

# Enhancing Cloud Computing Security and Privacy

---

Nabil Giweli

A thesis submitted in fulfilment for the degree of  
Master of Science (Honours)

School of Computing, Engineering and Mathematics  
University of Western Sydney

2013

# Publications from this Thesis

---

N. Giweli, S. Shahrestani, and H. Cheung, "Enhancing Data Privacy and Access Anonymity in Cloud Computing," Communications of the IBIMA, vol. 2013 (2013), p. 10, 2013.

URL:

<http://www.ibimapublishing.com/journals/CIBIMA/2013/462966/a462966.html>

N. Giweli, S. Shahrestani, and H. Cheung, "Cloud Computing: Preserving Data Privacy and Managing Access Control," presented at the 19th International Business Information Management Conference, Barcelona, Spain, 2012.

URL: <http://www.ibima.org/spain2012/papers/nabi.html>

## Copyright

Permission is granted to use this work or any portion of it for any purpose as long as sufficient context is given around the citation to correctly explain the author's original intent.

## Dedication

To my mother Zohra Banon and my father Mohamed Jawelli.

# Acknowledgements

---

I owe my deepest gratitude to my supervisor, Dr. Seyed Sharhestani, for his encouragement, guidance and support. If it was not for the motivation and the guidance of Dr. Seyed, completing this thesis would not have been possible. His suggestions helped me improve my problem solving capabilities and research skills.

Thanks are further due to my Co-supervisor Dr. Hon Cheung, for his help in completing this thesis and for his continuing advice. I also thank him for providing me with valuable suggestions and comments during my work.

I owe a lot to my colleagues in the Networking, Security, and Cloud Research group for encouragement and valuable discussions during this work.

I warmly thank the School of Computing, Engineering and Mathematics, In particular, Noshir Bulsara, campus support officer and Bruce Kneale, systems specialist, for their help and technical support.

Finally, I would like to thank my parents and family for all the sacrifice they have done for me in order to succeed throughout my life and particularly during the completion of this thesis.

# Statement of Authentication

---

I certify the work presented in this thesis is, to my best knowledge and belief, original except as acknowledged in this thesis. I hereby declare that I have not submitted this material, either in whole or in part, for a degree at this or any other institution.

**Nabil Giweli**



# Table of Contents

---

List of Figures .....	iv
List of Tables .....	v
List of Acronyms .....	vi
Abstract .....	viii
Chapter 1: Introduction .....	1
1.1    Motivations and Background .....	1
1.2    Research Objectives and Stages .....	2
1.3    Research Scope and Questions .....	3
1.4    Research Outcomes .....	4
1.5    Thesis Structure .....	6
Chapter 2: Cloud Computing and Security .....	8
2.1    Introduction .....	8
2.2    Cloud Computing Definition and Characteristics .....	8
2.3    Cloud Computing Service Delivery Models .....	10
2.3.1    Software as a Service (SaaS) .....	10
2.3.2    Platform as a Service (PaaS) .....	11
2.3.3    Infrastructure as a Service (IaaS) .....	11
2.4    Cloud Computing Service Deployment Models .....	12
2.4.1    Private Cloud .....	12
2.4.2    Community Cloud .....	12
2.4.3    Public Cloud .....	12
2.4.4    Hybrid Cloud .....	13
2.4.5    Virtual Private Cloud .....	13
2.5    Cloud Computing: Issues and Challenges .....	13
2.6    Trends and Directions of Solutions .....	17
2.6.1    Protecting Data Privacy and Integrity from Cloud Providers .....	18
2.6.2    Cryptography Techniques Fitting the Cloud Model .....	20
2.6.3    Trusted Computing (TC) .....	21
2.6.4    Data Centric Security (DCS) .....	21
2.7    Summary .....	22

Chapter 3: Data Centric Security Approach for Cloud Computing .....	23
3.1    Introduction .....	23
3.2    Data-Centric Security Approach .....	23
3.2.1    Classifying Security Solutions for Cloud Computing.....	25
3.2.2    DCS Characteristics .....	32
3.2.3    DCS Conceptual Framework .....	39
3.3    DCS Expected Benefits .....	43
3.4    Challenges of Applying DCS Approach .....	45
3.4.1    Challenges Related to Secure Access Control .....	45
3.4.2    Challenges Related to Queries and Search on Encrypted Data.....	47
3.4.3    Challenges Related to Outsourcing Computation .....	48
3.4.4    Challenges Related to Overhead and Performance .....	48
3.5    Identifying the Scope of Applying the DCS Approach.....	49
3.6    Essential Security Requirements for Public Cloud Storage .....	50
3.6.1    Privacy Preserving of Data Sharing and Access Control.....	51
3.6.2    Secure Searching on Encrypted Data .....	55
3.6.3    Data Integrity Proof.....	58
3.7    Evaluating the Reviewed Security Techniques .....	59
3.8    Summary .....	62
Chapter 4: A Data Centric Solution to Cloud Privacy and Security Issues .....	64
4.1    Introduction .....	64
4.2    Chinese Remainder Theorem and its Applications .....	65
4.3    Cryptography Access Control and Key Sharing Using the CRT .....	68
4.3.1    Enhancing Keys and Files Security.....	69
4.3.2    Secure Access Control Procedure .....	70
4.3.3    Granting and Revoking Procedure .....	73
4.4    Secure Search Ability .....	75
4.5    Access Procedure with Secure Search Ability .....	75
4.6    Constructing a DCS file with Integrity and Authenticity Proofs.....	77
4.7    Privacy Preserving and Integrity of the Proposed Solution .....	79
4.8    Summary .....	80
Chapter 5: Implementation Issues and Evaluation.....	81
5.1    Introduction .....	81



5.2	Implementation Tools and Experiment Environment .....	81
5.3	Client Side Implementations .....	83
5.3.1	Symmetric Encryption of the Original File.....	83
5.3.2	Implementing the Calculating CRT Solution.....	84
5.3.3	Asymmetric key Encryption .....	87
5.3.4	Encrypting the Keywords.....	89
5.3.5	Calculating the Integrity and Authenticity Proof .....	90
5.3.6	Creating a Secure File (DCS file) by the Data Owner .....	91
5.3.7	Operations at the Authorized User Side.....	96
5.4	Server Side Implementation and Experiments .....	102
5.5	Results and Discussion .....	104
5.5.1	At the Data Owner Side .....	104
5.5.2	At the Authorized User Side .....	111
5.6	Summary and Conclusion .....	111
Chapter 6: Conclusions and Future Work.....		113
Appendix A: The Implementation Code .....		118
References .....		141

# List of Figures

---

Figure 1.1 Research Scope .....	4
Figure 2.1 Issues Ranked by Cloud Computing Costumers .....	14
Figure 2.2 Basic Architecture for Preserving Data Privacy in the Cloud .....	19
Figure 3.1 Data Centric and System Centric Security .....	27
Figure 3.2 Possible Security Levels .....	27
Figure 3.3 DCS Framework .....	42
Figure 3.4 A Simple Way of Securely Sharing Data .....	52
Figure 4.1 Access Control Procedure.....	72
Figure 4.2 Secure Search Procedure .....	76
Figure 4.3 Constructing the DCS file .....	78
Figure 5.1 Experiment Environment for the Proposed Solution.....	83
Figure 5.2 Snapshot of the RSA Encryptions for Ten Users .....	89
Figure 5.3 Snapshot of Calculating HMAC-SHA256 for Five Keywords .....	90
Figure 5.4 Snapshot of Calculating CRT Solution for Five Users.....	93
Figure 5.5 Snapshot of Calculating File Digest and Signature for 40.7 MB File .....	94
Figure 5.6 Snapshot of Computing the $C_r    K_s$ Value from $X_r$ Value .....	98
Figure 5.7 Snapshot of Verifying Integrity and Authenticity .....	101
Figure 5.8 Snapshot of Running Search Process .....	104
Figure 5.9 The Size of $X_r$ in Bits Related to Number of Users .....	106

# List of Tables

---

Table 3.1 Heterogeneous Descriptions of Data-Centric Security .....	28
Table 5.1 The Code for Finding CRT Solution.....	85
Table 5.2 The Code for Calculating the Modular Multiplicative Reverse.....	86
Table 5.3 The Code for Calculating HMAC for a Keyword.....	89
Table 5.4 The Code for Calculating the Integrity and Authenticity Proof .....	91
Table 5.5 Code Description of Constructing the DCS file .....	94
Table 5.6 Code Description of Computing the Secret value $C_r    K_s$ from the Shared Value $X_r$ .....	97
Table 5.7 Code Description of Reading DCS file and Recreating the Original File..	99
Table 5.8 Code Description of Verifying Integrity and Authenticity .....	100
Table 5.9 The Code of Calculating the HMAC-SHA256 for an Input keyword .....	103
Table 5.10 Storage and Time Overheads for the AES Encryption .....	105
Table 5.11 Storage and Time Overheads for Calculating the Shared Value $X_r$ .....	106
Table 5.12 Storage and Time Overheads for Adding Encrypted Keywords.....	107
Table 5.13 Storage and Time Overheads for Adding Integrity and Authenticity proof .....	108
Table 5.14 Total Storage Overhead for Creating DCS file for Five Users and with Five Keywords .....	109
Table 5.15 Time Overhead for Copying the Encrypted File Content to DCS file ...	109
Table 5.16 Total Time Overhead Required for Creating DCS file .....	110
Table 5.17 Time Execution of the Operations at the User Side for different DCS files .....	111

# List of Acronyms

---

<b>ABE</b>	Attribute-based encryption
<b>AES</b>	Advanced Encryption Standard
<b>API</b>	Application Programming Interface
<b>AWS</b>	Amazon Web Services
<b>BDH</b>	Bilinear Diffie-Hellman
<b>CA</b>	Certified Authority
<b>CCA</b>	Chosen-Ciphertext Attack
<b>CIA</b>	Confidentiality, Integrity and Availability
<b>CPA</b>	Chosen-Plaintext Attack
<b>CP-ABE</b>	Ciphertext-policy ABE
<b>CPU</b>	Central Processing Unit
<b>CRT</b>	Chinese Remainder Theorem
<b>CSP</b>	Cloud Service Provider
<b>DaaS</b>	Database as a Service or Data storage as a Service
<b>DCS</b>	Data Centric Security
<b>DDH</b>	Decisional Diffie–Hellman
<b>DH</b>	Diffie-Hellman
<b>EEA</b>	Extended Euclidean Algorithm
<b>EHR</b>	Electronic Health Records
<b>HMAC</b>	Hash-based Message Authentication Code
<b>HTTP</b>	Hypertext Transfer Protocol
<b>IaaS</b>	Infrastructure as a service
<b>IBE</b>	Identity Based Encryption
<b>ICS</b>	Information Centric Security
<b>ICT</b>	Information and Communication Technology

<b>IDC</b>	International Data Corporation
<b>IPMaaS</b>	Identity and Policy Management as a Service
<b>IT</b>	Information Technology
<b>KMS</b>	Key Management Server
<b>KP-ABE</b>	Key-policy ABE
<b>MIM</b>	Main-In-the Middle
<b>NaaS</b>	Network as a Service
<b>NHS</b>	National Healthcare System
<b>NIST</b>	US National Institute of Standards and Technology
<b>OS</b>	Operating System
<b>PaaS</b>	Platform as a service
<b>PEKS</b>	Public-key Encryption with Keyword Search
<b>PKI</b>	Public key Infrastructure
<b>RSA algorithm</b>	Rivest Shamir Adleman Algorithm
<b>SaaS</b>	Software as a Service
<b>SFO</b>	Secure File Object
<b>SLA</b>	Service-Level Agreement
<b>SSE</b>	Symmetric Searchable Encryption
<b>TC</b>	Trusted Computing
<b>TCG</b>	Trusted Computing Group
<b>TMI</b>	Trusted Multi-Tenant Infrastructure
<b>TPM</b>	Trusted Platform Module
<b>TTP</b>	Trusted Third Party
<b>VM</b>	Virtual Machine
<b>VPC</b>	Virtual Private Cloud
<b>VPN</b>	Virtual Private Network
<b>XaaS</b>	anything-as-a-Service

# Abstract

---

Cloud computing paradigms are gaining widespread acceptance due to the various benefits they offer. These include cost-effectiveness, time savings and efficient utilization of computing resources. However, privacy and security issues are among the major obstacles holding back the widespread adoption of this new technology. The nature of these paradigms requires the customers to move their data to the cloud. As security and privacy of the data are usually handled by the service providers, the data owners may not even be fully aware of the underlying security challenges and solutions. The research to find solutions to these issues is very active in several directions. Some research is focused on improving the security at the application, operating system, Virtual Machine (VM) or hardware levels. These solutions do not normally provide a comprehensive solution and they still keep the data security measures under the control of the cloud provider. Another direction of research is based on Trust Computing (TC) concepts. In essence, these provide a set of trusted third party technologies to secure the VM from the cloud provider. While these approaches provide the users with tools to monitor and assess the security aspects of their data, they do not provide the users with much control capability. In contrast, Data Centric Security (DCS) is an emerging approach that aims to provide data owners with full control of their data security from within the data itself, throughout the data's lifecycle on the cloud. However, the concept of the DCS approach is interpreted in various ways in the literature and there is not yet a standardized framework of applying this approach to the cloud model. This thesis aims to enhance cloud computing security by proposing a solution based on the DCS approach.

The research towards achieving this aim starts with a systematic review of the literature to establish a framework to utilize DCS concepts for improving data security and privacy in cloud environments. The DCS concept is based on providing security at the data level. Hence, the data are self-describing, self-defending and self-protecting during their lifecycle in the cloud environments. The data owner is solely responsible to set and manage the data privacy and security measures. These requirements can be achieved without depending on trusting the cloud provider or/and a trusted third party assistance. Then, this conceptual framework is developed into an applied solution. The proposed solution is based on the Chinese Remainder

Theorem (CRT) and utilizes symmetric and asymmetric encryption techniques. To reduce the computational and management overheads, access control policy enforcement and sharing the symmetric key of encrypted data are accomplished in an efficient manner based on the CRT. For enhancing security, the data owner is able to use a unique symmetric key for encrypting each set of data and to attach it securely to the encrypted data. Only authorized users are given access to the key. Additionally, the privacy of access is improved by keeping the number of authorized users and their identities hidden even from the cloud provider. Moreover, secure search capabilities on the encrypted data are an integral part of the proposed solution. All the required security parameters, including integrity and authenticity proof parameters, are attached to the encrypted data to create a secure file container, which is referred to as a DCS file. Only authorized users can search and access DCS files, based on the embedded policies that are set and managed exclusively by the data owner.

This work also examines the relevant implementation issues and overheads of the proposed solution, mainly in terms of the required computation and storage capabilities. The experimental evaluations and the implementations use Java for the main operations. These operations include; creating the DCS files at the data owner side, searching through them at the server, and decrypting their contents at client side. The implementation and experiments show that the proposed solution can be used practically and efficiently.

In summary, one of the main contributions of this work is to take advantage of the benefits of the DCS approach in achieving practical solutions to security and privacy issues encountered in the cloud computing environments. In approaches developed here, all the security measures are created and managed by the data owner and are tightly attached to the data without requiring additional key management overhead or complex computations. The solutions strengthen the security to the level that even the cloud provider cannot compromise the integrity and privacy of users' data.

## Chapter 1: Introduction

### 1.1 Motivations and Background

Cloud computing technology has become a popular alternative to conventional computing technologies. This technology provides a new concept of a pay-per-use utility model of computing resources based mainly on virtualization technology. The fundamental features of cloud computing services are namely: on-demand self-service, broad network access, resource pooling, measured service, rapid elasticity and location independence [1-4]. Numerous benefits result from these features, such as cost-effectiveness, time saving, scalability and green IT. Despite these benefits, the wide adoption of this new technology faces several obstacles including security and privacy challenges [5]. In addition to the traditional security risks experienced by computing systems connected to the Internet, cloud systems have specific security and privacy issues because of the cloud's virtualization and multi-tenancy nature [6]. Due to the fact that cloud computing resources are shared amongst different customers, a possible malicious customer may exploit virtualization vulnerabilities to perform security attacks on other customers' data and applications. Moreover, cloud users have limited control over their data in the cloud, while a cloud provider gains excessive ability to control a customer's data including enforcing access control policies on the data [7]. Furthermore, in cloud computing, a user's data may be moved between different cloud providers, as a consequence of using other providers' services under pre-subcontracts [8]. Moreover, a cloud provider can be self-interested, un-trusted and possibly malicious with regard to customers' data [9, 10]. Because of these issues, data owners are reluctant to move their sensitive data to the cloud because of privacy and data integrity concerns. Accordingly, the need for stronger data security in the cloud environment leads to research on methods which protect customers' data in the cloud even from the cloud providers themselves [7, 11-14].

Some methods of securing customers' data from cloud providers are based on Trusted Computing (TC) technologies, such as the Trusted Platform Module (TPM) [15-19]. The protection of customers' data by these methods is controlled by a third party and does not give the data owners full control over the protection of their data.



Another approach, referred to as the Data Centric Security (DCS) or Information Centric Security (ICS) approach, aims to protect the data itself before the data are outsourced to the cloud. This approach has been suggested as the most promising approach in securing customers' data in the cloud [11, 15, 20-23]. The concept of this approach is still developing and there are different views about how to apply it to the cloud model. The cryptography technologies are the main security tools for the DCS approach [13]. Encrypting data before they are sent to the cloud is a straightforward way to protect the privacy of data. However, some usual features are lost as a result of the data being encrypted. For example, encrypted data cannot be searched nor used in computations without being decrypted first. Therefore, one of essential requirements is how to securely perform searches on the encrypted data without revealing the data content or the search query information [13]. In addition, access control policies on a customers' data are private information and they should not be revealed by the cloud server. Hence, researchers are working on how a cloud server enforces the access control policies that are defined and managed by data owners without the policies being revealed to any entities including the cloud server [5]. In general, a new security is required to enable cloud resources to be utilized efficiently by costumers without exposing their sensitive data and private information to unauthorized entities including the cloud providers. In addition, the data owners are able to control their outsourced data security and privacy, and verify security conditions, such as the integrity of the data, at any time.

## **1.2 Research Objectives and Stages**

The objective of this research is to develop a solution which is able to enhance the security and privacy of customers' data stored in cloud computing systems. To achieve this objective, the significance of security and privacy issues in the cloud model will be investigated. The DCS approach is adopted in this thesis to achieve the main objectives of this study. The first task before using this approach is identifying the characteristics and criteria of this approach to gain the most benefits from using it. Based on the identified criteria, the available security techniques and algorithms are reviewed and their suitability to be used in the DCS approach is investigated. Then the best techniques and algorithms to provide the most fine-grained security and privacy requirements are adopted and enhanced for developing a feasible

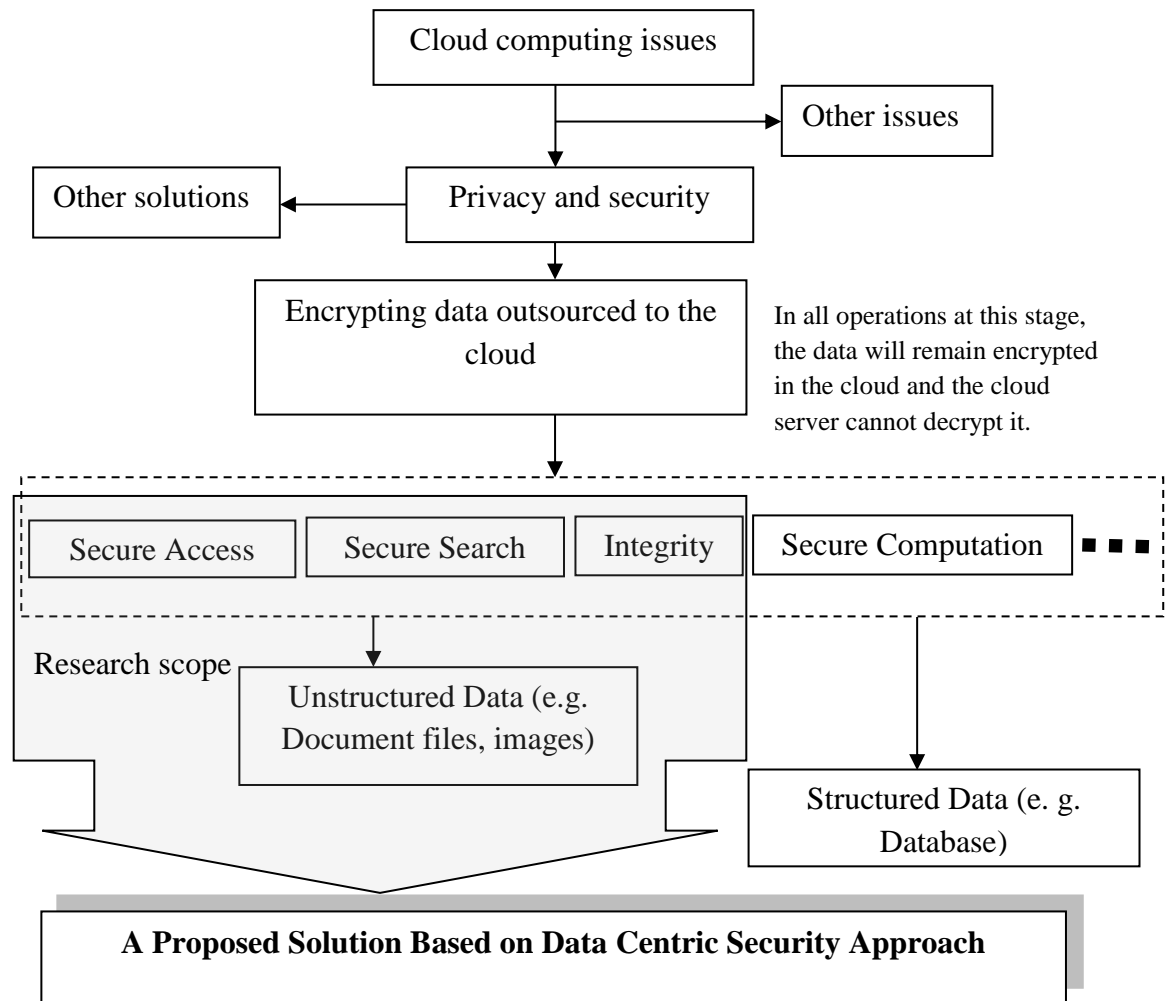
solution. Finally, the proposed solution will be evaluated in a test bed cloud environment.

### 1.3 Research Scope and Questions

As discussed in Section 1.1, the DCS approach has the characteristics to address security and privacy issues with regard to customers' data being stored and processed in the cloud. To develop a technique based on this approach the following research questions are first formulated:

- 1. How can existing security techniques be used to develop a solution to enhance privacy and security in the cloud based on the DCS approach?**
- 2. How can data outsourced to the cloud be secured in respect to the following characteristics or features:**
  - **Security of data is ensured during access and sharing by authorized users.**
  - **The access control policies on the data are protected from users and the service providers.**
  - **Protected data are searchable without compromising their confidentiality.**
  - **Data integrity is verifiable by authorized users.**
  - **Privacy of data is enhanced in such a way that even the cloud provider is not able to compromise it.**
  - **The security requirements on the protected data are created and controlled by the data owner and attached to the actual data.**

As shown in Figure 1.1, this research focuses on addressing security and privacy issues in the cloud by basically encrypting data before sending them to the cloud. The scope of the proposed solution designed to be applied to the most popular scenario of using cloud services where unstructured data, such as document, image and video files, are stored and shared in the cloud. The three research areas that will be covered are: securely sharing and accessing encrypted data, securely searching the encrypted data and protecting data integrity. The proposed solution will be based on the DCS approach.



**Figure 1.1 Research Scope**

## 1.4 Research Outcomes

The first contribution of this research is to develop a conceptual framework of the DCS approach for the cloud computing model. The essential characteristics of the DCS approach are identified including its benefits and the technical challenges in applying this approach to the cloud environments.

The second contribution is to develop a solution based on the DCS approach that includes these features:

- The security and privacy of the data outsourced to public cloud storage are enhanced by using existing security techniques mainly: Chinese Remainder Theorem (CRT), asymmetric encryption, symmetric encryption and hashing algorithms.

- The access control policies, search capability and data integrity and authenticity proof are all attached to each file itself independently from other files. The cloud provider is not required to store or expose any information about the data. Hence, the proposed solution is not dependant on security measures and facilities provided by the cloud provider or a third party.
- The access control policies are hidden even from the cloud server responsible for enforcing them. Only data owners can set and manage these policies throughout the data life time in the cloud.
- The data owner is able to use a unique key for encrypting each file without an extra key management overhead by attaching the key securely to the data and authorized users can retrieve this key during the secure access enforcement.
- File confidentiality and integrity remains protected in the cloud environment against any possible inside or outside security attacks. Any potential adversary who obtains a protected file cannot read its content or reveal the identities of users authorized to access this data.
- Only authorized users can verify the integrity and authenticity of the files.
- Authorized users are able to search the encrypted file based on the encrypted keywords attached to the data. These keywords are protected even from the cloud provider.
- The proposed solution is designed to be simple and does not require complex operations.
- Data owners create the protection of their data security and privacy before it is outsourced to the cloud.
- The implementation and evaluation of the proposed solution show that it can be used practically and efficiently.

The important features of the DCS approach can be summarized in three aspects: the security requirements are provided from within the data, the data owners possess more control of the security of their data in the cloud and the security measures do not rely on the cloud providers or a trusted third party.

## 1.5 Thesis Structure

The remaining contents of the thesis are organized as following:

Chapter 2 contains background information about important aspects of cloud computing, including its definition, essential characteristics, models of cloud services and security issues. The chapter also discusses the trends and directions of different approaches in addressing cloud security issues.

Chapter 3 describes the concept of the DCS approach and how the approach can be applied to the cloud model to enhance customers' data stored in the cloud. The characteristics of the DCS approach are reviewed from available literatures. A conceptual framework of the DCS approach is developed and discussed. Desirable features and potential challenges in applying this approach to enhance the security of customers' data are outlined. In this chapter, the scope of applying the DCS approach in this study is identified and accordingly the essential security requirements are defined. Then existing security techniques that can be used in the DCS approach to provide these security requirements are reviewed.

Chapter 4 describes the proposed solution of this study in applying the DCS approach to the cloud storage of customers' outsourced data. The chapter provides background knowledge about the Chinese Remainder Theorem (CRT) algorithm used by the solution. Then it describes how the CRT is used for providing secure access control and key sharing. In addition, it describes how a data owner can update the data's access control policies when granting access to certain data for a new user or revoking the access right for a user. The secure search capability and how it is integrated with the access control procedure are also explained. The chapter describes how to create a secure data file container called a DCS file that contains the encrypted data and all the security parameters required for secure access control enforcement and secure search besides integrity and authenticity proof.

Chapter 5 describes the implementations of the proposed solution at the client and server sides. The implementation tools used and the setup of the test bed cloud environment for conducting experiments are described and illustrated in this chapter.

When the client side is the data owner, the implementation of operations required to create the DCS file is explained. These operations involve the implementations of calculating the security parameters required for access control and key sharing by using the asymmetric encryption and CRT algorithm, encrypting the keywords used for search capabilities, encrypting the data file and calculating the encrypted data digest. The implementations of the operations required when the client is an authorized user are described including reading the security parameters in the DCS file for verifying the data integrity. At the server side, the implementation for searching on the DCS files is described and tested. From experiments carried out using the implementations, data are collected to show and analyse the storage and computation overheads at the client and server sides. All Java codes used for implementation are attached in Appendix A.

Chapter 6 concludes the research reported in this thesis and gives a summary of further research work and future research directions in enhancing the security and privacy of customers' data stored in a cloud computing environment.

## Chapter 2: Cloud Computing and Security

### 2.1 Introduction

This chapter discusses the fundamental basics of cloud computing technology and issues related to security, mainly privacy, in cloud computing services. Understanding the cloud computing concept and how this concept is implemented in different service delivery and deployment models helps to identify the security issues facing this new technology. In this chapter the definition of cloud computing and its main characteristics are discussed in Section 2.2. Then the service delivery and deployment models of cloud computing services are described in Sections 2.3 and 2.5. In Section 2.5, challenges and issues in cloud computing are investigated in general and some specific examples are provided. Solution trends of cloud security issues are discussed in Section 2.6. Summary of the chapter is given in the last section.

### 2.2 Cloud Computing Definition and Characteristics

There are a number of definitions of the term cloud computing given in the literature. In fact, the term itself refers to a new technology concept that is still changing and developing. Although several definitions are proposed to reflect the essential characteristics of this new computing paradigm, each definition focuses on certain features and characteristics. Vaquero, Roderio-Merino et al. in [24] point out that a set of minimum features found in most definitions lists virtualization, pay-per-use utility model and scalability, with virtualization considered as the main technology behind cloud computing. Nevertheless, there are many other features and characteristics, such as provisioning, usability, and data and process outsourcing, that should be considered to produce a more comprehensive definition. Although the 16<sup>th</sup> version of the definition of cloud computing published by the US National Institute of Standards and Technology (NIST) is widely adopted and well-defined [4], the definition of cloud computing proposed in [24] is more comprehensive because it covers more aspects, such as usability, and the definition is as follows:

“Clouds are a large pool of easily usable and accessible virtualized resources (such as hardware, development platforms and/or services). These resources can be dynamically re-configured to adjust to a variable load

(scale), allowing also for an optimum resource utilization. This pool of resources is typically exploited by a pay-per-use model in which guarantees are offered by the Infrastructure Provider by means of customized Service-Level Agreements (SLAs)” [24].

Formulating the definition can be another approach for achieving a better understanding of the fundamental elements of cloud systems from a technical point of view. For example, the initial formulation definition proposed in [25] conceives the cloud model as equations that reflect the relation between the essential cloud components, such as datacentres, hardware machines and locations, which are used to construct the cloud.

A definition of cloud computing must recognize the fundamental characteristics that make cloud computing services valuable and distinguishable. The NIST pointed out most of the characteristics that are widely used among the cloud computing community [1-4] and these characteristics include:

- On-demand self-service. Cloud computing resources (e. g. CPU, storage, software) are provided as needed and scheduled without requiring human interactions with a service provider. The key points of this feature are time saving, cost effectiveness, usability and the range of services provided.
- Broad network access. Cloud services are accessed through a widely accessible network, mainly the internet, which uses standard protocols and mechanisms to support various types of devices and platforms e.g., smart phones, thin clients, and PDAs.
- Resource pooling. The physical cloud resources, based on virtualization technology, are shared among cloud users, depending on their consumption demands. The users are not aware of the physical limitations of resources as they are virtually provisioned and de-provisioned automatically according to demand.
- Measured service. As the service is provided under the pay-as-you-use business model, the usage of services and resources can be metered and automatically billed for each particular user session.



- Rapid elasticity. The provisioning and de-provisioning of cloud services and resources are done rapidly and elastically for each cloud user in real time.
- Location independence [3]. The cloud resources can be located physically at any geographical location as long as the communication capability is available. Also the cloud users can reach the service from anywhere with the same condition.

These characteristics are the main features behind numerous benefits provided by cloud technology. The delivering and deployment of cloud computing services can be in different types and models which are discussed in the next section.

### 2.3 Cloud Computing Service Delivery Models

The main technology on which cloud computing technology relies is virtualization [1, 6]. Virtualization provides the ability to share services and resources amongst users while the services and resources allocated to a user are virtually isolated from the other users. These services and resources start from the application level to an in-depth hardware level. This feature allows IT companies to provide cloud computing services for customers as they are needed. On the one hand, service providers build high performance platforms and look after them proficiently to provide high quality, up-to-date services. On the other hand, customers share these services according to their needs. For example, this substantially reduces the cost of building datacentres in locations physically near the customers, and also the time in implementing, updating or adding services. Cloud computing offers a highly flexible and scalable IT environment while most of the maintenance, implementation and management of the infrastructure is performed by Cloud Service Providers (CSPs). There are several types of services and resources that can be provided by CSPs. The most common services can be classified into three types [7]: Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS).

#### 2.3.1 Software as a Service (SaaS)

In this type of cloud computing, a CSP provides licensed software ready to be used by customers through a network. Customers do not own or run the software on their machines. Usually, the expense of this service depends on how many users and how long they use the service. This service is provided using multi-tenant software architecture where the software is capable of running for each client with its own

virtual data and configurations [26]. Moreover, it allows a provider to manage and price individual customers. Overall, for customers, the service enables them to reduce hardware and software resources, time and effort in managing patches and upgrades and human resources. An example of this type of service is Google Docs.

### **2.3.2 Platform as a Service (PaaS)**

In this level of service, customers have more control and management of the resources provided by a CSP at the system level. These resources include operating systems and applications such as web servers. The CSP manages resources at the hardware level. Hardware requirements, such as CPU, memory and network, will be allocated to the rented platform when they are needed by the customer [26]. In this type of service, customers use scalable and flexible platforms hosted by CSPs. In addition, customers usually access the rented platform using a web browser. Google App Engine is an example of this kind of cloud service [27].

### **2.3.3 Infrastructure as a Service (IaaS)**

In this type of service, a customer rents an infrastructure of resources which include various hardware facilities. These resources such as CPU, memory and storage space are flexible in the sense that they can be scaled up or down as required by the customer [26]. This type of service enables a customer to deploy an IT infrastructure without the expensive start-up cost required. Amazon S3 and Sun's cloud services are examples of this kind of service [28]. They both provide their services through a web service via the Internet. Some other companies provide their services for domestic customers through Virtual Private Networks (VPNs) particularly for remote clients.

Some offered real world cloud computing services may belong to different types of delivery services. For example, Microsoft Azure platform is provided as PaaS in [28], while in [26] it is presented as an example of IaaS. In terms of the common access technology offered to access cloud service, Web browsers are offered to access SaaS while Web service technology is commonly offered to access IaaS. In PaaS both access technologies are commonly used [29].

The above types of cloud computing services are the more common ones, with the definition closely corresponding to the level of resources being provided. It is

possible to provide other information communication technology (ICT) services and resources through the cloud computing concept, such as Network as a Service (NaaS), Data storage as a Service or Database as a Service (DaaS), Identity and Policy Management as a Service (IPMaaS) or almost Anything as a Service (XaaS) [28, 30]. Examples of these kinds of services include Amazon S3, Google BigTable, and Apache HBase.

## **2.4 Cloud Computing Service Deployment Models**

Cloud services can be deployed in several models which are classified based on who owns the service and who uses this service. The following is a list of typical deployment models in cloud computing:

### **2.4.1 Private Cloud**

The cloud computing services and infrastructure are owned and served by an organisation. A private cloud is used to maximize the utilization of computing resources within the organization. An organization with high security and privacy concerns may find that the private cloud model is the preferred option over other cloud models which involve sharing resources with other organizations. Furthermore, organizations with mission-critical applications may prefer to rely on their own ability to manage and control their in-house infrastructure.

### **2.4.2 Community Cloud**

The cloud resources are owned and used by several organizations that have common requirements and applications [2]. On the one hand, the community cloud adds more cost-effective value than the private cloud as the cost is shared among the organizations involved. On the other hand, the community member organizations need to have some trust relation, especially on the member organization which is responsible for the management of the community cloud. Privacy and security concerns are limited to the community member organizations.

### **2.4.3 Public Cloud**

The term cloud computing is usually used to refer to the public cloud deployment model. The cloud resources are available to the public mainly through the Internet and provided by commercial companies. The definition and characteristics discussed in Section 2.2 are more related to this delivery model which offers consumers the

most cost-effective services but also faces more challenges, particularly in terms of security and privacy. Most of the work of this thesis is about this popular model of deployment.

