain. The DRM schemes manage the domain by using two counters: the first controls the number of devices in the domain and the second controls the number of devices that have ever joined the domain. The last counter was introduced to stop domain owners from keeping adding and then removing devices from a domain. Our schemes, on the other hand, manage domains by using a domain-specific public key list to validate a devices identity in the domain. This is because in our scheme we care about the devices identity rather than the number of devices in a domain. A key point in DRM schemes is to strongly bind the domain to its owner. Our schemes, on the other hand, do not bind the domain to a specific user, i.e. the organization policy and process workflow controls the users who can access the domain content.

All these differences introduce major differences in our scheme functions and protocols in comparison with the DRM [7] scheme functions and protocols.

9.2 Proposed Scheme Relation with ERM

ERM schemes as we discussed in Chapter 3 provide rights management solutions for organizations to manage and enforce content access rights wherever content is transferred and used. At this stage we rely on these schemes to provide rights management to our schemes. However, ERM does not address content leakage when users share their credentials. The schemes proposed in this thesis restrict accessing content to a group of selected devices, whereby content is accessed subject to predefined fine-grained access rights associated with the content. In other words, associating our schemes with ERM assures organizations that if users share their credentials, content cannot be accessed even with valid rights object on any unauthorized device. To conclude, combining our scheme with ERM provides organizations with rights management and enforcement solution together with addressing the content leakage problem when users share their credentials. In addition, it helps in managing the dynamic nature of organizations by the usage of the dynamic domain concept. Our planned future work (as discussed in Section 9.5) is to enhance the rights management part of the ERM to address the weaknesses in ERM. We then integrate the enhanced right managements scheme in our schemes to mange the dynamic and global domain's rights.

9.3 System Analysis

In this section we discuss the proposed scheme based on the organizational requirements, which we identified in Section 2.5. We consider how the proposed scheme meets each of our requirements.

Requirement 1: Support for **organization dynamic groups**. To achieve this requirement, our scheme proposes the dynamic domain concept which covers the following two points.

- Organization grouping structure. A dynamic domain groups devices and content together to facilitate secure content sharing between devices. In our proposed scheme, organizations can create groups and projects in the form of dynamic domains which can be created or diminished at any time as needed. An employee might participate in multiple dynamic domains at the same time.
- Organization dynamic nature. In our proposed scheme, a dynamic domain can be created, expanded, and/or diminished any time when needed. In addition, an organization can dynamically move devices between dynamic domains based on its needs.

Requirement 2: **Content Sharing**. Our proposed scheme provides the capabilities to authorized employees to exchange and access content. Authorized employees who are members of a dynamic/global domain can transfer content between each other and share it. Devices that require access to shared content must join all dynamic domains where shared content is bound.

Requirement 3: **Content protection.** The dynamic and global domain in our proposed scheme limits the effects of insiders on content confidentiality. It moves the fundamental access control assumption from "the need to trust authorized users to not share their credentials" (not necessarily trustworthy and cannot measure their trustworthiness) to "the need to trust authorized devices", where trustworthiness can be measured and attested. This is achieved in our scheme by binding content with an organization's authorized trusted devices. This in turn provides stringent controls on sharing content with unauthorized users when authorized users attempt sharing their credentials.

Our proposed scheme limits the internal and external leakage defined in Chapter 2 as follows.

Limit the internal leakage threat: Our proposed scheme limits content leakage accidentally or deliberately to unauthorized employees. We achieve this requirement by binding content to dynamic domain devices where authorized employees can access content. Each device in the dynamic domain possesses a copy of the domain key, which is used to protect a pool of content that needs to be shared between the dynamic domain devices. In the proposed solution we ensure that the domain key is only released to authorized devices which behave as expected and whose public keys exist in the dynamic domain public key list. The domain key is a very important key that needs to be protected from being revealed to unauthorized devices. In our scheme, we achieve this by securely generating, transferring, and storing the dynamic domain key. Content cannot be transferred unprotected to other devices in the organization because the application is trustworthy, which means devices will always receive protected content. In this case the recipient device either could be an authorized device for accessing content or it could be a device that is not authorized to access the content. Authorized devices can decrypt the content because they already possess a copy of the content protection key. However, unauthorized devices are not able to access the content because they do not have a copy of the key. By this, our proposed scheme limits the threat of internal content leakage.

Limit the external leakage threat: Our proposed scheme limits content leakage accidentally or deliberately to unauthorized employees outside the organization. We meet this requirement by binding content to global domain devices where authorized employees can access content. How global domain protects content is the same way as the dynamic domain does which is discussed above.

Requirement 4. **Rights management and enforcement**. The organization types that we described in Section 2.3 do not necessarily require employees who are members of a project to have a unified access rights, i.e. employees could have different access rights in comparison with each other. Assigning employees access rights on a project's content would be subject to the roles required to achieve the project functions. Therefore, any scheme proposed for this kind of organization will require: (a.) a mechanism to manage employees' access rights, and (b.) a mechanism to enforce access rights wherever content is transferred and moved between client devices. ERM schemes provide dynamic rights management which is achieved by having a centralized right management server. Client devices interact with the rights management server and download content access rights. Although ERM schemes have weaknesses in managing access rights when addressing the type of organization we are working on; however, at this stage of our research we still rely on ERM to manage access right of users in organizations that wish to implement our schemes. Such integration is a planned area of future work that is highlighted later in this chapter.

In addition to the above requirements there are other important sub requirements. These include performance, interpretability, ease of use, and scalability. Assessing these requirements would be based on the scheme's implementation. The additional requirement are as follows.

Performance. The performance of the dynamic and global domain management functions (i.e. domain establishment, and adding devices to and removing devices from a domain) is implementation-dependent. The scheme requires content to be encrypted using a symmetric technique. Symmetric encryption techniques are simple and fast to compute by comparison with asymmetric techniques.

Interpretability. The framework does not require use of a particular device type. For example, content can run on any device in a dynamic and global domain, as long as the device satisfies the platform properties described in Section 4.2.5, which are based on TCG specifications.

Ease of use. Domain establishment involves initializing a domain, adding devices to a domain, and removing devices from a domain. The details of these phases are implementation-dependent. Unlike home users, the scheme proposed in this thesis is expected to be used by highly professional individuals. This is because the scheme has been proposed to be implemented in organizational environment in which they have specialized employees to perform their job.

Resilience and Scalability. These important properties ensure that the system functions are maintained in case of a single point of failure or increased load. The main component that should provide these features in our scheme is the master controller. Our scheme can work by having multiple fully coordinated master controllers (i.e. a single master controller and multiple slave controllers), as discussed in [5]. This type of architecture would address these features.

9.4 Limitations

The limitations of our scheme are as follows.

- The proposed scheme requires trusted computing technology. We discussed this point in Section 3.6.4, and conclude that trusted computing is not far from being applied in practice. Great support for trusted computing technology is emerging from the open source community, and from collaborative research projects (e.g. OpenTC[1] and EMSCB[2]). Open source trusted virtualization layers are being developed by both the Xen and L4 communities [18]. Moreover given that enterprise infrastructure is more advanced and more managed in comparison with home network environments it seems likely that the technology will succeed first in enterprise settings rather than for home use.

- The proposed scheme does not address the analog hole problem. We have the following comments on this point.

  (a) Organizations could implement physical measures, which may provide some limitations on the freedom a user can have to share content. The more an organization cares about its content, the more physical protection measures it could provide to mitigate the analog hole problem (e.g. CCTV, stronger authentication, and employees organizations in different offices so that employees assigned to different projects cannot interact with each other).

  (b) We cannot deny that there are threats coming from the analog hole problem. However threats arising from transferring digital content are likely to have more of an impact. This is because copying analog content is typically more time consuming. On the other hand, transferring digital content can be done quickly and easily. This could affect a large amount of content as users in some cases can transfer thousands of files, pictures, or designs. Most importantly, executable files cannot be exploited by the analog hole threat.

- Our proposed scheme requires binding content to devices. This might have an impact with mobile users who want to work from a distance and for users who use portable devices to access data. In our scheme content is bound with a domain of devices and not to a specific single device. This in turn would enable mobile users to add their devices to all domains they are authorized to access. Some might argue that by doing so a user can give his portable device to a third party to access content. Although this is true, we reduce the probability of this as follows.

[1]http://www.opentc.net/
[2]http://www.emscb.com/

   (a) Content will still be bound to the device and will not be transferred to a third party's device. The user may not leave his device with the third party for long, as his company will likely discover that. The device must regularly communicate with a master controller to refresh key values. In other words, if a device is hacked it will not be able to access content for long because administrators take devices off the list.

   (b) Organizations which work with confidential content just allow the devices that are really needed to work from outside the organization and not all devices.

- As we assume in Section 4.2.10 each device is assigned a specific user. This might impose limitations for organizations that require employees to share a specific device. We acknowledge this problem, and are working on developing our scheme further to bypass this limitation.

- The last point is that our scheme relies on trusting the administrators to implement and manage the proposed scheme. One might argue that the proposed scheme still relies in some point on trusting users instead of devices. For this point we have the following comments:

   (a) In our scheme we reduce the need to trust all employees, to the need to trust a very small and specific group of employees, i.e. the security administrators. This will reduce the impact of content leakage when authorized employees attempt to share their credentials.

   (b) As long as the administrators are the only party who are allowed to manage the scheme and secure auditing mechanisms are in place, these will act as deterrent measures. This is because if any breach occurs, administrators will be the first to be asked and their detection will be easier.

   (c) We acknowledge that in any scheme there should be a starting point of trust. In our scheme (as discussed in Section 4.2.4) we recommend adding restrictions on the administrators which reduce their privileges (i.e. in our scheme we require the presence of N out of M of administrators to perform any administrative activity). Thus the root of trust does not reside with a single administrator.

