A Dynamic Cloud with Data Privacy Preservation

by

Mehdi Bahrami

A dissertation submitted in partial satisfaction of

the requirements for the degree of

Doctor of Philosophy

in

Electrical Engineering and Computer Science

in the Graduate Division

of the

University of California, Merced

Committee in charge:

Professor Mukesh Singhal, Chair

Professor Florin Rusu

Professor Dong Li

[1]Professor Hamid R. Arabnia

[1](External Committee Member, University of Georgia)

Fall 2016

A Dynamic Cloud with Data Privacy Preservation

Mehdi Bahrami

The Dissertation of **Mehdi Bahrami** is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

Professor Mukesh Singhal

_____

Professor Dong Li

_____

Professor Florin Rusu

_____

Professor Hamid R. Arabnia

(University of Georgia)

_____

_____

Co-Chair (if applicable)

Professor Mukesh Singhal

_____

Chair

University of California, Merced

2016

Abstract

A Dynamic Cloud with Data Privacy Preservation

by

Mehdi Bahrami

Doctor of Philosophy in Electrical Engineering and Computer Science

University of California, Merced

Professor Mukesh Singhal

The emerging field of Cloud Computing provides elastic on-demand services over the Internet or over a network. According to the International Data Corporation (IDC), cloud computing has two major issues: i) architecture issues, such as a lack of standardization, a lack of customization; and ii) users' data privacy. In this study we focus on these issues.

We are facing an increasing demand for migration of varieties of traditional databases and computation services to cloud computing environments, i.e., database-as-a-service. Although each service offers a new feature, it escalates standardization and customization issues due to the lack of standardization between cloud vendors and service customization because each cloud-based service has its own features, requirements and outputs. In the first part of this study, we propose a cloud architecture based on a Service-Oriented Architecture (DCCSOA) that enhances our ability to do standardization and customization in the cloud. The proposed architecture uses a single layer, which is called Dynamic Template Service Layer (DTSL), that provides the following operations and advantages:

i) enables a single service layer to interact with all native cloud services (e.g., IaaS, PaaS, SaaS and any cloud-based services);
ii) provides a standardization for existing services and future services in the cloud;
iii) customizes native cloud services based on users' group requests.

The second part of this study focuses on users' data privacy preservation on the proposed architecture. Users' data privacy can be violated by the cloud vendor, the vendor's authorized users, other cloud users, unauthorized users, or external malicious entities. Encryption of data on client side is one of the solutions to preserve data privacy in the cloud; however, encryption methods are complex and expensive for mobile devices to encrypt and decrypt each file, such as smart phones. We propose a novel light-weight data privacy method (DPM) by using a chaos system for mobile cloud users to store data on multiple clouds. The proposed method enables mobile users to store data in the clouds while it preserves users' data privacy.

We consider different technologies to deploy our proposed data privacy preservation method on DCCSOA, including the mobile devices, the Internet-of-things (IoT), and Graphic Processing Unit (GPU)-based computing. We also consider different use case scenarios for the proof of concept, including data privacy preservation for users' photos in smart phones, sensitive electronic health records protection in the cloud, and data privacy preservation for cloud-based databases.

We evaluate both the proposed dynamic architecture and the proposed data privacy preservation method. Our experimental results show that on the one hand DCCSOA enhances standardization by offering a flexible cloud architecture and minimizing the modification on the native cloud services; on the other hand, DPM achieves a superior performance over regular encryption methods in regard to computation time.

To my Mother, Sisters, Brother, and Yuliya

for their endless love, invaluable support, motivation and encouragement

In Memory of My Father

who was a great and kind teacher, and an invaluable supportive person and

who always encouraged me to study, to teach and to complete this work

This work would not have been possible without all your support.

Contents

List of Figures

List of Tables

List of Algorithms

List of Codes

Acknowledgements

A Ph.D. program is a long journey and this work would not be completed without receiving support from numerous people. It has been a privilege to work and collaborate with faculties, students, communities and staffs at University of California, Merced.

First, all that has been completed was advised by Professor Mukesh Singhal, who is a bright professor, leader, friend and an incredible supervisor. His patience, constructive feedback, and inspiration has encouraged me to complete this work. His invaluable support and encouragement allows me to successfully move forward in the cutting-edge areas of cloud computing, data security and data privacy preservation. I learned from Professor Singhal, how to do a research, and how to transfer my research from the lab to the real world.

I received valuable support and direction from Professor Florin Rusu, Professor Dong Li, Professor Arabnia in the fields of database, parallel computing, and distributed computing respectively.

I have been delighted to work with Ms. Kathleen Cadden and Dr. Kelvin Lwin as a teaching assistant. Ms. Kathleen Cadden helped me to deliver a set of high-tech software tools to students with diverse computer science backgrounds. Dr. Kelvin Lwin helped me to prepare a set of software engineering projects.

I would like to thank the Margo F. Souza Leadership Center, which prepared me to be a leader in my field. I have been awarded several honors, generous leadership awards, and fellowship awards from the center, including the Margo Souza Entrepreneur in Training Award and the Distinguished Leadership Award.

I would like to thank Mr. Akira Itoh, Dr. Wei-Peng Chen, and Mr. Takaki Kamiya from Fujitsu Laboratory of America, Inc. for their direction in the field of security of Information-Centric Networking and Natural Language Processing. My collaboration with these incredible scientists resulted in the best demo award at ACM ICN 2016, two United States patents pending, and three technical papers.

I would also like to thank Dr. Liguang Xie from Virginia Tech, Dr. Arshia Khan from University of Minnesota, and Dr. Ashish Kundu from IBM Thomas J. Watson for their collaboration and their constructive feedback.

I had a chance to meet several people from different departments who helped me with their support. First, I would like to thank Ms. Belinda Braunstein from the Center for Engaged Teaching and Learning for her kindness, giving valuable time and supportive direction in teaching. She was one of the most valuable people I met during orientation day and I would be happy to working with her in the future. Second, she

introduced me several supportive people from different departments including Ms. Leanna Peralta, Ms. Letha Goger, and Mr. Ian Hill.

I would like to thank my colleagues at UC Merced Cloud Lab, Dr. Santosh Chandrasekhar, Ms. Mina Naghshnejad, Ms. Chandrayee Basu, and Mr. Kai Sun.

I would like to thank the UC Merced Graduate Division, School of Engineering and their staff for their invaluable support as several research and teaching fellowship awards. These awards have always encouraged me to enhance my services to both communities of UC Merced and the fields of computer science.

I am deeply grateful for receiving all of this invaluable support at the University of California, Merced.

Organization of this thesis and contributions

This thesis is organized based on several published peer-reviewed technical papers. We organized chapters in this thesis along with correlation to each other. In each chapter, we introduce the related work, challenges and motivation of the study as well as our solution for the challenges. We also investigate each proposed solution from different perspectives including software engineering evaluations, proof of data security, the performance of the proposed solution for different workloads.

In the first chapter, we review the concept of cloud computing and big data tools (Bahrami and Singhal 2015a). This chapter introduces key points of cloud computing, the architecture of cloud computing, and available big data tools in the cloud computing environment.

The second chapter introduces a novel dynamic cloud architecture *(DCCSOA)* (Bahrami and Singhal 2015b and Bahrami et al. 2015d). The architecture offers several advantages to both cloud vendors and mobile users. The dynamic feature of the architecture allows cloud vendor to modify their own architecture based on users' request. Therefore, users are able to freely move their data and applications from one vendor to another vendor without modifications.

Another challenge in cloud computing is users' data privacy which is the second goal of this study. In Chapter 3, we introduce a novel light-weight data privacy method for mobile cloud users (Bahrami and Singhal 2015c and Bahrami 2015f). By end of this chapter, the reader understands the current data privacy issue in mobile cloud computing as well as our proposed privacy preservation method.

In Chapter 4, we introduce a parallel implementation of *DPM* when the method is securely processed on multiple GPU-cores and the result of this study appeared in (Bahrami et al. 2016a). This chapter allows the reader to understand the concept of DPM parallelization and the computation overheads for processing different numbers of GPU-cores.