#### **2.4.4 Hybrid Cloud**

This model is a combination of two or more other deployment models, i.e., private, community and public clouds. By using more than one model to provide cloud services, the benefits of each model can be utilized. For example, an organization may use public cloud for part of their application that requires high computation resources but not high security requirement. For applications that require a more secure environment, a private cloud can be used to host the data and applications that require security and privacy. However, moving data between different cloud models or providers faces issues of standardization and cloud interoperability [2].

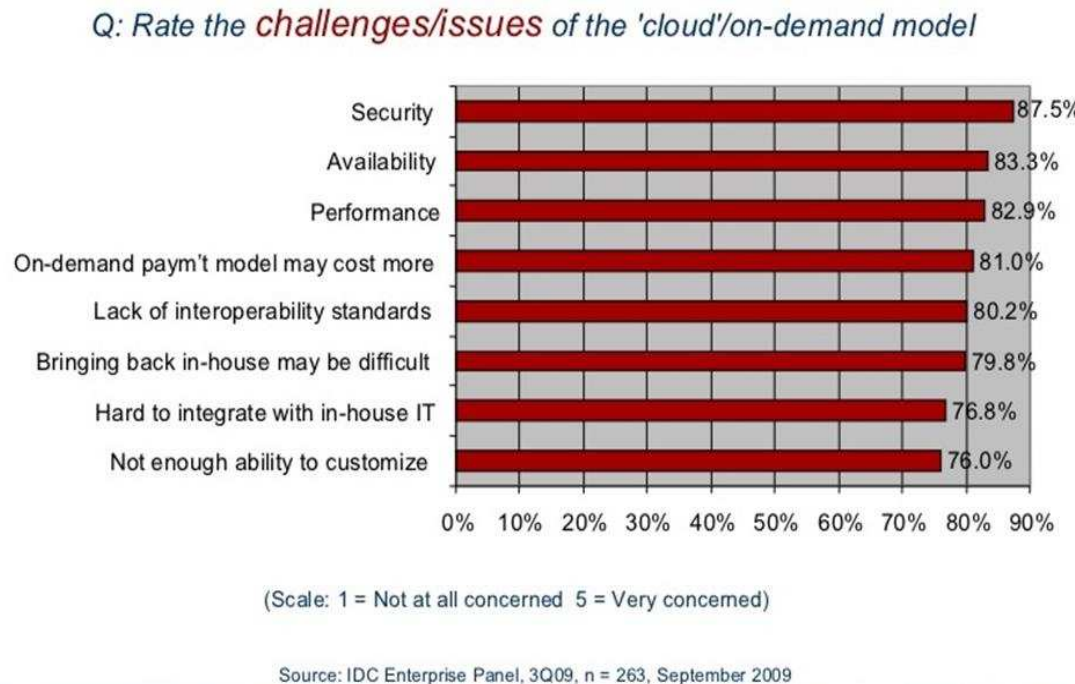
#### **2.4.5 Virtual Private Cloud**

The Virtual Private Cloud (VPC) deployment model was first used by Amazon Web Services (AWS) to provide cloud resources via a Virtual Private Network (VPN). The VPN is mainly controlled and configured by the customer [31]. The VPC terminology is not recognized yet in the NIST definition of cloud computing. The AWS offers dedicated physical hardware as an option for VPC customers but it is not an obligation of the VPC service model. The VPC can be considered as an attempt to reach a balance between the benefits of the private cloud and the public cloud that mainly represent the balance between security issues and the benefits of the business model of the cloud [2]. Although the VPC is still a new concept under development, it demonstrates a need to keep the rapid elasticity and cost-effectiveness of the public cloud services while giving the consumer of this service more control of their resource mainly to enhance the privacy and security of their data and applications in the cloud.

### **2.5 Cloud Computing: Issues and Challenges**

In cloud computing services, customers are concerned about moving their sensitive data and applications from their own private computing environments to a cloud environment which is shared by different users and which is usually accessible via a public network. A survey conducted in September 2009 by International Data

Corporation (IDC) [32] shows several issues that concern cloud computing customers, as shown in Figure 2.1. Security was ranked the highest concern.



**Figure 2.1 Issues Ranked by Cloud Computing Costumers** (adapted from [32])

Security issues in cloud computing are generally related to the core technology components on which cloud computing relies. These components are [1]:

- **Web Applications and Services** are the most used technologies to access cloud computing services.
- **Virtualization** is the main technology behind the existence of cloud computing. Both SaaS and PaaS are based on virtualization of the infrastructure provided at IaaS level.
- **Cryptography** techniques are currently the most common techniques to achieve a satisfactory level of security requirements for cloud computing.

Hence, any known vulnerabilities of the above three core technology components can be considered as vulnerabilities of cloud computing systems. For example, HTTP protocol used in web technologies is exposed to session-riding and session-hijacking. Therefore, cloud computing systems are vulnerable to this kind of attack and need to overcome this weakness. Virtualization is another vulnerable concern.

An attacker can break through the virtual separation and access data and resources either passively by observing data or actively by changing data and configurations [1]. In addition, there are various other possible vulnerabilities which can be present in cloud computing systems and they are related to its infrastructure and environment. Since cloud services are usually provided via the Internet, all expected problems related to the Internet are also related to cloud computing. Vulnerabilities in operating systems and other software programs installed in a cloud infrastructure can also be seen as being related to cloud computing vulnerabilities.

This section reviews several specific security vulnerabilities and issues due to the nature of cloud computing:

**Unauthorized access to the management interface [1]:** In cloud computing, management interfaces are typically accessible through public networks for authorized users and possibly unauthorized attackers, while conventional datacentres are usually only accessible by authorized administrators directly or via private networks. Moreover, management access is usually conducted via a Web application or service technologies, consequently the cloud management interface is probably subjected to the vulnerabilities of these technologies.

**Data recovery problem [8]:** Due to the nature of virtualization and sharing of cloud services at the hardware level, memory and storage areas that have been rented by previous customers can be reallocated to new customers. It is possible that those new customers can recover data from these memory and storage areas which may contains sensitive information belonging to previous customers.

**Virtual machine (VM) template image vulnerability [1, 33]:** A new VM is usually created by cloning a template image of a preconfigured VM as this way saves time and effort. Thus many customers will rent VMs with the same configurations. An attacker can gather information about cloud system template images by becoming a customer of a cloud with administration rights. Once the attacker has access to the template images, he/she can search for vulnerabilities in those images which are also used by other customers.

**Data leakage possibility [1]:** Another vulnerability issue related to the VM template images is that cloud providers can use templates created by other customers for new

customers. These templates may contain secret backdoors created by an attacker pretending to be a client and allow the attacker to gain access to other customers' virtual machines.

**Injection vulnerabilities [1]:** Since most cloud services use web application services, it is possible to input malicious codes into a cloud system by making use of the vulnerabilities in such web application services to hack into the web servers housing these services. There are many examples of hacking methods making use of malicious codes, such as SQL codes, OS command or JavaScript codes. Once a web server is compromised, it can be used as a stepping stone by the attacker to hack into the other targets in the system and these targets include databases and operating systems.

**Security metric and monitoring challenges [1]:** Customers are required to be able to measure and monitor the security situation of their cloud services and resources. However, providing such capabilities to cloud customers is still a challenge because the available traditional standard tools are not suitable yet for the cloud environment [1]. Because the cloud environment has complex and dynamic hierarchical services that may involve different cloud providers, the cloud environment requires novel distributed monitoring capability to fit that nature [34].

**Difficulty in digital key-management and random numbers [35]:** In a cloud system, there are different types of keys and random numbers that are necessary for cryptography operations. Managing and storing different keys in a cloud environment are difficult tasks because there is no completely physical isolation between storage resources allocated to different customers. Efficiency in the generation of random numbers mostly depends on the hardware clock used by the random number generator. A lack of such efficiency can be experienced in a cloud environment where, in different sessions, a number of cloud users use the same generation resources at the same time. This may overload the random number generation resources or might result in weak number generation. Therefore, implementing standard security mechanisms, such as a hardware security module which relies on an efficient random number generation resource, to cloud systems is a security challenge [1].

**Cloud interoperability issue [2]:** This issue is about how different cloud vendors allow data owners to seamlessly move their data from one vendor to another or from a cloud provider back to their local resources whenever they need. Without interoperability amongst cloud providers, a data owner may lock in a certain vendor and cannot easily move to other vendors or optimize the services among different providers.

**Observing activity patterns [33]:** Activity patterns of one cloud user can be observed either by other users on the same cloud or by the cloud provider. This observation can be a step for a security attack or can be used to reveal business activities that cannot be exposed in normal circumstances. For example, sharing information between two companies can be an indication of planning to merge.

**The need of mutual audit-ability, accountability and trustworthiness [33]:** Trust needs to be built between providers and customers in a cloud environment. Transparency via a high level of audibility and accountability is essential to build a trust relation. The trust relation will be more complicated when cloud providers delegate some of the cloud services to subcontracts. Cloud computing consumers should know if there are any subcontracts that can also be responsible for their data and applications behind the cloud [8]. For example, Linkup had provided online storage service via another subcontracted provider called Nirvanix before it shut down as a result of losing a remarkable amount of customers' data. Probably a subcontractor was responsible for the loss of the customers' data [20].

## 2.6 Trends and Directions of Solutions

Cloud computing technology faces different challenges that cannot be addressed directly by traditional solutions. A suitable solution should adapt to the specific characteristics of this new computing paradigm. The research directions of addressing cloud computing issues are various according to which kinds of cloud problems the researchers focus on. At the virtualization level of the technology, it is claimed that issues related to the isolation between VMs in the same physical machine need more attention in terms of security and performance [33]. For a higher level, concerns about the complexities of multi-party trust and the requirement for mutual audit-ability in cloud computing can be the new essential challenges [33, 36]. In the same direction, paper [37] claims that accountability and audit-ability are



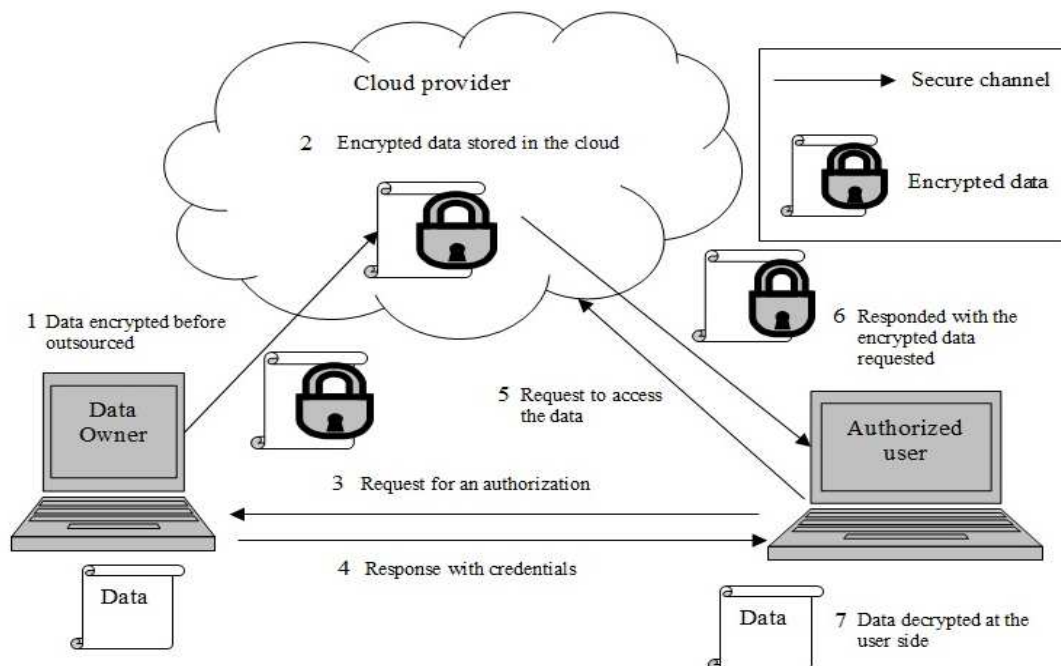
essential factors to improve trust relations in a cloud computing environment and to reduce several high concern threats. Paper [11] claims that the major issues of cloud security centre around cloud infrastructure, software platform and user data, in addition to related issues of access control, identity management and data integrity without neglecting the physical security of datacentre and processes. However, it can be claimed that the privacy and integrity of data are the essential security requirement particularly in un-trusted clouds. Also in trusted or partially trusted clouds, a strong privacy mechanism can be the key for establishing trustworthiness. An un-trusted cloud server can expose customers' private data and activity patterns, and return incorrect data from computational processes to customers. It is also possible that an un-trusted provider can manipulate the legitimate way of handling users' requested operations to their advantage [38]. Therefore, protecting data, particularly its privacy and integrity, from both the cloud providers and external attackers, is expected to result in stronger cloud security architectures that encourage wider adoption of cloud services.

### **2.6.1 Protecting Data Privacy and Integrity from Cloud Providers**

Privacy and integrity are important requirements for various applications such as e-government and EHR (Electronic Health Record) [39]. Cloud computing customers are not only worried about the compromising of privacy and integrity of their data from possible attackers, but also from potential curious cloud providers. Unfortunately, security breaches counted in 2011 and listed in [21] show that big companies such as Google, EMC/ RSA, Sony, UK National Healthcare System (NHS) and Amazon EC2 all experienced security incidents. In cloud computing, customers' data are outsourced to cloud providers which can be either trusted or un-trusted. The term un-trusted may be used to indicate that the cloud providers cannot be fully trusted. For instance, un-trusted cloud providers may not alter users' data but they can passively compromise data privacy or stealthily change the protocols for their financial benefit [38]. In other words, a cloud provider server can be considered as a honest-but-curious server [40]. Hence, it is trustworthy in providing the services, in terms of data availability, enforcing basic security control requirements and processing honestly authorized queries on stored data and returning the correct results. Nevertheless, possible malicious actions from inside the cloud can be carried out from a malicious administrator or employee.

Toward the more widespread adoption of cloud services, researchers are investigating and developing novel techniques that preserve privacy and integrity of outsourced data without fully depending on the cloud providers to provide these security requirements. The solutions should provide customers more control over the protection of their data and protect the data from the providers as well [7]. As a cloud provider server hosting outsourced data may not be fully trusted, several researchers (e.g. [38, 41-44]) have proposed methods to deal with this kind of situations. In general, their proposed solutions are based on encrypting data before the data are sent to the cloud provider server.

Although encrypted data are secured from unauthorized access, the encrypted data cannot be fully useful unless they are decrypted. For example authorized users cannot search for keywords in the encrypted data, use the encrypted data as input to computation or comparison operations. Because decrypting data at the cloud may possibly expose its content to the provider servers at least, it is more secure to decrypt data only in trusted machines controlled by the user who is authorized to access these data.



**Figure 2.2 Basic Architecture for Preserving Data Privacy in the Cloud**

Figure 2.2 shows the basic architecture of encrypting data for privacy protection before sending it to the cloud. Then the data remain encrypted in the cloud and only users authorized by the data owner can get the credential for accessing the encrypted data. The encrypted data can be decrypted only after they are downloaded to an authorized user machine. In such a scenario, the privacy of the data does not depend on an implicit assumption of trust of the server or of the service level of agreement (SLA). Instead, the protection of privacy depends on the encryption techniques used to protect the data [45]. The remaining issues are how to allow the data owner and authorized users to share and search the encrypted data, and use them for some computations, according to their access rights. All these functions should be done in a secure manner without exposing any private information to unauthorized entities including cloud providers. New cryptography techniques, trusted computing schemes and information centric security approaches [11] can be the promising solutions to overcome several cloud computing security challenges.

### **2.6.2 Cryptography Techniques Fitting the Cloud Model**

New cryptography technologies are required to fit the cloud model and to increase data security with less impact on its usability. For example, searchable encryption [13] is one area where researchers develop the capability of search of encrypted data without decrypting it. Only authorized users can query and retrieve encrypted data without exposing any private information either about the data or the query which may contain information about the data [46]. Another example is encryption techniques that enable computation operations of encrypted data without decrypting them. In such techniques, the results from the computation, which usually contain information about the protected data, are also protected. An example of these techniques is Homomorphic encryption [13, 47]. There are other cryptography techniques such as Identity Based Encryption (IBE) and Attribute Based Encryption (ABE) [46] that can improve key management and access control of cloud systems. Chow, Golle et al. [20] believe that these new cryptography solutions can be the most suitable tools to address several security issues and to give the cloud clients more security control of their data in the cloud by considering how these techniques can be improved to adapt them to the practical cloud environment.

### 2.6.3 Trusted Computing (TC)

The IT community, particularly the Trusted Computing Group (TCG), is attempting to build a set of technologies, e.g., authentication, data encryption, identity and access management, password management, network access control, and disaster recovery, to provide assurance that computer systems will fulfil the desired way of operation [28]. The TCG applies the trusted computing scheme to the Trusted Multi-Tenant Infrastructure (TMI) to establish trust in an un-trusted environment such as in a public cloud computing service. The new concept aims to enable clients to assess the trustworthiness of cloud providers by applying a set of hardware and software technologies. The remote server attestation is one of these technologies that allows clients to attest hosts [11].

The TCG starts building their solutions based on establishing a standard hardware module, the Trusted Platform Module (TPM), which carries out basic cryptography functions such as the hash function and RSA. These cryptography functions are required to establish a trust state in a computing hardware [18, 48]. In other words, the TCG aims to provide standard hardware and software technologies for trusted computing including cloud computing. Therefore, several of the security solutions, particularly those proposed to secure virtual isolation between VMs and a VM from the server provider, are based on TC technologies [17-19, 48]. These technologies still face several challenges and cannot be used solely to provide a universal solution to all cloud security problems [49]. For instance, if the TPM, which is the core component of TC, is compromised, the entire solution will be affected as pointed out by Christopher Tarnovsky in 2010 [50]. In addition, the TC concept does not provide the cloud users enough control of their data in terms of privacy and security policies.

### 2.6.4 Data Centric Security (DCS)

In traditional techniques of protecting data, security is provided by the server which stores the data. The methods used to protect the data as well as management of the protected data are controlled by the administrators of the server. This kind of approach can be classified as a system-centric approach, which is not suitable for protection of clients' data in the less trusted cloud environment. A data centric approach is expected to be more effective and adaptive for cloud services [20, 41, 51]. The term data centric security indicates in general that the focus of the security

protection is around the data. This terminology can be used differently. For cloud computing, the approach of data centric security is to secure data from the inside so that the data, according to their value and classification, have their security requirements built in to the actual data to provide optimal data security at any stage of the data life time, regardless of the environment where the data are stored [23].

## 2.7 Summary

The cloud computing concept offers a new computing paradigm where physical resources can be shared by customers on the Internet on the one hand, and customers have their own computation spaces on the other hand using techniques of virtualization. The general concept of cloud computing is that the ICT services and resources are provided by CSPs through broadband networks, mainly the Internet, and customers use the resources and services as they need and pay only for what they consume. The cloud services can be delivered in different models based on the type of the delivered ICT service. The main three cloud service delivery models are: Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS). In general, there are five known deployment models of the cloud computing service, namely; private cloud, community cloud, public cloud, hybrid cloud and virtual private cloud. Nowadays, the public cloud model is the most popular commercial cloud model. On the one hand, this technology provides great benefits, such as cost-effectiveness, time saving and scalability. On the other hand, this technology faces a variety of issues and challenges where the privacy and security issues can be considered as the most challenging. Therefore, the research trend is to protect data privacy and integrity in the cloud even from the cloud providers themselves and enable cloud users to mandate more control of their data security policies in the cloud. The proposed solutions for enhancing data security in the cloud have different directions: some focus on the tools used, primarily cryptography algorithms, for improving security of data in the cloud, others focus on more comprehensive solutions combining various security techniques based on mainly two approaches; Trusted Computing (TC) and Data Centric Approach (DCS).

## **Chapter 3: Data Centric Security Approach for Cloud Computing**

### **3.1 Introduction**

In this Chapter, the Data Centric Security (DCS) approach is discussed in more detail. This approach is the fundamental one used in this thesis for enhancing cloud computing security and privacy. The Chapter starts with reviewing and classifying possible security solutions, based on the DCS concepts in the cloud computing model. Section 3.2.3, then builds on these, to form the conceptual framework for DCS implementations proposed in this research. The expected benefits of applying the DCS approach to the cloud computing environments are discussed in Section 3.3. Several of the possible emergent challenges of the application are highlighted in Section 3.4. The scope of the application of the DCS approach to the cloud computing model for this thesis is identified in Section 3.5. In Section 3.6, the essential security requirements of applying the DCS approach to this scope are further clarified. Moreover, the available technologies that can be used to achieve these requirements are reviewed in that section. Based on such reviews, in Sections 3.7, the suitability of these technologies is assessed to derive a novel solution, which is among the main contributions of this research. Lastly, the summary of this chapter is briefly presented in the last section.

### **3.2 Data-Centric Security Approach**

In cloud computing, customers' data are stored mostly in virtual storages in a cloud service provider's cloud infrastructure. In the public SaaS and DaaS models, customers only own the stored data while all the hardware and software involved in storing and processing of the data are owned by the service providers. In other models such as the public IaaS and PaaS models, software handling of the data and applications is also owned by the customers, while the hardware is not. Therefore, from the cloud customers' perspective, the most valuable asset in the cloud environment is their data, specifically data that contain sensitive information such as government, healthcare, and financial data. With the benefits offered by the use of cloud computing, customers who used to store their sensitive data in a secure private computing environment become increasingly attracted to outsourcing their sensitive data to the cloud. Like any services on the Internet, cloud services are also subject to

security attacks. While compromising the availability of a cloud service usually results in short term effects and the damage can be recovered, compromising data confidentiality and privacy of cloud consumers can lead to long term effects and any loss can be hard to be recovered. For example, when several admin account passwords of UK's National Healthcare System (NHS) were hacked in June 2011, the NHS system was shut-down by health officials to protect the patient records [21]. This shows that in such cases data privacy is more important than the normal operation of the system itself. In addition to the risk that the data face from outside the cloud in which the data are stored, the data are also at risk from inside the cloud. Internal risks can be from malicious users using the same cloud service and the mitigation of the risks relies wholly on the cloud provider and is out of the control of the data owners. From the point of view of data owners, the cloud environment is invisible and a data owner is not certain about how his/her data are protected from these security risks. For instance, according to its nature, the files can be moved across service providers, unknown to the data owners or across different countries with different data privacy legal jurisdictions. As a consequence of these issues, cloud clients experience limited control over their data and lack confidence regarding the security of the data. From another point of view, the cloud service provider has excessive control of the clients' data [7], especially its security and privacy. Consequently, data owners are concerned about the security and privacy of their data and have a desire to keep their data secure even from these cloud storage service providers. In addition, they prefer to manage the security policies of their data in a secure manner as if the data were stored in their own machines.

The traditional security concept ordinarily centres around the technologies and devices used to store and handle data. This concept can be difficult to be adapted to provide the appropriate security level required particularly for sensitive and confidential data [23]. Though cloud security and privacy have recently drawn the attention of the research community, proposed solutions have mostly focused on securing the underlying Operating Systems (OSs) and Virtual Machines (VMs) that host cloud services [9, 16, 17, 19, 52-54]. Hence, most solutions are still based on the traditional security concept and most focus on either system-centric or VM-centric security [22]. Some of these solutions are based on the Trusted Computing (TC) concept which is introduced and developed by the Trusted Computing Group

(TCG). The TCG aims to develop a set of technologies and standards, such as Trusted Platform Module (TPM), and blind processing and self-encrypting drive technologies, which can keep the clients' data and applications handled within the cloud hardware in a claimed trusted secure container which is secured even from the cloud system administrators [15, 16]. The TCG, based on their technologies, focuses on providing a set of tools which can be used to assist the cloud users to evaluate the trustworthiness of cloud providers, monitor compliance to policy, and enable transparency regarding the physical location of data in the cloud [55]. However, the cloud users have to trust the TCG technologies first in terms of assessing the trustworthiness of the cloud provider and if the TPM, which is the basic component to build this trust, gets compromised, the entire solution will be affected. In 2010, Christopher Tarnovsky claimed that he had succeeded in compromising the TPM [50]. Consequently, research and proposed solutions based on the TPM may need to be re-evaluated. Even if the TPM and other TCG tools are security assurance, the focus of these tools is not on providing users with the desired control of the security and privacy of their data. Instead, from a customer's perspective, an implementation of the TC concept allows customers to monitor or audit the operations, including access control policies, of the cloud server through trusted tools that can provide proof of compliance to the TC concept to the users [20]. A new concept has been suggested by several researchers for addressing the specific security issues of cloud computing by shifting the focus of securing customer's data to the data itself and they call it Data Centric Security (DCS) or Information Centric Security (ICS) [11, 15, 20-23]. This concept is still developing and there are different views about how to apply it to the cloud model.

### **3.2.1 Classifying Security Solutions for Cloud Computing**

In this thesis, and in terms of understanding the DCS concept, data security solutions for the cloud computing paradigm can be classified based on two criteria: the first classification is based on which level the security is provided at, and the second classification is based on who is responsible for providing the security.

On the right-hand side of Figure 3.1, the levels that security functions can be provided at, in relation to data, are illustrated. In general, solutions focusing on providing security from outside the data level are classified as system-centric security. If the

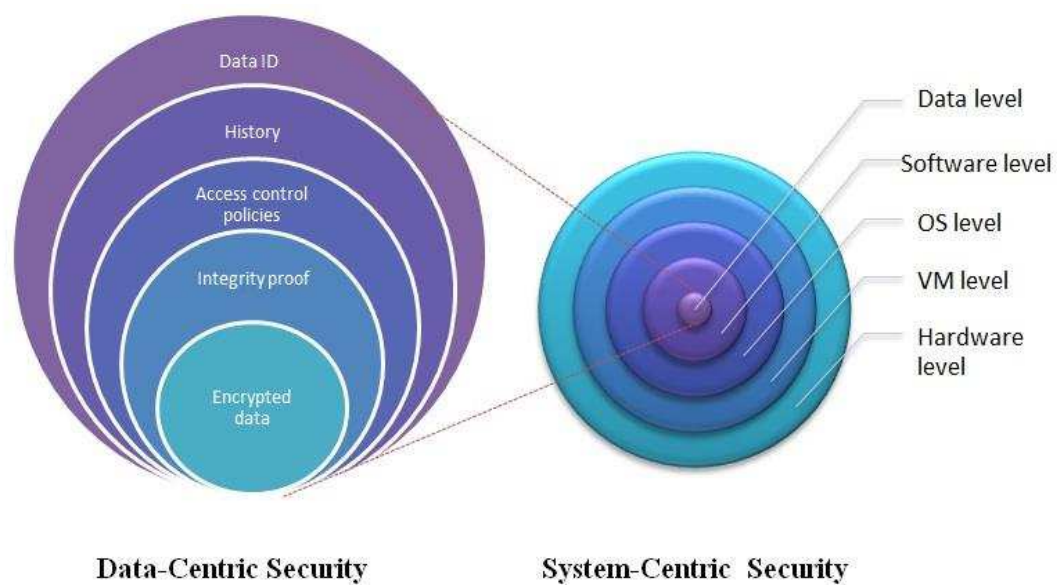


solutions focus on a particular level, they are classified according to that specific level. For example, solutions that aim to improve the security isolation between VMs at the VM (hypervisor) level can be classified as VM-centric security solutions. On the other hand, solutions aiming to provide data security from within the data itself, as shown on the left-hand side of Figure 3.1, are classified as data-centric security solutions. The levels shown in Figure 3.1 are typical levels. There can be more or fewer levels in a practical system, based on the actual requirements and implementation.

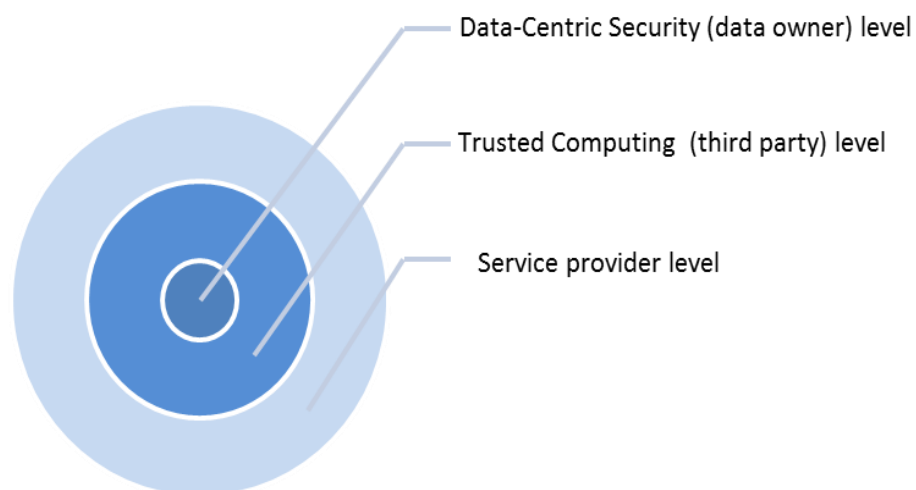
An example of the second classification is illustrated in Figure 3.2. There are three responsibility levels of security:

- Service provider level where security is provided and sponsored by the cloud provider.
- Trusted Computing level where security is provided and sponsored by a third party.
- Data-centric security level where security is provided and sponsored by the data owner.

For a comprehensive security solution, all these three security classifications should be integrated and be adapted to the cloud model. However, the dependency of one level of security on that of another level must be minimised. For example, the security functions provided at the data-centric security level should not rely on the other security levels, particularly the VM (hypervisor) and hardware levels as these two levels, in the public cloud, are controlled by the cloud provider in all the cloud delivery models. Moreover, from the responsibility perspective, the security data-centric security level must be managed only by the data owner as the data in the cloud are owned by the data owner regardless of which cloud model is hosting them.



**Figure 3.1 Data Centric and System Centric Security**



**Figure 3.2 Possible Security Levels**

In terms of investigating the diversity view of understanding the DCS (or ICS) concept, several researchers' description and understanding of the concept are listed in Table 3.1. The list also shows how the concept has been evolved over time by the IT research community.

**Table 3.1 Heterogeneous Descriptions of Data-Centric Security**

<b>Ref.</b>	<b>Security level</b>	<b>Descriptions</b>
(Bilger, O'Connor et al. 2006)	Outside data [56]	“The IBM data-centric security model (DCSM): The focus in the DCSM is on deriving the right security level, based on a business analysis of the data being handled. This data classification then drives the properties and access control policies governing the use of data by applications that implement business processes. Security services and their underlying mechanisms can be abstracted into interfaces that directly support data-handling policies.” [56].
(Jennifer 2009)	Outside data [57]	“Data-centric technology is a meaningful buzzword that, if it evolves according to its current vision, could provide technology building blocks that would allow security professionals to label data and restrict data access to authorised use cases.” [57]  “Data-centric security starts with a hard look at what data the business must protect and why, or an exercise in information classification.” [57]
(Ransom and Werner 2009)	Inside data [23]	“The focus of data-centric security is to provide at all times the fitting security level for each data set, according to its value, allowing for continuous optimal data security, regardless where the data is communicated, stored, or used. For this purpose, each data set has its security requirements attached to the data.” [23]
(Chow,	Inside	“We propose shifting from protecting data from the

Golle et al. 2009)	data [20]	outside (system and applications which use the data) to protecting data from within. We call this approach of data and information protecting itself information-centric. This self-protection requires intelligence be put in the data itself. Data needs to be self-describing and defending, regardless of its environment. Data needs to be encrypted and packaged with a usage policy. When accessed, data should consult its policy and attempt to re-create a secure environment using virtualization and reveal itself only if the environment is verified as trustworthy (using Trusted Computing). Information-centric security is a natural extension of the trend toward finer, stronger, and more usable data protection.” [20]
(Zhou, Sherr et al. 2010)	Outside data [22]	“This paper argues that a comprehensive solution has to go beyond OS and VM-centric security solutions and, in particular, must provide mechanisms for securely sharing, verifying, and tracing data as they flow between cloud users.” [22]
(Ko, Kirchberg et al. 2011)	Outside data [21]	“By data-centric, we refer to the need to trace data and files from the time they are created to the time they are destroyed; thus, data and information is viewed independent from the environmental constraints. This is reflective of the very elastic nature of cloud computing. With the transfer of control of data into the cloud, the providers have the mandate to ease the minds of consumers by facilitating them with the capabilities to track their assets.” [21]
(Sengupta, Kaulgud et al. 2011)	Inside data [11]	“Information centric security (ICS): As information in the public cloud is stored outside of organizational boundaries, we need to insert context specific access metadata in the information itself.” [11]
(Chen	Outside	“We define data-centric security as ‘the enabling and

2012)	data [58]	<p>enforcement of data-specific security policies’. In general there are three aspects related to data-centric security, i.e., primary authorization, secondary dissemination and lifetime policy enforcement:</p> <ul style="list-style-type: none"> <li>• Only authorized users and programs get access to the data (which we call primary authorization).</li> <li>• Authorized users are not allowed to violate the data’s confidentiality policy, e.g., sending the data to unauthorized recipients (which we call illegal secondary dissemination by authorized recipients).</li> <li>• The data’s confidentiality policy is enforced throughout the lifetime of the sensitive data, irrespective of the user or application accessing it, across a distributed computing environment.” [58]</li> </ul>
(Agrawal, Abbadi et al. 2012)	Both [15]	<p>“We identify the desirable features for ensuring a secure and privacy-preserving data service in the cloud. We focus on the critical features of ensuring data confidentiality and access privacy, as they usually conflict with the normal functioning and performance of data services in the cloud, and thus give rise to numerous research challenges.” [15]</p>
(Chen and Hoang 2012)	Inside data [59]	<p>“The active data-centric framework facilitates data to defend external attack even though it is located in an untrusted environment. Comparing to computing-centric or application-centric data protection models, some verification and data operation functions in data-centric framework are executed by data rather than a third party service provider.” [59]</p>

All the references listed in Table 3.1 agree on focusing on data when designing the reasonable level of security required. However, the researchers have different views about how to provide the required level. Is it provided from outside the data (the

application, hardware, etc., as shown in the right hand side of Figure 3.1 during its lifecycle or from inside the data (as shown in the left hand of the same Figure)? The first two references on the list are focused on providing the security from outside data as the DCS concept is applied to business process with traditional computing systems. Their goal is that if the system has multiple levels of security policies and mechanisms that are designed independently, then the system will be able to apply the proper security level to data according to its classification [56].

When applying the DCS concept to a cloud computing system, the issue of securing the data from the cloud system itself is involved. Consequently, on the one hand, in some studies [21, 22, 56-58] the DCS concept is applied to the cloud while still focusing on providing the proper security level from the outside of the data and they attempt to protect their security mechanisms and secure the data from the cloud system that hosts the data based on the TC concept. On the other hand, in other studies [11, 20, 23, 59] the DCS concept for cloud computing is based on providing the security features from within the data, i.e., inside data, so it becomes self-describing, self-protecting and self-defending. However, in [20], the DCS depends on TC technologies for assessing the environment trustworthiness hence the data security requires an outside data security measure. In [15] the data centric view of securing and preserving privacy of data in the cloud is presented by providing data confidentiality and privacy access to the data using either the TC approach or the DCS approach where data are self-protected. In terms of security responsibility, most of the data centric views of security listed in Table 3.1 consider that the data owner is responsible for evaluating the security requirements of the data and securing the data accordingly before the protected data is sent to the cloud. However, after the data have moved to the cloud the researchers are not agreed about what is responsible, i.e., the data owner, the cloud provider or the third party, for maintaining and managing data security under the data centric security concept.

In this thesis, the DCS approach is defined as one which is based on providing security requirements from within the data hence the data protection and all the information required for the protection are attached to data. Moreover, the security requirements are the data owners' responsibility. This DCS approach will meet the classification criteria of the DCS concept in two dimensions: at which level the security is provided and who is responsible for it (see Figure 3.1 and Figure 3.2). The

next section describes the major security features of the DCS approach defined in this thesis at the conceptual level.