9.5 Future Research

We conclude this thesis by noting some areas for further research. One important issue that is not addressed in the proposed schemes is access rights management in the domain schemes. This is an important area which we are aiming to work on in the near future. ERM schemes have rights management weaknesses in addressing the type of organization we are working on as follows.

(a) ERM requires defining, for every item of content, a specific set of access rights and a specific content protection key. An employee within a project most likely would have unified access rights for the project content. Therefore, the content creator in ERM schemes would need to associate with each item of content specific access rights for each user. Similarly, an employee would also need to download, for all project content, each item of content access rights. This would affect system ease-of-use and performance. This is especially the case in our organization definition as each employee would have unified access rights on all project content. However, in our scenario different employees might have different access rights. To show the effect of this in a simple example, an employee might participate in multiple projects, each consisting of many items of content. Assume an organization has a project that consists of 10 employees, and that the project has 1000 items of content. This would mean that the content creator needs to assign 1000 * 10 access rights objects for this project. Also, updating or revoking the access rights for a single employee would require updating 1000 rights objects.

(b) ERM schemes require employees to either be always connected online or periodically sync all rights objects stored on their devices. Such a process would affect network performance. This is especially the case in large organizations consisting of thousands of employees working on tens of thousands of items of content.

(c) Moreover, Adobe Rights Management ES is restricted to PDF type of document, limiting interpretability. An organization would possess different type of documents, e.g. Microsoft Word. Also, executable content are not protected in this scheme. For example, software products are not protected using Adobe Rights Management ES.

We are planning to update the master application proposed in our scheme to provide access rights management, and the client application proposed on our scheme to enforce the access rights. For example, to address the weaknesses of rights object is bounded with each item of content we plan to bind rights objects with a pool of content which is allowed to be accessed by each user. We are working on these details and we have finished an initial draft of a paper discussing this.

The proposed protocols have only been subjected to informal analysis, and security vulnerabilities may remain. Therefore, a more formal security analysis of the security protocols would be highly desirable.

I have produced proof of concept of the main blocks of the proposed schemes. Full prototyping of the overall scheme would enable the practicality of the schemes to be tested, and would also enable an assessment of their performance characteristics.

Another important area is adapting our schemes to work in a virtualized environment. Virtual machines are associated with important and desirable properties (e.g. migration); however, such a property requires careful consideration when adapting schemes. This is because we bind accessing content to physical platforms with certain properties. Moving to a virtual machine which could move between physical platforms might break our solution if not considered carefully.

Finally, we are planning to focus on mitigating insider impact on content integrity and availability.

Bibliography

[1] Data Protection Act, 1998. http://www.opsi.gov.uk/ACTS/acts1998/19980029.htm.

[2] Property attestation–scalable and privacy–friendly security assessment of peer computers. Technical report, RZ 3548, IBM Research, May 2004.

[3] Insider threat study: Illicit cyber activity in the government sector, 2008. www.cert.org/archive/pdf/insiderthreat_gov2008.pdf.

[4] Insider threat study: Illicit cyber activity in the information technology and telecommunications sector, 2008. www.cert.org/insider_threat/study.html.

[5] Imad M. Abbadi. Digital rights management for personal networks. Technical Report RHUL-MA-2008-17, Royal Holloway, University of London.

[6] Imad M. Abbadi. Authorised domain management using location based services. In Adrian David Cheak, Peter H J Chong, Winston Seah, and Shum Ping, editors, *Mobility '07: proceedings of the 4th International Conference on Mobile Technology, Applications & Systems*, pages 288–295. ACM Press, NY, September 2007.

[7] Imad M. Abbadi. Digital rights management using a master control device. In I. Cervesato, editor, *ASIAN '07: Proceedings of the 12th Annual Asian Computing Science Conference Focusing on Computer and Network Security*, volume 4846 of *Lecture Notes in Computer Science*, pages 126–141. Springer-Verlag, Berlin, December 2007.

[8] Imad M. Abbadi and Muntaha Alawneh. DRM domain authentication using electronic payment systems. In *ICEC '08: Proceedings of the tenth international conference on Electronic commerce*, pages 185–194. ACM Press, NY, August 2008.

[9] Imad M. Abbadi and Muntaha Alawneh. Preventing insider information leakage for enterprises. In *Proceedings of the Second International Conference on Emerging Security Information, Systems and Technologies*, pages 99–106. IEEE, 2008.

[10] Imad M. Abbadi and Chris Mitchell. Digital rights management using a mobile phone. In *ICEC '07: Proceedings of the ninth international conference on Electronic commerce*, pages 185–194. ACM Press, NY, August 2007.

[11] Adobe. LiveCycle ES Services, July 2008. http://help.adobe.com/en_US/livecycle/8.2/services.pdf.

[12] Adobe Reader, 2010. http://www.adobe.com/products/reader.

[13] Muntaha Alawneh and Imad M. Abbadi. Combining DRM with trusted computing for effective information access management. In *Proceedings of the 2008 workshop on Practice and Theory of IT Security (PTITS)*, 2008.

[14] Muntaha Alawneh and Imad M. Abbadi. Preventing information leakage between collaborating organisations. In *ICEC '08: Proceedings of the tenth international conference on Electronic commerce*, pages 185–194. ACM Press, NY, August 2008.

[15] Muntaha Alawneh and Imad M. Abbadi. Sharing but protecting content against internal leakage for organisations. In *DAS 2008*, volume 5094 of *Lecture Notes in Computer Science*, pages 238–253. Springer-Verlag, Berlin, 2008.

[16] Muntaha Alawneh and Imad M. Abbadi. Defining and analyzing insiders and their threats in organizations. In *Proceedings of the 2011 International Workshop on Security and Privacy in Internet of Things (SPIoT 2011)*, pages 785 – 794. IEEE, 2011.

[17] Apple Inc. Apple Fairplay, 2006. http://www.apple.com/lu/support/itunes/authorization.html.

[18] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. In *SOSP '03: Proceedings of the nineteenth ACM symposium on operating systems principles*, pages 164–177, New York, NY, USA, 2003. ACM.

[19] Matt Bishop, Dieter Gollmann, Jeffrey Hunker, and Christian W. Probst. Countering insider threats. In *Dagstuhl Seminar Proceedings 08302*, pages 1–18. RAND Corp., Santa Monica, California, 2008.

[20] Vijay Bollapragada, Mohamed Khalid, and Scott Wainner. *IPSec VPN Design*. Cisco Press, 2005.

[21] Richard C. Brackney and Robert H. Anderson. Understanding the insider threat. In *Proceedings of a March 2004 Workshop*, pages 1–110. RAND Corp., Santa Monica, California, 2004.

[22] Glenn Bruns, Daniel S Dantas, and Michael Huth. A simple and expressive semantic framework for policy composition in access control. In *FMSE '07: Proceedings of the 2007 ACM workshop on Formal methods in security engineering*, pages 12–21, New York, NY, USA, 2007. ACM.

[23] Luigi Catuogno, Hans Lhr, Mark Manulis, Ahmad-Reza Sadeghi, Christian Stble, and Marcel Winandy. Trusted virtual domains: Color your network. *Datenschutz und Datensicherheit - DuD*, 34:289–294, 2010. 10.1007/s11623-010-0089-0.

[24] Luigi Catuogno, Hans Löhr, Mark Manulis, Ahmad-Reza Sadeghi, and Marcel Winandy. Transparent mobile storage protection in trusted virtual domains. In *Proceedings of the 23rd conference on Large installation system administration*, LISA'09, pages 12–12, Berkeley, CA, USA, 2009. USENIX Association.

[25] David Chadwick, Gansen Zhao, Sassa Otenko, Romain Laborde, Linying Su, and Tuan Anh Nguyen. PERMIS: a modular authorization infrastructure. *Concurr. Comput. : Pract. Exper.*, 20(11):1341–1357, 2008.

[26] Liqun Chen, Rainer Landfermann, Hans Löhr, Markus Rohe, Ahmad-Reza Sadeghi, and Christian Stüble. A protocol for property-based attestation. In *STC '06: Proceedings of the first ACM workshop on Scalable trusted computing*, pages 7–16, New York, NY, USA, 2006. ACM.

[27] Peter M. Chen and Brian D. Noble. When virtual is better than real. In *Proceedings of the Eighth Workshop on Hot Topics in Operating Systems*, pages 133–138, Washington, DC, USA, 2001. IEEE Computer Society.

[28] J. Crampton and M. Huth. Detecting and countering insider threats: Can policy-based access control help? In *Proceedings of 5th International Workshop on Security and Trust Management*, 2009.

[29] J. Crampton and M. Huth. Towards an access-control framework for countering insider threats. In M. Bishop, D. Gollmann, J. Hunker, and C. Probst, editors, *Insider Threats in Cybersecurity – And Beyond*, pages 173–196. Springer, 2010.

[30] Enterprise & Regulatory Reform Department for Business. Information security breaches survey, 2008. http://www.pwc.co.uk/pdf/BERR_ISBS_2008*sml*.pdf.

[31] EMC. EMC Documentum Information Rights Management, December 2006. http://www.emc.com/collateral/software/white-papers/h3395_irm_tech_overview_wp.pdf.

[32] D. Ferraiolo, R. Chandramouli, and R. Kuhn. *Role-Based Access Control.* Artech House, Norwood, MA, USA, 2003.

[33] J. Davidson Frame. *Managing projects in organizations: how to make the best use of time, techniques, and people.* 3rd edition.