Recently, use of cloud-assisted IoT devices has become popular around the world. Due to storage and computation limitation on *IoT* devices, cloud computing provides an opportunity to these tiny computing machines to outsource their data and computation to cloud environments; however, there is two challenges:

First, *how to use heterogeneous cloud computing architectures for interacting with variety of IoT devices*; and second, *how a user may maintain own data privacy on IoT devices*.

Chapter 5 answers the first question, regarding to cloud-assisted *IoT* paradigm, by providing a *DCCSOA*-based architecture for interacting *IoT* devices with heterogeneous

cloud computing systems. The results of these study will appear in (Bahrami and Singhal 2016d).

Chapter 6 answers the second question regarding users' data privacy preservation by introducing a novel *DPM*-based solution for Cloud-assisted *IoT* devices (Bahrami et al. 2016b).

Chapter 7 introduces a use case of *DPM* and *DCCSOA* for electronic healthcare systems. A novel platform has been described in this chapter which is presented in (Bahrami et al. 2016b). The platform preserves patients' data privacy in cloud-based electronic health record systems.

Chapter 8 introduces a new cloud-based database schema which uses *DPM* to maintain data privacy of a database. The proposed schema works well on both SQL and NoSQL databases. By end of this chapter, the reader understands the key point of the proposed schema and its advantages, as well as the performance evaluation on both SQL and NoSQL databases. The result of this chapter appeared in (Bahrami et al. 2016c).

Finally, Chapter 9 concludes this study and provides future research directions on cloud computing architecture and data privacy preservation for mobile cloud users and cloud-assisted *IoT* devices.

Chapter 1

Introduction

In this chapter, we describe the definition of cloud computing architecture, different levels of the cloud architectures and the role of big data in each level. This introduction prepares the reader for the next chapter which describes our novel dynamic cloud computing architecture.

1.1 Introduction

Capturing data from different sources allows a business to use Business Intelligence (BI) (Matheson 1998) capabilities. These sources could be consumer information, service information, products, advertising logs, and related information such as the history of product sales or customer transactions. When an organization uses BI technology to improve services, we characterize it as a "smart organization" (Matheson 1998). The smart features of these organizations have different levels which depend on the accuracy of decisions; greater accuracy of data analysis provides "smarter" organizations. For this reason, we are collecting a massive amount of data from people, actions, sensors, algorithms, and the web which forms "Big Data." This digital data collection grows exponentially each year. According to (Manyika 2011), big data refers to datasets whose size is beyond the ability of typical database software tools and applications to capture, store, manage and analyze.

An important task of any organization is data analysis which is able to change a large volume of data to a smaller amount of valuable data but still it requires to collect a massive amount of data.

Big data has become a complex issue in all disciplines of science. In scientific big data, several solutions have been proposed to overcoming big data issues in the field of life sciences (Buscema et al. 2008 and Howe at al. 2008), education systems (Hanna 2004), material sciences (Wilson 2013), social networks (Tan 2013) and.

Some examples of the significance of big data for generating, collecting and computing are listed as follows:

Producing and collecting Big Data:

- It is predicated that data production will be 44 times greater in 2020 than it was in 2009. This data could be collected from variety resources, such as traditional databases, videos, images, binary files (applications) and text files;
- It is estimated 235 Terabytes of data were collected by the U.S. Library of Congress in April 2011;
- Facebook stores, accesses, and analyzes 30+ Petabytes of user generated data which includes a variety of data, such as images, videos and texts.

Computing big data:

- In 2008, Google was processing 20,000 Terabytes of data (20 petabytes) per day (Schonfeld 2014).
- Decoding the human genome originally took 10 years to process; now it can be achieved in one week with distributing computing on big data.
- IDC[1] estimates that by 2020, business-to-business and business-to-consumer transactions on the Internet will reach 450 billion per day.
- Big data is a top business priority and drives enormous opportunities for business improvement. Wikibon's own study projects that big data will be a $50 billion business by 2017 (Rigsby 2014).
- Macy's Inc. provides a real-time pricing. The retailer adjusts pricing in near real-time for 73 million items for sale based on demand and inventory (Davenport 2013).
- The major VISA process more than 172,800,000 card transactions each day (Fairhurst 2017).

The most public resource data are available on the Internet, such as multimedia steam data, social media data and text. This variety of data shows we are not facing only structured data, but also unstructured data, such as multimedia files (including video, audio and images), and Twitter and Facebook comments. Unstructured data causes complexity and difficulty in analyzing big data. For example, a corporation analyzes user comments and user shared data on social media that could recognize customer favorites and provide best offers.

[1] International Data Corporation (IDC) is an American market research, analysis and advisory firm specializing in information technology, telecommunications, and consumer technology.

To collect and process big data, we can use Cloud Computing Technology. Cloud computing is a new paradigm for hosting clusters of data and delivering different services over a network or the Internet. Hosting clusters of data allows customers to store and compute a massive amount of data on the cloud. This paradigm allows customers to pay on pay-per-use basis and enables them to grow (or shrink) their computing and storage needs on demand. These features allow customers to pay the infrastructure for storing and computing based on their current capacity of big data and transactions.

Capturing and processing big data are related to improving the global economy, science, social affair, education and national security; processing of big data allows us to propose accurate decisions and acquire knowledge from raw data.

This chapter aims to show the role of cloud computing in dealing with big data and intelligent computing. This chapter is organized as follows: Section 1.2 discusses a definition and characteristics of big data. In Section 1.3, we discuss important opportunities and challenges in handling big data. In Section 1.4, we discuss cloud computing and key architectural components for dealing with big data. In this section, we review how each service layer of a cloud computing system could handle big data issues. In Section 1.5, we provide a list of cloud-based services and tools for dealing with big data. A summary of implementation cloud computing models is described in Section 1.6. We review some major cloud computing issues in Section 1.7. We summary the chapter in Section 1.8.

1.2 Big Data Definition

Often big data is characterized by "4 V's" (McAfee 2012) which stand for:

- "**Volume**" which indicates a very large volume of data;
- "**Velocity**" which indicates the speed for data processing in terms of response time. This response time could be a batch, real-time or stream response-time;
- "**Variety**" which indicates heterogeneity in data that we have collected for processing and analysis this data variety includes structured, unstructured and semi-structured data;
- "**Veracity**" which indicates level of accuracy in the data. For example, a sensor that generates data can have a wrong value rather than provides an accurate data.

Big data could have one or multiple of the above characteristics. For example, storing and computing on social data could have a very large volume of data (*volume*) and specific response-time for computing (*velocity*) but it may not have *variety* and *veracity* characteristics.

Another example, analyzing public social media data regarding the purchase history of a customer could provide a future favorite purchase list when she searches for a new product. In this case, big data have all characteristics: *volume of data*, because collecting a massive amount of data from public social media networks; *velocity*, because response-time limited to near real-time when a customer search a product; *variety*, because big data may come from different sources (social media and purchase history); *lack of veracity*, because data from customers in social media networks may have uncertainty. For instance, a customer could like a product in a social media network, not because this is the product of her choice, but because of this product is used by her friend.

Another important question in big data is, *"How large is big data?"* We can answer this question based on our current technology. For example, (Jacobs 2009) states that in the late 1980s at Columbia University that they stored 100 GB of data as big data via an IBM 3850 MSS (Mass Storage System), which costs $40K per GB. In 2010, the Large Hadron Collider (LHC) facility at CERN produced 13 *peta*bytes of data (Gewin 2008). So what we call big data depends on the cost, speed and capacity of existing computing and storage technologies. For example, in the 1980s, 100 GB was big data because the storage technology was expensive at that time and it had low performance. However, by 2010, the LHC processed 13 Petabyte as a big data which has $1.363*10^5$ times more volume than IBM 3850 MSS big data in 1980s.