### **3.2.2 DCS Characteristics**

In this section, the DCS approach is discussed in more detail. Several of its characteristics are highlighted and discussed to produce a clearer view about the DCS conceptual requirements. All these characteristics are related to privacy and security issues in protecting clients' data in cloud computing. Traditionally, security basically refers to addressing three main concerns: confidentiality, integrity and availability. Commonly, CIA is used as an acronym for these three important security requirements. In terms of data security, the CIA requirements are on data confidentiality, data integrity and data availability. In cloud computing or any computing system, the CIA requirements are also applied to the system with regard to system confidentiality, system integrity and system availability.

#### **3.2.2.1 Data classification**

A data owner is normally the best entity to evaluate the security requirements of his or her own data. For example, if the data are connected to a business model, the security requirements of the data should result from analysing their use in the business process [56]. Typically, the data should be treated according to their value as the following security principle states: "the value of what is being protected affects the measures taken to protect it" [23]. Hence, the data are classified based on the data owner evaluation and security requirements. For instance, the data can be simply classified as top secret, secret or confidential. Based on that, the access control policies and security properties required to handle the data are provided according to those security requirements. The data classification at a conceptual level can be presented as information attached to data or presented on the required security measures applied to data.

#### **3.2.2.2 Data set**

Within a computing environment, data are created in different forms that can be separated in the physical and application levels. In terms of storage and transmission, the data set can be a block, a file or a directory containing files [21]. In terms of security, the data set is the collection of information that can be protected and securely treated as independent of the data. In general, data in cloud computing can

be classified into three categories: unstructured, structured and semi-structured data [60]. Unstructured data do not have a pre-defined data model and/or do not present as relational tables, such as document, image or video files. In contrast, structured data are represented in a strict format. An example of structured data is a database which stores the data in a table format where each row is called record and each column is called a field. Semi-structured data are a form of structured data that has a hierarchical construction of the content information presented in records and fields and separated by markers usually called tags [61]. Typically, XML, HTML and email are considered examples of semi-structured data [61]. Therefore, the data set may be used to refer to any of the above forms of data. On the contrary, the term resource has been used instead of data set, especially in articles dealing with data access control, because the resource term includes the meaning of the data set and any other physical or virtual component of the computer system [62]. In a practical design of the DCS approach, it is essential to specify which data set format the approach will be used for: whether it is a file of structured or semi-structured data or a field or record in a database or semi-structured data.

### **3.2.2.3 Data lifecycle**

In the cloud computing environment, each data set has several process stages starting from creation to the time the data set becomes useless and must be destroyed. The lifecycle, sometimes called lifetime, refers to the entire stages of any data set processing which typically goes through the following stages:

1. **Data generation/creation:** this is the first stage of the data lifecycle where the information can be represented in different digital data set forms in the computing environment. In the traditional computing environment, usually the data are generated and managed by users or an organization within their IT infrastructure owned by them. In cloud computing, the data can be generated at the owner side before they are sent to the cloud or they can be generated in the cloud. In both cases, how to maintain the data ownership within the cloud environment is highly controversial [63]. The term data owner or simply owner, in this thesis, refers to the entity, either a person or an organization, which legally owns the data and mostly generates them and then uses the cloud resources to handle these data. Privacy and security issues must be considered from this first stage when designing a DCS



solution. For example, data classification, described in Section 3.2.2.1, should be involved at this stage as security measures to be used are established accordingly.

2. **Data transmission/transfer:** at this stage the data may be moved from one place to another within the cloud or the data owner domain or between them. The data transferred within the data owner domain have less privacy and security concerns. In the cloud computing environment, the data can be transferred within VMs between different hardware components, e.g. CPU, memory and storage, that are shared with other users because of the multitenant architecture used in cloud computing. Moreover, the data may be also moved between the cloud data centres or nodes within the same or different geographical locations. Usually, the data are transferred from the user to the cloud or vice versa through a public communication network mainly the Internet, which may also be used if the data transmission across different cloud providers is required. For all data transmission cases, data confidentiality and integrity should be ensured in order to prevent data from being tapped and tampered with by unauthorized users. In particular, when data are transmitted through a public network, the risk of compromising their confidentiality and integrity becomes greater. In addition, during data transmission, private information can be exposed by unauthorized entities. Therefore, it should be ensured that the transmission process is established in a secure manner without revealing any private information. Using the DCS approach, since the data owner applied security measures to the data before the data were sent to cloud, the data are secured in any subsequent transmission between the data owner and the cloud provider, or between VMs or various cloud computing nodes.
3. **Data store/archive/backup:** the data are stored in a hardware storage media in the cloud; usually the hardware storage is shared by cloud users and their data are isolated virtually. This nature of the cloud storage leads to several security issues some of which have been discussed in Chapter 2. Cloud providers, particularly storage service providers, should protect data availability by backup and archive strategic plans. This may involve

replicating data or transferring copies of the data to off-site storage. On the other side, data owners need to be certain about how their data are protected during these operations. For example, data owners need to know how many copies of each data set the cloud maintains and their physical locations. Furthermore, the data owners need to be certain when a data set gets changed or deleted, copies of the data cannot be accessed any more in the cloud. Preserving data confidentiality and integrity, at this stage, requires the data to be archived without restricting the cloud provider from conducting the required backup or archive strategies.

4. **Data use/process:** the data are in use or process when computing actions are dealing with the contents of the data and involve the CPU. These actions may require reading, modifying or producing new data from arithmetic or logic operations on the old data. Static data simply outsourced to the cloud for storage purpose only, still require several operations to facilitate the storage service, such as indexing, searching on data, compressing and decompressing data. Traditionally, these operations cannot be conducted on encrypted data; hence the data must be un-encrypted first and that will reveal the data contents at least to a cloud server. In terms of confidentiality, the situation is worse when data are outsourced for cloud services that require data processing, such as, arithmetic and logic operations. The main challenge is how to retain data confidentiality during this stage with less impact on its usage efficiency.
5. **Data sharing:** this is when the data owner extends the data usage by authorizing other parties to use this data. These parties become authorized users who can use the data according to the access control policies rendered by the data owner. In terms of privacy, the sharing mechanism has to maintain data and in some applications, authorized users' privacy may also need to be protected. Furthermore, the original security measures of the data should be maintained in the stage, e.g., after the data are retrieved by authorized users [63], the data are protected during the whole process.

6. **Data deletion/destroy:** the data are permanently deleted from the cloud, either because the data owner does not need the data to be stored in the cloud or the data owner is going to store the data with another cloud provider. In both cases, the data owner's data in the cloud service provider needs to be deleted and all copies of the data, e.g., backup copies, also need to be removed. If these data still contain sensitive information, the data owner has to be certain that even if the deleted data is restored, the sensitive information cannot be disclosed.

#### ***3.2.2.4 Data history/tracing/track***

The data owner requires transparency about his or her data lifecycle in the cloud. In order to provide such a feature, each data set composes its lifecycle history along with the time and represents it as a set of metadata information [23]. The history must cover the entire lifecycle of the data set, including its generation, every transmission and different usage actions or modifications until its lifetime is expired. In addition, this history information is useful for allowing a data owner to trace and track their data in the cloud at any time. Depending on the update history information, the data owner may re-evaluate each data set and react based on this kind of information [23]. The actual data are also able to send a notification of important usage actions regardless of the data location. This will improve auditability and accountability of data usage in the cloud [59]. The data history metadata is also essential for data provenance which in turn enables debugging and unusual behaviour detection capability in the cloud [22]. Enabling all these features efficiently and effectively in the cloud environment is highly challenging [21].

#### ***3.2.2.5 Data access policies and their enforcement***

The access control policy is the core security feature where the security requirements are specified for each data set according to security policies which may include any potential legal obligations [23, 56]. These policies include rules and restrictions that control the access, usage, and flow of data [23]. The essential concept is about controlling who or what can access the data and with what granted rights, e.g. read, write, that comply with the traditional concept of access control policies [23]. The concern is not only about defining these policies but also about enforcing these policies in a secure manner that preserves users' privacy when they access or share the data [22]. In the DCS approach, the access control policies and the mechanism of

their enforcement are specified by the data owner and the cloud system is responsible to comply with the enforcement mechanism. The challenge is how the access control policies and their enforcement can be extensible and easily modifiable to adapt to the dynamic changes of data usage and sharing requirements in a cloud computing environment [22]. As each data set has its specific access policies, other complex issues arise about the interoperability of different access control policies and the ownership of a new data set which is produced in the cloud from processing several data sets with different policies and data owners [23].

#### **3.2.2.6 Self-protection**

The data set contents privacy is protected against any unauthorized entities even the cloud provider. To self-protect the privacy of the data, they are encrypted by a sufficient encryption technique and they remain encrypted until they are transferred to a fully trusted domain and decrypted by an authorized user. Only authorized users can have access to the secret key which is attached to the data set and is also protected. The encrypted data are packaged with their security parameters, such as the protected secret key and the data's usage policy. Therefore, each self-protecting data set has its access policies embedded and these policies are consulted when an authorized user requests access to the protected data. By the use of self-protecting where the access policies are embedded in the data, access to the protected data can be made in any location as long as a mechanism exists to execute the embedded access policies. The access control mechanism will be rudimentary in its implementation at a location. Only the data owner can manage the access control policies and other parameters and functions related to the self-protecting characteristic of each data set.

#### **3.2.2.7 Data integrity verification**

It is very important that the integrity of a data set can be verified at any stage. When stored in a public domain, a data set is subjected to modification by an unauthorized entity, which can even be the cloud provider itself, accidentally or intentionally during its lifecycle. Therefore, data owners and authorized users require this feature to assure that the data have not been inappropriately modified. In the DCS approach, information for the verification of the integrity of the data is embedded in the data and is also protected. The verification does not depend on information provided from outside the data itself. Consequently, directly verifying the integrity of data in the

cloud without requiring downloading it is a great challenge [63], especially, if the data are dynamically changed in the cloud.

#### ***3.2.2.8 Privacy and Confidentiality***

Confidentiality means that only authorized parties have the ability to access protected data with the appropriate rights and privileges defined by the data owner [64]. The risk of compromising data confidentiality increases in the cloud model because of the increased number of parties involved including other users in the same cloud. Furthermore, the control of data is delegated to the cloud provider and the cloud lacks hardware separation between users. These lead to an increase in the risk of confidentiality compromise, as the data become out of the data owner control and accessible from other parties [65].

Privacy has various definitions in the literature. Privacy can be used as a synonym for confidentiality, while some researchers consider privacy as a different concept [66]. The concept of privacy is viewed differently around the world and that is reflected in different privacy laws. Privacy seems a wider concept than confidentiality because it covers what, based on law or culture, is considered to be private information related to organizations and peoples even if this information itself is not confidential. This wider concept can be found in one of the privacy definitions provided by the American Institute of Certified Public Accountants and the Canadian Institute of Chartered Accountants in the Generally Accepted Privacy Principles standard which is “The rights and obligations of individuals and organizations with respect to the collection, use, retention, and disclosure of personal information” [63]. For example, the name or location of an ordinary person is not usually considered to be confidential, but may be considered private in some circumstances. Therefore, the protection of privacy goes further than protecting the actual data from unauthorized access. It also prevents any leakage of any information which is considered as private information during handling of the data. For example, the attached access control policies are considered to be private information and the cloud provider should not be able to disclose them [66]. Applying this kind of protection to the access control process results in a privacy access approach that can be used to prevent leakage of even partial information about data during an access process [15]. Therefore, any security solution must take into account addressing the

confidentiality of data and the privacy of cloud consumers according to the different regulations and requirements.

### **3.2.2.9 Availability**

In cloud computing, availability refers to cloud computing services being available and accessible for authorized users whenever they require them. In addition, these services may be related to the processing of critical data and the running of important applications that must be available all the time and capable of serving a large number of users [67]. The availability of cloud computing services depends on the continuous healthy operation of infrastructures and network resources all the time [65]. Therefore, a cloud provider should have robust and redundant strategies to maintain availability of the system [67]. A cloud provider is also responsible for carrying on its services even if there is an internal or external threat targeting the system's availability. The other availability issue is availability of a client's data for access by the data owner or by authorized users. An authorized user should be able to obtain data from the cloud at any time. As mentioned before in data lifecycle stages, a cloud provider should have an effective backup and archive strategy to protect data availability. From a data owner's perspective, the data are the most important asset they own in the cloud. The proposed DCS approach in this thesis gives a data owner a more substantial amount of responsibility and control in protecting their data which are to be stored in a cloud computing environment.

### **3.2.3 DCS Conceptual Framework**

A holistic DSC conceptual framework is established in this section. This framework is based on reviewing the various previous research works on the DCS concepts applied to the cloud model, listed in Table 3.1, and based on the DCS characteristics previously defined in Section 3.2.2. In the framework based on the concept, the data are secured before leaving the trusted domain of the data owner and their security parameters are attached to the data as a part of their metadata. In the ideal situation, only the credentials used to undo or access through the protection are kept outside the data and by the data owner and authorized users according to their access rights. Any other data security parameters required must be attached to the data. For example, in cases where the data are dynamically updated, a user who accesses the data needs to verify that the data up-to-dated [64]. This requirement should be provided from the actual data or from the metadata attached to the data. For instance,

the history feature is sub-level from the data level, as shown in Figure 3.1. As another example, proof of the authenticity of the data origin can also be added together with integrity proof by typically including a data owner's digital signature. Any extra feature can be added as a new sub-level under the data level or included as one of the sample sub-levels illustrated in the left-hand side of Figure 3.1. In addition, all these security metadata are provided at the data level and are created by the data owner. They also remain protected throughout the data lifecycle. An individual data set has its security metadata independent from other data sets. The following list highlights the criteria that any provided security solution based on the DCS approach must fulfil:

1. Each data set is self-describing, self-protecting and self-defending (i.e. secure data set). Therefore, each data set's security requirements and features are provided from within it and do not depend on facilities outside the data set, apart from some basic data handling processes.
2. The data protection does not rely on the cloud provider or Trusted Third Party (TTP).
3. Only the data owner is responsible for creating and managing these security requirements and features for each data set from the time of creating it until the end of the data set's lifecycle.
4. All operations related to access to the protected data set by authorized users and enforcement of the security policies on the data are performed without compromising the users' privacy or data confidentiality.

A sample of the DCS conceptual framework for the cloud computing model is illustrated in Figure 3.3. The creation of data is done in the trusted domain at the data owner side. At the creation stage the data are classified based on the security requirements and accordingly the security metadata are attached to the data, such as, their access control policy, integrity proof, the record of their history and tracing capability. In the creation stage, a secure data set is produced and it encapsulates the protected data contents and the corresponding metadata. The data management and tracing controller allows the data owner to manage and trace the data set, e.g., locations, usages and access history, from the time of creation until it is removed from the cloud environment. In addition, transmissions between the trusted domain where the user is and the cloud domain are normally via the Internet and are

protected by the underlying protection mechanism provided by the DCS approach or by an Internet security protocol. The secure data set during its lifecycle in the cloud has the ability, by executing codes attached to it, to inform the data owner if there are any changes of its condition. In the meantime, the secure data set includes an executable code ready to receive and respond to the data owner requests for tracing information and management commands. Even if the communication between the cloud domain and the trust domain is down, since the data set contains all the necessary information to execute the underlying security policies related to the data, the secure data set can be continually accessed by authorized users without the involvement of the data owner.

As shown in Figure 3.3, the enforcement of the access policies will be carried out by the cloud server resources in the cloud domain but based on the security parameters attached to the secure data set. The cloud provider rule in the enforcement is to allow the execution of these access policies without revealing the access control policies details or data set content. The data owner can verify the integrity of the secure data set content including its history record.

When the secure data set is to be retrieved by an authorized user, the cloud server in the cloud domain will verify the users' access rights to the data using the related attached security parameters. Only authorized users are allowed to download the secure data set. Then the authorized users are able to verify the integrity and originality of the downloaded data by using the attached integrity and authenticity proofs. This conceptual DCS framework allows on the one hand to optimize the security configuration of the security parameters for each data set, and on the other hand, it reduces the complexity of the overall security management. This is particularly so when cloud security responsibility is also optimally distributed among the cloud system potential parties (i.e. cloud providers, users and TTPs) according to the three levels of security responsibilities illustrated in

Figure 3.2. For example, the cloud providers are responsible to secure their systems, the TTPs responsible to asset the security measures provided by the providers and the data owners are responsible to secure their data in the cloud. Other expected benefits of applying the DCS to the cloud model are discussed in the next section.



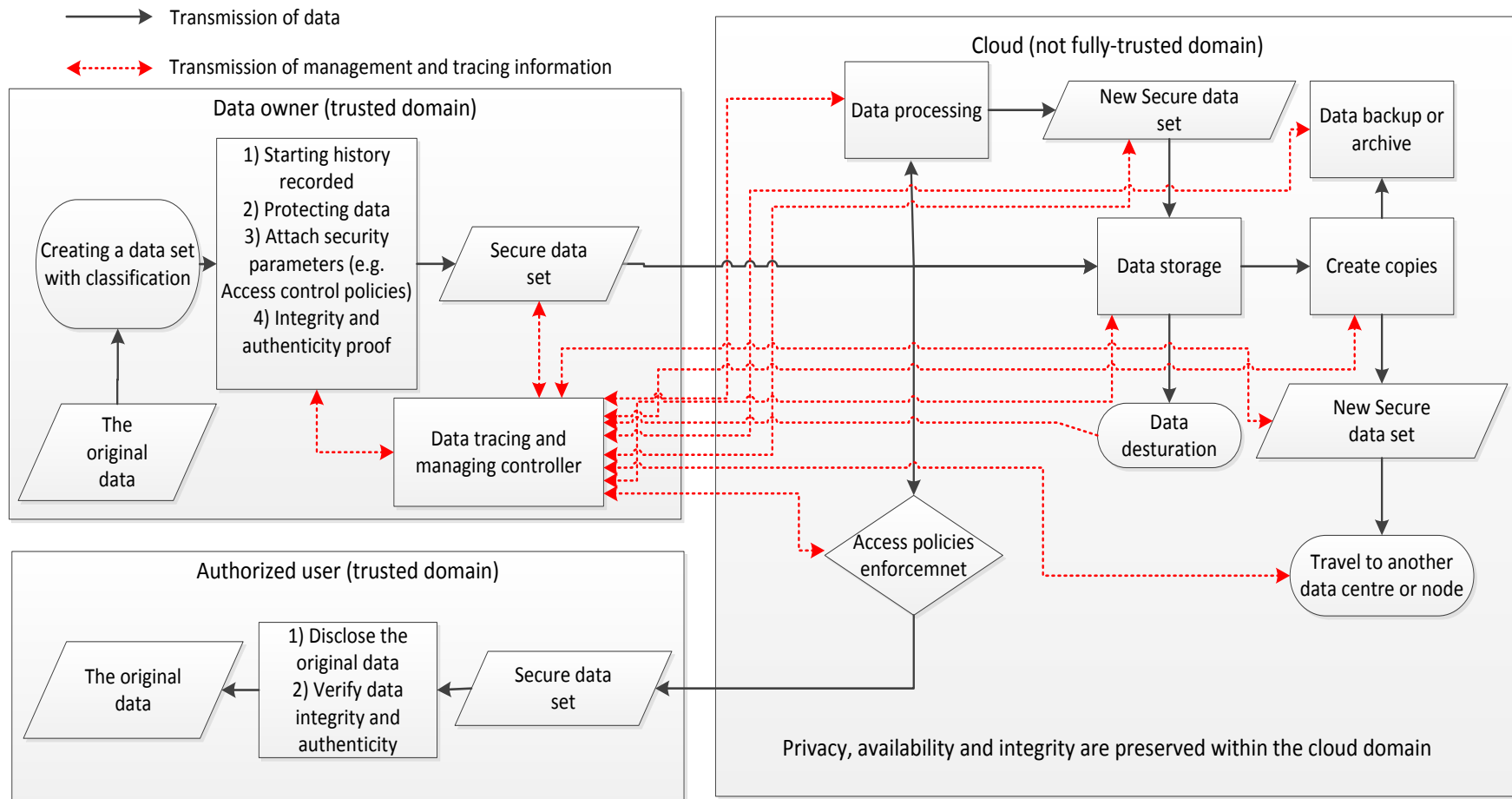


Figure 3.3 DCS Framework

### 3.3 DCS Expected Benefits

This section will list and discuss the expected benefits of using solutions having characteristics of the DCS approach in a cloud computing environment. The DCS approach goes beyond the security measures provided by cloud servers at the underlying operating system and virtualization platform. A comprehensive security solution should allow cloud clients to securely store, share, search and control their data within the cloud environment [22]. This approach is expected to achieve these benefits:

- Data confidentiality is preserved. The data are self-protected and the protection cannot be undone in the cloud or anywhere else by any authorized entity. For example, by keeping the data encrypted all the time at the cloud, even the cloud provider cannot decrypt. The data confidentiality is protected even if there are security breaches in the cloud. Outsourcing encrypted data to the cloud can be considered more secure even than keeping it unencrypted in-house [67]. Hence, the information remains secure whether the cloud server gets compromised either from inside or outside
- The complexity of the security management is reduced. The security parameters of each data set can be optimally specified independently from other data sets' or external security facilities [23]. For instance, changing the security requirement for a certain data set is simply achieved by changing the security parameters attached to the data set directly.
- Accountability and auditability are provided from program codes attached to the data itself. Access logs are attached to the data and the logs are protected against modifications. The logs provide information to the data owner upon request based on predefined configuration of the attached program [34].
- Transparency of the data usage and location of the data in the cloud can be provided by information for ubiquitous monitoring and tracking functionality attached to the data [59].
- Privacy of data access policies is preserved as the access to data is enforced under the secure policies hidden inside the data. Access is performed without the necessity of exposing these policies or any sensitive information to a cloud provider or TTP.

- Integrity of data is protected throughout its lifecycle. Authorized users are able to verify the integrity and authenticity for each data set individually.
- Data privacy and integrity rely on the strength of the cryptographic methods used more than on trust of the cloud service providers and do not require a TTP.
- Scalability can be measured in several aspects. Because in the DCS approach, each set of data has its own access control enforcement, the data set can be moved within the cloud environment without restriction related to access control enforcement. Furthermore, self-protected data allow the cloud to distribute information resources efficiently with less worry about trust requirements [68]. For example, in the case of read only resources, multiple copies of high demand information can be securely generated and distributed effectively among cloud nodes to enhance the read performance [68].
- The approach allows either a client or a cloud provider to improve data availability by keeping multiple copies of the data in different geographic locations, e.g. different data centre locations of the same provider or different providers [41]. Therefore, this strategy will provide a higher level of fault tolerance since data are available even if a server in one location fails [69]. The DCS approach allows this kind of strategy to be used as copies of a data set contain exactly the same self-protection information.
- Virtual machine (VM) data leakage possibility will not expose sensitive data for unauthorized entities because the data in the cloud are encrypted and can only be decrypted outside a cloud environment. Although a data set including its security parameters can be leaked, an unauthorized entity cannot access the data contents without breaking its encryption.
- The problems related to the legal jurisdictions for both providers and customers are reduced. Since data are encrypted and are not normally revealed to the public, they may not be subject to regulations imposed on Internet data in different countries. In this case, cloud providers are more flexible to optimize the use of their storage resources around the world even in countries with more stringent regulations on data contents on the Internet [12].

- The approach reduces the data owner involvement in the security process when the data are in the cloud, because the data have all the security parameters attached to them.

To design and implement a solution based on the DCS approach that is able to provide all these expected benefits, many challenges have to be overcome. In the next section, several of these challenges are pointed out and discussed.

### **3.4 Challenges of Applying DCS Approach**

There are several challenges to overcome in applying the DCS approach to the cloud model. This is predictable as the DCS approach attempts to provide a comprehensive solution to improving data privacy and security in the cloud and the design of the solution has to cover all the related data security issues. Some of these issues can be found at the design phase of the solution and others can only be discovered at the implementation phase. As discussed in Chapter 2, the encrypted data must remain encrypted in the cloud all the time. To achieve this requirement, data usability and accessibility are affected when the data remain encrypted in the cloud. Methods need to be designed to manage the attached access control policies and the keys used to encrypt data that will be accessed especially when the encrypted data will be shared among authorized users. Besides all other security requirements, such as tracking ability, these features require consumption overhead of at least one of the cloud system resources, which include cloud resources, network resources and client resources. How to apply and implement the DCS approach for the cloud practically and efficiently requires addressing several challenges. Some of these challenges have already been mentioned in Section 3.2.2. The following sub sections discuss other several possible challenges:

#### **3.4.1 Challenges Related to Secure Access Control**

Traditionally, to control the distribution of sensitive data, users store the data in a trusted server. Then, they delegate that server to become an omniscient reference monitor for defining and enforcing access control policies for their data [5]. Since the access control policies are implemented by the server, the server already knows the access rights and other parameters that an authorized user must have in order to be allowed access to the data. In a more traditional access control scheme, the server may also know all the users' Internet IDs and their data access histories. From a security point of view, this kind of access control scheme is not acceptable for data

sharing in the cloud, because it allows the cloud server to know what may be considered to be private information of the customers [46]. When the data are outsourced to the cloud, in many applications, data owners, especially large organizations, require the data to be shared among a large number of authorized users. The sharing usually includes managing different data access privileges for these authorized users. Using the DCS approach has to consider the following challenges:

- How to enforce data access without relying on cloud servers [5]?
- How to strongly couple data with their access control?
- How to keep the data encrypted and access control details hidden? The data access policy details are hidden from the cloud server. The hidden policy details include the number of users allowed to access each data set and their privileges. During access by an authorized user to the data set in the cloud domain, the server processes the access request, but during the processing, no data contents and private metadata contents are revealed to the cloud server.
- How to hide data access patterns to protected data? It is more challengeable to hide the data access patterns to the protected data from a cloud server. For example, to prevent traffic analysis of access patterns to protected data by unauthorized users, a cloud server needs to enforce access control policies without the ability or any information to infer which data has been accessed by an authorized user [70]. This also includes hiding the time of an access request and the result of the access request [71].
- How to establish content-level fine-grained access control for users [46]? This challenge raises the key management issue that typically faces sharing encrypted data among multiple users [26]. Because the data are encrypted, the access privileges for a certain data set are now related to a secret key to decrypt the data set. Therefore, if data sets have different access privileges, they need different secret keys to decrypt them, respectively. This can lead to encrypting each data set, such as a record in a database, with a different key which, in turn, can lead to the need to manage a large number of keys [46].

### 3.4.2 Challenges Related to Queries and Search on Encrypted Data

In a normal storage server in a trust domain, the server processes queries for data, searches for keywords and indexes the data based on its contents. If the data are stored in a cloud environment and are self-protecting, the data contents are not available to the cloud server because of the self-protection that the cloud server cannot undo. In this case, a cloud server cannot perform any search operation for stored data. Several of these challenges are listed below:

- The first challenge is how to enable a cloud server to search on the encrypted data basically for the data owner only and more advanced for other users without decrypting them or disclosing part of their content in the cloud domain. The different forms of data (i.e. structured, unstructured and semi structured) should be considered as the search mechanism differs for these forms when dealing with this challenge.
- How to prevent the query itself exposing any sensitive information? For instance, when an authorized user sends a request to a cloud server for searching for a certain word in a self-protected data set, this word must be hidden from the server or any other unauthorized entities. In the database, this word can be an attribute value which can be a number, string, or a value of any other data type. This includes hiding any other query details that can be used by the cloud server or an adversary to learn the user's behaviour in searching for data [36]. For example, other query details include query restrictions on date ranges or file types. It becomes more difficult to securely conduct advanced search queries. For example, searching encrypted data using conjunctive keywords or keywords with wildcard '\*', or searching for similar semantic matches of keywords if there are no exact matches [72].
- How to prevent a cloud server from gradually learning information about search results and using statistical analysis to infer the data contents [15]? A cloud server can store all the search queries and their results if they are not hidden. Then the cloud uses this information for statistical analysis to figure out what keywords or attributes are common in these results.
- How to assure query result correctness? A user receives search results from a cloud but he or she may not be certain that the cloud server returns all the correct results. The user can check that the received results match the query

but it is possible that the cloud server hides some results or has not conducted the search over all the stored data. In this case, an end-to-end verification is required to ensure query results correctness [22].

### 3.4.3 Challenges Related to Outsourcing Computation

One of the important services offered by the cloud computing paradigm is computation outsourcing. The cloud users use this literally unlimited computing resource as they require. However, as discussed before in Section 3.2.2, the problem is how to allow self-protected data to be processed in the cloud domain. One of the possible solutions is the homomorphic encryption which is still in its theoretical stage and still far from practical use [73]. To perform computations on self-protected data in the cloud domain requires several challenges to be addressed so the DCS approach can be practically applied to computation outsource. Below are several challenges related to this issue:

- How the homographic encryption scheme can be used to allow practical computation operations on encrypted data? This requires both sides, the user and the cloud server, to use this scheme with acceptable computation and time overheads, as compared to having the data in the local trust domain and the same computations performed on them [5].
- How to perform these operations without causing poor data security? In other words, the scheme should maintain a good level of balance between security requirements and useful performance [5].
- How the resulting encrypted data maintains the original security measures? For example, the resulting encrypted data after any computations have been done on them should maintain the access control policies of the original encrypted data.

### 3.4.4 Challenges Related to Overhead and Performance

As the cloud is typically a commercial service, the challenges should be addressed within the trade-off between security overheads and cloud benefits. Security overheads can happen in the trusted domain, either at the data owner or the authorized user sides, as well as the cloud domain. The overheads on these two domains are primarily measured in terms of three performance criteria; storage overhead, communication cost, and computational complexity [74]. The common challenge is how to design a security solution for the cloud model that has reasonable

overheads on these three criteria for both domains. The security solution should also include a flexible configuration capability that can meet different requirements. For example, tracing data capability should be designed in a way that the user is able to configure it to optimise consumption of the communication bandwidth and system resources [59]. Another important feature is the scalability capability of any security solution designed for the cloud model. Scalability is one of the essential characteristics of cloud services and any security solution should be able to be adapted to a larger system without a significant increase in complexity. The cloud platform has to deal with a massive amount of storage data, a huge number of users and high computation operations. Hence, any security solution must be able to scale up to cope with these requirements. Therefore, sharing of data and access control mechanisms should be able to serve a large number of users. Another scalability requirement is how users are able to deal with a huge amount of their data while their data remain encrypted, particularly, when a cloud user prefers to directly verify the integrity of their data in the cloud without downloading them [63]. The lifetime of data is another aspect which must be considered when designing a security solution. The lifetime of data can be potentially up to 100 years or more. Thus the protection of privacy must cover this possible length of the lifetime of data optionally [75]. The challenge is on how the protection maintains the desired level along this large scale of time.

The above mentioned challenges that are possibly faced when designing any security solution, particularly the ones based on the DCS approach, for commercial public cloud computing, are by no means comprehensive. Therefore, to limit the number of challenges that have to be overcome, in this thesis, the scope of applying the DCS approach has to be identified along with its essential requirements.

### **3.5 Identifying the Scope of Applying the DCS Approach**

This section focuses on identifying the scope of applying the DCS approach to the cloud model. Thereafter, the security requirements can be identified based on the identified scope. The formats and types of data to be stored by a cloud service provider are two important aspects when determining the scope the proposed DCS approach will cover. In general, the techniques used to handle data differ according to the form of the data which can be structured, semi-structured or unstructured. The techniques used to provide security for data may vary depending on the data form.



On the other hand, the security requirements can differ according to the cloud delivery and deployment model.

Nowadays, it is commonly known that most data, about 80%, are produced in unstructured form and the use of unstructured data is growing faster than that of structured data [76]. Hence, this thesis chooses unstructured data as the target for applying the DCS approach and also because the final solution can be applied to semi-structured and even structured data with modifications to fit their structure. Besides, the public cloud storage service is chosen as the application scope because it is the most popular cloud service and it covers most other clouds as far as security measures are concerned for storing and sharing this type of data. Most of the big cloud companies such as Microsoft, Google, and Dropbox, offer this cloud service to the public.

### 3.6 Essential Security Requirements for Public Cloud Storage

The essential security requirements, derived from the research questions, which are stated in Chapter 1 are discussed in more detail in this section. Then security techniques that can be used to achieve these requirements are investigated. The investigation aims to produce a security solution which preserves the privacy and integrity of unstructured data stored and accessed in public cloud storage based on the DCS approach. Therefore, the security requirements must take into account the DCS approach where the data are self-protected with built-in security policies. Accordingly, the essential security requirements are listed in the following:

- The data must be encrypted before being outsourced to the cloud and remain encrypted in the cloud. Encrypted data are never decrypted at the cloud server to prevent any unauthorized entities from exposing the data contents and these unauthorized entities may include the cloud server. The data can be decrypted only at an authorized user's side, i.e., a trusted domain.
- The data can be accessed securely by authorized users without revealing any users' credentials to the cloud provider that can be used to decrypt data.
- The access control policies have to be hidden from server providers even during access enforcement.

- The encrypted data can be searched by authorized users in the cloud domain without revealing any sensitive information to the cloud server hosting the data.
- The integrity of the data can be verified by an authorized user. This feature does not involve the cloud provider. The integrity of the data must be protected against any unauthorized violation including any possible malicious action from the cloud provider itself.
- Each data set has the entire security requirements contained in it. These requirements are not dependent on facilities outside the data set, except the execution of these requirements. There is no information or other security measures required from the cloud provider or TTP.
- The data owner is responsible to set and manage all these security requirements for each data set throughout its lifecycle. The data owner is able to update the access control policies and other security parameters of each data set without revealing sensitive information to unauthorized entities including the cloud server.
- The security requirements should be achieved with reasonable overhead for both the users and the cloud providers.

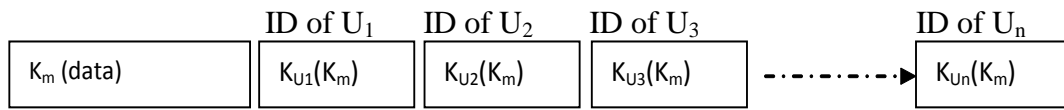
To meet these essential requirements, security techniques in three main areas of research are to be investigated: privacy preserving techniques for shared data and access control, secure searching on encrypted data, and verification of data integrity. Techniques must be suitable for protecting data which are stored in a cloud server and the outsourced server is not fully trusted, i.e., honest-but-curious. Techniques that are based on encrypting the outsourced data before their transmission to the server are considered the most prevalent ones. The following three subsections present a literature review of the proposed security techniques for each area respectively.

### **3.6.1 Privacy Preserving of Data Sharing and Access Control**

In this section, techniques that preserve data and users' privacy during data sharing among authorized users according to the data access control policies are discussed. In addition, privacy should be assured during sharing and access enforcement even from the cloud provider. Hence, the access control policies should be hidden from

the server hosting the data. The access control policies should be managed mainly by the data owner as if the data were stored in-house.

There are a number of approaches for securing data and their access control policies in the cloud. One of the approaches is to encrypt each data set with a different key and to reveal the corresponding decryption key only to an authorized user [5]. For example, the data owner encrypts the data set with the Key  $K_m$  and then encrypts the key  $K_m$  with each authorized user's public key  $K_{U_i}$  for  $i=1, 2 \dots n$  where  $n$  is the number of authorized users of this data set.