[34] Eimear Gallery. An overview of trusted computing technology. In Chris J. Mitchell, editor, *Trusted Computing*, chapter 3, pages 29–113. IEE, 2005.

[35] Eimear Gallery and Chris J. Mitchell. Trusted computing: Security and applications. *Cryptologia*, 33(3):217–245, 2009.

[36] Tal Garfinkel and Mendel Rosenblum. When virtual is harder than real: security challenges in virtual machine based computing environments. In *Proceedings of the 10th conference on Hot Topics in Operating Systems - Volume 10*, pages 20–20, Berkeley, CA, USA, 2005. USENIX Association.

[37] Yacine Gasmi, Ahmad-Reza Sadeghi, Patrick Stewin, Martin Unger, Marcel Winandy, Rani Husseiki, and Christian Stüble. Flexible and secure enterprise rights management based on trusted virtual domains. In *STC '08: Proceedings of the 3rd ACM workshop on Scalable trusted computing*, pages 71–80, New York, NY, USA, 2008. ACM.

[38] G. Scott Graham and Peter J. Denning. Protection: principles and practice. In *AFIPS '72 (Spring): Proceedings of the May 16-18, 1972, spring joint computer conference*, pages 417–429, New York, NY, USA, 1972. ACM.

[39] David Grawrock. *Dynamics of a Trusted Platform.* Intel, 2008.

[40] Vivek Haldar, Deepak Chandra, and Michael Franz. Semantic remote attestation: a virtual machine directed approach to trusted computing. In *VM'04: Proceedings of the 3rd conference on Virtual Machine Research And Technology Symposium*, pages 3–3, Berkeley, CA, USA, 2004. USENIX Association.

[41] Chris Hibbert. A copy protection and content management system from the DVB, 2002. http://www.dvb.org/documents/newsletters/DVB-SCENE05CopyProtectionArticle.pdf.

[42] Seong Oun Hwang, Ki Song Yoon, Kyung Pyo Jun, and Kwang Hyung Lee. Modeling and implementation of digital rights. *Journal of Systems and Software*, 73(3):533–549, April 2003.

[43] International Organization for Standardization. *ISO/IEC 9798-3, Information technology — Security techniques — Entity authentication — Part 3: Mechanisms using digital signature techniques*, 2nd edition, 1998.

[44] International Organization for Standardization. *ISO/IEC 18033-2, Information technology — Security techniques — Encryption algorithms — Part 2: Asymmetric ciphers*, 2006.

[45] International Organization for Standardization. *ISO/IEC FCD 19772, Information technology — Security techniques — Authenticated encryption mechanisms*, 2007.

[46] Ram Krishnan, Ravi Sandhu, Jianwei Niu, and William H. Winsborough. Foundations for group-centric secure information sharing models. In *SACMAT '09: Proceedings of the 14th ACM symposium on Access control models and technologies*, pages 115–124, New York, NY, USA, 2009. ACM.

[47] Ulrich Kühn, Marcel Selhorst, and Christian Stüble. Realizing property-based attestation and sealing with commonly available hard- and software. In *STC '07: Proceedings of the 2007 ACM workshop on Scalable trusted computing*, pages 50–57, New York, NY, USA, 2007. ACM.

[48] Klaus Kursawe, Dries Schellekens, and Bart Preneel. Analyzing trusted platform communication. In CRASH Workshop: CRyptographic Advances in Secure Hardware, 2005.

[49] Butler W. Lampson. Protection. *SIGOPS Oper. Syst. Rev.*, 8:18–24, January 1974.

[50] P. England M. Peinado and Y. Chen. An overview of ngscb. In Chris J. Mitchell, editor, *Trusted Computing*, pages 115–141. IEE, 2005.

[51] Jonathan M. McCune, Yanlin Li, Ning Qu, Zongwei Zhou, Anupam Datta, Virgil D. Gligor, and Adrian Perrig. Trustvisor: Efficient TCB reduction and attestation. In *IEEE Symposium on Security and Privacy*, pages 143–158, 2010.

[52] Microsoft Corporation. Microsoft Windows Rights Management Services, 2005. http://download.microsoft.com/download/8/d/9/8d9dbf4a-3b0d4ea1905b92c57086910b/RMSTechOverview.doc.

[53] Raymond E. Miles and Charles C. Snow. *Organizational Strategy, Structure and Process.* Stanford University Press, 2003.

[54] Karissa Miller and Mahmoud Pegah. Virtualization: virtually at the desktop. In *Proceedings of the 35th annual ACM SIGUCCS fall conference*, SIGUCCS '07, pages 255–260, New York, NY, USA, 2007. ACM.

[55] Derek Gordon Murray, Grzegorz Milos, and Steven Hand. Improving xen security through disaggregation. In *Proceedings of the fourth ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, pages 151–160, New York, NY, USA, 2008. ACM.

[56] Open Mobile Alliance. *DRM Specification — Version 2.0*, 2006.

[57] Oracle. Information rights management — managing information everywhere it is stored and used, June 2008. http://www.oracle.com/technology/products/content-management/irm/IRMtechnicalwhitepaper.pdf.

[58] Joon S. Park, Keith P. Costello, Teresa M. Neven, and Josh A. Diosomito. A composite RBAC approach for large, complex organizations. In *SACMAT '04: Proceedings of the ninth ACM symposium on Access control models and technologies*, pages 163–172, New York, NY, USA, 2004. ACM.

[59] Joon S. Park and Shuyuan Mary Ho. Composite role-based monitoring (CRBM) for countering insider threats. In H. Chen et al., editor, *Intelligence and Security Informatics*, volume 3073 of *Lecture Notes in Computer Science*, pages 201–213. Springer-Verlag, Berlin, 2005.

[60] S. Pearson. *Trusted computing platforms: TCPA technology in context.* Publisher: Prentice Hall PTR, 2002.

[61] Pierangela Samarati Ravi S. Sandhu. Access control: Principles and practice. *IEEE Communications Magazine*, 32(9):40–49, September 1994.

[62] E. Rescorla. Diffie-Hellman key agreement method. RFC 2631, Internet Engineering Task Force, June 1999.

[63] Robert Richardson. The 12th annual computer crime and security survey, 2007. http://i.cmpnet.com/v2.gocsi.com/pdf/CSISurvey2007.pdf.

[64] Carsten Rudolph. Covert identity information in direct anonymous attestation (DAA). In H. Venter, M. Eloff, L. Lebuschagne, J. Eloff, and R. von Solms, editors, *Proceedings of the IFIP TC-11 22nd International Information Security Conference (SEC 2007)*, volume 232 of *Lecture Notes in Computer Science*, pages 443–448. Springer-Verlag, Berlin, 2007.

[65] A. Sadeghi. Trusted computing — special aspects and challenges. In V. Geffert et al., editor, *SOFSEM*, volume 4910 of *Lecture Notes in Computer Science*, pages 98–117. Springer-Verlag, Berlin, 2008.

[66] Ahmad-Reza Sadeghi and Christian Stüble. Property-based attestation for computing platforms: caring about properties, not mechanisms. In *NSPW '04: Proceedings of the 2004 workshop on New security paradigms*, pages 67–77, New York, NY, USA, 2004. ACM.

[67] Ravi Sandhu, Ram Krishnan, Jianwei Niu, and William Winsborough. Group-centric models for secure and agile information sharing. In Igor Kotenko and Victor Skormin, editors, *Computer Network Security*, volume 6258 of *Lecture Notes in Computer Science*, pages 55–69. Springer Berlin, 2010.

[68] Ravi Sandhu, Kumar Ranganathan, and Xinwen Zhang. Secure information sharing enabled by trusted computing and pei models. In *ASIACCS'06: Proceedings of the 2006 ACM Symposium on Information, Computer and Communications Security*, pages 2–12, New York, NY, USA, 2006. ACM Press.

[69] Ravi S. Sandhu. Lattice-based access control models. *Computer*, 26(11):9–19, 1993.

[70] Ravi S. Sandhu, Xinwen Zhang, Kumar Ranganathan, and Michael J. Covington. Client-side access control enforcement using trusted computing and pei models. *J. High Speed Networks*, 15(3):229–245, 2006.

[71] B. Smyth, M. Ryan, and L. Chen. Direct anonymous attestation (DAA): Ensuring privacy with corrupt administrators. In F. Stajano, C. Meadows, S. Capkun, and T. Moore, editors, *Proceedings of Security and Privacy in Ad-hoc and Sensor Networks: 4th European Workshop, ESAS 2007, Cambridge, UK*, volume 4572 of *Lecture Notes in Computer Science*, pages 218–231. Springer-Verlag, Berlin, 2007.

[72] Kyle E. Stewart, Jeffrey W. Humphries, and Todd R. Andel. Developing a virtualization platform for courses in networking, systems administration and cyber security education. In *Proceedings of the 2009 Spring Simulation Multiconference*, SpringSim '09, pages 65:1–65:7, San Diego, CA, USA, 2009. Society for Computer Simulation International.

[73] Trusted Computing Group. *Infrastructure Working Group Architecture, Part II, Integrity Management. Specification version 1.0 Revision 1.0*, 2006.

[74] Trusted Computing Group. *TPM Main, Part 1, Design Principles. Specification version 1.2 Revision 103*, 2007.

[75] Trusted Computing Group. *TPM Main, Part 2, TPM Structures. Specification version 1.2 Revision 103*, 2007.

[76] Trusted Computing Group. *TPM Main, Part 3, Commands. Specification version 1.2 Revision 103*, 2007.

Appendix A

Verification

This appendix provides a prototyping of part of the algorithms provided in this thesis.