1.3 Opportunities and Challenges

On one hand, when we collect big data, we have an opportunity to make an accurate decision through BI. BI is a set of theories and technologies that aim to transfer data from raw-data into meaningful and useful information for business processes (BP). BI became popular in the 1990s, and Business Analytics (BA), which is an analytical component in BI, became popular in the 2000s. In the traditional model, the queries are pre-defined to confirm or refuse a query's hypotheses, but Online Analytical Processing (OLAP) analysis emerges as an approach to answer complex analytical queries. For example, in a car accident we can make a decision about the incident based on driver information. However, when we collect GPS information, engine information and driver information, we can make a more accurate decision about an accident. Also, if we collect more information, we can trust our decision more (***veracity***). In a second example, Volvo provided performance and fault monitoring for predictive warranty analysis (Young 2014). In another example, sensor data from a cross-country flight (New York to Los Angeles) generate 2.499 billion Terabyte per year (Kelly 2014) (***volume***) from different sensors (***variety***), which could be provided from reliable sensors (***veracity***) or unreliable sensors (***lack of veracity***). Often the processing of this data is real-time (***velocity***) and this computing could be processed by an aircraft's server or by a ground's servers.

Collection of information cannot only help us to avoid car accidents but also could help us to make an accurate decision in any systems, such as business financial systems (Rigsby 2014), education systems (Siegel 2000), and treatment systems, e.g. Clinical Decision Support Systems (Berner 2007).

Some important opportunities are provided by big data. They are listed as follows:

- Analyze big data to improve business processes and business plans, and to achieve business plan goals for a target organization (The target organization could be a corporation, industry, education system, financial system, government system or global system.)
- Reduce bulk data to a valuable smaller amount of data
- Provide more accurate decisions by analyzing big data
- Prevent future system failures by predicting big data

On the other hand, we have several issues with big data. The challenges of big data happened in various domains including *storing of big data*, *computing on big data* and *transferring of big data*. We discuss these issues below:

I. **Storage Issues**

A database is a structured collection of data. In the late 1960s, flat-file models which were expensive and slow, used for storing data. For these reasons, relational databases emerged in the 1970s. Relational Database Management Systems (RDBMS) employ Structured Query Language (SQL) to store, edit and retrieve data.

Lack of support for unstructured data led to the emergence of new technologies, such as BLOB (Binary Large Object) in the 2000s. Unstructured data may refer to multimedia data. Also unstructured data may refer to irregularly or randomly repeated column patterns that vary from row to row within each file or document. BLOB could store all data types in most RDBMS.

In addition, a massive amount of data could not use SQL databases because retrieving data and analyzing data takes more time for processing. So “NoSQL”, which stands for “Not Only SQL” and “Not Relational”, was designed to overcome this issue. NoSQL is a scalable partitioned table that could distribute data over many servers. NoSQL is implemented for cloud computing because in the cloud, a data storage server could be added or removed anytime. This capability allows for the addition of unlimited data storage servers to the cloud.

This technology allows organizations to collect a variety of data but still increasing the volume of data increases cost investment. For this reason, capturing high-quality data that could be more useful for an organization rather than collecting a bulk of data.

II. **Computing Issues**

When we store big data, we need to retrieve, analyze and modify it. The important part of collecting data is analyzing big data and converting raw data into valuable information that could improve a business process or decision making. This challenge can be addressed by employing a cluster of CPUs and RAMs in cloud computing technology.

High-Performance Computing (HPC) is another technology that provides a distributed solution by different computing models, such as traditional (e.g. Grid Computing) or cloud computing for scientific and engineering problems. Most of these problems could not process data in a polynomial time-complexity.

III. **Transfer Issues**

Transfer of big data is another issue. In this challenge, we are faced with several sub-issues: Transfer Speed, which indicates how fast we can transfer data from one location/site to another location/site. For example, transferring of DNA, which is a type of big data, from China to the United States has some delay in the backbone of the Internet, which causes a problem when they receive data in the United States (Marx 2013). BGI (one of the largest producers of genomic data, Beijing Genomics Institute in Shenzen, China) could transfer 50 DNAs with an average size of 0.4 terabyte through the Internet in 20 days, which is not an acceptable performance (Marx 2013).

Traffic Jam: transfer of big data could happened between two local sites, cities or worldwide via the Internet but between any locations this transfer will result in a very large traffic jam.

Accuracy and Privacy: Often we transfer big data through unsecured networks, such as the Internet. Data transfers through the Internet must be kept secure from unauthorized access. Accuracy aims to transfer data without missing any bits.

1.4 Cloud Computing

Several *traditional* solutions have been emerged for dealing with big data such as Supercomputing, Distributed Computing, Parallel Computing, and Grid Computing. However, elastic scalability is important in big data which could be supported by cloud computing services. Cloud computing has several capabilities for supporting big data which are related to handling of big data. Cloud computing supports two major issues of

big data, which are described in the following sections including storing of big data and computing of big data. Cloud computing provides a cluster of resources (storage and computing) that could be added anytime. These features allow cloud computing to become an emerging technology for dealing with big data.

In this section, we first review important features of cloud computing systems and a correlation of each of them to big data. Second, we discuss a cloud architecture and the role of each service layer in handling big data.

The next section, we review implementation models of cloud computing systems as they are related for handling big data.

The major characteristics of cloud computing as defined by the U.S. National Institute of Standards and Technology (NIST) (Liu et al. 2011) are as follows:

I. **On-demand Elastic Service**

This characteristic shows the following features: (i) an economical model of cloud computing which enables consumers to order required services (computing machines and/or storage devices). The service requested service could scale rapidly upward or downward on demand; (ii) it is a machine responsibility that does not require any human to control the requested services. The cloud architecture manages on-demand requests (increase or decrease in service requests), availability, allocation, subscription and the customer's bill.

This feature is interesting for a start-up business, because this feature of cloud computing systems allows a business to start with traditional data or normal datasets (in particular start-up business) and increase their datasets to big data as they receive requests from customers or their data grows during the business progress.

II. **Resource pooling**

A cloud vendor provides a pool of resources (e.g., computing machines, storage devices and network) to customers. The cloud architecture manages all available resources via global and local managers for different sites and local sites, respectively.

This feature allows big data to be distributed on different servers which is not possible by traditional models, such as supercomputing systems.

III. **Service Accessibility**

A cloud vendor provides all services through broadband networks (often via the Internet). The offered services are available via web-based model or heterogeneous client applications (Singhal 2013). The web-based model could be an Application Programming

Interface (API), web-services, such as Web Service Description Language (WSDL). Also heterogeneous client applications are provided by the vendors. Customers could run applications on heterogeneous client systems, such as Windows, Android and Linux. This feature enables partners to contribute to big data. These partners could provide cloud software applications, infrastructure or data. For example, several applications from different sites could connect to a single-data or transparent multiple-data warehouse for capturing, analyzing or processing of big data.

IV. **Measured Service**

Cloud vendors charge customers by a metering capability that provides billing for a subscriber, based on pay-per-use model. This service of cloud architecture manages all cloud service pricing, subscriptions and metering of used services. This capability of cloud computing system allows an organization to pay for the current size of datasets and then pay more when dataset size increases. This service allows customers to start with a low investment.

1.4.2 Cloud Architecture

Cloud computing technology could provide by a vendor that enables IT departments to focus on their software development rather than hardware maintenance, security maintenance, recovery maintenance, operating systems and software upgrades. Also, if an IT department establishes a cloud computing system in their organization, could help them to handle big data.

The Architecture of a cloud computing system is specific to the overall system and requirements of each component and sub-components. Cloud architecture allows cloud vendors to analyze, design, develop and implement big data.

Cloud vendors provide services through service layers in cloud computing systems. The major categories are divided into four service layers: Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS), Software-as-a-Service (SaaS) and Business Intelligence (BI) and other service layers assigned to the major service layers as shown in Figure 1.1, such as Data-as-a-Service(DaaS) assigned to IaaS layer. Description of each service discussed in Section 1.5.