**Figure 3.4 A Simple Way of Securely Sharing Data**

In Figure 3.4, each of the  $K_{U_i}(K_m)$  is labelled with the ID of an authorized user  $U_i$  for  $i=1, 2 \dots n$ . When an authorized user,  $U_i$ , requests access to the data, the cloud server checks the user's ID against the list of authorized users and if the user is on the list, the server returns the protected data  $K_m(data)$  and the  $K_{U_i}(K_m)$  for the user  $U_i$ . The user decrypts the  $K_{U_i}(K_m)$  using his or her private key to obtain the secret key  $K_m$ . Although the data remain encrypted and only authorized users can decrypt them, the number of authorized users and their IDs are known to the cloud server. In addition, access details to the data by authorized users are not hidden from the cloud server. In addition, the access enforcement still requires a cloud server to authenticate an authorized user's identity before the user is given the encrypted data.

A two-layered encryption (or proxy re-encryption) model has been proposed by several researchers [68, 77, 78]. In the first layer, encryption of data is performed by its owner. In the second layer, as part of the access control enforcement mechanism, another encryption is done by the cloud server. In this technique, the access control policies are represented by a key derivation method. The structure of the key derivation method changes according to variations in the access control policies. This mechanism allows a server to dynamically respond to these changes. However, the policy updates expose the system to possible collusion attacks where collusion between a user and a server allows that user to access unauthorized resources. While

this approach provides some clear benefits, there are issues in its implementation. In particular, this approach is computationally expensive as the encryption of data is performed twice. In addition, the process of deriving the keys is considered to be complex. There are enhancements to improve its performance by reducing the need for re-encrypting the data when changing access control policies [79, 80]. However, it still requires re-encryption and complex key derivation and is also vulnerable to collusion attacks during access policy updates.

Some other methods are based on keeping a database of the access policies on the server and hiding the contents of the database. For example, Dai, Luo et al. [81] focus on keeping both users' identities and access control policies confidential from the service provider. The service provider is considered curious but not malicious and thus it remains trusted to enforce these policies honestly but without being able to know them. They proposed a privacy preserving access control solution based on PKI access control mechanisms with TTP server as Certificate Authority (CA) for issuing certificates containing the key pairs of the authorized users. A user's identity is simply hashed by the CA when it issues the user's certificate and the relation between the user's ID and its hash value are kept secret by the CA. The same hash value of a user's ID is used for representing users' ID on the access control at the data owner side before transferring it to the server along with the encrypted data. During access by an authorized user, the server authenticates the user based on the certificate that had been issued for that user by the CA which contains the encrypted ID of the user. If the user is authorized, the server re-encrypts the data using a session key before the data are sent to the authorized user. This is to prevent an unauthorized or revoked user from accessing the encrypted data because only an authorized user can reveal the session key. However, a revoked user can collude with the server to reveal the encrypted data because the proposed scheme uses the same key generated by the data owner to encrypt all the outsourced data. Other solutions attempt to solve this issue by using, for example, a complex computational method to generate and manage the keys, for encrypting/decrypting data, and the method is based on the Diffie-Hellman (DH) algorithm [82, 83]. Recently, homomorphic encryption is proposed to secure the database of the access policies and any update operations on the database [71]. As discussed before in 3.4.3, using homomorphic encryption is not yet a practical approach. The previous solutions hide the contents of

the access control database except the locations of data. A shuffle index approach is proposed to shuffle the contents of data at the server and reflect that at the database for the purpose of disguising data locations from the server [70].

Attribute-Based Encryption (ABE), specifically Ciphertext-Policy ABE (CP-ABE), is chosen in [84] to cryptographically enforce access control policies. The CP-ABE scheme allows encrypted data to be decrypted only by users who are granted specific attributes. These attributes are decided by the data owner during the encryption of the data using attribute based access policy. In [84], additional functions are incorporated into the scheme, including scalable and fine-grained attributes and a user revocation mechanism. The scheme uses dual encryption and CP-ABE methods to perform access control. A trusted authority is responsible for managing and generating the attribute keys. When these are given to the authorized users, proper access rights are also given to the users according to the attributes held by the users. Hence, the trusted authority has to be fully trusted by all the entities in the system. Then the service provider is responsible of managing attribute group keys per each attribute group that represents the collection of users sharing this attribute. Although the solution provides an immediate update of revoked or new authorised users, it requires complex key generation methods that rely on a trusting relation between two authorities, i.e., the attribute generation authority and the server provider. Although the scheme is resistant to collusion attack between users, it is not resistant to collusions between the service provider and revoked users. In addition, the ABE relies on the Bilinear Diffie-Hellman (BDH) cryptography algorithm, which incurs computational overheads during the encryption and decryption processes. Most of the solutions based on the ABE experience these overheads. There are also efficiency issues in the updating of the access policies in the ABE scheme [46, 85, 86].

The Chinese Remainder Theorem (CRT) is used to enforce access control on outsourced data [87, 88]. The CRT is mainly used for protecting the access control policies from being revealed by the server storing the data or by other unauthorised users. An authorised user using his or her pre-shared key with the cloud server can prove to the server that he or she is authorized to access certain resources in the cloud by computing a secret value from a shared value related to each resource and presenting the value to the server. In comparison to the previous methods described in this section, the CRT approach has less computational complexity at both the user

and server sides. For instance, the CRT approach does not require performing re-encrypting or more complex key derivation mechanisms.

Several solutions rely on using the same secret key to encrypt all of the protected resources [81, 87]. This can have serious security ramifications, since if this key is compromised, all resources lose their protection. Moreover, any user can collude with the cloud server to access a resource that they are not authorized to access. On the other hand, if the data owner uses a unique key to encrypt each resource, distribution of the keys to authorized users can be a problem. Therefore, sharing and distribution of the keys in a secure manner, among all authorized users needs to be considered. The proposed DCS method in this thesis overcomes these problems without complex computation and key management overheads, by enhancing the works previously reported [87, 88]. Details of the method will be discussed in the following chapters.

### 3.6.2 Secure Searching on Encrypted Data

This section reviews techniques that can be used to search securely on encrypted data in the cloud domain without revealing any private information to the cloud server. The work in [89] can be considered as the first technique where a cryptography scheme was used for searching words on encrypted text documents in the cloud domain without allowing the un-trusted cloud server to have access to the plaintext. The technique mainly uses pseudorandom number generation and block cipher encryption. Although the technique provides the capability to search for any words in an encrypted file, it suffers from several problems due to the fact that it has been designed for searching by the data owner only. Furthermore, it is inefficient for searching a large amount of data because the search complexity increases linearly  $O(n)$  where  $n$  is the document length, while the average complexity for such operations nowadays is  $O(n)$  where  $n$  is the number documents [90]. A secure index approach is used to speed up the search operation and to support multi-user settings [91]. The secure index approach is based on using indices to map encrypted keywords to the files containing these keywords. However, there are issues in the secure index approach in terms of updating the indexes when the files are dynamically changed (e.g., edited, replaced and deleted). Moreover, data confidentiality can be compromised because of the amount of data that can be

learned by the cloud server on each index update, using statistical analysis techniques.

Waters et al. [92] developed two secure search mechanisms. The first one is based on Symmetric Searchable Encryption (SSE) that supports search capability for data owners on their encrypted data in the cloud domain. The first method is not secure against the server hosting the data as the symmetric key is revealed to the server during the search process. Furthermore, there is a possibility of false positive results from the search process (i.e., files that do not contain the keywords), because the hash algorithm which has been used may produce, with low probability, the same hash for different keywords. The SSE method supports only a single user so only the data owner can search securely in his or her encrypted data. The second method is based on Asymmetric Searchable Encryption (ASE) that supports multi-users to search the encrypted data. The ASE method is computationally expensive because it uses the Identity Based Encryption (IBE) scheme to perform encryption. The IBE scheme, in turn, uses Tate pairings over super-singular elliptic curves which do not have an efficient implementation. Although the scheme is designed to keep data secure from unauthorised accesses, curious or un-honest servers are still able to decrypt the encrypted data as the secret key is revealed to the server during a search process.

Chang and Mitzenmacher [93] improve previous work started by Dawn Xiaoding, Wagner et al. [89]. The scheme used in [59] also depends on the use of pseudo-random numbers. Any symmetric encryption can be used to encrypt data files but a search can only be used for keywords that were generated prior to encrypting the files. The data owner has to index each keyword in a file and then uses the pseudo-random function to encrypt the indices before sending them with the encrypted file to the server. The method is reported to be efficient because it does not use any asymmetric encryption algorithm which is more computational expensive compared to other encryption methods. The method is designed to support only search by the data owner on his or encrypted data stored in the cloud. In addition to speed up the search speed, the advantage of using an index is that the index can be used for different types of files (e.g. images and audio files) and the compressed versions of them. The scheme does not support efficiently and securely advanced search functions such as searching for multiple keywords in one query or searching for word

patterns instead of exact keyword match. With regard to security, the scheme is vulnerable to statistical analysis on the indices stored on the server and the server is capable of reusing previous queries to search on old files.

Dong et al. [14] designed an encryption scheme to allow data owners to share their data with authorised users based on a public key proxy encryption technique built on the El Gamal encryption algorithm. The security of the scheme relies on the increased computational complexity of the Decisional Diffie-Hellman (DDH) method which is an improved variant of the basic Diffie-Hellman technique. In the scheme, a Key Management Server (KMS) needs to be fully trusted with key management. The KMS uses a master key to generate a set of keys for users and the server provider. The scheme is not secure against possible collusion between a server and authorized users to work out the master and the server can observe the users' access patterns to the encrypted data. Several ideas to address these issues are discussed by the authors.

Kamara et al. [94] developed a technique which provides privacy protection as well as integrity protection on search queries and results from encrypted data in the cloud domain. The integrity of queried results from a server is verifiable by a user. The technique uses the Symmetric Searchable Encryption (SSE) scheme for search functionality and privacy protection. In the SSE scheme, a search uses an encrypted and hashed invert index which maps a keyword to its location in a document or set of documents stored at the server. The search performance is improved by returning a subset of search results instead of all the search results for more efficient bandwidth consumption. The technique allows only the data owner to verify the integrity of data stored in the server without the need to download the entire data file. This allows a data owner to detect whether a cloud provider has altered the data in the file or has modified query results. However, the search functionality relies on keyword indexes stored on the server and the indexes can be targeted by statistical attacks. Moreover, the search capability is only within the server hosting the index database. If the data are moved to another server, they cannot be searchable. Furthermore, the SSE-based technique is not practical to allow data owners to share their data securely with others because it only uses a symmetric key and hence sharing it among users is a security issue [12].



Koletka and Hutchison [95] implemented a mechanism for securely sharing and searching protected data stored in a cloud by authorized users. The mechanism uses a container called Secure File Object (SFO) that contains a number of metadata fields. An SFO can be used to improve the privacy and integrity of enclosed data, by a mechanism which manages 'read' and 'write' rights. In practice, the mechanism can only detect unauthorized writing to the protected data, but cannot prevent writing by authorized users that have only read rights. In addition, malicious users can create a new SFO that contains the encrypted contents of another authorized user's data to pretend that he/she is the owner of the SFO. For users authorized to access an SFO, their public keys are listed in the SFO similar to what is shown in Figure 3.4. As a result, the scheme does not protect the access policies or users' identities from the cloud server.

Recently, Li, Jia et al. [96] improve access control and search on encrypted data by developing a two-layer access control scheme. The first access layer is controlled by the trusted private cloud, while the second layer is managed by the data owner. Hence, the access procedure requires a trusted private cloud between the public cloud and the data owner. Moreover, the scheme is based on IBE which depends on the management of keys by a third party key management [82].

### 3.6.3 Data Integrity Proof

Data integrity is an important security feature particularly when data are outsourced to the cloud storage. Traditionally, data integrity is protected by computing the digest of data using one of the hash algorithms (e.g. Hash-based Message Authentication Code (HMAC)) and then encrypting it with the data owner private key. The resulting value is called a digital signature and is usually attached to data before the data are outsourced to the cloud. Digital signatures are widely used in several schemes that keep encrypted data in the cloud and the data are only decrypted after they are downloaded by authorised users [86, 94, 95].

Some researchers [97-99] focus on reducing bandwidth and computational overheads of the integrity verification process. Their methods do not involve the encryption of the entire data. Instead, only a few chunks of each set of data are encrypted and these chunks have their integrity proofs added as metadata to the data set. The data owner only verifies the integrity of these small selected chunks of the data in terms of

verifying the integrity of the whole set of data. Only the data owner knows which part of the data set corresponds to the metadata integrity proofs. Other solutions depend on verifying data integrity by using TC technologies, as in [16], or essentially involving TTP, as in [100].

### 3.7 Evaluating the Reviewed Security Techniques

This section discusses the security techniques reviewed in the previous section. The techniques are evaluated on how they can be used in the proposed DCS approach for the cloud model. The evaluation is based on the set of criteria stated in Section 3.2.3 and on whether they achieve the essential security requirements listed in Section 3.6. The following list organizes the main points of evaluations and discussions:

- The techniques for securely enforcing access control on outsourced encrypted data are mainly used to enforce read access rights when authorized users have to download the data and then decrypt them so they become able to read these data. However, the user can write to the downloaded data in the trusted domain and then re-encrypt it before resubmitting it to the server as new data or updated data.
- Solutions that require an encrypted access control database to be stored on the server are not compatible with the DCS approach. Instead, the access control policies must be at the data level.
- Solutions that require two or more encryption layers for the data outsourced are computationally expensive, particularly for large amounts of data. The cloud server in such a solution becomes an essential entity of securing data and hence there is more possibility of collusion attack. Therefore, these techniques are not complying with DCS approach criteria as the data access control policies rely on the cloud server.
- Attribute based encryption is a promising method for cryptography access control and sharing encrypted data. However, several open problems remain in the practical use of this technique. The generation and management of attributes and related keys, revoking a user, preventing collusion attack and other security issues related to the different ABE constructions are the major problems for practical applications. In addition, the ABE techniques add trust architecture overheads while one of the desired benefits of the DCS approach is to reduce dependency on the trust factor. Hence, we avoid using this

technique in our approach. In [81], the authors attempt to tackle some practical problems of using ABE under the DCS concept but still more work has to be done to successfully solve these problems.

- For unstructured data, search mechanisms can be designed to search attached predefined keywords for each data set or search a reconstructed encrypted index table of keywords and files related to these keywords. Although the second approach, in general, is more efficient in terms of search capability, the first one is more flexible as each data set has the search capability attached to it and does not depend on an outside index. Therefore, the first one is compatible with the DCS concept.
- From a security point of view, the encrypted keyword index method for data stored on a cloud provider server exposes the indices and the search queries for statistical and dictionary attacks. In addition, this is not compatible with the DCS approach as the secure search capability is provided from outside data.
- Search techniques, particularly Asymmetric Searchable Encryption (ASE) schemes, are weak against both the Chosen-Ciphertext Attack (CCA) and the Chosen-Plaintext Attack (CPA). When the public key of the authorized user is available to the public, the attacker may perform the CCA by choosing a ciphertext then attempt to figure out its plaintext. In the CPA, the attacker performs the other way around by encrypting a chosen plaintext by an authorized user public key and analyses the resulted ciphertext, in attempting to compromise the user secret key. They rely on third party KMS for revoking and granting privileges, while in the DCS approach, these operations must not rely on a third party. The AES scheme is also considered inefficient and less secure compared to the Symmetric Searchable Encryption (SSE) scheme [12]. However, the SSE scheme is basically for a support (single writer/single reader) case.
- Search methods that return false positive search results produce another issue. On the one hand, it can be one of the ways to disguise the search results from the cloud server. On the other hand, it costs more bandwidth, computation overheads and time because a user needs to download the false positive results, decrypt them and then search for the keyword before realizing that

false search results have been returned. Hence, it is more efficient to avoid this issue.

- The proof of the data integrity must be attached to each data set where authorized users are able to check it at any time. Thus, the techniques that rely on the cloud server or TTP or the TC to verify integrity of outsourced data are not compatible with the DCS approach. Also the techniques that allow only the data owner to verify the data integrity are not suitable to use in the DCS approach which should allow all authorized users to verify the data integrity.

As a result of the evaluations and discussions, the preferred techniques can now be selected and developed to form the basis of the solution in this work. As such, the Chinese Remainder Theorem (CRT) is chosen as it provides an efficient and flexible method to enforce the access control policies on the outsourced data. Using the CRT for access control, does not restrict the users' choices of encryptions techniques. Hence, any symmetric encryption scheme key can be used for encrypting the data. The CRT was used in [88] for outsourced data access control. But in that approach, the identities of authorized users are revealed to the server during the access process. Then CRT for access control was used in [87] to hide the identities and number of authorized users but the problem was that all the data are encrypted with the same key. The proposed solution in this thesis overcomes this problem because it allows the data owner to use different symmetric keys to encrypt each data set for more security and without an extra key management overhead. The symmetric key is securely attached to each data set using the CRT. This is explained in more detail in the next chapter.

The proposed solution in this thesis provides a search capability by including the Searchable Symmetric Scheme (SSE) adopted from [92]. Because the SSE was designed to support only the data owner, it modified and combined with the access control mechanism, based on CRT and asymmetric encryption algorithms, to allow all authorized users to search on the encrypted data. The proposed solution makes use of the advantage of each used technique and reduces their disadvantages. As symmetric encryption is more efficient, it is used in the proposed solution for encrypting the data. The combination of techniques is designed to enhance security

and privacy with as minimum overheads as possible. Moreover, the solution is designed according to the DCS concept so access control, search capability and data integrity proof are attached to each encrypted data set individually. The proposed solution is under the assumption that the cloud server is honest but curious. Hence, the server may attempt to read the contents of clients' data and to observe any information related to it. On the other hand, the server should protect its reputation by providing the services. For example, it executes access control procedures and search queries.

### 3.8 Summary

This chapter describes the DCS approach and how it is applied to the cloud model. In this thesis, the DCS approach rests on three foundation concepts:

- The security requirements are provided from within data.
- The data owner is responsible for these security requirements throughout the data lifetime.
- The security requirements do not rely on security measures provided from outside data.

These concepts distinguish the DCS approach from other approaches for providing data security and privacy in the cloud. The other approaches provide their security measures at the hardware, platform or application level relying on a cloud provider and/or a third party including TC technologies. In contrast, the DCS approach provides a data security and privacy solution to cover the entire data lifecycle. Accordingly, each data set is self-describing, self-protecting and all the security specifications attached to it and the enforcement of these specifications are accomplished in a secure manner.

Throughout the lifecycle of data sets, the data owner is able to manage their security specifications and to keep track of them, including their usage history. Most importantly, in the first stage of the data lifecycle, the data set creation stage, a data owner is responsible for constructing each data set with all the security requirements before sending it to the cloud. During the remaining stages until the destruction stage, the data set maintains compliance with the data owner security policies. Applying this approach to the cloud model is expected to improve data privacy and

security in the cloud to fit its scalability and elastic nature. This expectation is supported by the list of references in Table 3.1 and points in Section 3.3. However, completely applying the ideal conceptual framework of DCS is still impeded by several challenges. The scope of applying the DCS approach in this study is determined to limit such challenges. Although the scope is limited to unstructured data stored and shared in the public cloud storage, the proposed solution covers the most prevalent situations in cloud services. Moreover, the solution can be modified for application to other data forms and in other cloud models. The essential security requirements for the proposed solution are listed in Section 3.6. They essentially relate to protecting privacy and integrity during accessing and searching the encrypted data stored in the cloud servers. Based on the DCS concepts, the security techniques which may satisfy these requirements are reviewed. The techniques meeting the security requirements will be adapted and used in the proposed solution in an efficient and practical way. In the next chapter, the proposed solution is explained in detail.

## Chapter 4: A Data Centric Solution to Cloud Privacy and Security Issues

### 4.1 Introduction

This chapter describes the proposed solution that addresses the main research questions of this study. The proposed solution is based on the Data Centric Security (DCS) approach discussed in Chapter 3. The aim of this chapter is to develop a solution that makes an original research contribution to the DCS concept which is believed to be an adequate approach to address privacy and security issues in cloud computing. The solution is designed to satisfy the following desired requirements for applications in a cloud computing environment:

- The data are encrypted and can be accessed by authorized users only.
- The data are searchable without compromising their privacy.
- The data are self-protected and contain the required security parameters.
- The access control parameters are hidden from cloud server providers and other users.
- The server provider does not know the number or the identity of users who are authorized to access the data.
- Unauthorized entities, including service providers, cannot gain access to the data or derive information about the data from authorized operations carried out on the data.
- The data contain all the necessary information for proving their integrity and authenticity to the authorized users who access the data.
- Interactions between owners and authorized users should be minimal, especially for key management purposes.

These requirements are specified by a set of modules each one of which specifies at least one of the requirements. All parameters required for accomplishing the security functions are attached to the data file and the result is a file called a DCS file. The security functions include privacy protection as well as integrity and authenticity verifications. This chapter starts with a background description about the Chinese Remainder Theorem (CRT) which is the core algorithm used in the proposed solution. Then, Section 4.3 describes the module that is used to provide and manage

parameters required for access control and key sharing using the CRT. It is then followed by Section 4.4 which explains how the search capability is incorporated into the algorithm and how it is integrated with the access control procedure. Section 4.6 describes the architecture of all modules involved in the generation of a DCS file from data with integrity and authenticity proofs. Finally, the benefits of the proposed solution are discussed and a summary of the chapter is given in the last section.

## 4.2 Chinese Remainder Theorem and its Applications

The CRT was founded in ancient manuscripts written by a Chinese mathematician for analyzing arithmetic problems [101]. It is known as one of the oldest theorems in mathematics that was used to create calendars during the first century AD [102]. The CRT had been developed by mathematicians around the world until it reached its current formula [102].

### Theorem 1. Chinese Remainder Theorem

For any given integers,  $a_1, a_2, \dots, a_k$ , the following system of simultaneous congruence has a unique solution  $X$ , such that  $0 \leq X < n = n_1 n_2 \dots n_k$ , provided the non-negative integers  $n_1, n_2, \dots, n_k$  are relatively prime.

$$\begin{aligned} X &\equiv a_1 \pmod{n_1} \\ X &\equiv a_2 \pmod{n_2} \\ &\vdots \\ X &\equiv a_k \pmod{n_k} \end{aligned} \tag{4.1}$$

If  $X$  is given in the above congruence, each  $a_i$  can be calculated by the equation:

$$a_i = X \pmod{n_i} \tag{4.2}$$

for  $i = 1, 2, \dots, k$ . The proof of this theorem can be found in a number of references, e.g. [103].

The unique solution  $X$  for the simultaneous congruence can be calculated by the following equations [104]:

$$X = \sum_{i=1}^k a_i M_i M_i^{-1} \pmod{M},$$



where

$$M = n_1 n_2 \dots n_k,$$

$$M_i = M/n_i,$$

$$M_i^{-1} \text{ is the multiplicative inverse of } M_i \bmod n_i, \text{ i. e. } M_i M_i^{-1} \equiv 1 \pmod{n_i}$$

Because of the fact that  $M_i$  is relatively prime to  $n_i$ , there is a unique multiplicative inverse in  $\bmod n_i$ . Hence, calculating the multiplicative inverse in modular is an essential part of determining the CRT solution. The Extended Euclidean Algorithm (EEA) can be used efficiently to compute the multiplicative inverse [104], and is an essential component of the Garner's algorithm which is generally used to find the CRT solution. Details of the Garner's algorithm can be found in a number of references such as in [105].

The following Garner's algorithm [6] is used in the proposed solution to determine the CRT solution [88]:

**Input:** positive integers  $n = \prod_{i=1}^k n_i > 1$ , where  $\gcd(n_i, n_j) = 1$  for all  $i \neq j$ , and a modular representation  $a_i = a_1, a_2, \dots, a_k$  can be any integers.

**Output:** an integer  $X$  in radix  $b$  representation.

Steps of the algorithm:

1. For  $i = 2$  to  $k$  do:
  - 1.1  $C_i \leftarrow 1$ .
  - 1.2 For  $j = 1$  to  $(i - 1)$  do:
    - 1.2.1  $u \leftarrow n_j^{-1} \bmod n_i$ .
    - 1.2.2  $C_i = u \cdot C_i \bmod n_i$ .
2.  $u \leftarrow a_1, X \leftarrow u$ .
3. For  $i = 2$  to  $k$  do:
  - 3.1  $u \leftarrow (a_i - X) \cdot C_i \bmod n_i, X \leftarrow X + u \cdot \prod_{j=1}^{i-1} n_j$ .
4. Return  $(X)$ .

where  $n_j^{-1}$  is the multiplicative inverse of  $n_j$  in modular  $n_i$ .

One of the useful features of the CRT is that there is minimal change to current congruence when a new modular is added to the congruence. For example, if the solution  $X$  of Equation (4.1) has already been found and if a new modular  $a_{k+1} \bmod n_{k+1}$  is added to the congruences, let the solution for new congruences be  $X'$ . The new solution can be found by calculating the solution of the two following congruences:

$$\begin{aligned} X' &\equiv X \bmod n_1 n_2 \dots n_k \text{ and} \\ X' &\equiv a_{k+1} \bmod n_{k+1}. \end{aligned}$$

There is no need to evaluate the other congruences to arrive at the new solution.

Similarly, if an existing congruence is removed from the congruences, the new solution can be derived by a simple calculation. For example, let the new solution be  $X''$  after the last congruence  $X \equiv a_k \bmod n_k$  in Equation (4.1) is removed. The new solution can be calculated by one modular operation  $X'' = X \bmod n_1 n_2 \dots n_{k-1}$  [88]. These features can be used to minimize the computation overhead of calculating the CRT solution in such cases. For example, this feature is used, in the proposed solution in this chapter, when granting a new user an access right to a certain resource. This will be discussed in Section 4.3.3.

Several applications in cryptography and secure communications use the CRT because of its arithmetic nature that does not consume high computation resources. Therefore, it is suitable for computing and communication applications with limited resources. A secret sharing is one of the basic security functions of using the CRT in cryptography. It allows sharing a secret by deriving from it a set of shares (or shadows) that can be recovered only with a certain prearranged set of values [101]. Many applications are based on this secret sharing scheme, such as threshold cryptography and e-voting [106].

In wireless networks where devices may have limited resources, especially computation and storage resources, proposed solutions based on the CRT can be employed in several schemes. The CRT can be used directly in authentication and

access control schemes. For example, wireless sensor network broadcast authentication based on the CRT is proposed in [107]. In another example an access control scheme is proposed and is based on the CRT for wireless multimedia sensor network [108]. The CRT also is used to decrease the energy consumed when transmitting packets between nodes of wireless sensor networks which result in increasing in the lifetime and energy efficiency of the network [109]. Mobile ad-hoc networks are other wireless systems where the CRT is proposed to provide security with less impact on the system resources [110]. There is also indirect use of the CRT in applications where the CRT is used to speed up and reduce resource consumption of public-key cryptosystem operations and/or secret key generations [111-114].

### 4.3 Cryptography Access Control and Key Sharing Using the CRT

In this thesis, the proposed solution uses the CRT and the public key cryptosystem to secure sharing of protected users' data stored in a cloud computing environment amongst authorized users. The data, which are sometimes referred to as resources or files, is protected by the symmetric encryption method, where a secret key,  $K_s$ , is used. In this thesis, a resource can be a data set or a file containing data of any type, including text, audio, image or video. To distribute the secret key to an authorized user, it is encrypted using the public encryption method with the user's public key. The encryption also includes another value,  $C_r$ , that will be used as a response to a challenge sent by the server when a user requests access to the resource. Only an authorized user, who has the corresponding private key, can decrypt the cipher-text of  $C_r$  and  $K_r$  to produce  $C_r$  to access the resource  $r$ . Parameters  $C_r$  and  $K_s$  are concatenated to form  $C_r || K_s$ , and are treated as one value. This value is encrypted with the public key  $E_{K_{pub\ i}}$  of each authorized user  $u_i$ , resulting in a cipher-text of  $\left(E_{K_{pub\ i}}(C_r || K_s)\right)$  for the user  $u_i$ , where  $i=1, 2, 3, \dots, k$ , and  $k$  is the number of users authorized to access the resource  $r$ .

To apply the CRT to the proposed solution, for users  $u_1, u_2, \dots, u_k$ , each authorized user is associated with a unique relatively prime number  $n_i = n_1, n_2, \dots, n_k$ , where  $k$  is the number of authorized users. All  $n_i$ ,  $1 \leq i \leq k$ , are relatively prime numbers. Then, the cipher-text of  $C_r || K_s$  produced for each user, i.e.  $\left(E_{K_{pub\ i}}(C_r || K_s)\right)$ , is

used to substitute  $a_i$  in Equation (4.1) to produce the following simultaneous congruence:

$$\begin{aligned}
 X_r &\equiv \left( E_{K_{pub\ 1}}(C_r \parallel K_s) \right) \bmod n_1 \\
 X_r &\equiv \left( E_{K_{pub\ 2}}(C_r \parallel K_s) \right) \bmod n_2 \\
 &\vdots \\
 X_r &\equiv \left( E_{K_{pub\ k}}(C_r \parallel K_s) \right) \bmod n_k
 \end{aligned} \tag{4.3}$$

The solution of this congruence  $X_r$ , such that  $0 \leq X_r < n = n_1 n_2 \dots n_k$ , is the shared value for the resource  $r$  and it is attached to the resource, and the resource and the shared value are stored together in a cloud server. When an authorized user  $u_i$  requests access to a resource  $r$ , the cloud server sends the shared value  $X_r$  to the user. When the user receives the shared value  $X_r$ , the user uses the corresponding private key to decrypt  $X_r$  to produce the  $C_r \parallel K_s$  value as shown in Equation (4.4) below:

$$C_r \parallel K_s = D_{K_{priv\ i}}(X_r \bmod n_i) \tag{4.4}$$

where  $D_{K_{priv\ i}}$  is the decryption operation using the private key of user  $u_i$ , and  $n_i$  is the relatively prime number specific for this user. The value  $C_r$  is then sent back to the server by the user to prove that the user is authorized to access the resource. The server then sends the resource to the user and the key  $K_s$  is used by the user to decrypt the file to obtain its contents.

#### 4.3.1 Enhancing Keys and Files Security

In terms of enhancing security in the proposed solution, a data owner must use a unique symmetric key,  $K_s$ , to encrypt each file before sending it for storing in the cloud. According to the DCS approach, all the security requirements are attached to the actual data hence each symmetric key is attached to its file in a secure manner. From Equation (4.3), the symmetric key  $K_s$  is protected by two levels: first by encrypting it with an authorized user, then by the CRT to find the shared value  $X_r$ . Only authorized users can calculate the  $K_s$  from the  $X_r$  using Equation (4.4) which required the related  $n_i$  and private key of an authorized user. For efficient key

management, the data owner adds himself as a user when calculating  $X_r$  for each file using Equation (4.3). Hence, neither the data owner nor the users are required to save the  $K_s$  but only needs their private key and assigned  $n_i$  value. In this way, the key,  $K_s$ , is securely and efficiently shared with authorized users, and is protected from unauthorized access including the service provider hosting the file.

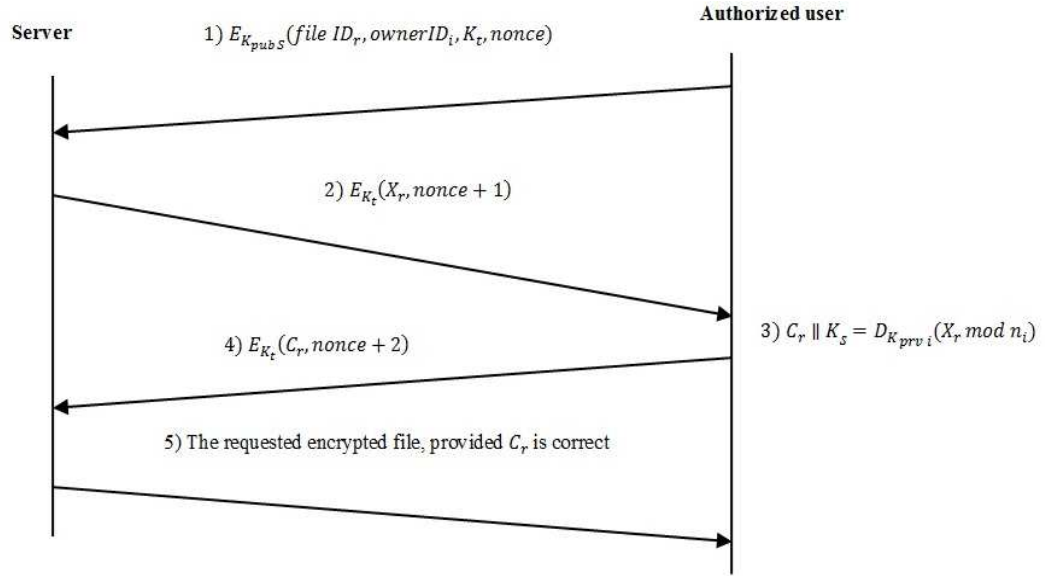
The secret key  $K_s$  as well as the value  $C_r$  are unique for each file. If the values of one file get compromised the other files remain safe. The secret value  $C_r$  is used by an authorized user to show to the server he or she is authorized to access the resource  $r$  (i.e. the file). The data owner securely attaches the secret value  $C_r$  to the file and also sends it inside the cipher-text  $X_r$  to the server. Only authorized users can calculate the secret value  $C_r$  using the shared value  $X_r$ . So, only authorized users can know and reveal  $C_r$  to the server and hence prove that they are authorized to access this particular file. The secret value  $C_r$  is unique for each file even if it is used by the same user for different files. As such, if one  $C_r$  for a particular file is compromised, no other files will be affected. Moreover, this feature is useful if the data owner wants to change the details of authorized users for a file, for instance, to add a new authorized user, the data owner only needs to change  $X_r$  for that file. As the parameter  $C_r$  can remain the same, it is not necessary to resend a new  $C_r$  to the server. This in turn, will reduce the possibility of compromising this value while maintaining a dynamic mechanism for updating the list of authorized users. However, when revoking a user from accessing a file that he/she already has accessed, the  $C_r$  value should be changed for security reasons. More details are described in Section 4.3.3.

#### 4.3.2 Secure Access Control Procedure

The proposed solution combines access control and key sharing in one mechanism using the CRT. The cloud server stores the encrypted data with the relevant secret  $C_r$  and shared value  $X_r$  calculated by the data owner. The provider can read the shared value  $X_r$  but cannot find out the  $K_s$  that was used to encrypt the data. Only authorized users can reveal the  $K_s$  from the  $X_r$ . The provider can also read the secret value  $C_r$  for each file. However, the number or identities of users that can access a file, even if they have already accessed the file, remain hidden from the provider.

To facilitate the needed calculations, it can be noted that when a user  $u_i$  is authorized to access file  $r$ , the data owner must have already sent it the relatively prime  $n_i$  assigned to it. The  $n_i$  is sent once to each user then the data owner uses it with the user's public key for all files that the user is authorized to access. For more security, the  $n_i$  should be sent to the user securely, for instance, by encrypting it with the user public key. So attackers will find it harder to discover the cipher-text ( $E_{K_{pub\ k}}(C_r \parallel K_s)$ ) from the shared value  $X_r$ . However, even if the  $n_i$  of any user is revealed to unauthorized entities, the files of that user remain secure as long as the private key of that user is not compromised. Therefore, before a user is authorized to access a file, the data owner of the file has the user's public key when calculating  $X_r$  of the file and the user has received his/her  $n_i$ .

The messages exchanged between an authorized user and the cloud server hosting shared files are shown in Figure 4.1. In the proposed solution, when the user  $u_i$  needs to access the file  $r$ , the user sends a request to the server, containing the ID of the file  $ID_r$ , the file owner's  $ID_i$ , a nonce (any random number), and a session key  $K_t$ . all encrypted with the public key of the server  $E_{K_{pub\ s}}$ . The result is represented as  $E_{K_{pub\ s}}(ID_r, ID_i, K_t, nonce)$ . The session key  $K_t$  is created by the user for securing the information exchange between the user and the server. The server locates the file with  $ID_r$ , reads its shared value  $X_r$ , increments the nonce to  $(nonce+1)$  and encrypts them with the session key  $K_t$ . The result, represented as  $E_{K_t}(X_r, nonce + 1)$  is returned to the user.