A.1 Introduction

In this appendix we propose, as a proof of concept, a prototype and its execution output for the initialization algorithms and the global domain algorithms discussed in this thesis. We chose those algorithms as they constitute the foundation of our scheme. To ease follow up we include many comments in the provided program and referred to the algorithms used in the thesis. We next summarize the hosting environment of our prototype.

Hosting machine: We used HP Centrino Duo, running Linux Operation System (ubuntu 10.04).

TPM chip: we executed the following command at the host machine to extract the TPM details:

```
# tpm_version
  TPM 1.2 Version Info:
  Chip Version:        1.2.1.0
  Spec Level:          2
  Errata Revision:     0
  TPM Vendor ID:       IFX
  TPM Version:         01010000
  Manufacturer Info:   49465800
```

TCG Software Stack: we used the IAIK jTSS[1] software stack as an implementation of the TCG Software Stack for the Java (tm) programming language. We decided to use IAIK jTSS as our prototyping Language is Java.

Language: we used java programming language. Following is the used Java version as extracted from the hosting machine:

```
# java -version
java version "1.6.0_15"
Java(TM) SE Runtime Environment (build 1.6.0_15-b03)
Java HotSpot(TM) Client VM (build 14.1-b02, mixed mode, sharing)
```

We now provide a summary of the IAIK jTSS which would help to understand the provided software prototyping. This summary is mainly based on trustedjava.sourceforge.net website. As we said earlier we use IAIK jTSS which implements the TCG Software Stack (TSS) in Java. TSS is the core software component for interaction with the TPM, which has been designed and standardized by TCG. TSS is composed of set of hierarchical stacks as illustrated in Figure A.1. Layer-1 is the TPM chip, layer-2 is the TPM device driver, layer-3 is the TSS Device Driver Library (TDDL) which provides standard interfaces, layer-4 is the TSS Core Services (TCS), and layer-5 is the TSS Service Provider (TSP). TPM is accessed (in our hosting machine) from (/dev/tpm0) via the TDDL. TDDL is first TSS component running in user space which provides a standard abstraction layer for TPM access regardless of the TPM manufacturer. TCS interacts with TDDL and provides command serialization to TPM, manage TPM resources, and intermediate the communication between TSP and TDDL that includes building TPM command messages from TSP sent messages. TSP interaction with the TCS revolves around contexts. Upper layers can send commands to the TCS using an TCS context object which manages resources, e.g. key handles, and allocated memory. TSP is a shared library providing a standardized C interface (TSPI) which can be used by application developers when interacting with the TPM. TSP does not only provides TPM access (via TCS) but also includes additional functionality such as signature verification and persistent user storage.

When interacting with the TPM, the application developer must first instantiate a TSP context object. The application would then need to use the context object and connect to the TCS. The context object allows the usage of the

[1]http://trustedjava.sourceforge.net/

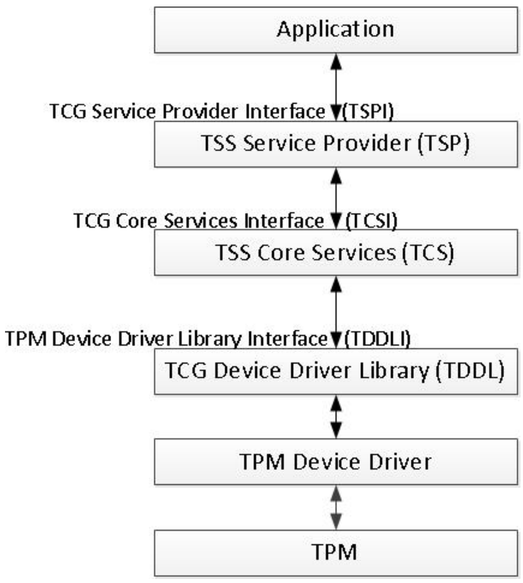

Application
TCG Service Provider Interface (TSPI)
TSS Service Provider (TSP)
TCG Core Services Interface (TCSI)
TSS Core Services (TCS)
TPM Device Driver Library Interface (TDDLI)
TCG Device Driver Library (TDDL)
TPM Device Driver
TPM

Figure A.1: The TSS Stack

TCS capabilities, used to create TSP objects (e.g. TPM, Policy, Key, Hash, EncData, and PcrComposite), registering and retrieving keys from user repository, and holds basic information about environment configuration. TPM functions such as data encryption require the knowledge of a usage secret. The association of a usage secret to TSP objects, for example to a key object, is managed at the TSP layer via a policy object. The policy object can be assigned to multiple TSP objects that use the same usage secrete.

A.2 Summary of Used Functions

In this section we provide the summary of the definition of the main TSP functions in our prototype, as provided in IAIK jTSS[2] (for further details please see their website).

- *connect()* — This method tries to connect the context to the default host (localhost).
- *getTpmObject()* — This method is used to obtain a TPM object that allows interaction with the system's TPM.
- *createEncDataObject(long initFlags)* — This method returns a new encdata objectwhere initFlags is used to specify further options for the new

[2]http://trustedjava.sourceforge.net/

object as defined by the TSS specification. Valid initFlags are: TcTss-Constants.TSS_ENCDATA_BIND, TcTssConstants.TSS_ENCDATA_SEAL, TcTssConstants.TSS_ENCDATA_LEGACY.

- *createHashObject(long initFlags)* — This method returns a new hash object; where initFlags is used to specify further options for the new object as defined by the TSS specification. Valid initFlags are: TcTssConstants.TSS_HASH_DEFAULT, TcTssConstants.TSS_HASH_OTHER, TcTss-Constants.TSS_HASH_SHA1.
- *createPcrCompositeObject(0)* — This method returns a new PCR object.
- *createPolicyObject(long initFlags)* — This method returns a new policy object; where initFlags is used to specify further options for the new object as defined by the TSS specification. Valid initFlags are: TcTssConstants.TSS_POLICY_MIGRATION and TcTssConstants.TSS_POLICY_USAGE.
- *createRsaKeyObject(long initFlags)* — This method returns a new key object; where initFlags - is used to specify further options for the new object as defined by the TSS specification. Valid initFlags are: TcTssConstants.TSS_KEY_SIZE_DEFAULT, TcTssConstants.TSS_KEY_SIZE_512, TcTssConstants.TSS_KEY_SIZE_1024, TcTss-Constants.TSS_KEY_SIZE_2048, TcTssConstants.TSS_KEY_SIZE_4096, TcTss-Constants.TSS_KEY_SIZE_8192, TcTssConstants.TSS_KEY_SIZE_16384, TcTss-Constants.TSS_KEY_TYPE_AUTHCHANGE, TcTssConstants.TSS_KEY_TYPE_BIND, TcTssConstants.TSS_KEY_TYPE_DEFAULT, TcTssConstants.TSS_KEY_TYPE_IDENTITY, TcTssConstants.TSS_KEY_TYPE_LEGACY (signing and binding), TcTssConstants.TSS_KEY_TYPE_SIGNING, TcTssConstants.TSS_KEY_TYPE_STORAGE, TcTssConstants.TSS_KEY_NON_VOLATILE, TcTssConstants.TSS_KEY_VOLATILE, TcTssConstants.TSS_KEY_NOT_MIGRATABLE (default), TcTssConstants.TSS_KEY_MIGRATABLE, TcTssConstants.TSS_KEY_CERTIFIED_MIGRATABLE, TcTssConstants.TSS_KEY_NOT_CERTIFIED_MIGRATABLE, TcTssConstants.TSS_KEY_NO_AUTHORIZATION (default), TcTssConstants.TSS_KEY_AUTHORIZATION, TcTssConstants.TSS_KEY_AUTHORIZATION_PRIV_USE_ONLY, TcTssConstants.TSS_KEY_STRUCT_DEFAULT (default), TcTssConstants.TSS_KEY_STRUCT_KEY, TcTssConstants.TSS_KEY_STRUCT_KEY12, TcTssConstants.TSS_KEY_TSP_SRK.
- *seal(TcIRsaKey encKey, TcBlobData data, TcIPcrComposite pcrComposite)* — This method encrypts a data blob in a manner that can only

be decrypted by unseal on the same system. The data blob is encrypted using a public key operation with the non-migratable key addressed by the given encryption key object. Additionally the seal operation allows software to explicitly state the future trusted configuration that the platform must be in for the encrypted data to be revealed and implicitly includes the relevant Platform Configuration Register (PCR) values when the seal operation was performed. Which PCR registers are going to be part of the seal operation is specified by the PCR composite object; where encKey is the key used for encryption, data is the data to encrypt, and pcrComposite is the PCR values the encrypted data should be sealed to (set to null to omit sealing to PCR values).

- *unseal(TcIRsaKey key)* — This method reveals data encrypted by Tspi_Data_Seal only if it was encrypted on the same platform and the current configuration (as defined by the named PCR contents of the encrypted data blob) is the one named as qualified to decrypt it. This is internally proofed and guaranteed by the TPM; where key is non-migratable key which is used to decrypt the data.
- *sign(TcIRsaKey key)* — This method signs the hash data of the object with the provided signing key; where key is the Key object which should be used for the signature.
- *setHashValue(TcBlobData hashValue)* — This method sets the hash value of the hash object; where hashValue is the hash value to be set.
- *setPcrValue(long pcrIndex, TcBlobData pcrValue)* — This method sets the digest for a given PCR index inside the PCR composite object. Multiple PCRs with different indices can be set by calling this method multiple times in the same PCR composite object; where pcrIndex is the index of the PCR to set, and pcrValue is the value of the PCR.
- *setSecret(long secretMode, TcBlobData secret)* — This method sets the authorization data of a policy object and defines the handling of its retrieval; where secretMode is a flag indicating the policy secret mode to set. Valid secretModes are: TcTssConstants.TSS_SECRET_MODE_NONE, TcTssConstants.TSS_SECRET_MODE_PLAIN, TcTssConstants.TSS_SECRET_MODE_POPUP, and TcTssConstants.TSS_SECRET_MODE_SHA1.
- *assignToObject(TcIAuthObject obj)* — this method assigns an object (working object) like TPM object, key object, encrypted data object

to a certain policy; where obj is the object to be assigned. Each of these working objects will utilize its assigned policy object to process an authorized TPM command. Note that there are two different policies that can be assigned to a working object, usage policy and migration policy.