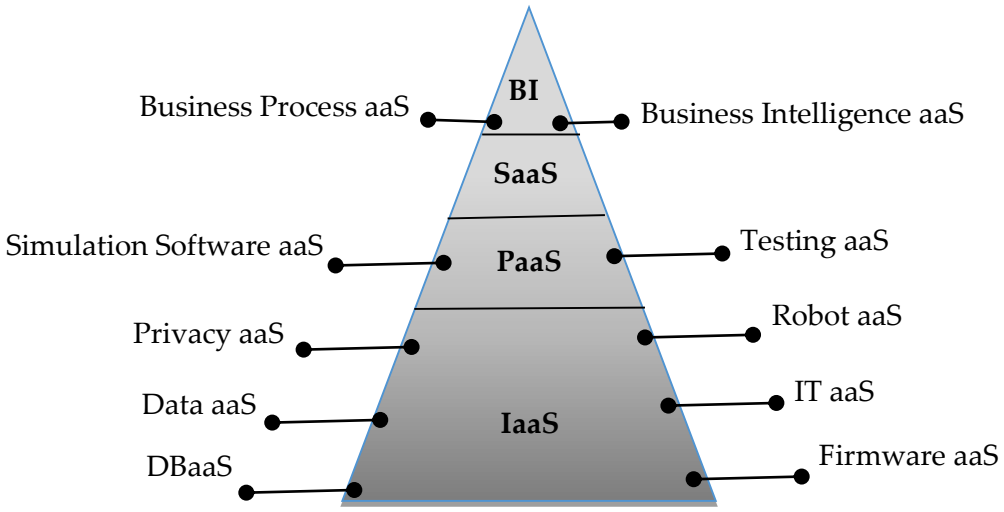


Figure 1.1. Cloud services

I. The Role of Infrastructure-as-a-Service (IaaS)

The IaaS model offers storage, processors and fundamental hardware to the cloud customers. This model covers several services, such as firmware, hardware, utilities, data, databases, resources and infrastructure. This model allows clients to install operating systems, receive quoted infrastructure, and develop and deploy required software applications. This model is often implemented via Virtualization, which enables multi users/tenants work on share machines with his own privacy.

The IaaS model provides several opportunities for big data:

i) ***storage data:*** this feature allows customers to store big data. Storage on the cloud computing system enables customers to store, retrieve and edit big data by employing a cluster of storage devices. These clusters could be added or removed dynamically;

ii) ***hardware:*** this feature enables customers have an access to a resource pool of hardware for big data. This feature could be used for capture data, such as through sensors, Radio-Frequency Identifications (RFIDs) or Communication-as-a-Service (CaaS). The CaaS is responsible for the required hardware and software for delivering Voice-over-IP (VoIP), audio and video conferencing. The hardware feature also provides network access and network traffic control that could to transfer big data.

Amazon Elastic Compute Cloud (Amazon EC2) provides virtual and scalable computing systems at the IaaS. Amazon EC2 customers could define instances of a variety of operating systems (OSs). Each OS and required hardware, such as CPUs and RAMs could be customized by a customer on the fly. Customers should create an Amazon Machine Image (AMI) in order to use Amazon EC2. The AMI contains the

required applications, operating systems (the customer could select various operating systems such as Windows or Linux versions), libraries, data and system configuration. Amazon EC2 uses Amazon S3, which is a cloud storage service and stores data and uploads AMI into S3.

The impact of big data in this service layer is higher than other service models in cloud computing systems, because IaaS users could access and define the required *data framework*, *computing framework* and *network framework*.

In a *data framework*, users could define structured data, unstructured data and semi-structured data. Structured and semi-structured data could be defined via traditional databases, such as RDBMS and OODBMS. In these models, structured data stored which has a schema before adding data to the databases. All of data frameworks and in particular unstructured data could be defined by cloud databases, such as Hadoop which is based on MapReduce programming model. MapReduce programming language technique allows storing data on a cluster of resources. The implementation model of MapReduce is provided by Hadoop which is provided a category of open-source database, applications and analytics tools.

In *computing framework*, users have full-permission for developing, installing and running new application for computing purposes. Each application could reserve a cluster of CPUs and RAMs. Several tools and databases with analysis tools emerge to provide computing framework on big data. For example, Hive is an open-access "SQL-like" BI tools that allows BI applications to run query on Hadoop data. Other example, Pig is another open-source platform that allows analyzing on big data by a "Perl-language-like" feature.

In *network framework, users have a* significant benefit, because they have access to required network control, such as network cards and the Internet connectivity. For example, they could access to regular network transfer infrastructure such as Optical Carrier (OC) 768 backbone (Cartier 2014), which is capable of transferring 39,813.12 Mbit/s.

This accessibility to data, computing and network framework allows the users to control require hardware like an administrator in IT department. However, these users could handle infrastructure without worrying about maintenance.

II. The Role of Platform-as-a-Service (PaaS)

PaaS is a platform that provided by cloud vendor. The PaaS model does not require users to setup of any software, programming language, environment application, designer, tools or application. Developers use vendor's platform, library and programming language for developing their applications. This model provides a software application

for outgrowth of the cloud applications delivery. PaaS allows developer to focus on software application development, without worrying about operating system maintenance like in IaaS. The PaaS provides services for software programmers to develop and deploy their applications with an abstraction on the hardware layer.

The role of PaaS in handling big data is less than IaaS, because some restrictions and limitations are applied to PaaS users in order to work on the data framework, computing framework and transfer frameworks. In this service layer, users are limited to cloud vendor frameworks. For example, Google App Engine provides a platform which supports Python, Java, PHP, Go and MySQL compatible Cloud SQL to develop applications. So, in this service layer, users could not access other languages, such as C# or C++ and server hardware. However, developers still could build, deploy and run their scalable applications on the cloud computing systems. These applications could capture a massive amount of data from anywhere and use a cluster of CPUs for computing and analytics of big data.

III. **The Role of Software-as-a-Service (SaaS)**

The traditional model of software is to purchase software applications and install them on the local computer. However, SaaS model provides applications in the cloud though a network and does not require customers to install applications on their local computers.

According to Microsoft, SaaS model could be divided to the following categories (lower-level to higher-level) (Rittinghouse 2009):

- *Ad-hoc/Custom,* which supports by minimum requirement to migrate traditional and client/server application to this level. Ad-hoc/Custom models allow developer to build their application based on ad-hoc or peer-to-peer technology;
- *Configurability,* which provides more flexibility through configuration metadata and supports peer-to-peer technology;
- *Multi-tenancy,* which adds multi-tenancy to the configuration level, and a single instance of application allows serving all the vendor's consumers;
- and *Scalability,* which supports all other lower-levels. In addition, this level supports scalability through architectural design that adds a capability of dynamic load-balancing for growing or shrinking cloud servers. Most applications in the cloud are developed at this level.

The impact of SaaS is less than PaaS, because in this service layer, users could use provided applications and resources. This service layer is limited to developers. However, users still could work on big data that could be added before or captured by provided infrastructure. For example, Google Apps, such as Gmail, provides services on

the web and users could not add or manipulate capturing data from server. Users are limited to web-based interface for email processes such as sending an email.

IV. The Role of Business Intelligence (BI)

The BIaaS layer sits on the top of cloud architecture service layers and aims to provide the required analytic models for cloud customers.

Information granularity as (Pedrycz 2013) defined, it is a structure which plays a key role in human cognitive and decision-making computing. The BI service layer could provide a platform for information granularity on the cloud computing and in particular granular computing, which is a processing of complex information entities. Unlike the traditional computing, cloud computing by granular computing on big data may provide a significant result. For example, (Bessis at al. 2010) propose a big picture by collecting big data and using cloud computing for managing disasters.

Cloud computing could provide the following information granularity and granular computing infrastructures (Pedrycz 2013):

- A granular description of data and pattern classification by non-SQL databases, such as SciDB (Cudré-Mauroux 2009);
- A representation of information granules by migrating traditional applications to the cloud;
- Different granular architecture and development by collecting information from different sources and computing with high quality rather than traditional models which were working with a limited computing resource;
- Collaborative and linguistic models of decision-making by collecting information from different sources at the cloud storages.

The information-processing level (Bargiela 2003), which is encountering a number of conceptual and algorithmic layers indexed by the size of information granular, could be high if a cloud application provides a computing model. However, if a cloud application provides only a storage model, this impact and granular computing will be low. For example, when an application provides a service for collecting data from financial consumers and running an analytical model on this data to make a decision about investment, cost and profit, this application has a high-level BIaaS impact. For instance, (Xu et al. 2009) present "*Big Cloud based Parallel Data miner (BC-PDM)*" which is a framework for integrating data mining applications on MapReduce and HDFS (Hadoop File System) platforms.