**Figure 4.1 Access Control Procedure**

After receiving the information, represented in message number two in Figure 4.1, from the server, the user calculates the secret parameter  $C_r$  using Equation (4.4) and returns  $C_r$  and  $(nonce + 2)$  to the server encrypted with the session key  $K_t$ , that is  $E_{K_t}(C_r, nonce + 2)$ . The server checks  $C_r$  and if they are matched with the one attached to the requested file, the server sends the encrypted file to the user. The user can decrypt the file by using key  $K_s$  which resulted previously from using Equation (4.4) at step three in Figure 4.1. All the messages between the user and the server are encrypted and an incremental nonce is used to prevent traditional communication attacks such as Man-In-the Middle (MIM). As mentioned above, for privacy access, the identity of the authorized user is hidden from the server hence the procedure avoids using the public key of the user to keep his identity confidential. In contrast, the user starts the communication session with the server with a message encrypted with the server's public key to authenticate the server. Therefore, if the user communicates with the exact server, only that server is able to decrypt the message to reveal the session key  $K_t$  and nonce, and then respond with an encrypted message which contains the nonce+1.

The access procedure shows that the enforcement of the access control policies and providing the authorized user with the secret key for decrypting data are accomplished in one mechanism. This technique reduces the overhead on both the

cloud server and the user in terms of computational and key management. Moreover, using the CRT in this mechanism is another factor for reducing the overhead as the CRT operations are simple modular arithmetic operations; for instance, there are no modular exponential operations. Another important feature reached by using the CRT, is that the CRT allows the data owner to hide the number of authorized users. Because of its nature as discussed in Section 4.2, the cipher-text  $E_{K_{pub\ i}}(C_r \parallel K_s)$  of each authorized user  $u_i$  for  $i=1, 2, \dots, k$  can be represented by using CRT in one value  $X_r$ . The server cannot find out how many users are presented in this  $X_r$  but without using CRT the data owner has to attach all the cipher-texts so the number of the users is revealed to the server.

### 4.3.3 Granting and Revoking Procedure

To grant a new user  $u_{k+1}$  access to a file  $r$ , a new congruence is added to the CRT, as shown in Equation (4.5). In such a case, the owner needs to recalculate the shared value  $X'_r$ .

$$\begin{aligned}
 X'_r &\equiv \left( E_{K_{pub\ 1}}(C_r \parallel K_s) \right) \bmod n_1 \\
 X'_r &\equiv \left( E_{K_{pub\ 2}}(C_r \parallel K_s) \right) \bmod n_2 \\
 &\vdots \\
 X'_r &\equiv \left( E_{K_{pub\ k}}(C_r \parallel K_s) \right) \bmod n_k \\
 X'_r &\equiv \left( E_{K_{pub\ k+1}}(C_r \parallel K_s) \right) \bmod n_{k+1}
 \end{aligned} \tag{4.5}$$

$K_{pub\ k+1}$  is the public key of the new user. As described in section 4.2, the solution  $X'_r$  of the above simultaneous congruence can be calculated from the  $X_r$ , that is already attached to the file, with less modular calculation if the new congruence system for the  $X'_r$  has the same modular of the previous  $X_r$  plus a new modular; in other words, if the data owner decides to add a new user to access an existent file and does not intend to change the previous values, including  $C_r$  and  $K_s$ , that were used to calculate the  $X_r$ . Hence, it is more efficient to find the  $X'_r$  by solving the following congruence:



$$X'_r \equiv X_r \bmod n_1 n_2 \dots n_k$$

$$X'_r \equiv \left( E_{K_{pub\ k+1}}(C_r \parallel K_s) \right) \bmod n_{k+1} \quad (4.6)$$

The data owner sends the new shared value  $X'_r$  with the file ID and data owner ID to the server to replace the old  $X_r$ . The granting mechanism of adding a new user is efficient for two reasons; it only requires one asymmetric encryption operation for a fixed length of information, i.e.  $C_r \parallel K_s$ , and it finds a solution for only two congruences, see Equation (4.6). Moreover, the granting mechanism does not disclose any secret value as the exchanged values are not considered to be secret. Although the shared value  $X'_r$  is based on secret values, only authorized users can access them.

To revoke the access privileges of a user  $u_k$  to the file  $r$ , the data owner has to change the secret value  $C_r$  to  $C'_r$  and to re-calculate  $X_r$  to  $X''_r$  for the remaining users from 1 to  $k-1$  as follows:

$$X''_r \equiv \left( E_{K_{pub\ 1}}(C'_r \parallel K_s) \right) \bmod n_1$$

$$X''_r \equiv \left( E_{K_{pub\ 2}}(C'_r \parallel K_s) \right) \bmod n_2$$

$$X''_r \equiv \left( E_{K_{pub\ k-1}}(C'_r \parallel K_s) \right) \bmod n_{k-1} \quad (4.7)$$

The data owner sends the new  $C'_r$  and  $X''_r$  values to the server with the file ID and data owner ID to replace them with the old values of that file. If the user has accessed the data before revocation, it is possible that the user and the server can collude to access the data after revocation. When an authorized user has accessed a file, the key of this file becomes known to him. Later if that user is revoked from accessing the file and the data owner only changed the  $C_r$  for that file, it is possible for the user to collude with the server so the server allows him to access the file and the user provides the server the old key  $K_s$  that still decrypts the file. Although, as discussed in Section 4.3.2, the scheme does not reveal the identities of the users to the server and that may reduce the chance of this type of collusion attack, it is strongly recommended to change the key  $K_s$  and re-encrypt the file particularly if the content has been changed. Then replace the old file with the new one that is encrypted with a

new key and has the new  $C'_r$  and  $X''_r$  attached to it. Consequently, the server and the revoked user cannot collude to access the new content of the file.

#### 4.4 Secure Search Ability

In this thesis we have adopted methods for searching encrypted data and these methods are compatible with the DCS approach as discussed in Chapter 3. The data owner has to specify one secret key  $K_w$  which is used in the encryption of each of the keywords. The file owner uses the keyed pseudorandom function (PRF) to encrypt the keywords. Let  $H_K(m)$  be a Hash-based Message Authentication Code (HMAC) that can use any cryptographic hash function, such as MD5, SHA1, SHA256 or SHA512, with a key  $K$  and input message  $m$ . Assume  $w_i$  is the  $i$ -th keyword in the keyword list,  $R$  is a random number,  $flag$  is a constant bit-pattern with fixed length  $l$  bits and padding is a random pattern of bits.

$$a_i = H_{K_w}(w_i), \quad b_i = H_{a_i}(R), \quad c_i = b_i \oplus (flag \parallel padding) \quad (4.8)$$

From each keyword  $w_i$  the data owner creates  $c_i$  and sends it with the random number  $R$  to the server attached to the related file. The  $c_i$  is the encrypted form of the keyword  $w_i$ ; therefore, the server cannot reveal the  $w_i$  from the  $c_i$ . The next section describes how users can securely search on their encrypted files by using these encrypted keywords.

#### 4.5 Access Procedure with Secure Search Ability

In our proposed solution for searching encrypted data stored in a cloud environment, when the user  $u_i$  needs to search for a keyword  $w_i$ , the user has to send a request for search capabilities to the data owner containing the keyword  $w_i$  encrypted with the owner's public key ( $K_{pub\ o}$ ) and signed by the user's private key ( $K_{priv\ i}$ ). The data owner then replies with  $T_w = H_{K_w}(w_i)$  encrypted with the user's public key ( $K_{pub\ i}$ ). The user decrypts the message by his private key and then encrypts  $T_w$  with the server's public key ( $K_{pub\ s}$ ) before sending it to the server as a search request with the owner's ID, nonce and a session key  $K_t$ , as shown in Equation (4.9).

$$E_{K_{pub\ s}}(T_w, ownerID, K_t, nonce) \quad (4.9)$$

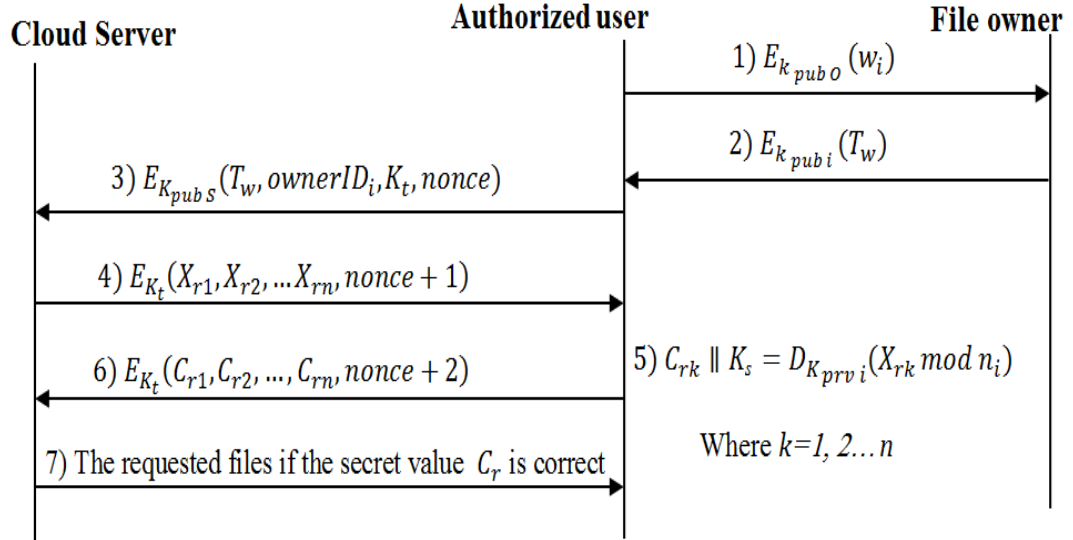


Figure 4.2 Secure Search Procedure

The server should search only the data related to the owner ID using  $T_w$  to compute  $P = H_{T_w}(R)$  and then for each  $c_i$  the server checks the result of  $P \oplus c_i$  to see if the first 1 bits are equal to the flag. Then the file ID related to this  $c_i$  is added to the search result list together with its two parameters  $X_r$  and  $C_r$ . The purposes and contents of the parameter items  $X_r$  and  $C_r$  are discussed in Section 4.3.

$$P = H_{T_w}(R), \quad \text{if } P \oplus C_i = \text{flag} \parallel \text{padding} \therefore \text{there is match} \quad (4.10)$$

After searching all files, the server sends only the  $X_r$  values  $(X_{r1}, X_{r2}, \dots, X_{rn})$ , where  $n$  is the number of files matching the search criteria, and  $nonce+1$  and all encrypted with a session key,  $K_t$ , back to the client.

$$E_{K_t}(X_{r1}, X_{r2}, \dots, X_{rn}, \quad nonce + 1) \quad (4.11)$$

The user extracts the secret value  $C_{ri}$  for each shared value  $X_{ri}$ , where  $i=1, 2, \dots, n$ , using Equation (4.4) and then returns the secret values and  $nonce+2$  to the server encrypted with the session key  $K_t$ .

$$E_{K_t}(C_{r1}, C_{r2}, \dots, C_{rn}, \quad nonce + 2) \quad (4.12)$$

The server checks the secret values  $(C_{r1}, C_{r2}, \dots, C_{rn})$  and sends the client only the files whose secret values match the ones sent back by the user. The user now can decrypt the files by using the secret key  $K_s$  extracted from  $X_{ri}$  for each file. Figure 4.2 illustrates the secure search procedure.

#### 4.6 Constructing a DCS file with Integrity and Authenticity Proofs

The DCS file concept creates a secure container that includes all the previous modules. A DCS file encapsulates an original data file before sending it to a cloud server and the encapsulation keeps the data file secure from any unauthorized access including the cloud server. Only authorized users can search and access the file content according to the attached security policies presented in the  $C_r$  and  $X_r$  that are set and managed mainly by the file owner. The creation process for a DCS file is assumed to be conducted in a fully trusted machine which is owned by the data owner. Before creating a DCS file there are four essential operations:

1. encrypting the original file content by a symmetric encryption algorithm (see Figure 4.3 block A)
2. using the CRT to create the shared value  $X_r$  (see Figure 4.3 block C)
3. generating the secret keywords for search capability (see Figure 4.3 block D)
4. creating integrity and authenticity protections for the encrypted original data file (see Figure 4.3 block B). The encrypted file is hashed and then the resulting digest value is digitally signed by encrypting it with the file owner's private key  $K_{prvO}$ .

Then, a DCS file is created from these four operations. Figure 4.3 (block F) shows the output structure of a DCS file. Only authorized users are able to obtain the original file from a DCS file.

All the encrypted keywords,  $c_i$  where  $i=1,2,\dots,z$  and  $z$  is the number of keywords, are attached to the DCS file along with the associated random numbers  $R_i$ .

The DCS file now can be sent to the cloud environment for storage. The privacy, integrity and authenticity of the original file are securely preserved and do not rely on the cloud server to provide the security requirements, except some of the operations in executing these requirements.

## Chapter 4: A Data Centric Solution to Cloud Privacy and Security Issues

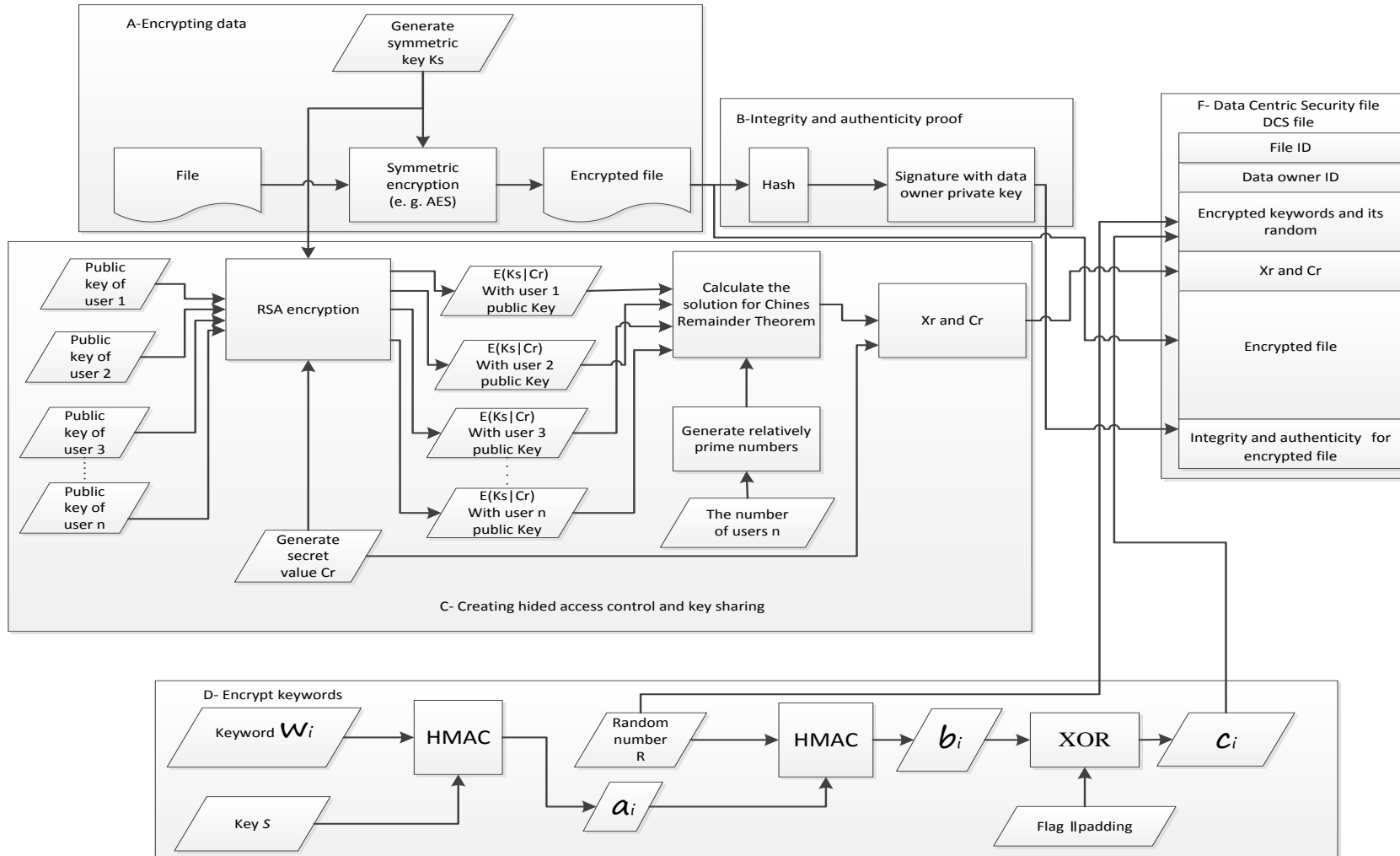


Figure 4.3 Constructing the DCS file

#### 4.7 Privacy Preserving and Integrity of the Proposed Solution

This section provides a summary of the features of the proposed solution in providing data privacy and security for users' data stored in the cloud. Compared to previous work in [87, 88], the proposed solution is more secure because we use a different symmetric key to encrypt each file and a different secret value to enforce the access control for each file. This improvement is reflected in several security and privacy aspects as listed below:

- A user cannot access the encrypted data before he/she provides securely a secret value to the server
- The identity of a user is always hidden.
- If a symmetric key  $K_s$  and/or secret value  $C_r$  of one file get compromised, other files remain secure.
- The proposed solution combines access control and sharing the secret key in one mechanism using the CRT which results in minimizing the computational and management overhead.
- The proposed solution is more robust against possible collusion attacks between revoked users and a cloud server. Because each file is encrypted with a different symmetric key, the server cannot collude with a revoked user to reveal the data content if the revoked user has not accessed it before revocation. Especially, if a revoked user has accessed the DCS file and revealed the symmetric key  $K_s$ , the data owner can delete the file and can use a new key  $K_s$  to create a new DCS file. Therefore, if the server is prevented from keeping a copy of the deleted DCS files, then it cannot collude with revoked users.
- When adding a new user to the list of authorized users who can access a file, the data owner does not need to exchange the secret value  $C_r$  with the server. Only the shared value  $X_r$  has to be exchanged. Hence, the possibility of compromising the value  $C_r$  is reduced. In addition, for revoking a user, the data owner only needs to change the  $C_r$  of that file because no other files use the same old value.
- When the data owner adds himself or herself as an authorized user for all the DCS files, the data owner and authorized users only need to keep their public-key pairs safe.

- The proposed solution provides integrity and authenticity verifications for each file and the parameters do not require changing when revoking an existing user and granting access to a new user.
- By using the proposed solution, all the security parameters are independently attached to each DCS file. This allows both a cloud provider and a user to deal with each DCS file more flexibly and efficiently. For example, either the user or the server can move the DCS file normally without worrying about its security or access control policies as these features are already part of it and does not depend on outside facilities. For instance, the DCS file's security is not dependent on a database or system access control thus it is not required to consider these facilities when moving data within the cloud or between clouds.

In general, computing the CRT solution is efficient compared to other cryptography methods used in cryptography access control as it uses simple modular operations without requiring exponential operations [111].

#### 4.8 Summary

This chapter has presented a solution that makes use of the DCS approach for cloud computing. The solution is designed as modules where each module provides a set of security services that are mainly managed by a data owner. The first module is for encrypting the data before it is outsourced. The second module enables a more efficient and effective method of managing the access control policies which are presented by the secret value and shared value used for sharing and access control of the data. The third module is used to provide secure search capability for the encrypted data by generating encrypted keywords. The fourth module produces integrity and authenticity proof ability. Then the outputs of all the previous modules are used to create a DCS file. The use of a DCS file is the result of finding a solution that enhances privacy and security suitable for the cloud computing environment. By the use of a DCS file, the proposed solution is able to preserve privacy, integrity and authenticity of data outsourced to the cloud even from the cloud provider itself with minimal impact in functionality as the encrypted data can be searched and shared by authorized users. More importantly, the security features offered in the DCS file rely on the cryptography algorithms used in the solution and managed by the data owner.

## Chapter 5: Implementation Issues and Evaluation

### 5.1 Introduction

This chapter examines the implementation issues of the proposed solution in Chapter 4. The objective of this chapter is to evaluate the effects of using this solution on the client and server sides mainly in terms of computation and storage resources. The implementation tools and experiment environment are described in Section 5.2. The implementation of the operations required for creating a DCS file on the data owner side and the operations required on an authorized user side are described in the Client Side Implementations section. In Section 5.4, the implementation and experiments in searching keywords on DCS files stored on a server side are discussed. The client side implementation issues and experiments results are discussed further in Section 5.5. Summary and conclusions of this chapter are provided in the last section.

### 5.2 Implementation Tools and Experiment Environment

The implementation tools and experiment environment were prepared for a server and a client side. The Java language was used for the implementation and was based on Eclipse IDE for the Java Developers' version: 1.4.2.20120213-0813. For the sever side, a virtual server platform representing a cloud server was installed in a dedicated hardware server for this implementation. The hardware server had the following specifications:

- Manufacturer: Supermicro
- Model (Motherboard) : H8DMR-82
- CPU: 2 × Dual-Core AMD Opteron <sup>TM</sup> Processor 2214
- Total memory: 12GB
- Total hard-disk storage: of 268GB (4 × 73GB Seagate SCSI U320 hard-disk drives configured in a spanned structure).

The VMware VSphere 4.1.0 was the virtualization platform for creating the virtual server. VMware is one of the global leaders in virtualization and cloud infrastructure [115]. The hardware server was located on the Intranet within the corporate network of University of Western Sydney at Penrith campus and the virtual server housed with the hardware server can be accessed via the Internet. The virtual server was configured with these specifications:



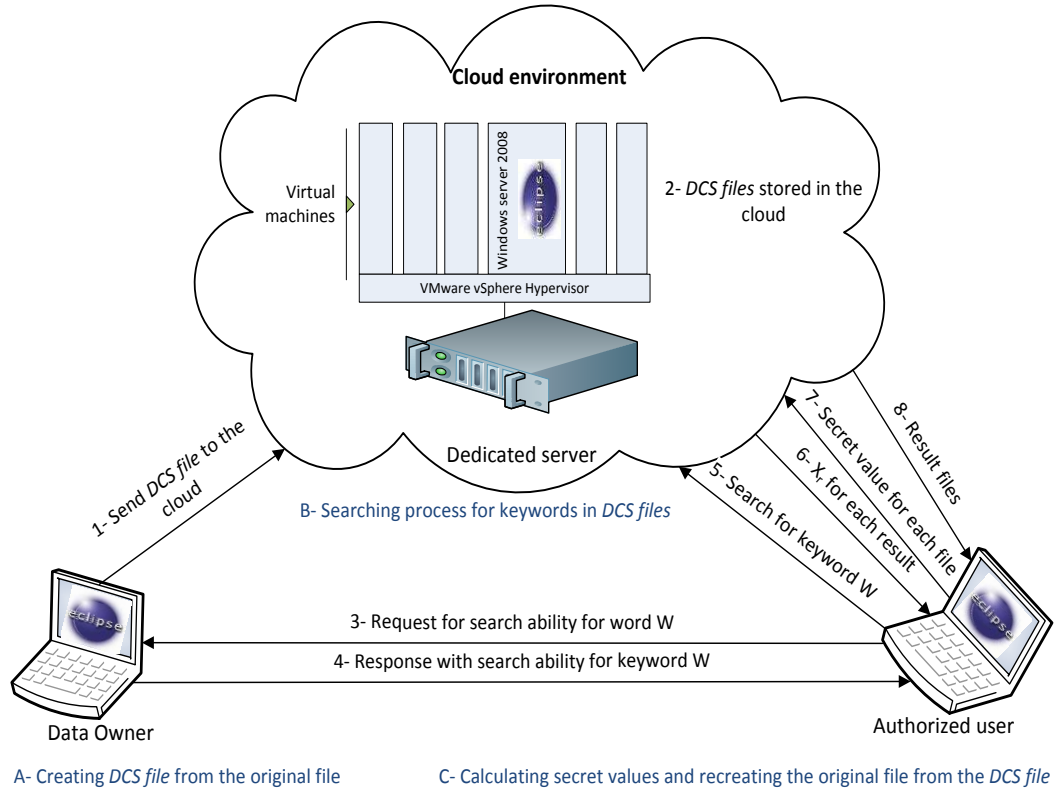
- OS: Windows Server 2008 R2 Enterprise 64-bit Edition
- Virtual Memory: 4GB
- Virtual storage space: 104GB.
- Processor: dual-core AMD Opteron™ processor 2214 2.2GHz.

Other virtual machines were installed in the same dedicated hardware server to create a virtual environment simulating a virtualisation environment used in a public cloud. As discussed in Chapter 2, in a public cloud, several clients share the same hardware server by having their virtual machines running on the same hardware server. These virtual machines are only isolated virtually by using a virtualization platform. In a commercial environment, a more powerful virtualisation platform such as VMware [115] is usually used.

Another physical computer was used in the experiment. The computer was used for simulating both the data owner side and an authorized user side. It was a notebook with the following specifications:

- Processer: Intel CORE i7 CPU 2.2GHz
- Memory: 8GB
- OS: Microsoft Windows 7 Professional 32-bit

For the data owner side, the operations for creating a DCS file before outsourcing it to the cloud server were implemented and tested. For an authorized user, the implementation and testing were conducted for the operations by calculating the value  $C_r || K_s$  from the shared value  $X_r$  using Equation (4.4) and by decrypting the data from a DCS file that contains the encrypted data. At the server side, the searching for keywords in a DCS file was implemented and carried out. Figure 5.1 shows the setup of the experiment environment, the messages passed amongst the entities, and the main operations on each side. In the implementation stage, one of the tasks was to measure the storage overhead and the computation overhead for carrying out the DCS operations.



**Figure 5.1 Experiment Environment for the Proposed Solution**

### 5.3 Client Side Implementations

The client side can be the data owner or an authorized user. Many functions, such as the asymmetric encryption algorithm, symmetric encryption algorithm and computing a hash of a file are used by both the data owner and the authorized user. Finding the CRT solution and encrypting the keywords are only done by the data owner. The followed sections describe the implementations of the required functions for the client side:

#### 5.3.1 Symmetric Encryption of the Original File

Encrypting the data with a symmetric encryption algorithm is the most important operation. For strong security, the data must be encrypted even if they will remain within the data owner premises. Therefore, the first action after classifying data as confidential data is encrypting data with an adequate encryption algorithm and key strength. In general, the symmetric encryption algorithm is used instead of asymmetric encryption for encrypting a large amount of data because it is more efficient. In this implementation, the files are encrypted by using the Advanced

Encryption Standard (AES) algorithm provided by an open source AES Crypt program for Windows platforms [116], using 256-bit key length, stronger key length for this algorithm. The next section describes the implementation of the CRT algorithm used to share the  $K_s$  with the secret value  $C_r$ .

### 5.3.2 Implementing the Calculating CRT Solution

The CRT is used to share, for each file, two secret values among authorized users; the symmetric key  $K_s$  used to encrypt a certain file and the secret value  $C_r$  used to enforce access control policies for that certain file. In the implementation, the length of  $C_r$  value should be fixed for all files so it is easy to distinguish between it and the symmetric key  $K_s$  when the two values are concatenated as one value  $C_r || K_s$ . These two values are encrypted with one of the asymmetric encryption algorithms  $E_{K_{pub} i}$  with the public key of each authorized user  $u_i$ , resulting in cipher-texts of  $a_i = \left( E_{K_{pub} i}(C_r || K_s) \right)$  for  $i=1, 2, \dots, k$  where  $k$  is the number of authorized users for a certain file. The resulting cipher-text for each user authorized to access the file will be the input for the CRT operation to calculate the shared value  $X_r$  for that file using Equation (4.3) in Section 4.2. Then the implementation requires finding the solution of the CRT simultaneous congruences represented in Equation (4.3). Consequently, for example, if we use the RSA algorithm to encrypt the value  $C_r || K_s$ , the minimum length of the result of the ciphertext will be at least 1024 bits. For implementation, the Java platform provides the class library `java.math.BigInteger` to deal with long integer values with length more than 64 bits [117]. In addition, the CRT requires a function to generate relative prime numbers  $n_i = n_1, n_2, \dots, n_k$  where  $k$  is the number of authorized users. The generation of relative prime numbers can be done once for all users and then assign each user a unique one. The program used to solve the CRT simultaneous congruences is based on Garner's algorithm [6] and the Extended Euclidean Algorithm (EEA) [104] is used for calculating the modular multiplicative inverse. The source codes for these two algorithms are available as an open source program in [118]. However, the original source codes are implemented to support integers with a 64-bit length. For the implementation of the proposed solution, the source codes are modified, based on the class library `java.math.BigInteger`, to support integers longer than 64 bits, classified in Java as `BigIntegers`. Therefore, the resulting program for solving the CRT can handle integers with length more than 64

bits. The CRTforBigInteger.java and cryptoBig.java are the two Java class files that contain the upgraded program codes for finding the CRT solution and for calculating the modular multiplicative inverse. The source codes for these two Java class files are documented in Appendix A.

The function CRTforBigInteger.findSolution(BigInteger[] a, BigInteger[] n) is used to find the CRT solution where a and n are the input BigInteger arrays of the  $a_i$  and  $n_i$  values. Table 5.1 shows the source code for finding the CRT solution that supports BigIntegers. In the findSolution function, the modular multiplicative inverse is calculated by using the function cryptoBig.inverse(BigInteger p, BigInteger g), as shown in Table 5.2. The returned value of the inverse function is the  $g^{-1} \pmod{p}$ . The Greatest Common Divisor (GCD) of two integers can be calculated by using the function BigInteger p.gcd(BigInteger g) included in the Class java.math.BigInteger library.

**Table 5.1 The Code for Finding CRT Solution**

```
import java.math.BigInteger;
public class CRTforBigInteger {
// Finding CRT solution Xr given Xr = a1 (mod n1), x = a2 (mod n2) ...
public static BigInteger findSolution(BigInteger[] a, BigInteger[] n)
{
    if(a==null || n ==null)
    {
        System.out.println("a or n must have at least 2 elements");
        return null;
    }
    if(a.length<2 || n.length<2)
    {
        System.out.println("Length of a or n are less than 2");
        return null;
    }
    if(a.length != n.length)
    {
        System.out.println("Length of a and n are not same");
        return null;
    }
}
```

```

        BigInteger x = a[0];
        BigInteger nInverse, y, z;
        BigInteger ni = BigInteger.ONE;
        int i = 0;
        while(i < a.length-1)
        {
            ni=ni.multiply(n[i]);

//check if the n's are relatively prime
if(!BigInteger.ONE.equals(n[i+1].gcd(ni))){
    System.out.println("The n's are not relatively prime->" + n[i+1] + ", " + ni);
        return null;
    }

    // calculating the modular multiplicative inverse
    nInverse = cryptoBig.inverse(n[i+1], ni);
    if(nInverse.compareTo(BigInteger.ZERO) == -1 )
        nInverse = nInverse.add(n[i+1]);
        z = a[i+1].subtract(x);
        z = z.multiply(nInverse);
        y = z.remainder(n[i+1]);
        if(y.compareTo(BigInteger.ZERO) == -1) // y < 0
            y = y.add(n[i+1]);
            x = x.add(ni.multiply(y));
            i++;
        }
        return x; // CRT solution  $X_r$ 
    }
}

```

Table 5.2 The Code for Calculating the Modular Multiplicative Reverse

```

//returns  $g^{-1} \pmod{p}$ 
    public static BigInteger inverse (BigInteger p, BigInteger g)
    {
        //g inverse =  $g^{p-2} \pmod{p}$ 

        BigInteger[] gInverse = ExtendedEuclid(p, g); //crypto.fastSq(p, g, p-2);
        if (gInverse[2].compareTo(BigInteger.ZERO) != -1)
            //return  $p+gInverse[2]$ ;
    }

```

```

        return p.add(gInverse[2]);
    return gInverse[2];

}

public static BigInteger[] ExtendedEuclid(BigInteger a, BigInteger b)

/* This function will perform the EEA to find the GCD of a and b. We
assume here that a and b are non-negative (and not both zero). This
function also will return numbers j and k such that d = j*a + k*b where
d is the GCD of a and b.*/
{
    BigInteger[] ans = new BigInteger[3];
    BigInteger q;
    if (b.equals(BigInteger.ZERO)){
        ans[0] = a;
        ans[1] = BigInteger.ONE;
        ans[2] = BigInteger.ZERO;
    }
    else
    {
        /* Otherwise, make a recursive function call */
        q = a.divide(b);
        ans = ExtendedEuclid (b, a.remainder(b));
        BigInteger s = ans[2].multiply(q);
        BigInteger temp = ans[1].subtract(s); // - ans[2]*q;
        ans[1] = ans[2];
        ans[2] = temp;
    }
    return ans;
}

```

### 5.3.3 Asymmetric key Encryption

The implementation used the RSA as the asymmetric key encryption algorithm to encrypt the value  $C_r || K_s$  for each authorized user. The length of the  $K_s$  can be 128, 192 or 256 bits and that for the secret value  $C_r$  should be chosen so the total length of  $C_r || K_s$  will be less than 1024 bits. This ensures that the encrypted value of  $C_r || K_s$  is limited to 1024 bits. Moreover, limiting the length of the  $C_r || K_s$  value will result in reducing the computational overhead of the RSA encryption and decryption operations. The Java platform provides the `java.security` package which contains

several class libraries that include functions used for generating RSA key pairs and RSA encryption/decryption operations [119]. From these functions, two Java class files were programmed: the class `GenerateRSAkey.java` and class `RSAforBytes.java`. The class `GenerateRSAkey.java` is for generating RSA pair keys and storing them in a file. The class `RSAforBytes.java` contains two function: `rsaEncrypt(byte[] data, String PubKeyFile)` for encrypting an array of bytes and `rsaDecrypt(byte[] data, String PrivKeyFile)` for decrypting this array of bytes. The codes of the two classes are listed in Appendix A.

Figure 5.2 shows some results of generating RSA key pairs and the encryption of the  $C_r || K_s$  value using the generated keys. The  $C_r || K_s$  value was selected to be '10444332252559723399888232537479'. This integer value was converted from `BigInteger` to byte array because the `rsaEncrypt` function is programed to encrypt data in a byte array format. The transformation resulted in 14 bytes and the encryptions of this value for ten users took 15 milliseconds, as shown in the program snapshot in Figure 5.2. The resulting encrypted value, for each user, has a length of 128 bytes (1024 bits) because the public key used was generated with a length of 1024-bit. Each encrypted output has to be converted back to `BigInteger`. Then the resulting encrypted `BigInteger` values will be used as input to the function `CRTforBigInteger.findSolution(BigInteger[] a, BigInteger[] n)` where the array `a[]` contains the values of  $a_i$  and `n[]` contains the values of  $n_i$  as represented in Equation (4.1) where  $a_i = \left( E_{K_{pub\ i}}(C_r || K_r) \right)$  and  $n_i$  is the relative prime for user  $u_i$  for  $i=1, 2, \dots, k$  where  $k$  is the number of authorized users for that certain file. Then, the resulting CRT solution  $X_r$  from `findSolution` will be attached to the protected data as part of the DCS file.