- *createKey(TcIRsaKey wrappingKey, TcIPcrComposite pcrComposite)* — This method creates a key pair within the TPM and wraps it with the key addressed by wrappingKey. The key must already be properly set up via the key init flags or TcIAttributes.setAttribData(long, long, TcBlobData) and TcIAttributes.setAttribUint32(long, long, long); where wrappingKey is the key used to wrap the newly created key, and pcrComposite if not omitted (i.e. set to null), the newly created key will be bound to the PCR values described within this object.

- *loadKey(TcIRsaKey unwrappingKey)* — This method loads the key blob into the TPM. The TPM will unwrap the key when it is loaded; where unwrappingKey is the key which should be used for unwrapping.

- *certifyKey(TcIRsaKey certifyingKey, TcTssValidation validation)* — This method signs a public key inside the TPM using TcTssConstants.TSS_SS_RSASSAPKCS1V15_SHA1; where certifyingKey is a certifying key which is used to sign the key, and validation is a structure of the type TcTssValidation. After successful completion of the call the validationData field of this structure contains the signature data of the command. The data field of the structure contains an instance of TcTpmCertifyInfo or TcTpmCertifyInfo2.

- *getRandom(long length)* — This method returns random data obtained from the TPM via the TSS; where length is the length of the data to be requested.

- *pcrRead(long pcrIndex)* — This methods reads a PCR register; where pcrIndex is the index of the PCR to read.

- *pcrExtend(long pcrIndex, TcBlobData data, TcTssPcrEvent pcrEvent)* — This method extends a PCR register and writes the PCR event log; where pcrIndex is the index of the PCR to extend, data is a data blob for the PCR extend operation, and pcrEvent contains the info for an event entry (if this is null no event entry is created and the method only executes an TPM extend operation

- *loadKeyByUuidFromSystem(TcTssUuid uuid)* — This method creates a key object based on the information contained in the key manager using the UUID and loads the key into the TPM. The persistent storage provides all information to load the parent keys required to load the key associated with the given UUID; where uuid is the UUID of the key to be loaded.
- *getUuidSRK()* — This method returns the UUID of the SRK (PS-system, no-auth, non-migratable).