Cloud based BI could reduce the total development cost, because cloud computing systems provide environment for agile development and reduce the maintenance cost. Also, the BI could not be implemented on a traditional system, because the current volume of data for analysis is massive. BI-as-a-Service (Zorrilla et al. 2013) is other example that shows how the BI could migrate to the cloud computing systems as a software application in the SaaS layer.

One of the major challenges with traditional computing is analysis of big data. Cloud computing at BIaaS layer could handle this issue by employing a cluster of computing resources. For example, SciDB (Cudré-Mauroux 2009) is an open-source and cloud-based database management system (NoSQL DBMS) for scientific application with several functions for analyzing of big data, such as astronomy, remote sensing and climate modeling.

V. Other Service Layers

The major service models of cloud computing are BIaaS, IaaS, PaaS and SaaS. As shown in Table 1.1, we assigned each service to the major service models.

1.5 Big Data Tools

The Table 1.2 shows a summary of big data open-source tools which are provided through cloud computing infrastructures. Most of the tools are provided by Apache[2] and released under the Apache License. We categorized each tool based on those applications of big data.

[2] http://apache.org/

Table 1.1. Other service layers in Cloud Architecture

Service name	Related to	Service Description	Role of Service in Big Data
Business-Process-as-a-Service (BPaaS) (Accorsi 2011)	BIaaS	Automated tool support	Analysis of big data
Business-Intelligence-as-a-Service (BIaaS) (Hunger 2010)	BIaaS	Integrated approaches to management support	Analysis of big data
Simulation Software-as-a-Service (SimSaaS) (Tsai 2011)	SaaS	Simulation service with a MTA configuration model	Analysis of big data
Testing-as-a-Service (TaaS) (Candea et al. 2010)	SaaS	Software testing environments	Test big data tools
Robot-as-a-Service (RaaS) (Chen et al. 2010)	PaaS	Service-oriented robotics computing	Action on big data
Privacy-as-a-Service (PaaS) (Itani et al. 2009)	PaaS	A framework for privacy preserving data sharing with a view of practical application	Big data privacy
IT-as-a-Service (ITaaS) (Foster et al. 2005)	IaaS	Outsource IT department's resource (on Grid infrastructure that time)	Maintaining of big data
Hardware-as- a Service (HaaS) (Stanik et al 2012)	IaaS	A transparent integration of remote hardware that is distributed over multiple geographical locations into an operating system.	Capturing and maintaining of big data
Database-as-a-Service (DBaaS) (Curino et al. 2011)	IaaS	(1) a workload-aware approach to multi-tenancy (2) a graph-based data partitioning algorithm (3) an adjustable security scheme	Storing big data
Data-as-a-Service (Daas) (Truong et al. 2009)	IaaS	Analyzing major concerns for data as a service	Storing big data
Big-Data-as-a-Service (Zheng 2014)	All layers	Service-generate for big data	Generate big data

Table 1.2. Cloud-based big data open-source tools

Big Data Tools	Description[3]
Data Analysis Tools	
Ambari[4]	A web-based tool for provisioning, managing, and monitoring Apache Hadoop clusters.
Avro[5]	A data serialization system.
Chukwa[6]	A data collection system for managing large distributed systems.
Hive[7]	A data warehouse infrastructure that provides data summarization and ad hoc querying.
Pig[8]	A high-level data-flow language and execution framework for parallel computation.
Spark[9]	A fast and general compute engine for Hadoop data. Spark provides a simple and expressive programming model that supports a wide range of applications, including ETL, machine learning, stream processing, and graph computation.
ZooKeeper[10]	A high-performance coordination service for distributed applications
Actian[11]	An Analytics Platform which accelerates the analytics value chain from connecting to massive amounts of raw big data all the way to delivering actionable business value.
HPCC[12]	Provide high-performance, data-parallel processing for applications utilizing big data.
Data Mining Tools	
Orange[13]	A data visualization and analysis for novice and experts.
Mahout[14]	A scalable machine learning and data mining library.
KEEL[15]	An assess-evolutionary algorithm for data mining problems.
Social Network Tools	
Apache Kafka	A unified, high-throughput, low-latency platform for handling real-time data feeds.
BI Tools	
Talend[16]	A data integration, data management, enterprise application integration and big data software tools and services.
Jedox[17]	An analyzing, reporting and planning functions.

[3] The description retrieved from each tools' official website
[4] http://ambari.apache.org/
[5] http://avro.apache.org/
[6] http://incubator.apache.org/chukwa/
[7] http://hive.apache.org/
[8] http://pig.apache.org/
[9] http://spark.incubator.apache.org/
[10] http://zookeeper.apache.org/
[11] http://www.actian.com/about-us/#overview
[12] http://hpccsystems.com/
[13] http://orange.biolab.si/
[14] http://mahout.apache.org/
[15] http://keel.es/
[16] http://www.talend.com/
[17] http://www.jedox.com/en/

Table 1.2. Cloud-based big data open-source tools (Cont.)

Big Data Tools	Description[18]
Data Analysis Tools	
BI Tools (Cont.)	
Pentaho[19]	A data integration, business analytics, data visualization and predictive analytics.
rasdaman[20]	A multi-dimensional raster data (arrays) of unlimited size through SQL-style query language.
Search Tools	
Apache Lucene[21]	An application for full text indexing and searching capabilities.
Apache Solr[22]	A full-text search, hit highlighting, faceted search, near real-time indexing, dynamic clustering, database integration, rich document (e.g., Word, PDF) handling, and geospatial search.
Elasticsearch[23]	A distributed, multitenant-capable full-text search engine with a RESTful web interface and schema-free JSON documents.
MarkLogic[24]	A NOSQL and XML database.
mongoDB[25]	A cross-platform document-oriented database system, JSON-like documents with dynamic schemas.
Cassandra[26]	A scalable multi-master database with no single point of failure.
HBase[27]	A scalable, distributed database that supports structured data storage for large tables.
InfiniteGraph[28]	A distributed graph database.

1.6 Implementation Models of Cloud Computing Systems

A Cloud computing system based on infrastructure location could be implemented as Private, Public or Hybrid cloud.

The ***private model*** is a local implementation of cloud computing system. In this model, hardware is located in local data centers and uses cloud software applications to provide service to local users. This model is the best option for consumers who needs cloud computing capabilities with low-risk in IT departments because this model allows

[18] The description retrieved from each tools' official website
[19] http://www.pentaho.com/
[20] http://rasdaman.eecs.jacobs-university.de/
[21] http://lucene.apache.org/
[22] http://lucene.apache.org/solr/
[23] http://www.elasticsearch.org/
[24] http://developer.marklogic.com/
[25] http://www.mongodb.org/
[26] http://cassandra.apache.org/
[27] http://hbase.apache.org/
[28] http://www.objectivity.com/

an IT department to migrate from the traditional model to the cloud computing system and does not require data to be migrated to another location (such as cloud vendor location). This model is implemented for local trusted users. This model still allows scalability, on-demanded self-service, and elastic service. However, this model requires high investment in maintenance, recovery, disaster control, security control, and monitoring.

In addition, the private cloud computing model enables an IT department to handle a local organization's big data by its own infrastructure, such as the storage of big data and computing big data. This model provides a flexible resource assignment and could enhance the resource availability.

Several open source applications have been developed for establishing private cloud computing based on IaaS and SaaS service layers. For example, CloudIA is a private cloud computing system at HFU (Doelitzscher et al. 2011). The targeted users of the CloudIA project are HFU staff and students running e-Learning applications, and external people for collaboration purposes.

The ***public model*** is a regular model of cloud computing system. This model is provided by cloud vendor who supports billing and a subscription system for public users. This model, unlike a private model, does not require high investment, because consumers could pay on pay-per-use basis for cloud storage or cloud computing services on demand.