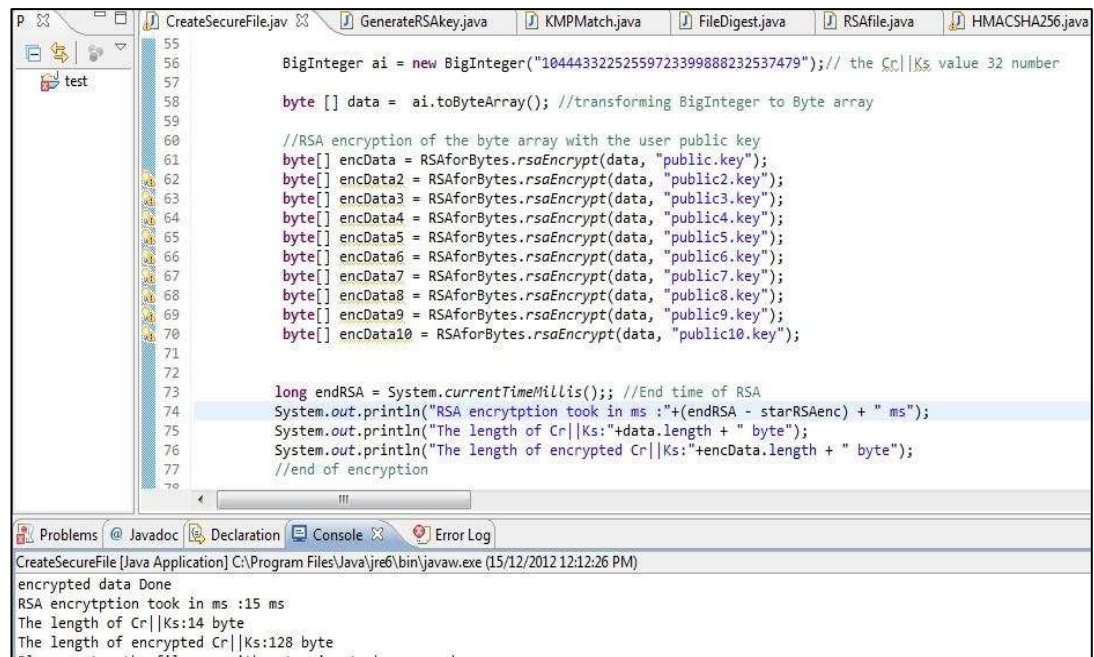


Figure 5.2 Snapshot of the RSA Encryptions for Ten Users

### 5.3.4 Encrypting the Keywords

The search keywords are encrypted by using Hash-based Message Authentication Code (HMAC) with the SHA256 hash algorithm in the implementation. The Java class HMACSHA256.java contains a function `encode(String key, String data)` that calculates the HMAC-SHA256 value for string input data with a key that is inputted as a string. The encode function returns the encrypted string in a byte array format with a length of 256 bits. The code is based on the classes `javax.crypto.Mac` and `javax.crypto.spec.SecretKeySpec` both of which are provided by the Java platform. Table 5.3 shows the code of the encode function. The key used for this function is the same for all keywords.

Table 5.3 The Code for Calculating HMAC for a Keyword

```

import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;

public class HMACSHA256 {
public static byte[] encode(String key, String data) throws Exception {
    // Specifying the Hash algorithm to be used
    Mac sha256_HMAC = Mac.getInstance("HmacSHA256");

```

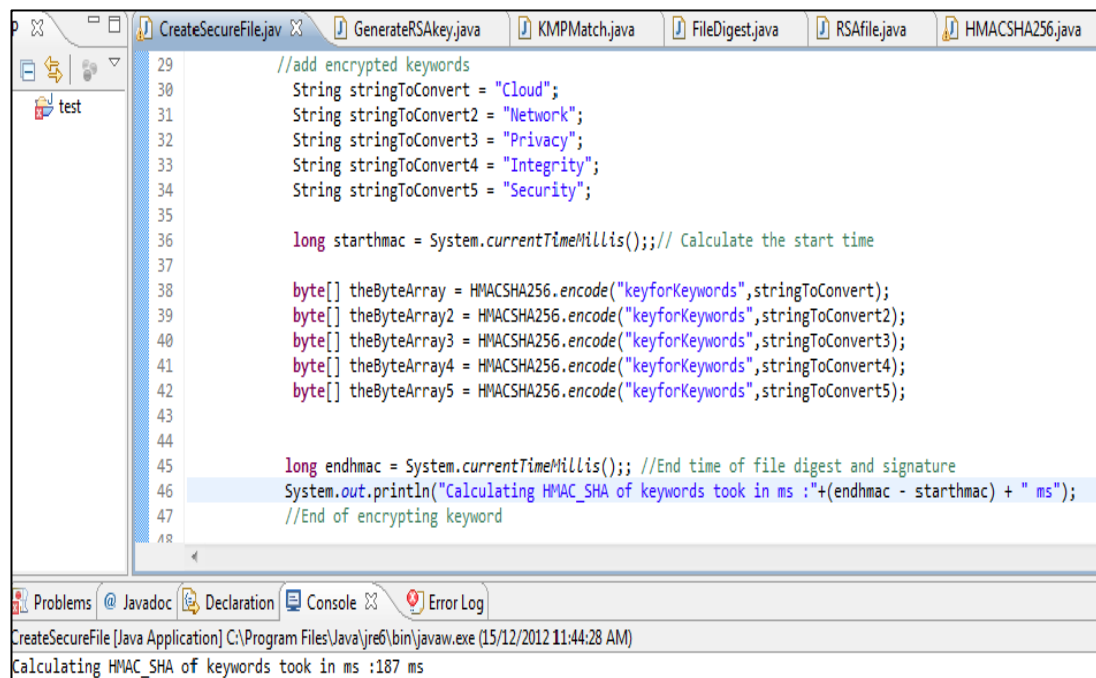


```

SecretKeySpec secret_key = new SecretKeySpec(key.getBytes(),
"HmacSHA256");
//Generating the key based on the entered key string
sha256_HMAC.init(secret_key);
return sha256_HMAC.doFinal(data.getBytes()); //Conducting the Hashing
    } }

```

Figure 5.3 shows a snapshot of running the program code for encrypting five keywords using the encode function. The encryption took 187 milliseconds.



**Figure 5.3 Snapshot of Calculating HMAC-SHA256 for Five Keywords**

### 5.3.5 Calculating the Integrity and Authenticity Proof

Integrity and authenticity proof is calculated by hashing the encrypted file and then encrypting the resulting digest by the data owner's private key. The function `createFileHash(String filename, String method)`, in class `Hash.java`, is used to calculate the digest of a file by using one of the hash algorithms, such as SHA-1, SHA-256 and MD5 (see the code of `Hash.java` in Appendix A). The code of the class `Hash.java` is based on the `java.security` package. The `createFileHash` takes as input the file name and the name of the hash algorithm (e. g. SHA-1, SHA-2 or MD5).

After calculating the digest of the given file according to the specified hash algorithm, the function returns the digest value in a byte array format. The length of the resulting digest depends on the algorithm used. For instance, when using SHA-256 the resulting digest length is 256 bits. The resulting digest is then encrypted by using the RSA encryption function `rsaEncrypt(byte[] data, String PrivKeyFile)` where the key is the private key of the data owner (i.e. data owner digital signature). The digital signature value will be 1024 bits when using a 1024-bit private key and 2048 bits when using a 2048-bit key. Table 5.4 contains the program code for calculating the value, `FSignature` in the program code, used for integrity and authenticity proof.

**Table 5.4 The Code for Calculating the Integrity and Authenticity Proof**

```
// Calculating the input file digest and data owner signature
String PrivKeyFile = "private1.key"; //Data owner private key

//Calculating the file digest
byte[] FileDigest = Hash.createFileHash(InputFile, "SHA-256");

// (Signature) Encrypting the file digest with data owner private key
byte[] FSignature = RSAforBytes.rsaEncrypt(FileDigest, PrivKeyFile);
```

### 5.3.6 Creating a Secure File (*DCS file*) by the Data Owner

The creation of a DCS file is mainly by the previously described operations namely: symmetric encryption, finding the CRT solution, HMAC-SHA256, RSA encryption and signing the encrypted message digest of the input file. All outputs of these operations and related information are included in constructing the content of a DCS file. The class `CreateSecureFile.java` is created and designed to be used for creating the DCS file from an encrypted file. Before running the `CreateSecureFile` program, the original file must be already encrypted by the `AES Crypt` program and the following parameters are defined, with examples of assigning a value to a parameter given for each of the parameters:

1. Predefining the data owner name or ID.

```
String Owner = "Nabil"; //Data owner ID
```

2. Predefining the search keywords' strings, number and the encryption key.

```
//add keywords
String stringToConvert1 = "Cloud";
String stringToConvert2 = "Privacy";

int keyword_no = 2; //Number of encrypted keywords

byte[] theByteArray =
HMACSHA256.encode("keyforKeywords",stringToConvert);
byte[] theByteArray2 =
HMACSHA256.encode("keyforKeywords",stringToConvert2);
```

3. Predefining the  $C_r||K_s$  value and the  $C_r$  separately.

```
BigInteger C_K = new BigInteger("10444332252559723399888232537479");

long Cr = 1044433225; // this number is the Cr value
```

4. Predefining all the relative prime values (i.e. BigInteger n[] array elements) of the authorized users of this file.

```
BigInteger n1 = new
BigInteger("215143111331331113151515157719135");//User1

BigInteger n2 = new BigInteger("235311001"); //User2
```

5. Predefining the files containing the public key for each of the authorized users.

```
byte[] encData = RSAforBytes.rsaEncrypt(data, "public.key"); //User1

byte[] encData2 = RSAforBytes.rsaEncrypt(data, "public2.key"); //User2
```

6. Predefining the file containing the private key of the data owner.

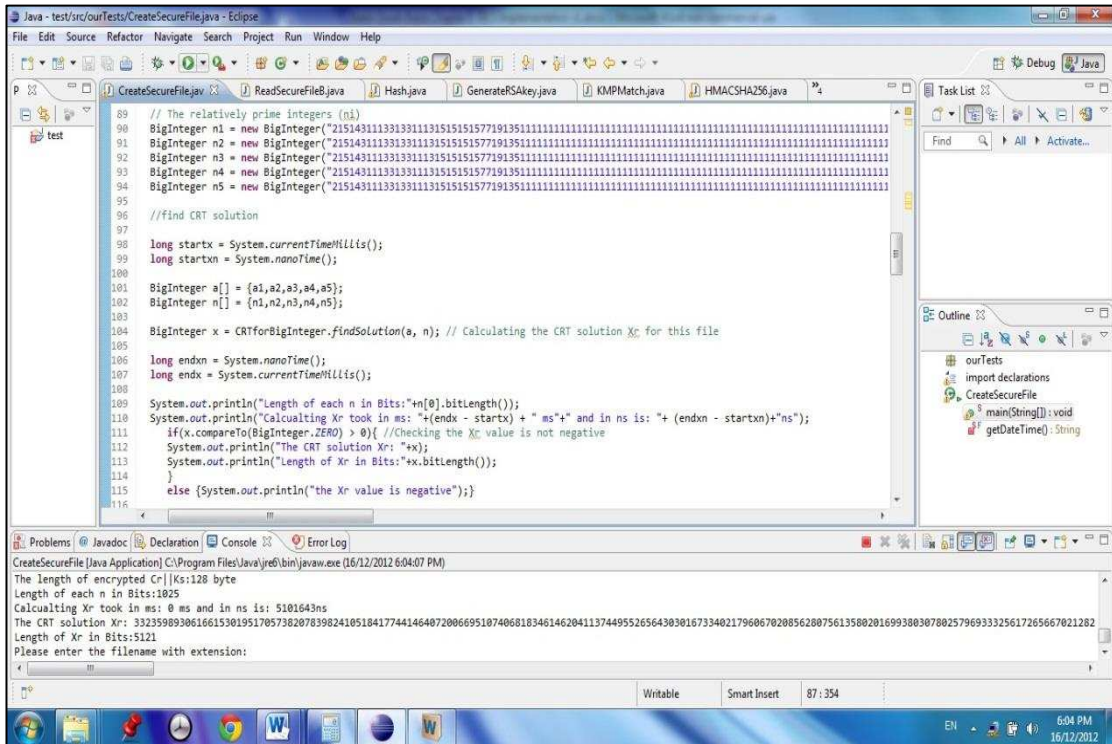
```
String PrivKeyFile = "privateOwner.key"; //Data owner private key
```

7. Predefining the hash algorithm (e.g. SHA-1 or SHA-256) for digesting the encrypted file.

```
byte[] FileDigest = Hash.createFileHash(InputFile, "SHA-256");
```

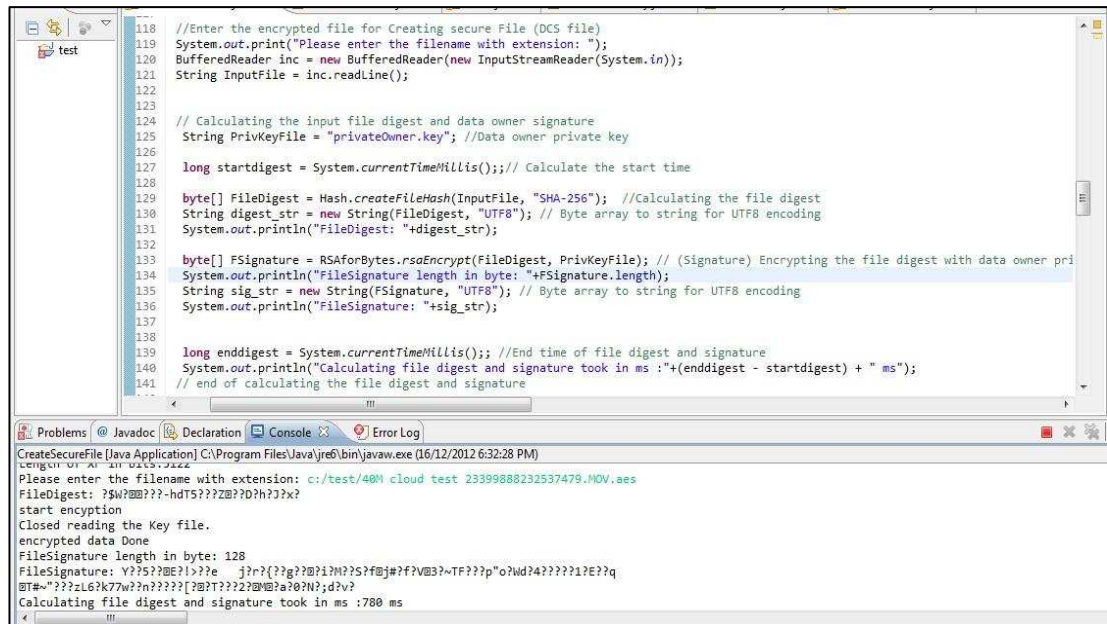
After defining the above parameters the program is ready for execution. The execution first calculates the HMAC-SHA256 of the keywords as shown in Figure 5.3 above. Then it encrypts the value  $C_r||K_s$  with each user's public key, an example of which is shown in Figure 5.2. The resulting encrypted values with the users' relatively prime integers are used to find the shared value  $X_r$  by using the function `CRTforBigInteger.findSolution`. For example, calculating the CRT solution

for five users, where the length of their relative prime numbers was 1025 bits and that of the  $C_r||K_s$  value was 1024 bits, took about 5.1 milliseconds and the resulted  $X_r$  was 5122 bits, as shown in Figure 5.4.



**Figure 5.4 Snapshot of Calculating CRT Solution for Five Users**

The program then prompts the user to enter the encrypted file which will be converted to a DCS file with the security parameters attached. The program calculates the digest of the input file and signs the digest with the data owner's private key. Figure 5.5 shows an example of using the developed program to calculate the digest of a file, with a size of 40.7 MBs, and to create a digital signature of the file. It took 780 milliseconds when the SHA-256 hash algorithm and 1024-bit RSA private key were used for this particular file.



**Figure 5.5 Snapshot of Calculating File Digest and Signature for 40.7 MB File**

After calculating all the required parameters, the program prompts the user to enter the path, name and extension for the output DCS file, with the default extension 'dcs'. Then the program starts creating the DCS file as the output. Table 5.5 shows the format type and length of parameters to be inserted into a DCS file. The complete program code of the CreateSecureFile.java can be found in Appendix A.

**Table 5.5 Code Description of Constructing the DCS file**

Code	Description
RandomAccessFile f = new RandomAccessFile(file, "rw");	Creates a random access file stream to write to the file specified by the File argument (i.e. file) which is the output DCS file.
f.writeInt(keyword_no)	Writes the number of keywords attached (this integer number occupies 4 bytes). This number will be used to calculate the range of searching on this DCS file and to move the pointer, when reading the DCS file, after the encrypted keywords.

<pre>f.write(theByteArray); f.write(theByteArray2); f.write(theByteArray3);</pre>	Writes the encrypted keywords resulting from the HMACSHA256. The total size occupied by the keywords = 32 byte × number of key words.
<pre>f.writeUTF(Owner);</pre>	Writes the data owner name as UTF-8 encoding string.
<pre>f.writeLong(Cr);</pre>	Writes the value of $C_r$ (this long integer number occupies 8 bytes). For more security this value should be hashed or encrypted with the public key of the cloud provider before inserted to the DCS file.
<pre>byte [] xr = x.toByteArray(); int xlength = xr.length; f.writeInt(xlength);</pre>	Converts the $X_r$ value from BigInteger to byte array format. Then calculates the number of bytes representing the $X_r$ . Then Writes this number as an integer to the output file (this occupies 4 bytes). These actions keep the $X_r$ value able to be easily read later.
<pre>f.write(xr);</pre>	Writes the $X_r$ value as byte array. The number of its bytes is specified in the previous step.
<pre>f.write(FSignature);</pre>	Writes the 32 bytes array resulting from signing the digest of the input encrypted file (the integrity and authenticity proof )
<pre>RandomAccessFile in = new RandomAccessFile(InFile, "r");</pre>	Creates a random access file stream to read from the file specified by the File argument (i.e. InFile) which is the input encrypted file.
<pre>byte[] buf = new byte[1024]; int len; while ((len = in.read(buf)) &gt; 0) { f.write(buf, 0, len); }</pre>	This set of codes reads the content of the input encrypted file (in) and writes it to the output DCS file (f). The write process starts at the last location of the output file pointer.

In the Java programming language, to insert a new bit stream in the beginning of a file, it is required to create a new file with the new metadata information of the file and the original file content [120]. Therefore, creating a DCS file requires creating a new file with the parameters listed in Table 5.5 as metadata and the encrypted file content. This will add a computational overhead, particularly for large files, in copying the content of an encrypted file to the DCS file.

### 5.3.7 Operations at the Authorized User Side

An authorized user requires mainly four operations, i.e., computing the  $C_r||K_s$  value from the shared value  $X_r$ , recreating the original encrypted file from the DCS file, verifying the integrity and authenticity proof, and finally decrypting the encrypted file to produce the file in plaintext. All these operations are implemented in one class file, ReadSecureFileB.java. The user has to define the following parameters. Examples in assigning a value to a parameter are given for each of the parameters:

1. The unique relatively prime of the user.

```
BigInteger N = new BigInteger("215143111331331113151515157719135");
```

2. The file containing the user's private key.

```
String PrivKeyFile = "private.key";
```

3. The file containing the data owner public key.

```
String PublicKeyFile = "publicOwner.key";
```

The user's relatively prime and private key are used to compute the  $C_r||K_s$  from the  $X_r$  value and the data owner's public key is used for verifying the integrity and authenticity of the DCS file.

#### 5.3.7.1 Computing the Secret value $C_r||K_s$ from the Shared Value $X_r$

Table 5.6 lists the names of the Java functions and the names of the output from the function, together with the description for the function of each of the Java functions for finding the secret value  $C_r||K_s$  from the shared value  $X_r$ .



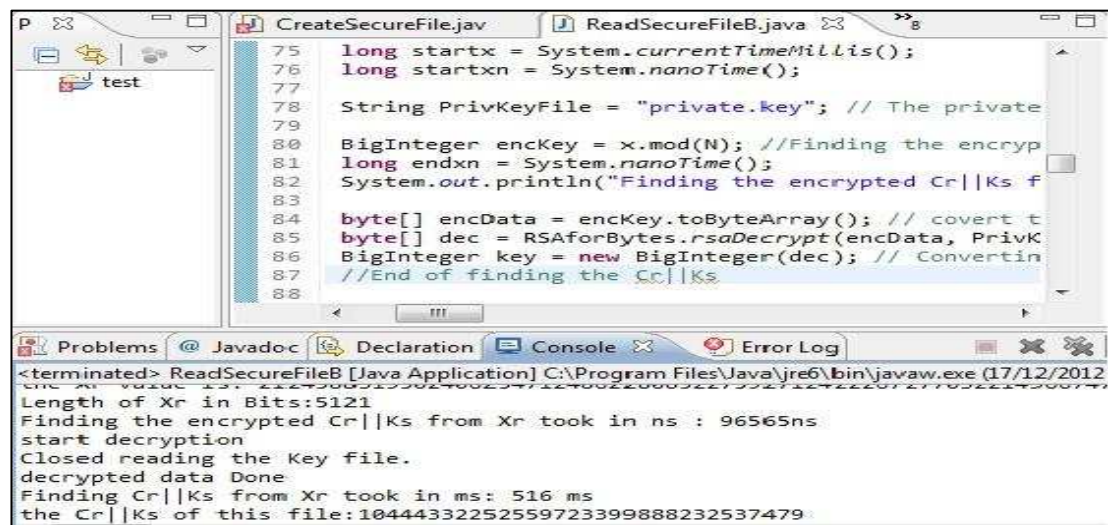
**Table 5.6 Code Description of Computing the Secret value  $C_r||K_s$  from the Shared Value  $X_r$** 

Code	Description
<code>BigInteger x = new BigInteger(xr);</code>	Converts $X_r$ (x) from byte array to BigInteger.
<code>BigInteger encKey = x.mod(N);</code>	Computes the encrypted value of $C_r  K_s$ for the user $u_i$ that has provided the (N) relative prime number. This represent the following equation: $E_{K_{pub\ i}}(C_r    K_s) = X_r \bmod N$ Where the $E_{K_{pub\ i}}(C_r    K_s)$ is the encrypted $(C_r    K_s)$ value by the public key of user $u_i$ .
<code>byte[] encData = encKey.toByteArray();</code>	Converts BigInteger result to a byte array so it can be decrypted in the next step.
<code>byte[] dec = RSAforBytes.rsaDecrypt(encData, PrivKeyFile);</code>	Decrypts the encrypted $(C_r    K_s)$ value by using the private key of the user $u_i$ . This represents the equation: $C_r    K_s = D_{K_{prv\ i}} \left( E_{K_{pub\ i}}(C_r    K_s) \right)$
<code>BigInteger key = new BigInteger(dec);</code>	Converts the $C_r    K_s$ value resulting from the private setup from byte array to BigInteger.

After revealing the  $C_r||K_s$  value, an authorized user  $u_i$  now can use the  $C_r$  for completing the access control procedure and the  $K_s$  for decrypting the original encrypted file after it has been downloaded. Since the communication part of the access control procedure is not covered in this implementation, the code in Table 5.6 is included in ReadSecureFileB.java. Therefore, the code reads the  $X_r$  directly from the DCS file while the server should do that and send it to the user during the access control procedure. The focus in this part of the implementation was on measuring the computation overhead caused from computing  $C_r||K_s$  at the user side. As shown in



Table 5.6 above, the program code mainly executes two functions. The first function is related to the CRT algorithm which is based on a simple modular function (i.e.  $X_r \bmod n_i$ ) and hence the calculation can be done very fast. For instance, using this function, calculating the  $X_r$  value with a 5121-bit length, for five users, and  $n$  with a 1024-bit length took about 96.565 microseconds, as shown in Figure 5.6. The second function is the RSA decryption operation for the resulting encrypted value from the first function, as shown in Table 5.6 above. This decryption operation took about 516 milliseconds, as shown in Figure 5.6.



### Figure 5.6 Snapshot of Computing the $C_r||K_s$ Value from $X_r$ Value

### 5.3.7.2 Reading the Embedded Information in the DCS file and Recreating the Original Encrypted File

The DCS file contains information, such as the  $X_r$  value, and the integrity and authenticity proof, in addition to the encrypted data file content. When the class `ReadSecureFileB.java` is running, the program will ask the user to enter the input DCS file and then the name and extension for the output file. The program code to read the information from the input DCS file and then output the encrypted data file is shown in Table 5.7.

**Table 5.7 Code Description of Reading DCS file and Recreating the Original File**

Code	Description
<code>RandomAccessFile f = new RandomAccessFile(file, "r");</code>	Creates a random access file stream to read from the file specified by the File argument (file) which is the input DCS file.
<code>RandomAccessFile out = new RandomAccessFile(Dest, "rw");</code>	Creates a random access file stream to write to the file specified by the File argument (Dest) which is the output file of the encrypted file content in the DCS file.
<code>int keyword_no = f.readInt();</code>	Reads the number of encrypted keywords attached. (first 4 bytes of the file)
<code>int l=keyword_no*32+4; f.seek(l);</code>	Calculates the length in bytes from the file beginning to the end of the encrypted key words. Then moves the file pointer to that end.
<code>String File_owner = f.readUTF();</code>	Reads the string of the data owner name. The file pointer moves to end of the bytes representing this string.
<code>long Cr = f.readLong();</code>	Reads the 8 bytes representing the $C_r$ value in Long integer.
<code>int x1 = f.readInt();</code>	Reads the 4 bytes representing the number of bytes of the $X_r$ in an Integer.
<code>byte[] xr = new byte[x1]; f.read(xr);</code>	Specifies the length of the $X_r$ byte array from the previous step. Then reads the $X_r$ according to its length.
<code>byte[] FSignature = new byte[128]; f.read(FSignature);</code>	Specifies the length of the byte array that represents the file signature (i.e. 128 bytes * 8 = 1024bit). Then reads file signature byte array according to the specified length.
<code>byte[] buf = new byte[1024]; int len; while ((len = f.read(buf)) &gt; 0)</code>	This set of codes reads the content of the input DCS file (f) that represents the encrypted file content and writes it to the

<pre>{     out.write(buf, 0, len); }</pre>	output file (out). This code resulted in creating the original encrypted file separated from its DCS file.
--------------------------------------------	------------------------------------------------------------------------------------------------------------

The code shown in Table 5.7 is for allowing an authorized user or the cloud provider to read from a DCS file, the data owner's name,  $X_r$ ,  $C_r$  and the signed digest of the encrypted data file. However, only authorized users can obtain the key  $K_s$  which can then be used to decrypt the encrypted file. In addition, an authorized user can also verify the data file's integrity and authenticity.

### 5.3.7.3 Verifying the Integrity and Authenticity

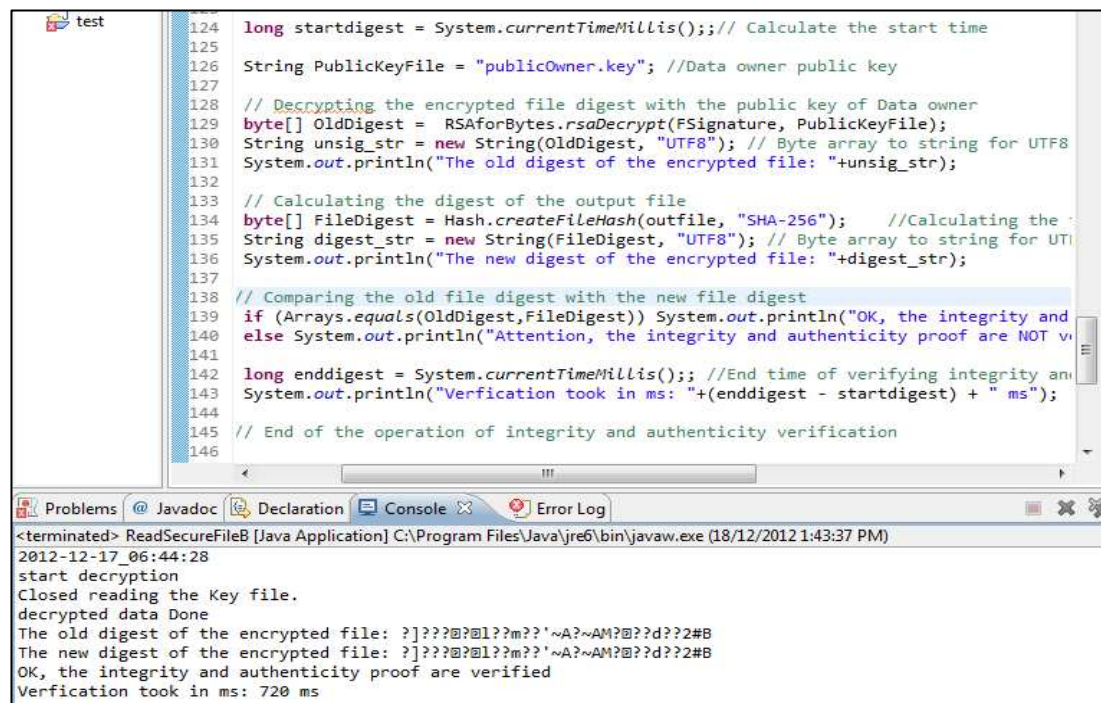
After obtaining the encrypted data file from a DCS file and reading the attached file signature, an authorized user can verify the integrity and authenticity of the encrypted data file. The code used for verifying integrity and authenticity is shown in Table 5.8.

**Table 5.8 Code Description of Verifying Integrity and Authenticity**

Code	Description
<pre>String PublicKeyFile =     "publicOwner.key";</pre>	Specifies the file containing the data owner public key.
<pre>byte[] OldDigest =     RSAforBytes.rsaDecrypt(FSignature,     PublicKeyFile);</pre>	Decrypts the attached encrypted (signed) file digest by using the data owner public key. The result of this operation represents the 256 bit of the original encrypted file digest in a byte array (i.e OldDigest).
<pre>byte[] FileDigest =     Hash.createFileHash(outfile, "SHA-     256");</pre>	Computes the recreated encrypted file (i.e. outfile) digest by hashing it with SHA-256 algorithm. The result of this operation represents the digest of the recreated file in a byte array (i.e. FileDigest).
<pre>If(Arrays.equals(OldDigest,FileDige</pre>	Compares the attached digest (i.e

<pre> st)) System.out.println("OK, the integrity and authenticity proof are verified"); else System.out.println("Attention, the integrity and authenticity proof are NOT verified"); </pre>	<p>OldDigest) with the new calculated digest (i.e. FileDigest). If they are equal, the encrypted file integrity is maintained and originated from its data owner. Otherwise, there is a concern about the file integrity.</p>
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

The verifying process incurs a computational overhead and the amount of overhead is according to the file size. For example, verifying the integrity and authenticity of a file with a size of 40.7 MBs took about 720 milliseconds, as shown in Figure 5.7.



```

124 long startdigest = System.currentTimeMillis(); // Calculate the start time
125
126 String PublicKeyFile = "publicOwner.key"; //Data owner public key
127
128 // Decrypting the encrypted file digest with the public key of Data owner
129 byte[] OldDigest = RSAforBytes.rsaDecrypt(FSignature, PublicKeyFile);
130 String unsig_str = new String(OldDigest, "UTF8"); // Byte array to string for UTF8
131 System.out.println("The old digest of the encrypted file: "+unsig_str);
132
133 // Calculating the digest of the output file
134 byte[] FileDigest = Hash.createFileHash(outfile, "SHA-256"); //Calculating the
135 String digest_str = new String(FileDigest, "UTF8"); // Byte array to string for UTF8
136 System.out.println("The new digest of the encrypted file: "+digest_str);
137
138 // Comparing the old file digest with the new file digest
139 if (Arrays.equals(OldDigest,FileDigest)) System.out.println("OK, the integrity and
140 else System.out.println("Attention, the integrity and authenticity proof are NOT v
141
142 long enddigest = System.currentTimeMillis(); //End time of verifying integrity and
143 System.out.println("Verification took in ms: "+(enddigest - startdigest) + " ms");
144
145 // End of the operation of integrity and authenticity verification
146

```

Problems @ Javadoc Declaration Console Error Log

<terminated> ReadSecureFileB [Java Application] C:\Program Files\Java\jre6\bin\javaw.exe (18/12/2012 1:43:37 PM)

2012-12-17\_06:44:28

start decryption

Closed reading the Key file.

decrypted data Done

The old digest of the encrypted file: ????@l??m??'~A?~AM?@??d??2#B

The new digest of the encrypted file: ????@l??m??'~A?~AM?@??d??2#B

OK, the integrity and authenticity proof are verified

Verification took in ms: 720 ms

**Figure 5.7 Snapshot of Verifying Integrity and Authenticity**

#### 5.3.7.4 Decrypting the encrypted file

After verifying the integrity and authenticity of an encrypted data file, the encrypted data file can be decrypted by using the AES Crypto software with the key  $K_s$  for this file. Most of the computational overhead comes from decrypting the encrypted file particularly for large size files as the results shown in Section 5.5.2.

## 5.4 Server Side Implementation and Experiments

At the server side, the main operation implemented is the search process on a DCS file for the attached encrypted keywords. Class `search.java` includes the code required to search DCS files. To speed up the searching process, the Knuth–Morris–Pratt (KMP) string searching algorithm is used. The KMP algorithm is one of the most efficient search algorithms and it can find the exact pattern matching in a range of digital data [121]. The class `KMPMatch.java` contains an open source code implementation of the KMP algorithm [122]. The `KMPMatch.java` provides the function:

```
indexOf(byte[] data, byte[] pattern)
```

The function uses the KMP algorithm to search for the first occurrence of a specific byte array pattern (i.e. `byte[] pattern`) on a given byte array range (i.e. `byte[] data`). This function is used in the `search.java` class to search for a given encrypted keyword in the DCS files on a cloud server. The implemented search algorithm, based on the KMP algorithm, is designed to search only on the part of the DCS file that contains the array of bytes containing the encrypted keywords. For instance, for five encrypted keywords, the range will be 164 bytes as the first 4 bytes of the DCS file represent the number of keywords and each encrypted keyword occupied 32 bytes as described in Table 5.5. Limiting the search range is very important to reduce the time required for searching especially for large files. If the search process is not limited to the range that contains the encrypted keywords, the search of the whole DCS file will waste computation resources and time as there are no searchable encrypted keywords in the encrypted data file.

Before running the Class `search.java`, the path of the directory that contains the DCS files to be searched has to be specified, as shown below:

```
String path = "c:\\search_test";
```

The code shown in Table 5.9 asks for a keyword string and then calculates the HMAC-SHA256 value for the keyword. The resulting value is put in a byte array `theByteArray`. This code is included in the `search.java` class for experiment purposes. In real life, this code is run at the data owner side. The cloud server only receives the

encrypted keywords (i.e. theByteArray) which are used to search for the DCS files containing the keywords.

**Table 5.9 The Code of Calculating the HMAC-SHA256 for an Input keyword**

```
// Buffer to read input text
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
try {
    System.out.println("Write the Keyword you want to search for:");
    String stringToConvert = br.readLine();
    byte[] theByteArray;
    theByteArray = HMACSHA256.encode("keyforKeywords",stringToConvert);
} catch (Exception e1) {
    e1.printStackTrace();
}
```

After entering the keyword and calculating its HMAC-SHA256, the search.java opens each of the DCS files and reads the number of encrypted keywords embedded in the file. From the integer number of keywords attached to a DCS file, the code calculates the range of the search and then searches for the keyword using the KMP algorithm. If there is a pattern matched, the code prints on the screen the path and name of the DCS file containing that keyword. The code repeats the same procedure for all the DCS files in the specified directory. For all the files, the code shows the time it takes to search for each of the files. The complete code of the class search.java is listed in Appendix A.