A.3 Program

```
#
# cat muntaha.java
import java.io.*;
import java.util.Arrays;
import iaik.tc.tss.api.tspi.*;
//import iaik.tc.tss.api.exceptions.common.TcTssException;
//import iaik.tc.tss.api.constants.tpm.*;
import iaik.tc.tss.api.constants.tsp.*;
import iaik.tc.tss.api.structs.common.*;
import iaik.tc.tss.api.structs.tpm.*;
import iaik.tc.tss.api.structs.tsp.*;

public class muntaha {

    public muntaha()
    {
    }

    public static void main(String[] args) throws Exception
    {
      TcIContext context_ = null;
      long current_time = System.currentTimeMillis();
      long took_time;

      try {

/*******************************************************************
*                                                                  *
*       Initialization steps                                       *
*                                                                  *
***********************************************************************
*/
System.out.println("****************************************");
System.out.println("*******  Initialization steps **********");
System.out.println("****************************************\n");

/**/
System.out.println("\ni) Create context and tpm objects.\n");
/**/
     context_ = new TcTssContextFactory().newContextObject();
                   context_.connect();
                   TcITpm tpm = context_.getTpmObject();

/**/
System.out.println("\nii) Define the policy object, set the secret in the");
System.out.println(" policy, and then assign it to the tpm object.\n");
/**/
    TcBlobData ownerSecret = TcBlobData.newString("muntaha");
    TcIPolicy policy = context_.createPolicyObject(
    TcTssConstants.TSS_POLICY_USAGE);
    policy.setSecret(TcTssConstants.TSS_SECRET_MODE_PLAIN,ownerSecret);
    policy.assignToObject(tpm);

/**/
System.out.println("\niii) Retrive the tpm's SRK and assign it to the defined");
System.out.println(" policy.\n");
/**/
    TcIRsaKey srkkey = context_.loadKeyByUuidFromSystem(
                      TcUuidFactory.getInstance().getUuidSRK());
    policy.assignToObject(srkkey);

/**/
System.out.println("\niv) Create a composite PCR object S_M which will be");
System.out.println(" used latter to bind the usage of the master");
System.out.println(" application key with it.\n");
/**/
    TcIPcrComposite S_M = context_.createPcrCompositeObject(0);
    S_M.setPcrValue(0, tpm.pcrRead(0));
    S_M.setPcrValue(1, tpm.pcrRead(1));
    S_M.setPcrValue(2, tpm.pcrRead(2));
    S_M.setPcrValue(3, tpm.pcrRead(3));
    S_M.setPcrValue(4, tpm.pcrRead(4));
    S_M.setPcrValue(5, tpm.pcrRead(5));
    S_M.setPcrValue(6, tpm.pcrRead(6));
    S_M.setPcrValue(7, tpm.pcrRead(7));
    S_M.setPcrValue(8, tpm.pcrRead(8));

/**/
System.out.println("\nv) Initialize the master application key object and");
System.out.println(" assign its policy.\n");
/**/
    TcIRsaKey MasterKey = context_.createRsaKeyObject(
                         TcTssConstants.TSS_KEY_TYPE_STORAGE |
                         TcTssConstants.TSS_KEY_SIZE_2048 |
                         TcTssConstants.TSS_KEY_NOT_MIGRATABLE);
    policy.assignToObject(MasterKey);
/**/
System.out.println("\nvi) Initialize the master application key pair");
System.out.println(" (PuM, PrM) pointed to by MasterKey object. We ");
System.out.println(" then load the key.\n");
/**/

    MasterKey.createKey(srkkey, S_M);
    MasterKey.loadKey(srkkey);

/**/
System.out.println("Initialize the AIK  key object, assign its policy,");
System.out.println("create it, and then load it.");
/**/
    TcIRsaKey AikSignKey = context_.createRsaKeyObject(
                         TcTssConstants.TSS_KEY_TYPE_SIGNING |
                         TcTssConstants.TSS_KEY_SIZE_2048 |
                         TcTssConstants.TSS_KEY_NOT_MIGRATABLE);
    policy.assignToObject(AikSignKey);
    AikSignKey.createKey(srkkey, S_M);
    AikSignKey.loadKey(srkkey);

    System.out.println("\nThe public part of the AikSignKey:\n"+
                   AikSignKey.getPubKey().toHexString());

    System.out.println("\nThe public part of the MasterKey:\n"+
                   MasterKey.getPubKey().toHexString());
    took_time= System.currentTimeMillis() - current_time;
    System.out.println("\nMaster initialization phase took: "+ took_time +" millisecond
    current_time = System.currentTimeMillis();
/****************************************************************************
```

```
*                                                                          *
*       Client Initialization at Client Device                             *
*                                                                          *
****************************************************************************
*/

System.out.println("****************************************");
System.out.println("*******  Client Initialization *********");
System.out.println("*******  at Client Device      *********");
System.out.println("****************************************\n");
/**/
System.out.println("\ni) Create a composite PCR object S_D which will be");
System.out.println(" used latter to bind the usage of the client");
System.out.println(" application key with it\n");
/**/
    TcIPcrComposite S_D = context_.createPcrCompositeObject(0);
    S_D.setPcrValue(0, tpm.pcrRead(0));
    S_D.setPcrValue(1, tpm.pcrRead(1));
    S_D.setPcrValue(2, tpm.pcrRead(2));
    S_D.setPcrValue(3, tpm.pcrRead(3));
    S_D.setPcrValue(4, tpm.pcrRead(4));
    S_D.setPcrValue(5, tpm.pcrRead(5));
    S_D.setPcrValue(6, tpm.pcrRead(6));
    S_D.setPcrValue(7, tpm.pcrRead(7));
    S_D.setPcrValue(8, tpm.pcrRead(8));

/**/
System.out.println("\nii) Initialize the client application key object and");
System.out.println(" assign its policy.\n");
/**/
    TcIRsaKey ClientKey = context_.createRsaKeyObject(
                              TcTssConstants.TSS_KEY_TYPE_STORAGE |
                              TcTssConstants.TSS_KEY_SIZE_2048 |
                              TcTssConstants.TSS_KEY_NOT_MIGRATABLE);
    policy.assignToObject(ClientKey);

/**/
System.out.println("\niii) Initialize the master application key pair");
System.out.println(" (PuM, PrM) pointed to by MasterKey object.");
System.out.println(" We then load the key.\n");
/**/

    ClientKey.createKey(srkkey, S_D);
    ClientKey.loadKey(srkkey);
    took_time= System.currentTimeMillis() - current_time;
    System.out.println("\nClient initialization phase took: "+ took_time +" millisecond
    current_time = System.currentTimeMillis();

/***************************************************************************
*                                                                          *
*       Admin Registration                                                 *
*                                                                          *
****************************************************************************
*/
System.out.println("****************************************");
System.out.println("*******  Admin Registration ************");
System.out.println("****************************************\n");

/**/
System.out.println("\ni) Retrive admins authentication credentials.\n");
/**/

    BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
    String [] adminname;
    String [] adminpass;
    int N;

    System.out.print("Enter number of admins (must be more than one):  ");
    N = Integer.parseInt(in.readLine());
    if(N < 2){N=2;}// Default number of admins

    adminname = new String[N];
    adminpass = new String[N];

    for (int i=0; i<N; i++) {
        System.out.print("Enter admin username: ");
        adminname[i] = in.readLine();
        System.out.print("Enter admin password: ");
        adminpass[i] = in.readLine();
    }


/**/
System.out.println("\nii) Seal the retrived authentication credentials.\n");
/**/
```

```
    TcBlobData adminname_blob = TcBlobData.newString(
                        Arrays.toString(adminname),true,"ASCII");
    TcIEncData encrypted_adminname_blob = context_.createEncDataObject(
                                       TcTssConstants.TSS_ENCDATA_SEAL);
    policy.assignToObject(encrypted_adminname_blob);
    encrypted_adminname_blob.setAttribData(
                        TcTssConstants.TSS_TSPATTRIB_ENCDATA_BLOB,
                        TcTssConstants.TSS_TSPATTRIB_ENCDATABLOB_BLOB,
                        adminname_blob);
    encrypted_adminname_blob.seal(MasterKey, adminname_blob, null);
    TcBlobData sealed_adminname_blob =
                        encrypted_adminname_blob.getAttribData(
                              TcTssConstants.TSS_TSPATTRIB_ENCDATA_BLOB,
                              TcTssConstants.TSS_TSPATTRIB_ENCDATABLOB_BLOB);

    TcBlobData adminpass_blob = TcBlobData.newString(
                        Arrays.toString(adminpass),true,"ASCII");
    TcIEncData encrypted_adminpass_blob = context_.createEncDataObject(
                                            TcTssConstants.TSS_ENCDATA_SEAL);
    policy.assignToObject(encrypted_adminpass_blob);
    encrypted_adminpass_blob.setAttribData(
                        TcTssConstants.TSS_TSPATTRIB_ENCDATA_BLOB,
                        TcTssConstants.TSS_TSPATTRIB_ENCDATABLOB_BLOB,
                        adminpass_blob);
    encrypted_adminpass_blob.seal(MasterKey, adminpass_blob, null);
    TcBlobData sealed_adminpass_blob =
                        encrypted_adminpass_blob.getAttribData(
                           TcTssConstants.TSS_TSPATTRIB_ENCDATA_BLOB,
                           TcTssConstants.TSS_TSPATTRIB_ENCDATABLOB_BLOB);

   System.out.println("\nAdmin credentials are sealed");
//In practice: sealed_adminname_blob would be saved to a file.

    took_time= System.currentTimeMillis() - current_time;
    System.out.println("\nAdmin Registration phase took: "+ took_time +" milliseconds\n
    current_time = System.currentTimeMillis();

/****************************************************************************
*                                                                           *
*       Admin Verification                                                  *
*                                                                           *
*****************************************************************************
*/
System.out.println("****************************************");
System.out.println("*******  Admin Verification ************");
System.out.println("****************************************\n");


/**/
System.out.println("\ni) UnSeal the admin authentication credentials.\n");
/**/

    TcIEncData  adminname_unseal_object = context_.createEncDataObject(
                                TcTssConstants.TSS_ENCDATA_SEAL);
    policy.assignToObject(adminname_unseal_object);
    adminname_unseal_object.setAttribData(
                    TcTssConstants.TSS_TSPATTRIB_ENCDATA_BLOB,
                    TcTssConstants.TSS_TSPATTRIB_ENCDATABLOB_BLOB,
                    sealed_adminname_blob);
    TcBlobData unsealed_adminname_blob =
                        adminname_unseal_object.unseal(MasterKey);


    TcIEncData  adminpass_unseal_object = context_.createEncDataObject(
                                TcTssConstants.TSS_ENCDATA_SEAL);
    policy.assignToObject(adminpass_unseal_object);
    adminpass_unseal_object.setAttribData(
                        TcTssConstants.TSS_TSPATTRIB_ENCDATA_BLOB,
                        TcTssConstants.TSS_TSPATTRIB_ENCDATABLOB_BLOB,
                        sealed_adminpass_blob);
    TcBlobData unsealed_adminpass_blob =
                        adminpass_unseal_object.unseal(MasterKey);



    String tempadmin = unsealed_adminname_blob.toStringASCII();
    tempadmin = tempadmin.replaceAll("(^\\[)","");
    tempadmin = tempadmin.replaceAll("\\].*$","");
    adminname = tempadmin.split(",\\s");

    tempadmin = unsealed_adminpass_blob.toStringASCII();
    tempadmin = tempadmin.replaceAll("(^\\[)","");
    tempadmin = tempadmin.replaceAll("\\].*$","");
    adminpass = tempadmin.split(",\\s");
```

```
/**/
System.out.println("\nii) Admin authentication.\n");
/**/
    System.out.print("Enter number of admins to authenticate (>2):  ");
    int M = Integer.parseInt(in.readLine());
    if(M<2) M=2;

    String username, password, match;
    match="F";
    for (int i=0; i<M; i++) {
        System.out.print("Enter admin username: ");
        username = in.readLine();
        System.out.print("Enter admin password: ");
        password = in.readLine();

        for(int j=0; j<N; j++){
            if(username.equals(adminname[j]) && password.equals(adminpass[j])){
              j=N;
              match="T";
            }
        }

        if(match.equals("F")) {
            System.out.println("\n Incorrect username or password,"
                                +" please re-enter them.");
            i--;
        }
        match="F";
    }
    System.out.println("\nAdmin credentials unsealed and then admins verified"+
                        " successfully.\n Username:"+
                        unsealed_adminname_blob.toStringASCII());
    System.out.println("\nPassword:"+unsealed_adminpass_blob.toStringASCII());