The **hybrid model** composes private and public clouds. This model could connect a private cloud to public cloud through network connection, such as the Internet.

This model has several advantages, which are listed below:

- *Collaboration between cloud computing systems:* Often collaboration between two clouds led to emergence of hybrid cloud model. An organization could keep their own cloud security and maintenance, and simultaneously have collaboration with other clouds. This collaboration could be permanent or temporary.
- *Scalability:* This model also is useful for extending the scalability of a private cloud computing system, because in case of limited resources at a peak time, a cluster of new resources could be added temporary from another cloud.

1.7 Cloud Computing Issues

The cloud computing technology is the best option for dealing with big data. However, cloud computing is still nascent state and we still needed to address some major issues. In this section, we review the major cloud computing issues based on (IDC Enterprise Panel 2013).

When big data costs customers, and a system disaster could cause organizational destruction in the digital age, migration applications and databases from traditional model are difficult to cloud, because:

- migration to the cloud computing system is difficult; Migration requires to redevelop applications, data and sometimes requires to use efficient programming models to save resources as well as resource costs;
- returning data to the IT department is difficult;
- connection is via an unsecured network, such as the Internet;
- cloud vendor administrator users could have an access to users' data;
- data warehouse location is transparent to consumers;
- We do not have a cloud computing standard and standard cloud architecture. It causes some big issues, such as different architectures, difficulty with migration data and application to another cloud vendor;
- We do not have any customization in cloud computing systems;
- We do not have a strong Service Layer Agreement (SLA) for customer satisfaction.

Cloud customers need to have a contract with one or more cloud vendor(s) -often one cloud vendor- and they should use the provided operating systems, middleware, APIs and/or interfaces. Data and application are dependent on the platforms or are provided by cloud vendor infrastructure. This dependency in cloud services has several issues. For example, "*Security*" is the major concern in cloud computing systems. Cloud features, such as a shared resource pool and multi-user/tenancy causes security issue because the resourced pool are shared through users and we could expose users' data and users' privacy to others.

Unsecured connection to the vendor, network access security, Internet access security and cloud vendors' user security emerged as other major security concerns based on accessibility to the cloud via the Internet.

"*Bringing back in-house may be difficult*" with 79.8% issue rate and "*Hard to integrate with in-house IT*" with 76.8% issue rate indicates customers are afraid of data and software application migration to the cloud computing systems, because the migration is difficult to integrate with IT departments and it is difficult to return data back to the IT department; "*Lack of interoperability standards*" with 80.2% is another cloud issue. This issue shows that cloud computing requires higher interoperability with other cloud computing systems; also as indicated in this report, "*Not enough ability to customize*" with

a 76.0% issue rates show, the cloud computing system requires dynamic architecture and customization.

Some studies, such as (Juve 2009) show existing cloud computing systems (Amazon EC2 in this case) could not be responsible with a cost-effective performance for HPC applications over using tightly-couple hardware, such as Grid Computing or Parallel Computing systems.

1.8 Chapter Summary

In this chapter, we discussed a definition of big data, the importance of big data, and major big data challenges and issues. We understand that, if we analyze big data with business intelligence tools, we may provide a catalyst to change an organization to a smart organization. We discussed the importance of cloud computing technology as a solution to handle big data for both computing and storage. We reviewed the capabilities of cloud computing systems that are important for big data, such as resource scalability, resource shrink-ability, resource pool sharing, on-demanded servicing, elastic servicing, and collaboration with other cloud computing systems. We explained cloud architecture service layers and role of each service layer to handle big data. We discussed how business intelligence could change big data to smaller valuable data by using cloud computing services and tools. Finally, we discussed major cloud computing system issues that need to be addressed for cloud computing to become a viable solution for handling big data.

Chapter 2

Dynamic Cloud Architecture

In the previous chapter, we describe the definition of big data. We introduce a set of cloud-based tools for collecting and analyzing big data. We also define the architecture of cloud computing which is divided into multiple layers. This chapter summarizes some critical challenges of cloud architecture as well as our proposed dynamic architecture to overcome the issues.

2.1 Introduction

Cloud computing is based on a distributed and parallel computing systems that provide elastic storage resources and computing resources over the Internet. As described in the previous chapter, the cloud computing paradigm allows customers to pay for their resource usage based on pay-per-use model, and enables customers to scale their storage and computing resources up or down on- demand.

An important aspect of cloud computing is *cloud architecture* that refers to the components (e.g., service layers), subcomponents (e.g., security or message passing in service layers), and overall system organization of cloud computing. Moving successfully into cloud computing requires an architecture that will support new capabilities for migrating different traditional services and applications to cloud computing systems. Such an architecture should support all user domains of a cloud computing system which includes cloud vendors, cloud developers, cloud customers or cloud vendors' partners, and end-users.

2.2 Motivation

Cloud computing services relies on the vendor infrastructure. This dependency causes several issues which are described in (IDC Enterprise Pnael 2009; Moreno-Vozmediano et al. 2013; Sasikala et al. 2013). For example, according to the IDC Survey (IDC Enterprise Pnael 2009) 79.8% people say "*Bringing back in-house may be difficult*" is another issue and

76.8% people say "*Hard to integrate with in-house IT*" is an issue. These issues indicate consumers are afraid of migrating to cloud computing systems because the migration is difficult to integrate with IT department services and it is difficult to return the data back to the IT department. The survey shows 80.2% of people say "*Lack of interoperability standards*" is another concern. Thus, cloud computing requires interoperability with other cloud computing systems; also as indicated in this report, 76.0% of respondents answer that "*Not enough ability to customize*" is an issue. Similar significant concerns around cloud computing are reported recently in other studies (Moreno-Vozmediano et al. 2013; Sasikala et al. 2013, Shayan 2013). Furthermore, all of these concerns show that cloud computing systems require flexibility in defining a variety of services that meet specific cloud users' requirements. The flexibility in defining services can be implemented by a customizable architecture that allows a vendor to define a service for each group of users.

Cloud vendors provide several services to their customers through a general multi-tier architecture (IaaS, PaaS and SaaS). Although this architecture is useful for several customers' requests, customers may have own specific request. Customers should adapt his request based on offered services, because each offered service intends to satisfy unique user requests. For example, when a customer requests a service in PaaS for developing an image processing application, the customer has the same accessibility to Application Programming Interfaces (*API*)) as other customers who develop a web mining application on a cloud. However, an image processing application requires specific functions (e.g., spatial transformations) that are different in type and not useful for a web mining application that requires more specific network functionality (e.g., spatial indices). This example shows that the customization of a service by a cloud vendor allows a cloud vendor to provide unique service to each customer. A customized service allows customers to have a simple system or *API* rather than a complex system or a complex *API* that intends to satisfy different users' demands. For example, a cloud vendor could define a customized service that only satisfies a small group of partners or users, such as a group of users who only need Voice-over-IP service (*VoIP*) in a cloud computing system.

Another concern in cloud computing is an increasing demand for the introduction and migration of a variety of services to cloud computing systems. Although each service provides a new feature, such as Simulation-as-a-Service (Tsai et al. 2011) or Robot-as-a-Service (Chen et al. 2010), it aggravates migration issues and complexity issues due to the lack of standardization and customization, respectively because each cloud-based service has its own features, requirements and output. For example, Robot-as-a-Service provides a platform to control robot devices through a cloud computing system. This

service requires different resources and it provides different outputs. A dynamic architecture allows vendors to add/edit their services and future **-as-a-service*s to their cloud computing systems with ease.

In this chapter, we propose a dynamic and customizable architecture that targets mentioned concerns, such as providing customizable and dynamical services, a standardization for different cloud vendors with different solutions, supporting different services in a cloud computing system, and a solution for cloud vendor lock-in issue.

2.3 Related Work

Currently, we do not have a generally accepted standard for cloud computing. Unlike the Internet which was developed by the U.S. government agencies (Kaufman 2012), such as *ARPA* (Leiner 2009), cloud computing has been developed by several open-source groups and leading business companies, such as Microsoft and Amazon. Therefore, several independent cloud architectures have been developed.