In the following experiment, the directory specified for the search contained 217 files, some of which were non DCS files, so the code cannot determine their search range because non DCS files do not contain the number of the keywords. Hence the search range was limited to five encrypted keywords. Consequently, all the DCS files on the search directory had five encrypted keywords or less. Figure 5.8 shows a snapshot of some of the outputs of the searching process in the directory containing files of different sizes. The search of each file took less than 1 millisecond and the total time for searching on all the 217 files took about 9633 milliseconds. As a result of limiting the search range, the search process is not affected by the size of a file as the results in this experiment show.



```

Write the Keyword you want to search for:
Integrity
Searchin start at time:2012-12-12_09:54:08
Searchin of the keyword in file no: 0 took in ns :662095 ns
Searchin of the keyword in file no: 1 took in ns :37993 ns
Searchin of the keyword in file no: 2 took in ns :38273 ns
Searchin of the keyword in file no: 3 took in ns :37435 ns
Searchin of the keyword in file no: 4 took in ns :37155 ns
Searchin of the keyword in file no: 49 took in ns :37994 ns
Searchin of the keyword in file no: 50 took in ns :37993 ns
Searchin of the keyword in file no: 51 took in ns :38831 ns
Keyword found in: c:\test\encrypted\key121.txt.aes
Searchin of the keyword in file no: 52 took in ns :271822 ns
Searchin of the keyword in file no: 53 took in ns :38552 ns
Searchin of the keyword in file no: 54 took in ns :38273 ns
Searchin of the keyword in file no: 55 took in ns :38274 ns
Searchin of the keyword in file no: 56 took in ns :38553 ns
Searchin of the keyword in file no: 57 took in ns :37994 ns
Searchin of the keyword in file no: 58 took in ns :8184280 ns
Searchin of the keyword in file no: 210 took in ns :7543 ns
Searchin of the keyword in file no: 211 took in ns :8940 ns
Searchin of the keyword in file no: 212 took in ns :7543 ns
Searchin of the keyword in file no: 213 took in ns :7822 ns
Searchin of the keyword in file no: 214 took in ns :7822 ns
Searchin of the keyword in file no: 215 took in ns :7543 ns
Searchin of the keyword in file no: 216 took in ns :7543 ns
Searchin of the keyword in file no: 217 took in ns :7543 ns
Searchin on all files took in ms :9633 ms
End time:2012-12-12_09:54:08

```

Figure 5.8 Snapshot of Running Search Process

## 5.5 Results and Discussion

This section discusses the results and implementation issues with regard to the overheads, in terms of storage and computation. The computation overhead is observed as the time taken for computation. The results are based on the assumption that all the parameter values are already generated, such as the RSA pair keys, the relative prime numbers,  $C_r$  and  $K_s$  values. This section focuses on the data owner side and authorized user side as the server side has already been covered in the previous Section 5.4.

### 5.5.1 At the Data Owner Side

This section discusses the implementation issues and overhead in creating a DCS file at the data owner's machine. The data file will first be encrypted with an AES (other encryption symmetric algorithms could be used) and the resulting encrypted data file will become the main body of the DCS file. Table 5.10 shows storage overheads and execution times of encrypting different types of data files using the AES algorithm with a key size of 256-bit. The storage overhead is around 300 bytes for all files. The computational overhead is increased with the increase of the file size to reach more than a minute for a file with 571 MB size. This operation is the core for any method based on encrypting data before the data are sent to a cloud. In addition, it is important to choose a strong encryption algorithm and a longer key for achieving higher security. However, data owners have the flexibility, according to their security

requirements, to reduce the computational overhead by using a shorter key or a weaker encryption algorithm with less computational overhead. In general, the encryption process incurs the most computational overhead at the data owner side in the proposed method in this thesis.

**Table 5.10 Storage and Time Overheads for the AES Encryption**

File name	Size in bytes	Size after encryption in bytes	Increase of size in bytes	AES encryption in seconds
Any.docx	14,074	14,386	312	<1
plan.vsd	44,032	44,338	306	<1
m.jpg	65,812	66,130	318	<1
Wildlife.wmv	26,246,026	26,246,338	312	1.6
40Mtest.mov	42,750,493	42,750,802	309	2.3
Case.avi	137,237,016	137,237,330	314	14.8
Tripoli_rev.mov	598,787,322	598,787,634	312	86.7

#### 5.5.1.1 Calculating the $X_r$ value

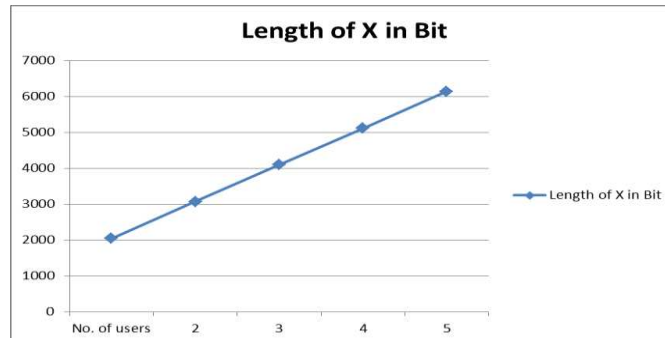
Calculating the CRT solution to determine the  $X_r$  value is the essential part of the proposed solution. The key sharing and access control policies are based on this value. The CRT calculation is not affected by the file size, but is affected by the number of users and the asymmetric encryption algorithm used to encrypt the  $C_r||K_s$  value. Table 5.11 shows the storage and time overheads in calculating the CRT solution  $X_r$  for different numbers of users from two to six. These calculations were done when a  $C_r||K_s$  value was encrypted with the 1024-bit RSA and each relative prime number was 1022-bit. The results show that the CRT operation itself requires a low execution time. For instance, for six users, computing the CRT solution took about 6 milliseconds. The execution time in calculating the  $X_r$  value is mainly caused by the RSA encryption process and increases with increase of the number of users. However, because the RSA encryption process is done for a fixed length value of  $C_r||K_s$  the total execution time in finding  $X_r$  is still low and satisfactory. For example, the total time was 31 milliseconds for six users and the total time increased by about



2 milliseconds for each extra user. The most overhead caused by the CRT operation is the storage overhead which increases with the increase in the number of users. Figure 5.9 shows, based on the results in Table 5.11, that the size of  $X_r$  increases linearly at about 1000 bits for each extra user, when the RSA key length is 1024 bits. For example, for six users, the size of  $X_r$  was 6142 bits.

**Table 5.11 Storage and Time Overheads for Calculating the Shared Value  $X_r$**

No of users	RSA (1024 bit) encryption for $C_r  K_s$ in ms	Calculating CRT Solution $X_r$ in ms	Total time for calculating $X_r$ in ms	$X_r$ length in bits
2	21	3	24	2049
3	22	4	26	3073
4	23	4	27	4097
5	24	5	29	5118
6	25	6	31	6142



**Figure 5.9 The Size of  $X_r$  in Bits Related to Number of Users**

### 5.5.1.2 Adding searchable encrypted keywords

Adding encrypted keywords to a DCS file is necessary for searching capability, as discussed in Section 4.3.

Table 5.12 shows the storage and execution time overheads incurred by adding from one to five encrypted keywords. The results show a moderate computational overhead, around 200 milliseconds, and the storage overhead increases 256 bits for

each extra keyword. Only one round of HMAC is done in the implementation while the proposed solution in Chapter 4 requires two rounds and an XOR operation. The reason for that is to reduce the computational overhead and also to use the KMP algorithm for searching on the server. When the XOR operation is used in encrypting the keywords, the KMP algorithm cannot be applied, because when using the XOR operation, the search process will involve the XOR operation before searching for the same pattern which is in conflict with the KMP mechanism of searching. Moreover, using one round of HMAC with SHA-256 or SHA-512 should provide a sufficient security level. The implementation allows users to use HMAC-SHA512 for more security but the storage overhead will increase twice. For example, when using HMAC-SHA256 for five keywords, the total storage required is 1280 bits. When using HMAC-SHA512 for five keywords, the storage required will be 2560 bits. The data owner has to trade-off between the security level required for the keywords and overheads in storage and execution time. Also the number of keywords required should be decided carefully to reduce the overhead without losing the usability of the searching capability.

**Table 5.12 Storage and Time Overheads for Adding Encrypted Keywords**

No Keywords	HMAC-SHA256 duration in ms	total output size in bits
1	196	256
2	196	512
3	201	768
4	197	1024
5	199	1280

### **5.5.1.3 Adding the Integrity and Authenticity Proof and Creating the DCS file**

Adding the integrity and authenticity proof to the DCS file is important but it causes an extra computation and storage overhead. Table 5.13 shows the overheads in storage and execution times in calculating the integrity and authenticity proof for files of different types and sizes. The hash function used for digesting the encrypted

file was SHA-256 and the RSA key for encrypting the digest was 1024-bit. Computational overhead is increased with the increase of the size of a file, while the storage overhead is fixed to 1024 bits which is the result of encrypting the 256-bit digest of the file by the RSA with a 1024-bit key length.

**Table 5.13 Storage and Time Overheads for Adding Integrity and Authenticity proof**

<b>Input file name</b>	<b>Size in bytes</b>	<b>Time of calculating digest and signature in ms</b>	<b>Storage overhead in bytes</b>
Any.docx.aes	14,386	51	128
plan.vsd.aes	44,338	51	128
m.jpg.aes	66,130	133	128
Wildlife.wmv.aes	26,246,338	1910	128
40Mtest.MOV.aes	42,750,802	775	128
Case.avi.aes	137,237,330	8714	128
tripoli_rev.mov.aes	598,787,634	50977	128

#### **5.5.1.4 Creating the DCS file**

Creating the DCS file requires that all the parameters are already computed. The total storage overhead depends on the number of users authorized to access the file, the number of keywords attached and the specifications of the algorithms used. Table 5.14 shows that the total storage overhead in creating a DCS file is a constant 952 bytes for files of different sizes. The DCS files were created with five attached encrypted keywords and to be shared by five authorized users. The algorithms were used with specifications defined in the implementations, i.e., RSA-1024, AES-256, SHA-256. For large files, this storage overhead can be neglected.

**Table 5.14 Total Storage Overhead for Creating DCS file for Five Users and with Five Keywords**

<b>Input file name</b>	<b>Size in bytes</b>	<b>Size of DCS file in bytes</b>	<b>Storage overhead in bytes</b>
Any.docx.aes	14,386	15,337	952
plan.vsd.aes	44,338	45,290	952
m.jpg.aes	66,130	67,082	952
Wildlife.wmv.aes	26,246,338	26,247,290	952
40Mtest.MOV.aes	42,750,802	42,751,754	952
Case.avi.aes	137,237,330	137,238,282	952
tripoli_rev.mov.aes	598,787,634	598,788,586	952

The process of creating a DCS file requires copying the encrypted file content to the new DCS file. This operation causes extra time as shown in Table 5.15. This time can be avoided if both the AES encryption and calculation of the encrypted file digest are done simultaneously in creating a DCS file. In other words, the original data file is encrypted in blocks, and then each block is digested, and used to construct the DCS file in one round. Therefore, this time is due to implementation and is not counted as an overhead as far as the proposed solution is concerned.

**Table 5.15 Time Overhead for Copying the Encrypted File Content to DCS file**

<b>Input file name</b>	<b>Size in bytes</b>	<b>Copying the encrypted file content to the DCS file in ms</b>
Any.docx.aes	14,386	32
plan.vsd.aes	44,338	43
m.jpg.aes	66,130	91
Wildlife.wmv.aes	26,246,338	253
40Mtest.MOV.aes	42,750,802	390
Case.avi.aes	137,237,330	1183
tripoli_rev.mov.aes	598,787,634	79460

The main operations that the data owner performs to create a DCS File are the AES encryption, calculating the  $X_r$  value, calculating the HMAC for the keywords, calculating the digest of the encrypted file, and encrypting the digest. Table 5.16 shows the total execution times taken by these operations for different files sizes. The  $X_r$  value was calculated for five users and five encrypted keywords were attached.

**Table 5.16 Total Time Overhead Required for Creating DCS file**

<b>Input file size in bytes</b>	<b>AES in seconds</b>	<b>Calculating <math>X_r</math> for 5 users in seconds</b>	<b>Calculating file digest and Signature in seconds</b>	<b>HMAC-SHA 256 for 5 Keywords in seconds</b>	<b>Total time in seconds</b>
14,074	<1	0.029	0.003	0.199	~ 1
44,032	<1	0.029	0.004	0.199	~ 1
65,812	<1	0.029	0.005	0.199	~ 1
26,246,026	1.6	0.029	0.474	0.199	2.302
42,750,493	2.3	0.029	0.720	0.199	3.248
137,237,016	14.8	0.029	5.050	0.199	20.078
598,787,322	86.7	0.029	17.897	0.199	104.825

The total execution time was less than 4 seconds for files less than 50 MBs and about 20 seconds for a file with 130-MB size. For a file with 571 MBs, the total execution time was about 1.7 minutes. From the results shown in Table 5.16, it is clear that computing the parameters required for access control, key sharing and searching capability have less computation impact. In contrast, the AES encryption followed by calculating the integrity and authenticity proof has the most computation impact. Therefore, in the proposed solution, the symmetric encryption operation incurs the most computational overheads especially for large files. However, any solution based on encrypting data before the data are sent to the cloud requires at least one round of encrypting the data.

### 5.5.2 At the Authorized User Side

An authorized user has to conduct three operations; computing  $C_r||K_s$  value from  $X_r$  value, verifying the integrity and authenticity of an encrypted data file, and decrypting the encrypted data file. Table 5.17 shows the execution times of the operations for different sizes of DCS files. The AES decryption needs the most execution time followed by verifying the integrity and authenticity of the encrypted data file. Computing the  $C_r||K_s$  value has a small execution time. Recreating the original encrypted data file from a DCS file can be done at the server side and then the server sends the original encrypted data file with its signed digest instead of the DCS file. Hence, the overhead of copying the encrypted content from the DCS file to new recreated file is avoided at the user side.

**Table 5.17 Time Execution of the Operations at the User Side for different DCS files**

Input DCS file name	Size in bytes	Creating original encrypted file in ms	Verifying integrity and authenticity in ms	Finding $C_r  K_s$ from $X_r$ in ms	AES decryption in seconds
Any.docx.dcs	15,337	214	3	29	<1
plan.vsd.dcs	45,290	220	4	29	<1
m.jpg.dcs	67,082	350	5	29	<1
Wildlife.wmv.dcs	26,247,290	1448	474	29	3.5
40Mtest.MOV.dcs	42,751,754	1457	720	29	2
Case.avi.dcs	137,238,282	11520	5050	29	32
tripoli_rev.mov.dcs	598,788,586	13198	17897	29	32.7

### 5.6 Summary and Conclusion

This chapter discusses the implementation issues of the proposed solution in Chapter 4. The focus was on the storage and computation overhead caused by using the proposed solution at the client side and server sides. An experimental platform was setup for both sides and the Eclipse platform for Java programming language was used for implementing the main operations of the proposed solution. For the data owner

operations, the main code for creating a DCS file from an encrypted file was written in the java class file named `CreatSecureFile.java`. This class computes the shared value  $X_r$  for any given users by calling the `findSolution` function from the `CRTforBigInteger.java` class that was implemented to compute the CRT solution when using integers larger than 64 bit length. Also the `CreatSecureFile` was implemented to compute other security parameters namely, the encrypted keywords for search capability and, integrity and authenticity proof. Then it attached all these parameters to a DCS file in a way that can be easily read later.

The `ReadSecureFile.java` was implemented to read the security parameters in the DCS file and recreate the original encrypted file form. Also it contains the operations required at the authorized user side namely, revealing the secret values  $C_r$  and  $K_s$  from the  $X_r$  and verifying the integrity and authenticity of the encrypted file. Open source AES Crypto software was used for encrypting the original file at the data owner side and then for decrypting it at the user side.

The `Search.java` class file was implemented at the server to search on the DCS files for a given keyword. For better performance when a DCS file was being constructed, information about how many keywords and the length of them was attached as well. Hence, the search function can calculate the range within the DCS file that contains the encrypted keywords so the search process will be limited to that range. Thus, the search time will not depend on the file length. Moreover, the KMP searching algorithm was used to enhance the search speed.

The results from the implemented operations show that the proposed solution is simple and does not require complex operations. Moreover, the storage overhead in creating a DCS file is low compared to the features provided. These features are: creating a DCS file with its search capability, hidden access control policies and integrity and authenticity proof regardless of where it is stored within the cloud. The proposed solution can be practically implemented with minimal computation and storage overheads. The proposed solution is proven to be very efficient as it does not require complex key derivation methods and the data file does not need to be encrypted more than once.

## Chapter 6: Conclusions and Future Work

This thesis investigates the Data Centric Security (DCS) concept as an important approach to enhance the security and privacy of stored clients' data in cloud computing systems, especially in the public cloud environment where data may be moved frequently from one cloud server to another and may be accessed by other end entities. This study focuses on securing the actual data from possible security risks in the cloud environment without fully relying on trusting the cloud provider or a third party. By contrast, approaches that focus on securing the hardware and applications handling the data in the cloud require the data owner to trust the cloud provider or a third party for providing the desired security and privacy requirements. As mentioned before, because of the nature of the public cloud, clients' data may be moved within the cloud environment between different providers or locations that have different security measures and privacy regulations. As discussed in Section 3.2, the DCS approach has not yet a standardized conceptual framework of applying it to the cloud computing paradigm and this study offers a contribution at this conceptual level.

This study proposes a conceptual framework for applying the DCS approach that aims to enhance the security and privacy of data in the cloud. The proposed framework is based on specifying the security requirements from actually within the data so the data are self-describing, self-protecting, self-defending throughout their lifecycle. The data owner retains the full responsibility of specifying and managing the security and privacy of their data outsourced to the cloud. Furthermore, the privacy of the data and the privacy of the users who access the data remain secure even from the cloud providers. The data security does not rely on trusting the cloud provider nor does it require a trusted third party. This proposed conceptual framework is expected to add a robust security level for the cloud data and consequently potential users may have more confidence to move their data to the cloud storages. As a result of the enhanced level of the security and privacy achieved by the use of the proposed framework, the benefits of using the cloud services can be achieved without losing control of the security and privacy of the data by the data owner or exposing the data to possible security and privacy breaches as opposed to keeping the data within the user's own private computing environment. In other words, the DCS approach attempts to provide cloud consumers with the ability to



protect their data as if the data were still under their control in terms of security and privacy. In Section 3.3, the expected benefits of applying the DCS approach to the cloud model were discussed. As discussed in Section 3.2, although several researchers' solutions in addressing cloud security concerns have been motivated by the DCS concept, there is no agreement amongst them as to any specified criteria and requirements of that concept. This research outlined in Section 3.2 provides a clearer and wider conceptual DCS framework for applying the DCS approach to the cloud model with three essential criteria. These three essential criteria are summarized below:

- All security measures and requirements are provided from within the data set itself.
- The data owner is responsible for configuring, managing and monitoring the data and their security specifications throughout the data's life time.
- The security and privacy of the data and users who access the data must not rely on the cloud providers or trusted third parties.

The second contribution of this work is the proposal of a solution based on the DCS approach to enhance the security and privacy of any type of data file, such as documents, images and videos, stored and shared in public cloud storage (see Chapter 4). The proposed solution is based on creating a file called a DCS file by the data owner for each data file to be outsourced to the cloud. Each DCS file contains the encrypted original data file and a set of security parameters that are used for enforcing access control policies, searching for keywords and verifying the integrity and authenticity of the data file. Exclusively, the data owner sets and manages these security parameters for each DCS file throughout its lifecycle. The DCS file keeps the attached data file secure within the cloud environment and only authorized users can access the encrypted data file within their trusted domains. Hence, the data remain secure against possible security risks from inside and outside the cloud environment even against possible malicious actions by the cloud provider itself.

Each DCS file has its own access control policies attached to it and the policies are hidden even from the cloud provider who is responsible for enforcing the access policies. In addition, the number and identities of the authorized users who can

access the data are hidden even from the cloud provider. The Chinese Remainder Theorem (CRT) is used efficiently to calculate a shared value  $X_r$  that contains two secret values. The proposed solution uses the CRT for the sharing of two secret values; one value is the secret key  $K_s$  for decrypting the data file and another secret value  $C_r$  for authenticating the authorized users within the access procedure without revealing the users' identities. As a result, the proposed solution combines the sharing of the secret key and the enforcement of the access control policies in one mechanism thus minimizing the computation and key management overheads. Each DCS file has its shared value  $X_r$  attached which embeds the specific access policies and the symmetric key  $K_s$  of this DCS file. Neither the data owner nor the users need to keep or exchange this key, as each key will be securely attached to its relevant file. Only authorized users can retrieve the  $K_s$  and  $C_r$  values from the shared value  $X_r$ . Hence only authorized users can access and decrypt the data file in the DCS file. Even the cloud server storing the DCS files cannot reveal the secret value  $K_s$  from the shared value  $X_r$ . Moreover, if one DCS file gets compromised, other DCS files will not be affected because each file has its unique secret values. As described in Chapter 4, only the data owner can change the access control policies of each DCS file by securely and efficiently changing its  $X_r$  value. The implementation and experiment results presented in Chapter 5 show that calculating  $X_r$  has low computational overhead.

The DCS file can also be securely searched by attaching encrypted keywords to it. Therefore, even non-text files, can be searched according to the secret keywords. The keywords are encrypted and attached to a DCS file by the data owner and only authorized users can search for the keywords in the DCS files. The attached keywords are encrypted by the data owner to hide them from unauthorized entities including the cloud provider. This search capability does not depend on a database stored at the cloud provider and only requires a normal encryption process and a simple search method. In the implantation phase, the search procedure is designed to speed up the search process by making use of an efficient string searching algorithm, i.e., the Knuth–Morris–Pratt (KMP) algorithm as discussed in Chapter 5. Results from the executions of the implementation show satisfactory search efficiency, while the keywords remain secure.

The proposed solution based on the DCS approach provides a platform independent of the computing environment and is suitable for the complex and dynamic nature of the cloud environment, especially in the case of the public cloud. The DCS file has its data file encrypted and the security requirements are attached to it as security parameters which are also referred to as security metadata. Each DCS file has its independent security parameters including the integrity and authenticity proof attached to it and can be individually managed by the data owner. Therefore, the data owners can be confident about the security of their DCS files even if the files are transferred between different cloud providers and different geographical locations. Regardless of the cloud server in which a DCS file is located in the cloud environment, authorized users can search, access and verify its integrity as these are embedded in the file and are independent of the computing environment. Moreover, when the cloud provider creates copies for backup or improving access by multiple users or for any other reasons, the security parameters of the original DCS file are maintained by the copied versions.

The results obtained from testing the implemented operations of the proposed solution show that the implementation, hence the corresponding solution, is simple and does not require complex operations that require high computational resources as well as high maintenance. In addition, the proposed solution can be practically used for any type of data file with moderate overheads in terms of storage and computational resources. In addition, the proposed solution allows a data owner to flexibly choose the encryption and hashing algorithms based on the required security strengths and applications.

A possible improvement on the proposed solution can be made so that it can be used for thin clients, such as smart phones and tablets. The main focus would be on reducing the consumption of the communication and computation resources. The major computation consumption in the proposed solution occurs from the encryption /decryption operations and the adding/verifying of the integrity and authenticity proofs, particularly for large data files. Hence, these operations require more improvement of the proposed method to be suitable to be used by thin clients.

Another security aspect which requires further work is the encrypted keywords. In the proposed solution, all the keywords are encrypted with the same secret key  $K_w$ .

As a consequence, the cloud provider is able to find the DCS files that contain keywords in common and can use that for statistical analysis to infer possible sensitive information about the data files' contents. The challenge is how to increase the security of the keywords and search results with a practical search performance.

For the long term, the current work will be expanded to include improving the proposed solution to overcome more technical challenges facing the DCS approach so it can be applied to more cloud service models. For example, one of the major challenges is how to legally modify the encrypted data file content without unencrypting it in the cloud. Another improvement which can be made concerns how the DCS approach can be integrated with other cloud security approaches while its major characteristics are maintained. For example, in improving the efficiency of searching the keywords, the method to embed and protect the keywords needs to be modified so that the cloud server may use the keywords to construct an index database which can be used for speeding up the search process. The challenge is how the keywords can remain secure from the cloud provider during the construction of the index database. In general, one of the most challenging aspects is how to conserve the important characteristics of the DCS approach proposed in this thesis when it is being expanded and generalized to cover more cloud computing applications and more general cloud computing environments.

## Appendix A: The Implementation Code

### The Client Side Code:

#### CreateSecureFile.java

```
package ourTests;

import java.io.BufferedReader;
import java.io.File;
import java.io.InputStreamReader;
import java.io.RandomAccessFile;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.TimeZone;
import ourTests.Hash;
import java.math.BigInteger;

public class CreateSecureFile {
    public static void main(String[] args) throws Exception{

        String Owner = "Nabil"; //Data owner ID

        //add keywords
        String stringToConvert1 = "Cloud";
        String stringToConvert2 = "Network";
        String stringToConvert3 = "Privacy";
        String stringToConvert4 = "Integrity";
        String stringToConvert5 = "Security";

        int keyword_no = 5; //Number of encrypted keywords

        long starthmac = System.currentTimeMillis();// Calculate the start time

        //Encrypting the keywords with HMAC-SHA256 algorithm
        byte[] theByteArray =
```

```

HMACSHA256.encode("keyforKeywords",stringToConvert1);
byte[] theByteArray2 =
HMACSHA256.encode("keyforKeywords",stringToConvert2);
byte[] theByteArray3 =
HMACSHA256.encode("keyforKeywords",stringToConvert3);
byte[] theByteArray4 =
HMACSHA256.encode("keyforKeywords",stringToConvert4);
byte[] theByteArray5 =
HMACSHA256.encode("keyforKeywords",stringToConvert5);

//End time of encrypting keywords
long endhmac = System.currentTimeMillis();
System.out.println("Calculating HMAC_SHA of keywords took in ms
:"+(endhmac - starthmac) + " ms");
//End of encrypting keyword

// The start time of RSA encryption
long starRSAenc = System.currentTimeMillis();

// the Cr||Ks value
BigInteger C_K = new BigInteger("10444332252559723399888232537479");
long Cr = 1044433225; // this number is the Cr value
System.out.println("The Cr value: "+Cr);

byte [] data = C_K.toByteArray(); //transforming BigInteger to Byte array

/* RSA encryption of the byte array Cr||Ks with each authorized the user
public key */
byte[] encData1 = RSAforBytes.rsaEncrypt(data, "public.key");
byte[] encData2 = RSAforBytes.rsaEncrypt(data, "public2.key");
byte[] encData3 = RSAforBytes.rsaEncrypt(data, "public3.key");
byte[] encData4 = RSAforBytes.rsaEncrypt(data, "public4.key");
byte[] encData5 = RSAforBytes.rsaEncrypt(data, "public5.key");

long endRSA = System.currentTimeMillis(); //End time of RSA
System.out.println("RSA encryption took in ms :"+(endRSA - starRSAenc) +
" ms");
System.out.println("The length of Cr||Ks:"+data.length + " byte");
System.out.println("The length of encrypted Cr||Ks:"+encData1.length + "

```

[illegible]

[illegible]



```

    }

    // Calculating the input file digest and data owner signature
    String PrivKeyFile = "privateOwner.key"; //Data owner private key

    long startdigest = System.currentTimeMillis(); // Calculate the start time

    //Calculating the file digest
    byte[] FileDigest = Hash.createFileHash(InputFile, "SHA-256");

    // (Signature) Encrypting the file digest with data owner private key

    byte[] FSignature = RSAforBytes.rsaEncrypt(FileDigest, PrivKeyFile);
    System.out.println("FileSignature length in byte: "+FSignature.length);

    //End time of file digest and signature
    long enddigest =
    System.currentTimeMillis();System.out.println("Calculating file digest and
    signature took in ms :"+(enddigest - startdigest) + " ms");
    // end of calculating the file digest and signature

    //output file for creating DCS file
    System.out.print("Please enter the output DCS filename with extension: ");
    BufferedReader outc = new BufferedReader(new
    InputStreamReader(System.in));
    String outfileExtension = outc.readLine();

    File file = new File(outfileExtension); //output file

    // Calculate the start time of creating DCS file
    long start = System.currentTimeMillis();
    System.out.println(getDateTime()); // show start time

    //Open output file for writing
    RandomAccessFile f = new RandomAccessFile(file, "rw");

    // Start writing the output DCS file
    f.writeInt(keyword_no); //The number of keywords

```

```

//writing encrypted keywords
        f.write(theByteArray);
        f.write(theByteArray2);
        f.write(theByteArray3);
        f.write(theByteArray4);
        f.write(theByteArray5);

        f.writeUTF(Owner); // writing the string of file owner name or ID

byte [] xr = x.toByteArray(); //Convert the BigInteger Xr to byte array
// calculating the length of the xr so we can read it later
int xlength = xr.length;
        f.writeLong(Cr); // this number is the Cr
        f.writeInt(xlength); // this is the length of Xr in bytes
        f.write(xr); //attaching the shared value Xr

//Attaching the digital file signature
        f.write(FSignature);

//reading the content of the input file
        RandomAccessFile in = new RandomAccessFile(InFile, "r");

        // Transfer bytes from in to out
        byte[] buf = new byte[1024];
        int len;

        while ((len = in.read(buf)) > 0) {

                f.write(buf, 0, len);
        }

        System.out.println("new DCS file length:"+f.length());

        f.seek(0);
        f.close();
        in.close();

        System.out.println("Creating teh DCS file Done");

```

```

        long end = System.currentTimeMillis();

System.out.println("Creating DCS file took in ms: "+(end - start) + "
ms");
        System.out.println(getDateTime()); // show end date and time

    }

    private final static String getDateTime()
    {
        DateFormat df = new SimpleDateFormat("yyyy-MM-
dd_hh:mm:ss");

        df.setTimeZone(TimeZone.getTimeZone("PST"));
        return df.format(new Date());
    }
}

```

## cryptoBig.java

```

package ourTests;
import java.math.BigInteger;

public class cryptoBig {

    //returns g^-1 (mod p)
    public static BigInteger inverse (BigInteger p, BigInteger g)
    {
        //g inverse = g ^ (p-2) (mod p)

        BigInteger[] gInverse = ExtendedEuclid(p, g); //crypto.fastSq(p, g, p-2);

        if (gInverse[2].compareTo(BigInteger.ZERO) != -1)
            return p.add(gInverse[2]);

        return gInverse[2];
    }
}

```

```

public static BigInteger[] ExtendedEuclid(BigInteger a, BigInteger b)
/* This function will perform the Extended Euclidean algorithm to find the
GCD of a and b. We assume here that a and b are non-negative (and not
both zero). This function also will return numbers j and k such that

$$d = j*a + k*b$$

where d is the GCD of a and b.
*/
{
    BigInteger[] ans = new BigInteger[3];
    BigInteger q;

    // if (b == 0)
    if (b.equals(BigInteger.ZERO)){
        ans[0] = a;
        ans[1] = BigInteger.ONE;
        ans[2] = BigInteger.ZERO;
    }
    else
    {
        /* Otherwise, make a recursive function call */
        q = a.divide(b);
        ans = ExtendedEuclid (b, a.remainder(b));
        BigInteger s = ans[2].multiply(q);
        BigInteger temp = ans[1].subtract(s); // - ans[2]*q;
        ans[1] = ans[2];
        ans[2] = temp;
    }
    return ans;
}
}

```

## RSAforBytes.java

```

package ourTests;

import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.IOException;

```

```

import java.io.InputStream;
import java.io.ObjectInputStream;
import java.security.spec.RSAPrivateKeySpec;
import java.security.spec.RSAPublicKeySpec;
import java.math.BigInteger;
import java.security.Key;
import java.security.KeyFactory;
import javax.crypto.Cipher;

public class RSAforBytes {

    static Key readKeyFromFile(String keyFileName) throws IOException {

        InputStream in = new FileInputStream(keyFileName);

        ObjectInputStream oin = new ObjectInputStream(new BufferedInputStream(
            in));

        try {

            BigInteger m = (BigInteger) oin.readObject();
            BigInteger e = (BigInteger) oin.readObject();
            KeyFactory fact = KeyFactory.getInstance("RSA");

            if (keyFileName.startsWith("public"))

                return fact.generatePublic(new RSAPublicKeySpec(m, e));

            else

                return fact.generatePrivate(new RSAPrivateKeySpec(m, e));

        } catch (Exception e) {

            throw new RuntimeException("Spurious serialisation error", e);

        } finally {
            oin.close();
            System.out.println("Closed reading the Key file.");
        }
    }
}

```

```

    }
}

// file_source is the key file name
public static byte[] rsaEncrypt(byte [] byteData, String file_source)

throws Exception {

    System.out.println("start encryption");

    Key pubKey = readKeyFromFile(file_source);

    Cipher cipher = Cipher.getInstance("RSA");

    cipher.init(Cipher.ENCRYPT_MODE, pubKey);

// use object for encryption
byte[] encryptedByteData = cipher.doFinal(byteData);
    System.out.println("encrypted data Done");
    return encryptedByteData;
}

// Use this PublicKey object to initialise a Cipher and decrypt some data

public static byte[] rsaDecrypt(byte[] encryptedByteData, String key_file)

    throws Exception {

    System.out.println("start decryption");
    Key priKey = readKeyFromFile(key_file);
    Cipher cipher = Cipher.getInstance("RSA");
    cipher.init(Cipher.DECRYPT_MODE, priKey);

// use object for encryption
byte[] decryptedByteData = cipher.doFinal(encryptedByteData);
    System.out.println("decrypted data Done");
    return decryptedByteData;
}
}

```

## GenerateRSAkey.java

```
package ourTests;

import java.security.KeyFactory;
import java.security.KeyPair;
import java.security.KeyPairGenerator;
import java.security.PrivateKey;
import java.security.PublicKey;
import java.security.spec.RSAPrivateKeySpec;
import java.security.spec.RSAPublicKeySpec;

public class GenerateRSAkey {
    public static void main(String[] args)
        throws Exception {

        generateKeys();
    }

    public static void generateKeys() throws Exception {

        KeyPairGenerator kpg = KeyPairGenerator.getInstance("RSA");

        kpg.initialize(1024); // 2048 or 1024

        KeyPair kp = kpg.genKeyPair();

        PublicKey publicKey = kp.getPublic();

        PrivateKey privateKey = kp.getPrivate();

        System.out.println("keys created");

        KeyFactory fact = KeyFactory.getInstance("RSA");

        RSAPublicKeySpec pub = fact.getKeySpec(publicKey,
            RSAPublicKeySpec.class);
```

```

    RSAPrivateKeySpec priv = fact.getKeySpec(privateKey,
        RSAPrivateKeySpec.class);

    // create files for the private and public keys
    RSAfile.saveToFile("public.key", pub.getModulus(),
    pub.getPublicExponent());

    RSAfile.saveToFile("private.key", priv.getModulus(),
    priv.getPrivateExponent());

    System.out.println("keys saved");
}
}

```

## HMACSHA256.java

```

package ourTests;

import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;

public class HMACSHA256 {

    public static byte[] encode(String key, String data) throws Exception {

        Mac sha256_HMAC = Mac.getInstance("HmacSHA256"); // Specifying the Hash
        algorithm
        SecretKeySpec secret_key = new SecretKeySpec(key.getBytes(),
        "HmacSHA256");
        sha256_HMAC.init(secret_key); //Generating based on the entered key string

        return sha256_HMAC.doFinal(data.getBytes()); //Conducting the Hashing
    }
}

```



## Hash.java

```
package ourTests;

import java.io.*;
import java.security.*;

public class Hash {

    public static byte[] createFileHash(String filename, String method) {
        try {
            try {
                InputStream fis = new FileInputStream(filename);

                byte[] buffer = new byte[1024];
                MessageDigest complete = MessageDigest.getInstance(method);
                int numRead = 0;
                while (numRead != -1) {
                    numRead = fis.read(buffer);
                    if (numRead > 0) {
                        complete.update(buffer, 0, numRead);
                    }
                }
                fis.close();
                return complete.digest();
            }
            catch (NoSuchAlgorithmException nsae) {
                return null;
            }
            // Do nothing for it.
        }
        catch (IOException e ) {
            return null;        }        // Do nothing for it.
        } // End of creating file hash (or digest)
    }
}
```