    took_time= System.currentTimeMillis() - current_time;
    System.out.println("\nAdmin verification phase took: "+ took_time +" milliseconds\n
    current_time = System.currentTimeMillis();

/*******************************************************************************
*                                                                              *
*       Initalize Global Domain Credentials (i_g, k_g, and pkl_g)              *
*                                                                              *
********************************************************************************
```

```
*/
System.out.println("*****************************************");
System.out.println("** Initalize Global Domain Credentials **");
System.out.println("** (i_g, k_g, and pkl_g) ****************");
System.out.println("*****************************************\n");

/**/
System.out.println("\ni) Create global domain credentials.\n");
/**/
    TcBlobData i_g = tpm.getRandom(128);
    TcBlobData k_g = tpm.getRandom(128);

    System.out.print("Enter number of device in the global domain:  ");
    int N_G = Integer.parseInt(in.readLine());

    String [] pkl_g = new String[N_G];

    for (int i=0; i<N_G; i++) {
        System.out.print("\nEnter Device "+i+" public key: ");
        pkl_g[i] = in.readLine();
    }

    System.out.println("\nGlobal domain credentials are created "+
                    "(k_g, i_g, and pkl_g).");
    System.out.println("\nk_g has the following value:\n"+k_g.toHexString());
    System.out.println("\ni_g has the following value:\n"+i_g.toHexString());
    System.out.println("\npkl_g has the following value:\n"+
                    Arrays.toString(pkl_g));

/**/
System.out.println("\nii) Seal global domain credentials.\n");
/**/

    TcIEncData encrypted_k_g_blob = context_.createEncDataObject(
                                   TcTssConstants.TSS_ENCDATA_SEAL);
    policy.assignToObject(encrypted_k_g_blob);
    encrypted_k_g_blob.setAttribData(
                    TcTssConstants.TSS_TSPATTRIB_ENCDATA_BLOB,
                    TcTssConstants.TSS_TSPATTRIB_ENCDATABLOB_BLOB,
                    k_g);
    encrypted_k_g_blob.seal(MasterKey, k_g, null);
    TcBlobData sealed_k_g = encrypted_k_g_blob.getAttribData(
                    TcTssConstants.TSS_TSPATTRIB_ENCDATA_BLOB,
                        TcTssConstants.TSS_TSPATTRIB_ENCDATABLOB_BLOB);

   System.out.println("\n k_g is sealed\n");

//In practice: sealed_k_g would be saved to a file.

    TcIEncData encrypted_i_g_blob = context_.createEncDataObject(
                                       TcTssConstants.TSS_ENCDATA_SEAL);
    policy.assignToObject(encrypted_i_g_blob);
    encrypted_i_g_blob.setAttribData(
                        TcTssConstants.TSS_TSPATTRIB_ENCDATA_BLOB,
                        TcTssConstants.TSS_TSPATTRIB_ENCDATABLOB_BLOB,
                        i_g);
    encrypted_i_g_blob.seal(MasterKey, i_g, null);
    TcBlobData sealed_i_g = encrypted_i_g_blob.getAttribData(
                        TcTssConstants.TSS_TSPATTRIB_ENCDATA_BLOB,
                        TcTssConstants.TSS_TSPATTRIB_ENCDATABLOB_BLOB);

   System.out.println("\n i_g is sealed\n");
//In practice: sealed_i_g would be saved to a file.

    TcBlobData pkl_g_blob = TcBlobData.newString(
                        Arrays.toString(pkl_g),true,"ASCII");
    TcIEncData encrypted_pkl_g_blob = context_.createEncDataObject(
                                       TcTssConstants.TSS_ENCDATA_SEAL);
    policy.assignToObject(encrypted_pkl_g_blob);
    encrypted_pkl_g_blob.setAttribData(
                        TcTssConstants.TSS_TSPATTRIB_ENCDATA_BLOB,
                        TcTssConstants.TSS_TSPATTRIB_ENCDATABLOB_BLOB,
                        pkl_g_blob);
    encrypted_pkl_g_blob.seal(MasterKey, pkl_g_blob, null);
    TcBlobData sealed_pkl_g_blob =
                        encrypted_pkl_g_blob.getAttribData(
                              TcTssConstants.TSS_TSPATTRIB_ENCDATA_BLOB,
                              TcTssConstants.TSS_TSPATTRIB_ENCDATABLOB_BLOB);

   System.out.println("\npkl_g is sealed");
//In practice: sealed_pkl_g_blob would be saved to a file.

    took_time= System.currentTimeMillis() - current_time;
    System.out.println("\nInitialize global domain credentials took: "+ took_time +" mi
    current_time = System.currentTimeMillis();
```

```
/******************************************************************************
*                                                                             *
*      Join a device to the Global Domain: this part mainly covers the master *
*      controller steps. The client steps would follow the same steps,        *
*      similar to the ones provided.                                          *
*                                                                             *
*******************************************************************************
*/

System.out.println("****************************************");
System.out.println("** Join a device to the Global Domain **");
System.out.println("****************************************\n");


/**/
System.out.println("\ni) Client sends a join domain request associated with a nonce");
System.out.println(" (nonce_from_client) and the global domain identifier i_g.");
System.out.println(" As we do not have a real client communication, we created");
System.out.println(" the nonce locally. In practice these values will be");
System.out.println(" communicated from client to the master application.\n");
/**/

    TcBlobData nonce_from_client = tpm.getRandom(128);
    TcBlobData i_g_from_client = i_g;
    if(!i_g_from_client.equals(i_g)){
        System.out.print("\nSend an error message to client device");
        System.exit(0);
    }

/**/
System.out.println("\nii)Mutual Authentication.");
System.out.println("This step include the usage of certifykey function");
System.out.println("to generate matster application key certicate bound with S_M\n");
/**/
    TcBlobData nonce_from_master = tpm.getRandom(128);
    TcTssValidation MasterKeyValidation = new TcTssValidation();

    MasterKeyValidation.setExternalData(nonce_from_client);

    AikSignKey.loadKey(srkkey);
    MasterKey.loadKey(srkkey);

    MasterKeyValidation = MasterKey.certifyKey(AikSignKey, MasterKeyValidation);
```

```
  System.out.println("\nThe MasterKey Validation includes the certified\n"+
            "master application key with S_D:\n"+
            MasterKeyValidation.getValidationData().toHexString());

//Sign nonce_from_master using AikSignKey
  TcIHash hash_object = context_.createHashObject(
             TcTssConstants.TSS_HASH_SHA1);
  TcTpmDigest master_nonce_digest = new TcTpmDigest(
             nonce_from_master);
  hash_object.setHashValue(
             master_nonce_digest.getDigest());
  TcBlobData sign_master_message =
             hash_object.sign(AikSignKey);


/**/
System.out.println("\niii) The master controller then sends the MasterKeyValidation,");
System.out.println(" nonce_from_master, and its signature (sign_master_message) to the"
System.out.println(" client device. The client device can then do the verifications.");
System.out.println(" We ommit some of the client side steps for conveniance.\n");
/**/

  match="F";
  for (int i=0; i<N_G; i++) {
    if(ClientKey.getPubKey().equals(pkl_g[i])){
      match="T";i=N_G;
    }
  }
  match="T"; //Temporarily for the purpose of prototyping
  if(match.equals("F")){
   System.out.print("\nERROR: Client is not part of the Global Domain");
  }


/**/
System.out.println("\niv) Encrypt k_g using ClientKey and send it over to");
System.out.println(" client application.\n");
/**/

  TcIEncData  k_g_unseal_object = context_.createEncDataObject(
                   TcTssConstants.TSS_ENCDATA_SEAL);
  policy.assignToObject(k_g_unseal_object);
```

```
    k_g_unseal_object.setAttribData(
                            TcTssConstants.TSS_TSPATTRIB_ENCDATA_BLOB,
                            TcTssConstants.TSS_TSPATTRIB_ENCDATABLOB_BLOB,
                            sealed_k_g);
    TcBlobData unsealed_k_g_blob = k_g_unseal_object.unseal(MasterKey);


    TcIEncData  encrypted_k_g_object_to_client = context_.createEncDataObject(
                                  TcTssConstants.TSS_ENCDATA_BIND);
    policy.assignToObject(encrypted_k_g_object_to_client);
    encrypted_k_g_object_to_client.setAttribData(
                            TcTssConstants.TSS_TSPATTRIB_ENCDATA_BLOB,
                            TcTssConstants.TSS_TSPATTRIB_ENCDATABLOB_BLOB,
                            unsealed_k_g_blob);

    encrypted_k_g_object_to_client.seal(ClientKey, unsealed_k_g_blob, null);

// send the encrypted_k_g_object_to_client to client.
    took_time= System.currentTimeMillis() - current_time;
    System.out.println("\nJoin device to the global domain took: "+ took_time +" millis
    current_time = System.currentTimeMillis();


/*****************************************************************************
*                                                                           *
*       At client device: k_c Generation and Sealing                        *
*                                                                           *
*****************************************************************************
*/
System.out.println("****************************************");
System.out.println("****** At client device: ***************");
System.out.println("****** k_c Generation and Sealing  *****");
System.out.println("****************************************\n");


    TcBlobData k_c = tpm.getRandom(128);

    TcIEncData encrypted_k_c_blob = context_.createEncDataObject(
                                  TcTssConstants.TSS_ENCDATA_SEAL);
    policy.assignToObject(encrypted_k_c_blob);
    encrypted_k_c_blob.setAttribData(
                    TcTssConstants.TSS_TSPATTRIB_ENCDATA_BLOB,
                    TcTssConstants.TSS_TSPATTRIB_ENCDATABLOB_BLOB,
                        k_c);
    encrypted_k_c_blob.seal(MasterKey, k_c, null);
    TcBlobData sealed_k_c =
                        encrypted_k_c_blob.getAttribData(
                             TcTssConstants.TSS_TSPATTRIB_ENCDATA_BLOB,
                             TcTssConstants.TSS_TSPATTRIB_ENCDATABLOB_BLOB);

   System.out.println("\n k_c is sealed");
   took_time= System.currentTimeMillis() - current_time;
   System.out.println("\nk_c generation and sealing took: "+ took_time +" milliseconds\
   current_time = System.currentTimeMillis();

/*****************************************************************************
*                                                                            *
*       Remove Device                                                        *
*                                                                            *
******************************************************************************
*/
System.out.println("*****************************************");
System.out.println("*******  Remove Device ******************");
System.out.println("*****************************************\n");


// Admins must be authenticated as explained earlier -
// we omit it to avoid repetation



    TcIEncData  pkl_g_unseal_object =
                        context_.createEncDataObject(
                                 TcTssConstants.TSS_ENCDATA_SEAL);
    policy.assignToObject(pkl_g_unseal_object);
    pkl_g_unseal_object.setAttribData(
                TcTssConstants.TSS_TSPATTRIB_ENCDATA_BLOB,
                TcTssConstants.TSS_TSPATTRIB_ENCDATABLOB_BLOB,
                sealed_pkl_g_blob);
    TcBlobData unsealed_pkl_g_blob = pkl_g_unseal_object.unseal(MasterKey);

    String temp_pkl_g = unsealed_pkl_g_blob.toStringASCII();
    temp_pkl_g = temp_pkl_g.replaceAll("(^\\[)","");
    temp_pkl_g = temp_pkl_g.replaceAll("\\].*$","");
    pkl_g = temp_pkl_g.split(",\\s");
```

```
// The master controller instructs the client device to remove ClientKey from
// its protected storage. It then continues with these steps.

    match="F";
    String more="Yes";
    String pk_device_remove = null;

      while(more.equals("Yes")){

         System.out.print("Enter client device public key to be removed : ");
         pk_device_remove = in.readLine();

         for (int i=0; i<N_G; i++) {
           if(pk_device_remove.equals(pkl_g[i])){
              pkl_g[i] = null;
              System.out.println("\nClient device is removed.\n");
              match="T";
              i=N;
           }
         }

         if(match.equals("F")){
           System.out.println("\nClient device does not exists.\n");
         }else{match="F";}

         System.out.print("If you want to remove more devices enter Yes : ");
         more = in.readLine();
      }

      took_time= System.currentTimeMillis() - current_time;
      System.out.println("\ndevice removal took: "+ took_time +" milliseconds\n");
      current_time = System.currentTimeMillis();


      } catch(Exception e) {
         System.out.println(e.getMessage());
         throw e;
      }  finally {
         context_.closeContext();
      }
   }
```

```
}
#
```