## ReadSecureFileB.java

[illegible]

```

BufferedReader outc = new BufferedReader(new
InputStreamReader(System.in));

    String outfile = outc.readLine();
    File Dest = new File (outfile);

    System.out.println(getDateTime()); // show start time

// Open the DCS file for reading
RandomAccessFile f = new RandomAccessFile(file, "r");

// Open the output file for writing
RandomAccessFile out = new RandomAccessFile(Dest, "rw");

long startreading = System.currentTimeMillis();// Calculate the start time

    try{

//Read the number of encrypted keywords attached
int keyword_no = f.readInt();
System.out.println("number of keywords: " + keyword_no);

/* Move the pointer after the encrypted keywords starting from the begging
of the file: 32 byte for HSA 256 and *the number of words + 4 bytes for
integer number of keywords */

    int l=keyword_no*32+4; // 4 byte for int + 32 byte * no of keywords
    f.seek(l);

    String File_owner = f.readUTF(); //Reading the data owner ID
    System.out.println("The file owner is: " +File_owner);

    long Cr = f.readLong(); //read the secret value Cr
    System.out.println("The Cr value: "+Cr);

    int x1 = f.readInt(); // Reading the length of Xr
    //System.out.println("the length of xr: " + x1);
    byte[] xr = new byte[x1];// Specify the length of Xr byte array

```

```

/* Reading the shared value Xr in Bytes array according to the specified
length */
f.read(xr);
BigInteger x = new BigInteger(xr); //Convert the Xr to BigInteger

//Checking that the Xr value is positive
if(x.compareTo(BigInteger.ZERO) > 0){
    System.out.println("the Xr value is: " +x);
    System.out.println("Length of Xr in Bits:"+x.bitLength());

    //Decrypting Xr value to reveal the Cr and Ks of the file
    long startx = System.currentTimeMillis();
    long startxn = System.nanoTime();

    //The private key of the authorised user
    String PrivKeyFile = "private.key";

    BigInteger encKey = x.mod(N); //Finding the encrypted Cr||Ks from Xr
    long endxn = System.nanoTime();
    System.out.println("Finding the encrypted Cr||Ks from Xr took in ns : "+
    (endxn - startxn)+"ns");

    byte[] encData = encKey.toByteArray(); // covert to byte array

    //Decrypting the encrypted Cr||Ks
    byte[] dec = RSAforBytes.rsaDecrypt(encData, PrivKeyFile);
    BigInteger key = new BigInteger(dec); //Converting Cr||Ks to BigInteger
    //End of finding the Cr||Ks

    long endx = System.currentTimeMillis();

    System.out.println("Finding Cr||Ks from Xr took in ms: "+(endx - startx) +
    " ms");

    //this show the value Cr||Ks
    System.out.println("the Cr||Ks of this file:"+ key);
    }
    else {System.out.println("the Xr value is incorrect");}

```

```

//the length of the file signature (128 bytes * 8 = 1024bit)
byte[] FSignature = new byte[128];
f.read(FSignature);

//From here start writing the encrypted content file to the output file

byte[] buf = new byte[1024];
int len;

while ((len = f.read(buf)) > 0) {
    out.write(buf, 0, len);
}

//End of writing to output file
System.out.println("new file length:"+f.length());

f.close();
out.close();

//End time of reading DCS file and creating the original one

long endreading = System.currentTimeMillis();
System.out.println("Reading the information from DCS file and creating the
original file took in ms :"+(endreading - startreading) + " ms");
System.out.println(getDateTime()); // show end time

// Verifying the integrity and authenticity of the output file

long startdigest = System.currentTimeMillis(); // Calculate the start time

String PublicKeyFile = "publicOwner.key"; //Data owner public key

// Decrypting the encrypted file digest with the public key of Data owner
byte[] OldDigest = RSAforBytes.rsaDecrypt(FSignature, PublicKeyFile);
String unsig_str = new String(OldDigest, "UTF8");
System.out.println("The old digest of the encrypted file: "+unsig_str);

// Calculating the digest of the output file
byte[] FileDigest = Hash.createFileHash(outfile, "SHA-256");
String digest_str = new String(FileDigest, "UTF8");

```

```

System.out.println("The new digest of the encrypted file: "+digest_str);

// Comparing the old file digest with the new file digest
if (Arrays.equals(OldDigest,FileDigest)) System.out.println("OK, the
integrity and authenticity proof are verified");
else System.out.println("Attention, the integrity and authenticity proof
are NOT verified");

    long enddigest = System.currentTimeMillis();

//End time of verifying integrity and signature
System.out.println("Verification took in ms: "+(enddigest - startdigest) +
" ms");

// End of the operation of integrity and authenticity verification
    }
    catch(Exception e){
        System.out.println("Reached EOF!");
    }
}

private final static String getDateTime()
{
    DateFormat df = new SimpleDateFormat("yyyy-MM-dd_hh:mm:ss");
    df.setTimeZone(TimeZone.getTimeZone("PST"));
    return df.format(new Date());
}
}

```

**The server side code:**

## Search.java

```

package ourTests;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.RandomAccessFile;
import java.text.DateFormat;
import java.text.SimpleDateFormat;

public class search {

    public static void main(String[] args) throws IOException
    {
        //Specify the directory path that contains the DCS files for searching
        String path = "c:\\search_test";

        // Buffer to read input text
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

        long start = System.currentTimeMillis();// Calculates the start time

        try {
            /* This part to enter the search keyword and encrypt it (it must done at
            data owner side) */
            System.out.println("Write the Keyword you want to search for:");
            String stringToConvert = br.readLine();
            byte[] theByteArray;
            theByteArray =
            HMACSHA256.encode("keyforKeywords",stringToConvert);
            // The server must only receive the encrypted keyword (theByteArray)

            //show start time
            System.out.println("Seachin start at time:"+ getDateTme());
        }
    }
}

```

```

String files;

File folder = new File(path);
//reads the list of files in specified folder
File[] listOfFiles = folder.listFiles();

    for (int i = 0; i < listOfFiles.length; i++)
    {
        // repeats the actions for all files in the list

        if (listOfFiles[i].isFile())
        {
            files = listOfFiles[i].getName(); //reads the path and name of the file

            try{
                //Open the file for reading the number of keywords
                RandomAccessFile rand = new RandomAccessFile(listOfFiles[i], "r");

                //create FileInputStream object for searching on file
                FileInputStream fin = new FileInputStream(listOfFiles[i]);

                /* This code could be used for automatically calculating the range of
                keywords for each DCS file if all the files in the search directory are
                DCS files */

                // Read the number of encrypted keywords attached
                int keyword_no = rand.readInt();
                /* Specifies the range of search = 32bytes* No for keywords + 4 byte
                (integer) */
                int Keywords_range = 32*keyword_no+4;
                System.out.println("Keyowrds range in bytes: "+Keywords_range);
                byte fileContent[] = new byte[Keywords_range];

                /* If not all files are DCS files Specifies the range size manually: for 5
                keywords the range = 32bytes*5 for keywords + 4 byte for the integer */
                // byte fileContent[] = new byte[164];

```



```

        fin.read(fileContent); //reads the byte array with the specified length

        long starthmac = System.nanoTime();// Starting time of search on a file

//Using the KMPMatch algorithm for search on each file
if ( ourTests.KMPMatch.indexOf(fileContent, theByteArray) != -1)
    System.out.println("Keyword found in: "+listOfFiles[i]);

    long endhmac = System.nanoTime(); //Ending time of search on a file
    System.out.println("Searchin of the keyword in file no: "+i+" took in ns
    :"+(endhmac - starthmac) + " ns");

    rand.close();
    fin.close();

}
catch(IOException e)
{
    System.out.println(e.getMessage());
}

}

long end = System.currentTimeMillis();
System.out.println("Searchin on all files took in ms :"+(end - start) +
" ms");
System.out.println("End time:"+ getDateTime()); // show end time
} catch (Exception e1) {
    // TODO Auto-generated catch block
    e1.printStackTrace();
}

}

private final static String getDateTime()
{
    DateFormat df = new SimpleDateFormat("yyyy-MM-dd_hh:mm:ss");
    df.setTimeZone(TimeZone.getTimeZone("PST"));
    return df.format(new Date());
}
}

```

## KMPPMatch.java

```

package ourTests;

/* Code source: http://www.fmi.uni-sofia.bg/fmi/logic/vboutchkova/sources/KMPPMatch\_java.html */

/**
 * The Knuth-Morris-Pratt Algorithm for Pattern Matching modified for
 * comparing
 * byte arrays.
 *
 * @version 2010-01-15
 */
class KMPPMatch {
    /**
     * Finds the first occurrence of the pattern in the text.
     */
    public static int indexOf(byte[] data, byte[] pattern) {
        int[] failure = computeFailure(pattern);

        int j = 0;
        if (data.length == 0) return -1;

        for (int i = 0; i < data.length; i++) {
            while (j > 0 && pattern[j] != data[i]) {
                j = failure[j - 1];
            }
            if (pattern[j] == data[i]) { j++; }
            if (j == pattern.length) {
                return i - pattern.length + 1;
            }
        }
        return -1;
    }

    /**
     * Computes the failure function using a boot-strapping process,

```

```
    * where the pattern is matched against itself.
    */
    private static int[] computeFailure(byte[] pattern) {
        int[] failure = new int[pattern.length];

        int j = 0;
        for (int i = 1; i < pattern.length; i++) {
            while (j > 0 && pattern[j] != pattern[i]) {
                j = failure[j - 1];
            }
            if (pattern[j] == pattern[i]) {
                j++;
            }
            failure[i] = j;
        }

        return failure;
    }
}
```

## References

- [1] B. Grobauer, T. Walloschek, and E. Stocker, "Understanding Cloud Computing Vulnerabilities," *Security & Privacy, IEEE*, vol. 9, pp. 50-57, 2011.
- [2] T. Dillon, W. Chen, and E. Chang, "Cloud Computing: Issues and Challenges," in *Advanced Information Networking and Applications (AINA)*, 2010 24th IEEE International Conference on, 2010, pp. 27-33.
- [3] M. Malathi, "Cloud computing concepts," in *Electronics Computer Technology (ICECT)*, 2011 3rd International Conference on, 2011, pp. 236-239.
- [4] P. Mell and T. Grance. (2011, 7/8/2012). NIST Definition of Cloud Computing Available: <http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf>
- [5] K. Ren, C. Wang, and Q. Wang, "Security Challenges for the Public Cloud," *IEEE Internet Computing*, vol. 16, pp. 69-73, 2012.
- [6] S. Pearson, "Privacy, Security and Trust in Cloud Computing," *Privacy and Security for Cloud Computing*, pp. 3-42, 2012.
- [7] C. Rong, S. T. Nguyen, and M. G. Jaatun, "Beyond lightning: A survey on security challenges in cloud computing," *Computers & Electrical Engineering*, 2012.
- [8] T. J. Lehman and S. Vajpayee, "We've Looked at Clouds from Both Sides Now," in *SRII Global Conference (SRII)*, 2011 Annual, 2011, pp. 342-348.
- [9] N. el-Khameesy and H. A. Rahman, "A Proposed Model for Enhancing Data Storage Security in Cloud Computing Systems," *Journal of Emerging Trends in Computing and Information Sciences*, vol. 3, 2012.
- [10] W. Cong, W. Qian, R. Kui, and L. Wenjing, "Ensuring data storage security in Cloud Computing," in *Quality of Service, 2009. IWQoS. 17th International Workshop on*, 2009, pp. 1-9.
- [11] S. Sengupta, V. Kaulgud, and V. S. Sharma, "Cloud Computing Security--Trends and Research Directions," in *Services (SERVICES)*, 2011 IEEE World Congress on, 2011, pp. 524-531.
- [12] S. Kamara and K. Lauter, "Cryptographic Cloud Storage Financial Cryptography and Data Security." vol. 6054, R. Sion, R. Curtmola, S. Dietrich, A. Kiayias, J. Miret, K. Sako, and F. Sebé, Eds., ed: Springer Berlin / Heidelberg, 2010, pp. 136-149.
- [13] I. Agudo, D. Nuñez, G. Giammatteo, P. Rizomiliotis, and C. Lambrinoudakis, "Cryptography Goes to the Cloud Secure and Trust Computing, Data Management, and Applications." vol. 187, C. Lee, J.-M. Seigneur, J. J. Park, and R. R. Wagner, Eds., ed: Springer Berlin Heidelberg, 2011, pp. 190-197.
- [14] C. Dong, G. Russello, and N. Dulay, "Shared and searchable encrypted data for untrusted servers," *Journal of Computer Security*, vol. 19, pp. 367-397, 2011.
- [15] D. Agrawal, A. E. Abbadi, and S. Wang, "Secure and privacy-preserving data services in the cloud: a data centric view," *Proc. VLDB Endow.*, vol. 5, pp. 2028-2029, 2012.

- [16] J. Naruchitparames and M. H. Gunes, "Enhancing data privacy and integrity in the cloud," in High Performance Computing and Simulation (HPCS), 2011 International Conference on, 2011, pp. 427-434.
- [17] K. Jinzhu, "A Practical Approach to Improve the Data Privacy of Virtual Machines," in Computer and Information Technology (CIT), 2010 IEEE 10th International Conference on, 2010, pp. 936-941.
- [18] K. Jinzhu, "Protecting the Confidentiality of Virtual Machines Against Untrusted Host," in Intelligence Information Processing and Trusted Computing (IPTC), 2010 International Symposium on, 2010, pp. 364-368.
- [19] F. Lombardi and R. Di Pietro, "Secure virtualization for cloud computing," *Journal of Network and Computer Applications*, vol. 34, pp. 1113-1122, 2011.
- [20] R. Chow, P. Golle, M. Jakobsson, E. Shi, J. Staddon, R. Masuoka, and J. Molina, "Controlling data in the cloud: outsourcing computation without outsourcing control," presented at the Proceedings of the 2009 ACM workshop on Cloud computing security, Chicago, Illinois, USA, 2009.
- [21] R. K. L. Ko, M. Kirchberg, and B. S. Lee, "From system-centric to data-centric logging - Accountability, trust & security in cloud computing," in Defense Science Research Conference and Expo (DSR), 2011, 2011, pp. 1-4.
- [22] W. Zhou, M. Sherr, W. R. Marczak, Z. Zhang, T. Tao, B. T. Loo, and I. Lee, "Towards a data-centric view of cloud security," presented at the Proceedings of the second international workshop on Cloud data management, Toronto, ON, Canada, 2010.
- [23] S. Ransom and C. Werner, "Towards Data-Centric Security in Ubiquitous Computing Environments," in Database and Expert Systems Application, 2009. DEXA '09. 20th International Workshop on, 2009, pp. 26-30.
- [24] L. M. Vaquero, L. Roderio-Merino, J. Caceres, and M. Lindner, "A break in the clouds: towards a cloud definition," *SIGCOMM Comput. Commun. Rev.*, vol. 39, pp. 50-55, 2008.
- [25] T. Grandison, E. M. Maximilien, S. Thorpe, and A. Alba, "Towards a Formal Definition of a Computing Cloud," in Services (SERVICES-1), 2010 6th World Congress on, 2010, pp. 191-192.
- [26] R. Mutavdzic, "Cloud computing architectures for national, regional and local government," in MIPRO, 2010 Proceedings of the 33rd International Convention, 2010, pp. 1322-1327.
- [27] A. Zahariev, "Google app engine," in TKK T-110.5190 Seminar on Internetworking, 2009, pp. 1-5.
- [28] S. A. Almula and Y. Chan Yeob, "Cloud computing security management," in Engineering Systems Management and Its Applications (ICESMA), 2010 Second International Conference on, 2010, pp. 1-7.
- [29] M. Jensen, J. Schwenk, N. Gruschka, and L. L. Iacono, "On Technical Security Issues in Cloud Computing," in Cloud Computing, 2009. CLOUD '09. IEEE International Conference on, 2009, pp. 109-116.
- [30] S. Patidar, D. Rane, and P. Jain, "A Survey Paper on Cloud Computing," in Advanced Computing & Communication Technologies (ACCT), 2012 Second International Conference on, 2012, pp. 394-398.
- [31] (13/08/2012). Amazon Virtual Private Cloud (Amazon VPC). Available: <http://aws.amazon.com/vpc/>
- [32] IDC. (2009, 24 October 2012). Cloud User Surveys. Available: <http://www.slideshare.net/JorFigOr/cloud-computing-2010-an-idc-update>

- [33] Y. Chen, V. Paxson, and R. H. Katz, "What's new about cloud computing security?," University of California, Berkeley Report No. UCB/EECS-2010-5 January, vol. 20, pp. 2010-5, 2010.
- [34] S. Sundareswaran, A. Squicciarini, and L. Dan, "Ensuring Distributed Accountability for Data Sharing in the Cloud," *Dependable and Secure Computing*, IEEE Transactions on, vol. 9, pp. 556-568, 2012.
- [35] K. Popovic and Z. Hocenski, "Cloud computing security issues and challenges," in *MIPRO, 2010 Proceedings of the 33rd International Convention*, 2010, pp. 344-349.
- [36] P. Mell, "What's Special about Cloud Security?," *IT Professional*, vol. 14, pp. 6-8, 2012.
- [37] R. K. L. Ko, B. S. Lee, and S. Pearson, "Towards Achieving Accountability, Auditability and Trust in Cloud Computing Advances in Computing and Communications," vol. 193, pp. 432-444, 2011.
- [38] C. C. Tan, Q. Liu, and J. Wu, "Secure Locking for Untrusted Clouds," in *Cloud Computing (CLOUD)*, 2011 IEEE International Conference on, 2011, pp. 131-138.
- [39] S. Haas, S. Wohlgemuth, I. Echizen, N. Sonehara, and G. Müller, "Aspects of privacy for electronic health records," *International Journal of Medical Informatics*, vol. 80, pp. e26-e31, 2011.
- [40] P. Samarati and S. D. C. d. Vimercati, "Data protection in outsourcing scenarios: issues and directions," presented at the *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*, Beijing, China, 2010.
- [41] B. Hay, K. Nance, and M. Bishop, "Storm Clouds Rising: Security Challenges for IaaS Cloud Computing," in *System Sciences (HICSS)*, 2011 44th Hawaii International Conference on, 2011, pp. 1-7.
- [42] J. Bethencourt, D. Song, and B. Waters, "New Techniques for Private Stream Searching," *ACM Trans. Inf. Syst. Secur.*, vol. 12, pp. 1-32, 2009.
- [43] L. T. A. Joseph, A. Samsudin, and B. Belaton, "Efficient search on encrypted data," in *Networks, 2005. Jointly held with the 2005 IEEE 7th Malaysia International Conference on Communication.*, 2005 13th IEEE International Conference on, 2005, p. 6 pp.
- [44] B. Amann and T. Fuhrmann, "Cryptographically Enforced Permissions for Fully Decentralized File Systems," in *Peer-to-Peer Computing (P2P)*, 2010 IEEE Tenth International Conference on, 2010, pp. 1-10.
- [45] S. D. C. D. Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati, "Encryption policies for regulating access to outsourced data," *ACM Trans. Database Syst.*, vol. 35, pp. 1-46, 2010.
- [46] Y. Lu and G. Tsudik, "Enhancing Data Privacy in the Cloud Trust Management V." vol. 358, I. Wakeman, E. Gudes, C. Jensen, and J. Crampton, Eds., ed: Springer Boston, 2011, pp. 117-132.
- [47] N. Sonehara, I. Echizen, and S. Wohlgemuth, "Isolation in Cloud Computing and Privacy-Enhancing Technologies," *Business & information systems engineering*, vol. 3, pp. 155-162, 2011.
- [48] F. Rocha, S. Abreu, and M. Correia, "The Final Frontier: Confidentiality and Privacy in the Cloud," *Computer*, vol. 44, pp. 44-50, 2011.
- [49] A.-R. Sadeghi, "Trusted Computing — Special Aspects and Challenges *SOFSEM 2008: Theory and Practice of Computer Science.*" vol. 4910, V.

- Geffert, J. Karhumäki, A. Bertoni, B. Preneel, P. Návrat, and M. Bieliková, Eds., ed: Springer Berlin / Heidelberg, 2008, pp. 98-117.
- [50] C. Tarnovsky, "Hacking the smartcard chip," Black Hat, 2010.
  - [51] H. Takabi, J. B. D. Joshi, and G. Ahn, "Security and Privacy Challenges in Cloud Computing Environments," *Security & Privacy, IEEE*, vol. 8, pp. 24-31, 2010.
  - [52] T. Junfeng and W. Zhijie, "A Trusted Control Model of Cloud Storage," in *Computer Distributed Control and Intelligent Environmental Monitoring (CDCIEM)*, 2012 International Conference on, 2012, pp. 78-81.
  - [53] S. M. Khan and K. W. Hamlen, "AnonymousCloud: A Data Ownership Privacy Provider Framework in Cloud Computing," in *Trust, Security and Privacy in Computing and Communications (TrustCom)*, 2012 IEEE 11th International Conference on, 2012, pp. 170-176.
  - [54] T. Kirkham, D. Armstrong, K. Djemame, M. Corrales, M. Kiran, I. Nwankwo, M. Jiang, and N. Forgo, "Assuring Data Privacy in Cloud Transformations," in *Trust, Security and Privacy in Computing and Communications (TrustCom)*, 2012 IEEE 11th International Conference on, 2012, pp. 1063-1069.
  - [55] (4 Nov 2012). Trusted Computing Group. Available: [http://www.trustedcomputinggroup.org/solutions/cloud\\_security](http://www.trustedcomputinggroup.org/solutions/cloud_security)
  - [56] M. Bilger, L. O'Connor, M. Schunter, M. Swimmer, and N. Zunic, "Data-centric security," IBM Corporation (December 2006), [http://www-935.ibm.com/services/us/cio/risk/gov\\_wp\\_data\\_centric.pdf](http://www-935.ibm.com/services/us/cio/risk/gov_wp_data_centric.pdf), 2006.
  - [57] B. Jennifer, "Data-centric security," *Computer Fraud & Security*, vol. 2009, pp. 7-11, 2009.
  - [58] Y. Y. Chen, "Architecture for Data-Centric Security," 2012.
  - [59] L. Chen and D. Hoang, "Active data-centric framework for data protection in cloud environment," in *ACIS 2012: Proceedings of the 23rd Australasian Conference on Information Systems 2012*, Geelong, Australia, 2012, pp. 1-11.
  - [60] I. Arora and A. Gupta, "Cloud Databases: A Paradigm Shift in Databases," 2012.
  - [61] (2012, 11 Nov 2012). Semi-structured data. Available: [http://en.wikipedia.org/wiki/Semi-structured\\_data](http://en.wikipedia.org/wiki/Semi-structured_data)
  - [62] D. Morley and C. S. Parker, *Understanding Computers: Today and Tomorrow*, Comprehensive: Course Technology Ptr, 2012.
  - [63] C. Deyan and Z. Hong, "Data Security and Privacy Protection Issues in Cloud Computing," in *Computer Science and Electronics Engineering (ICCSEE)*, 2012 International Conference on, 2012, pp. 647-651.
  - [64] A. Albeshri, C. Boyd, and J. Gonzalez Nieto, "A Security Architecture for Cloud Storage Combining Proofs of Retrievability and Fairness," 2012, pp. 30-35.
  - [65] D. Zissis and D. Lekkas, "Addressing cloud computing security issues," *Future Generation Computer Systems*, vol. In Press, Corrected Proof, 2010.
  - [66] E. Ferrari, "Database as a Service: Challenges and solutions for privacy and security," in *Services Computing Conference, 2009. APSCC 2009. IEEE Asia-Pacific*, 2009, pp. 46-51.
  - [67] Z. Minqi, Z. Rong, X. Wei, Q. Weining, and Z. Aoying, "Security and Privacy in Cloud Computing: A Survey," in *Semantics Knowledge and Grid (SKG)*, 2010 Sixth International Conference on, 2010, pp. 105-112.

- [68] S. D. C. d. Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati, "A data outsourcing architecture combining cryptography and access control," presented at the Proceedings of the 2007 ACM workshop on Computer security architecture, Fairfax, Virginia, USA, 2007.
- [69] D. J. Abadi, "Data management in the cloud: Limitations and opportunities," *IEEE Data Eng. Bull.*, vol. 32, pp. 3-12, 2009.
- [70] S. De Capitani di Vimercati, S. Foresti, S. Paraboschi, G. Pelosi, and P. Samarati, "Efficient and Private Access to Outsourced Data," in *Distributed Computing Systems (ICDCS)*, 2011 31st International Conference on, 2011, pp. 710-719.
- [71] M. Harbach, S. Fahl, M. Brenner, T. Muders, and M. Smith, "Towards privacy-preserving access control with hidden policies, hidden credentials and hidden decisions," in *Privacy, Security and Trust (PST)*, 2012 Tenth Annual International Conference on, 2012, pp. 17-24.
- [72] J. Li, Q. Wang, C. Wang, N. Cao, K. Ren, and W. Lou, "Enabling efficient fuzzy keyword search over encrypted data in cloud computing," in *Proc. Of IEEE INFOCOM10 Mini-conference*, San Diego, CA, USA, 2009.
- [73] M. Naehrig, K. Lauter, and V. Vaikuntanathan, "Can homomorphic encryption be practical?," in *Proceedings of the 3rd ACM workshop on Cloud computing security workshop*, 2011, pp. 113-124.
- [74] K. Yang and X. Jia, "Data storage auditing service in cloud computing: challenges, methods and opportunities," *World Wide Web*, vol. 15, pp. 409-428, 2012.
- [75] V. A. Oleshchuk and G. M. Koien, "Security and privacy in the cloud a long-term view," in *Wireless Communication, Vehicular Technology, Information Theory and Aerospace & Electronic Systems Technology (Wireless VITAE)*, 2011 2nd International Conference on, 2011, pp. 1-5.
- [76] K. Murthy, P. M. Deshpande, A. Dey, R. Halasipuram, M. Mohania, P. Deepak, J. Reed, and S. Schumacher, "Exploiting evidence from unstructured data to enhance master data management," *Proc. VLDB Endow.*, vol. 5, pp. 1862-1873, 2012.
- [77] X. Chen and Y. Li, "Efficient Proxy Re-encryption with Private Keyword Searching in Untrusted Storage," *International Journal of Computer Network and Information Security (IJCNIS)*, vol. 3, p. 50, 2011.
- [78] S. Chow, J. Weng, Y. Yang, and R. Deng, "Efficient unidirectional proxy re-encryption," *Progress in Cryptology—AFRICACRYPT 2010*, pp. 316-332, 2010.
- [79] A. V. D. M. Kayem, P. Martin, and S. G. Akl, "Efficient enforcement of dynamic cryptographic access control policies for outsourced data," in *Information Security South Africa (ISSA)*, 2011, 2011, pp. 1-8.
- [80] M. Nabeel and E. Bertino, "Privacy preserving delegated access control in the storage as a service model," in *Information Reuse and Integration (IRI)*, 2012 IEEE 13th International Conference on, 2012, pp. 645-652.
- [81] J. Dai, S. Luo, and H. Liu, "A privacy-preserving access control in outsourced storage services," in *Computer Science and Automation Engineering (CSAE)*, 2011 IEEE International Conference on, 2011, pp. 247-251.
- [82] S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, G. Pelosi, and P. Samarati, "Encryption-Based Policy Enforcement for Cloud Storage,"



- in Distributed Computing Systems Workshops (ICDCSW), 2010 IEEE 30th International Conference on, 2010, pp. 42-51.
- [83] A. Zych, M. Petković, and W. Jonker, "Efficient key management for cryptographically enforced access control," *Computer Standards & Interfaces*, vol. 30, pp. 410-417, 2008.
  - [84] H. Junbeom, "Attribute-Based Access Control with Efficient Revocation in Data Outsourcing Systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, pp. 1214-1221, 2011.
  - [85] N. Attrapadung and H. Imai, "Attribute-Based Encryption Supporting Direct/Indirect Revocation Modes Cryptography and Coding." vol. 5921, M. Parker, Ed., ed: Springer Berlin / Heidelberg, 2009, pp. 278-300.
  - [86] T. Yang, P. P. C. Lee, J. C. S. Lui, and R. Perlman, "Secure Overlay Cloud Storage with Access Control and Assured Deletion," *Dependable and Secure Computing*, *IEEE Transactions on*, vol. 9, pp. 903-916, 2012.
  - [87] P. Tourani, M. A. Hadavi, and R. Jalili, "Access control enforcement on outsourced data ensuring privacy of access control policies," in *High Performance Computing and Simulation (HPCS)*, 2011 International Conference on, 2011, pp. 491-497.
  - [88] Y. Kong, J. Seberry, J. Getta, and P. Yu, "A Cryptographic Solution for General Access Control Information Security." vol. 3650, J. Zhou, J. Lopez, R. Deng, and F. Bao, Eds., ed: Springer Berlin / Heidelberg, 2005, pp. 461-473.
  - [89] S. Dawn Xiaoding, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *Security and Privacy*, 2000. S&P 2000. Proceedings. 2000 IEEE Symposium on, 2000, pp. 44-55.
  - [90] H. Hacıgümüş, B. Hore, B. Iyer, and S. Mehrotra, "Search on Encrypted Data Secure Data Management in Decentralized Systems." vol. 33, T. Yu and S. Jajodia, Eds., ed: Springer US, 2007, pp. 383-425.
  - [91] E. J. Goh, "Secure indexes," An early version of this paper first appeared on the Cryptology ePrint Archive on October 7th, 2003.
  - [92] B. Waters, D. Balfanz, G. Durfee, and D. K. Smetters, "Building an Encrypted and Searchable Audit Log," in *NDSS'04*, 2004.
  - [93] Y.-C. Chang and M. Mitzenmacher, "Privacy Preserving Keyword Searches on Remote Encrypted Data Applied Cryptography and Network Security." vol. 3531, J. Ioannidis, A. Keromytis, and M. Yung, Eds., ed: Springer Berlin / Heidelberg, 2005, pp. 391-421.
  - [94] S. Kamara, C. Papamanthou, and T. Roeder, "Cs2: A searchable cryptographic cloud storage system," *Technical Report MSR-TR-2011-58*, Microsoft 2011.
  - [95] R. Koletka and A. Hutchison, "An architecture for secure searchable cloud storage," in *Information Security South Africa (ISSA)*, 2011, 2011, pp. 1-7.
  - [96] J. Li, C. Jia, J. Li, and Z. Liu, "A Novel Framework for Outsourcing and Sharing Searchable Encrypted Data on Hybrid Cloud," in *Intelligent Networking and Collaborative Systems (INCoS)*, 2012 4th International Conference on, 2012, pp. 1-7.

- [97] R. Sravan Kumar and A. Saxena, "Data integrity proofs in cloud storage," in Communication Systems and Networks (COMSNETS), 2011 Third International Conference on, 2011, pp. 1-4.
- [98] G. S. Reddy, A. Dileep, P. N. Rao, and K. Kavitha, "Security Challenges in Cloud Storage," IJECCE, vol. 3, pp. 685-690, 2012.
- [99] R. Sugumar and K. Gopinath, "Recognition and security guidance of data integrity in cloud storage," International Multidisciplinary Research Journal, vol. 2, 2012.
- [100] Q. Wang, C. Wang, J. Li, K. Ren, and W. Lou, "Enabling Public Verifiability and Data Dynamics for Storage Security in Cloud Computing Computer Security – ESORICS 2009." vol. 5789, M. Backes and P. Ning, Eds., ed: Springer Berlin / Heidelberg, 2009, pp. 355-370.
- [101] S. Iftene, "General secret sharing based on the Chinese remainder theorem," ed: Citeseer, 2006.
- [102] O. Knill, "A multivariable Chinese remainder theorem," arXiv preprint arXiv:1206.5114, 2012.
- [103] S. Y. Yan., "Number theory for computing," in Computers & Mathematics with Applications. vol. 46, Second Edition ed: Elsevier Ltd, 2003, pp. 502-503.
- [104] X. Zheng, C.-T. Huang, and M. Matthews, "Chinese remainder theorem based group key management," presented at the Proceedings of the 45th annual southeast regional conference, Winston-Salem, North Carolina, 2007.
- [105] K. O. Geddes, S. R. Czapor, and G. Labahn, Algorithms for computer algebra: Springer, 1992.
- [106] S. Iftene, "General Secret Sharing Based on the Chinese Remainder Theorem with Applications in E-Voting," Electronic Notes in Theoretical Computer Science, vol. 186, pp. 67-84, 2007.
- [107] S. S. Mhatre, V. B. Salve, and S. J. Mane, "Enhanced Chinese Remainder Theorem based Broadcast Authentication in Wireless Networks," International Journal of Computer Applications, vol. 50, pp. 10-14, 2012.
- [108] M. WEN, J. LEI, J. LI, Y. WANG, and K. CHEN, "Efficient User Access Control Mechanism for Wireless Multimedia Sensor Networks," Journal of Computational Information Systems, vol. 7, pp. 3325-3332, 2011.
- [109] G. Campobello, A. Leonardi, and S. Palazzo, "On the Use of Chinese Remainder Theorem for Energy Saving in Wireless Sensor Networks," in Communications, 2008. ICC '08. IEEE International Conference on, 2008, pp. 2723-2727.
- [110] D. Sinha, U. Bhattacharya, and R. Chaki, "A CRT based encryption methodology for secure communication in MANET," International Journal of Computer Applications, vol. 39, pp. 20-25, 2012.
- [111] B. Wang, Y. Wei, and Y. Hu, "Fast public-key encryption scheme based on Chinese remainder theorem," Frontiers of Electrical and Electronic Engineering in China, vol. 4, pp. 181-185, 2009.
- [112] Z. Huang and S. Li, "Design and Implementation of a Low Power RSA Processor for Smartcard," International Journal of Modern Education and Computer Science (IJMECS), vol. 3, p. 8, 2011.
- [113] C.-C. Chang and Y.-P. Lai, "A fast modular square computing method based on the generalized Chinese remainder theorem for prime moduli," Applied Mathematics and Computation, vol. 161, pp. 181-194, 2005.

## References

- [114] W. J. Wang, H. Y. Jiang, X. G. Xia, P. C. Mu, and Q. Y. Yin, "A wireless secret key generation method based on Chinese remainder theorem in FDD systems," Science China Information sciences, pp. 1-12, 2012.
- [115] (2012). vmware. Available: <http://www.vmware.com/solutions/cloud-computing/index.html>
- [116] AES Crypt. Available: <http://www.aescrypt.com/download.html>
- [117] (2010, 11 February 2012). Class BigInteger. Available: <http://docs.oracle.com/javase/1.4.2/docs/api/java/math/BigInteger.html>
- [118] (2011, 3 February 2012). cryptocodes. Available: <http://code.google.com/a/eclipselabs.org/p/cryptocodes/source/browse/trunk/Crypto/src/ChineseRemainder.java>
- [119] (2010, 5 April 2012). Package java.security. Available: <http://docs.oracle.com/javase/1.4.2/docs/api/java/security/package-summary.html>
- [120] (3 July 2012). Random Access Files. Available: <http://docs.oracle.com/javase/tutorial/essential/io/rafs.html>
- [121] M. Zubair, F. Wahab, I. Hussain, and M. Ikram, "Text scanning approach for exact string matching," in Networking and Information Technology (ICNIT), 2010 International Conference on, 2010, pp. 118-122.
- [122] (2010, 3 May 2012). The Knuth-Morris-Pratt Algorithm. Available: [http://www.fmi.uni-sofia.bg/fmi/logic/vboutchkova/sources/KMPMatch\\_java.html](http://www.fmi.uni-sofia.bg/fmi/logic/vboutchkova/sources/KMPMatch_java.html)