A.4 Execution Output

The following is the result of executing the provided program. We captured the execution output from Unix command prompt and copy it as follows.

```
#
# java muntaha
****************************************
*******  Initialization steps **********
****************************************


i) Create context and tpm objects.


ii) Define the policy object, set the secret in the
 policy, and then assign it to the tpm object.


iii) Retrive the tpm's SRK and assign it to the defined
 policy.


iv) Create a composite PCR object S_M which will be
 used latter to bind the usage of the master
 application key with it.


v) Initialize the master application key object and
 assign its policy.


vi) Initialize the master application key pair
 (PuM, PrM) pointed to by MasterKey object. We
 then load the key.

Initialize the AIK  key object, assign its policy,
create it, and then load it.
```

```
The public part of the AikSignKey:
 00 00 00 01 00 01 00 02 00 00 00 0c 00 00 08 00
 00 00 00 02 00 00 00 00 00 00 01 00 82 dc 72 9e
 54 ce 3b fa ac bb c9 7a 38 39 19 10 89 96 99 55
 71 0f be fd 96 c7 2c 06 7d 09 47 1a 1c 1a 03 e1
 45 89 81 f8 9a 75 2c e8 b6 26 27 5e 30 76 9d 3e
 53 62 d5 08 e6 ec 17 2d d1 d5 16 22 20 ad 82 8f
 bf 62 03 61 00 6e 4a 8d 5e 0b 17 0b 4c d3 b9 f0
 af a3 7b 26 ac de 40 f3 d3 51 e8 62 a2 8c 5b ac
 bb d0 07 ed fd 33 ce 86 7a 46 74 f3 a9 80 cb 20
 17 bf 26 d3 db 8c bc df 3f c7 05 d2 d3 5d 3e 9b
 d9 ee 46 4a 97 a7 58 42 a4 7e 01 5d 63 be 34 30
 e7 c2 ce 1f 71 f1 cf 4b dd 07 fc 43 58 a3 4b d9
 32 bd cd f4 7d a5 cd a8 d2 c8 95 e1 09 da 7c 44
 56 62 cc 62 23 b1 a3 28 28 a8 49 48 b1 0c 6e ea
 3f d2 d9 ac 2c 02 a2 c6 dd e3 48 a9 a5 d4 09 f2
 12 44 80 74 87 86 56 52 ec fd 55 98 c1 ff 2f fa
 97 dc 5f 5a 2b 84 91 94 1f f8 f4 17 e1 b6 e1 32
 be a4 3c dc 07 cd 52 e6 a7 9a 21 63

The public part of the MasterKey:
 00 00 00 01 00 03 00 01 00 00 00 0c 00 00 08 00
 00 00 00 02 00 00 00 00 00 00 01 00 85 31 02 9f
 ae 81 c0 e1 7b e9 75 dc 88 41 7c 09 2f 69 7a 1c
 ea 13 9c e6 8f 41 1d 54 61 dc b3 a8 9e 5b d9 f7
 c3 30 b4 60 01 74 1c b8 25 ad 48 9d 96 bf 4e a4
 b9 52 64 43 07 dd d3 8e b2 67 c6 2b 1a 01 57 52
 9a 7f 25 3c 07 6e b4 58 4a f7 35 a4 9e f3 06 e1
 01 55 3a 66 0b b0 91 7c c7 15 2e f7 86 ef 5e cb
 ef b8 e2 f9 a9 e8 40 cd 54 6c 03 e5 41 17 f9 66
 ae 11 13 dd bf 2a 0c 64 26 93 3c 80 2e c0 e2 98
 a5 0b 26 3c 0f 93 c8 62 df c2 69 1c 37 8c 2c e0
 eb 98 1c ee 5f 83 d3 7a e2 ea c2 47 e4 59 81 22
 c0 d6 25 cb 25 9e 59 f8 bd d5 6a 21 90 0e 2a 4f
 05 22 99 85 49 67 88 89 76 91 46 cd 8c 03 d7 f8
 6c 74 ec e7 97 70 f6 61 8c ab 6c 5b b1 97 01 f6
 e4 2f 2d b4 71 24 44 df 89 cb ae 51 28 c8 34 6c
 fc 30 52 28 13 71 5c 9c d2 51 1c 50 9d e5 c7 80
 f7 ad 80 f3 fd 07 8b 5a c0 fc d7 37

Master initialization phase took: 6425 milliseconds

****************************************
```

```
*******  Client Initialization *********
*******  at Client Device      *********
****************************************


i) Create a composite PCR object S_D which will be
 used latter to bind the usage of the client
 application key with it


ii) Initialize the client application key object and
 assign its policy.


iii) Initialize the master application key pair
 (PuM, PrM) pointed to by MasterKey object.
 We then load the key.


Client initialization phase took: 13082 milliseconds

****************************************
*******  Admin Registration ************
****************************************


i) Retrive admins authentication credentials.

Enter number of admins (must be more than one):  4
Enter admin username: user1
Enter admin password: pass1
Enter admin username: user2
Enter admin password: pass2
Enter admin username: user3
Enter admin password: pass3
Enter admin username: user4
Enter admin password: pass4

ii) Seal the retrived authentication credentials.


Admin credentials are sealed
Admin Registration phase took: 25481 milliseconds


*****************************************
*******  Admin Verification ************
******************************************


i) UnSeal the admin authentication credentials.


ii) Admin authentication.

Enter number of admins to authenticate (>2):  3
Enter admin username: user1
Enter admin password: pass4

 Incorrect username or password, please re-enter them.
Enter admin username: user1
Enter admin password: pass1
Enter admin username: user2
Enter admin password: pass2
Enter admin username: user3
Enter admin password: pass3

Admin credentials unsealed and then admins verified successfully.
 Username:[user1, user2, user3, user4]

Password:[pass1, pass2, pass3, pass4]

Admin verification phase took: 24047 milliseconds

******************************************
** Initalize Global Domain Credentials **
** (i_g, k_g, and pkl_g) ****************
******************************************


i) Create global domain credentials.

Enter number of device in the global domain:  3

Enter Device 0 public key: pub0
Enter Device 1 public key: pub1

Enter Device 2 public key: pub2

Global domain credentials are created (k_g, i_g, and pkl_g).

k_g has the following value:
 88 81 84 be 3b 6a e6 1f 60 5a 83 94 41 36 57 18
 7a 59 22 1a d4 22 51 64 3e 54 5c fa 37 ab a3 07
 35 4e 6c db af 22 0c c2 38 ed 35 fb f3 9c c2 c0
 fe e7 7d 1b 25 ed e9 61 c3 a4 48 92 d3 4a be 31
 e9 31 99 e4 d9 2f 05 4a 7a b8 95 01 57 b7 20 c6
 f2 8b 2e 58 49 31 49 38 7a 23 06 ae 6b 9f ae 0f
 49 d5 cf 23 5c 19 4f e7 98 27 b9 74 63 11 e9 03
 3d 6f 6f 06 9a 46 8d bd e0 e3 63 bc 32 44 e1 19

i_g has the following value:
 7b 81 79 20 cc 16 21 c6 a1 3d 70 79 f3 a6 d5 db
 15 06 a9 d0 0a 12 0e 3e 86 e5 d2 32 e6 0e cc d1
 77 56 e0 75 97 13 20 d4 41 9c de f2 13 9e 11 76
 95 3b 4e b7 b0 72 3c 51 b4 2d ee 0f 7c eb 4d 41
 ea 8b 5b b3 71 fc ba 10 86 22 21 2e 3b ae ba 7a
 92 6b 4d a2 0b 4b 09 bd 2e 93 51 ce 68 7a 4d d0
 3b b8 50 f6 ba 0d 19 e2 9d 9e 3d 7a c1 29 2f cd
 05 1c 4a 87 06 27 21 ec 40 b0 7a 32 32 63 d4 93

pkl_g has the following value:
[pub0, pub1, pub2]

ii) Seal global domain credentials.


 k_g is sealed


 i_g is sealed


pkl_g is sealed

Initialize global domain credentials took: 12373 milliseconds
```

```
****************************************
** Join a device to the Global Domain **
****************************************


i) Client sends a join domain request associated with a nonce
 (nonce_from_client) and the global domain identifier i_g.
 As we do not have a real client communication, we created
 the nonce locally. In practice these values will be
 communicated from client to the master application.


ii)Mutual Authentication.
This step include the usage of certifykey function
to generate matster application key certicate bound with S_M


The MasterKey Validation includes the certified
master application key with S_D:
 6f 33 8e 85 d1 7e b0 13 da 49 10 47 d6 73 43 ca
 35 3d 70 ea 03 d2 d4 49 ea e9 84 d1 63 56 96 58
 0d 31 68 08 64 c7 4c a0 d6 44 30 01 46 aa c7 fc
 35 00 c6 a6 cc 9f 03 50 08 4b 7f 47 ae e6 ab 6a
 63 3a 35 ea 62 6a aa 87 52 58 bb 52 31 92 14 11
 0d dc 3b 22 30 df 2a 1f 20 4e e4 f8 50 dd 26 ef
 51 66 2f 04 88 82 9a 8a 2d 33 ee 0f 30 d4 6c cd
 ab 28 ac 50 89 1b 9e db 1d 1e 78 07 4e 32 33 0e
 a2 0a 34 96 2f 6e 99 f8 67 21 c1 d8 49 72 a3 93
 48 5d b2 24 30 b1 b8 ec 33 d5 ae 87 3f 50 84 1b
 32 32 46 d1 41 a7 cf f1 19 18 86 e7 41 3c d2 db
 59 60 0f ec 59 35 ef d8 ea 29 52 9f f8 b3 09 3a
 5d cb 99 91 91 ab 20 1e de 8d a3 1a 02 4b a6 13
 45 91 23 15 be b4 be 97 9c 9a 4d cd 24 ac d9 b0
 18 28 15 93 a0 e0 4e 96 3e da a4 53 67 a0 18 d9
 62 5d 99 a3 bb 5a 81 64 1d d4 10 e2 48 4d 15 e2

iii) The master controller then sends the MasterKeyValidation,
 nonce_from_master, and its signature (sign_master_message) to the
 client device. The client device can then do the verifications.
 We ommit some of the client side steps for conveniance.


iv) Encrypt k_g using ClientKey and send it over to
```

```
 client application.


Join device to the global domain took: 6664 milliseconds

****************************************
****** At client device: ***************
****** k_c Generation and Sealing  *****
****************************************


 k_c is sealed

k_c generation and sealing took: 804 milliseconds

****************************************
*******  Remove Device *****************
****************************************

Enter client device public key to be removed : pub2

Client device is removed.

If you want to remove more devices enter Yes : No

device removal took: 13203 milliseconds
#
```