To the best of our knowledge, no effective architecture exists that supports dynamic customization. As previously discussed, the lack of ability for customization is one of the major issues in existing cloud architectures. This drawback of existing cloud architecture creates other issues, which are discussed in Section 2.2, such as migration issues. We have several solutions to overcome this drawback by implementing customization at different level of cloud computing systems. As shown in Figure 2.1, we divided customization of cloud computing systems into conceptual level, architecture level and implementation level. In the following section, we review related work in each level of customization.

2.3.1 Conceptual Level

Conceptual level provides a high-level definition of a customized system. Based on customization at the conceptual level, we can define an architecture and its implementation. For example, one of the conceptual customization is Mass Customization (*MC*) (Pine 1999) which is based on marketing and manufacturing. MC focuses on developing one product with different features. For instance, (Hu et al. 2013) proposed a mass customization for their proposed cloud architecture (*CCRA*), which enables a cloud vendor to define a cloud architecture requirements and its implementation. In their architecture, different models of one object could be defined by a conceptual model. Each object has different features. Their concept provides different services through a dynamic domain with different abstractions which is called a model. Although Hu et al. provide a customization model in cloud computing, the model is not

adoptable because authors did not provide the specific detail of implementation methods for a diverse environments.

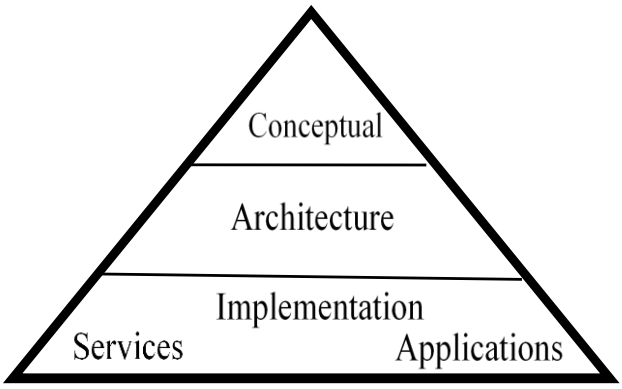


Figure 2.1. Customization levels in cloud computing

2.3.2 Architecture Level

Existing cloud architectures are static, and are divided into the following categories:

(i) Service-Oriented Architectural (*SOA*)-based (Perrey et al 2003): (Tsai et al. 2010) provided *SOCCA* which is a combination of Enterprise *SOA* style and cloud style and Zhang et al. provide *CCOA* (Zhang et al. 2009) architecture based on *SOA* with a scalable service feature, but these cloud architectures do not provide customization on each service layer;

(ii) Cloud Reference Architecture (*CRA*) (Liu et al. 2011)which is developed by *NIST*. This architecture has five primary actors: Cloud Service, Consumer, Cloud Service Provider, Cloud Broker, Cloud, Auditor and Cloud Carrier;

(iii) Open forums, such as *OGF* Open Cloud Computing Interface (Metsch et al. 2010), Cloud Computing Interoperability Forum (CCIF)[29], Deltacloud (Bist et al. 2013), DMTF[30], Open Stack (Bist et al. 2013), Open Cloud Consortium[31] and Open Cloud Computing Interface (OCCI)[32] (Grossman et al. 2010).

The idea behind most of these open source clouds is to provide a *common interface* that includes major cloud platforms. However, in this chapter, we propose an architecture that allows vendors to define and implement their own specific service through a

[29] Available from: http://www.cloudforum.org/
[30] Available from: http://dmtf.org/standards/cloud
[31] Available from: http://opencloudconsortium.org
[32] Available from: http://occi-wg.org/about

standardized layer cross all other vendors' platforms. In the proposed architecture, the vendor uses a layer to provide standard services to their customers. The vendors are not required to modify their platform and they can provide an extension layer on the top of their cloud platform.

Existing cloud architectures do not provide any solution for facilitating different services. In addition, existing cloud architectures are static and could not easily provide a customization on services.

2.3.3 Implementation Level

Customization at implementation level allows a vendor to define several separate services and applications. Customization at this level is often tied to a vendor's platforms and infrastructures.

Major customization at implementation level has been developed by using *Object-Oriented* (*OO*) paradigm. The concept of *OO* enables developer to implement an application based on different objects which are closely linked. For customization reasons, several implementation models have been developed for cloud computing systems. For example, (Bahga et al. 2013) provide a *Cloud Computing Model* (*CCM*) which is a component-based model for cloud computing systems. The *CCM* allows a developer to provide multiple components which are connected via *Uniform resource identifier* (*URI*) and uses message passing. Although, the model provides a customization for cloud applications, *CCM* is relied on cloud architecture.

The *CCM* has several drawbacks. For example, if an architecture is non-functional, then the implementation model cannot provide an efficient model. For example, limitation on network access at *PaaS* layer it causes limitation on *CCM* application (i.e., the lack of accessibility to a protocol). Implementation of *CCM* also has some drawbacks because the model depends on the architecture with specific requirements, such as type of programming language. These issues show disadvantages of a cloud architecture could be caused issue in the implementation.

2.4 The Proposed Architecture

This section presents a *Dynamic Cloud Computing Service-Oriented Architecture* (*DCCSOA*) that allows cloud vendors to analyze, design, develop and implement a cloud computing system. The *DCCSOA* provides a dynamic service layer that allows a vendor to add new customized services on-demand.

A dynamic architecture for cloud computing allows cloud vendors to customize their services. As shown in Figure 2.2, the architecture is based on *SOA*. The *SOA* features enable an architecture to provide several independent services that work together as a system and can be run on different cloud computing systems. The proposed architecture can customize value-added cloud services (offered resources on a cloud computing system). In the proposed architecture, a dynamic layer represents all heterogeneous services, and it can customize services on-demand.

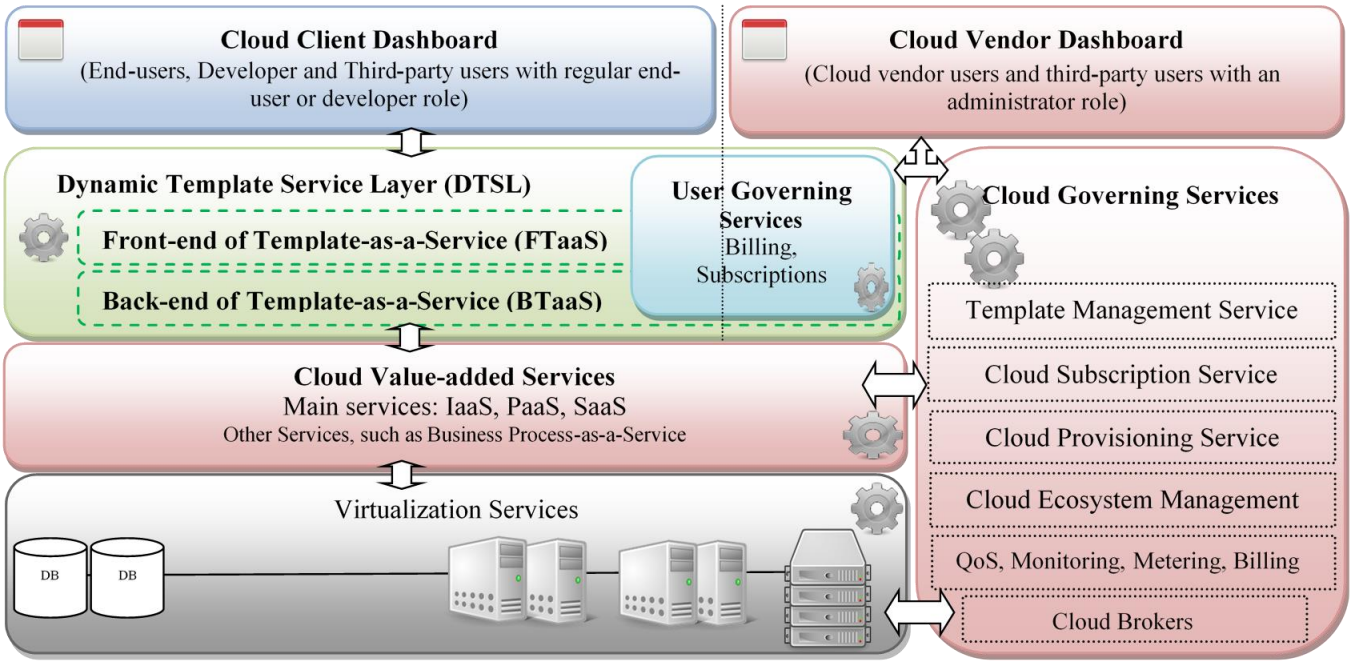


Figure 2.2. The Architecture of DCCSOA

2.4.1 DCCSOA Components

The *DCCSOA* has several service layers that are discuss as follows:

I. **Dynamic Template Service Layer (DTSL):**

The DTSL provides a dynamic and customizable bridge between all value-added services in a cloud computing system and all cloud user groups, such as cloud vendor users, cloud customers (partner of cloud vendors), cloud developers and cloud end-users. The *DTSL* is a primary component of the proposed architecture and it provides a service layer which we call "*Template-as-a-Service (TaaS)*". The *TaaS* provides a dynamic customization on value-added services. The *DTSL* is divided into two sub-layers as follows: "*Front-end of*

Template-as-a-Service (FTaaS)" and "*Back-end of Template-as-a-Service (BTaaS)*". The *FTaaS* provides customized value-added cloud services to cloud clients by *Cloud Client Dashboard*. The *BTaaS* is only available to cloud vendors and it interacts with all cloud services, such as all traditional service (IaaS, PaaS and SaaS), other service layers (e.g., Firmware-as-a-Service, Robot-as-a-Service). The classification of *DTSL* into *BTaaS* and *FTaaS,* makes a cloud architecture progressively deployable alongside existing cloud technologies without significant barriers or overhead because the *BTaaS* defines a dynamic layer which can be modified and customized by a cloud vendor. The *BTaaS* can be developed alongside of existing cloud service layers. The *FTaaS* forms the customer interface and subscriber audits.

A cloud vendor defines several different services on-demand at *DTSL*. Each defined service is a *Template* which is integrated with one or multiple value-added cloud services. Cloud vendors can set up, configure and provide different *templates* to their customers based on different value-added service layers in a cloud computing system. As illustrated in Figure 2.3, a *template* at the back-end of *DTSL* is dynamic, and it interacts with one or multiple value-added cloud services.

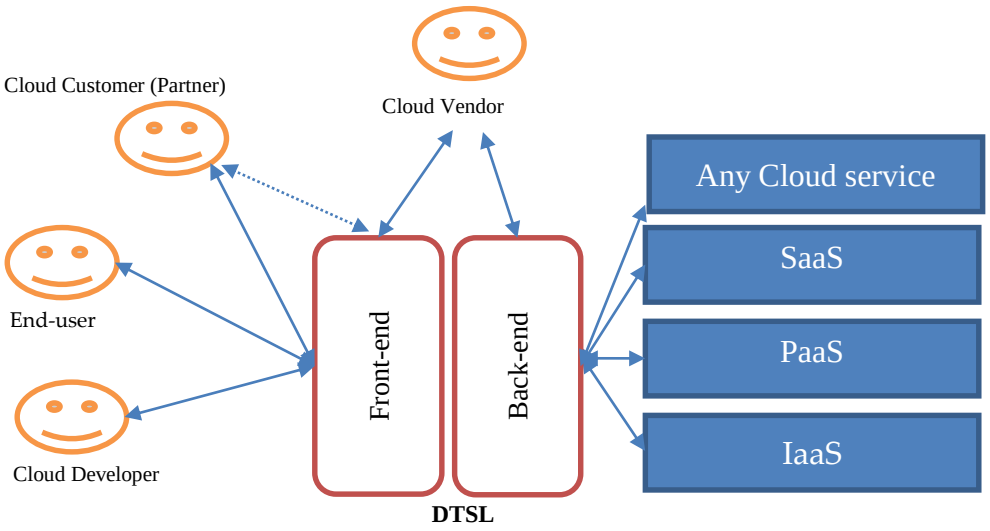


Figure 2.3. One snapshot of DTSL layer and connection to cloud value-added services

A cloud vendor can define several *templates* at *FTaaS* where each *template* provides cloud services to end-users. The *FTaaS* allows different vendors to define the same template to their customers. This feature provides independent value-added service to customers who need data and applications migration from one cloud to another cloud. Cloud vendor are able to define their own service layer with a *BTaaS*. The *BTaaSs* differ from vendor to vendor and they provide a transfer from heterogeneous services to *general templates.* For example, in Figure 2.4, if two vendors (V_1 and V_2) provide different *IaaSs*

(*IaaS*$_1$ and *IaaS*$_2$), each vendor can provide a template as *IaaS*$_x$ at *FTaaS*. *BTaaS* in V_1 is different from *BTaaS* in V_2. Both vendors should use his own *BTaaS* to configure the back-end of his *IaaS*$_x$.

The dynamic customization feature of the *BTaaS* layer enables a cloud vendor to customize their own services and it provides standard services through the *templates*.

A cloud vendor can edit a layer by adding, editing or removing a *template* as shown in Figure 2.5. In this figure, rows represent cloud-value added services (traditional services layers) are static, such as *IaaS*, *PaaS* and *SaaS* or other service layers, such as future services. In this figure, columns represent *templates* and are dynamic that can be defined by a cloud vendor on-demand. Four templates are defined in Figure 2.5. For example, a user who has access to T_1 can use *SaaS* layer, or a user has access to T_3 can access to *SaaS* and *PaaS* layers.

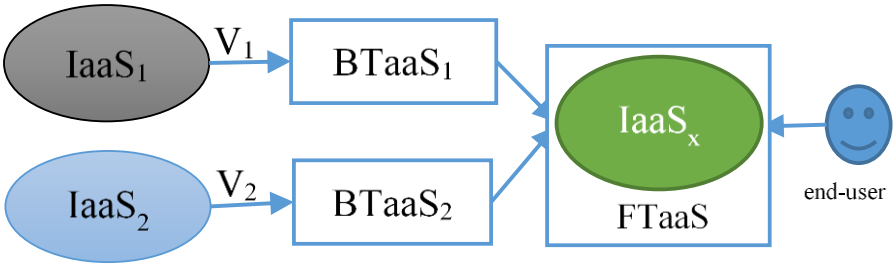


Figure 2.4. An example of a *template* (*IaaS*$_x$ *template*) with two different back-ends for two different platforms

The *templates* can be implemented for any kind of cloud services and traditional services. For instance, a cloud vendor can define several services as a *template*, such as Business-Intelligence-as-a-Service (*BIaaS*) and *IaaS*. In this case, Figure 2.5 will be changed and rows represent *IaaS* and *BIaaS* layers, and columns can be defined by a vendor.

The number of columns is dynamic, and is defined by a vendor. Each column stands for a *template*. The vendor defines several *templates* which make use of resources in one or multiple layers in a cloud computing system. For example, in Figure 2.5, T_1, T_2, T_3 and T_4 are cloud *templates* (orange colors). T_1 interacts with *SaaS* value-added service layer, T_2 interacts with all value-added service layers in a cloud computing system, T_3 interacts with two value-added service layers (*SaaS* and *PaaS*) and finally T_4 interacts with two lower-level value-added service layers (*PaaS* and *IaaS*).

The customer groups include end-users, developers and third-party users (with end-user, or developer role). They use *Cloud Client Dashboard* for interacting with *FTaaS* to use cloud

resources. Each user has an option for working on several cloud value-added services simultaneously by interacting with a *template*. For instance, a developer who uses T_4 *template* in Figure 2.5, can work on *PaaS* and *IaaS* simultaneously. The developer can work on *IaaS* to install a new application (App_1) on the server and she has access to *PaaS* simultaneously for developing a Mashups application which is required App_1.

II. Cloud Client Dashboard (CCD)

This component provides an interface to a group of end-users, developer and third-party users. Although the third-party users are collaborating with a cloud vendor to develop or provide cloud services or applications, they may use resources as regular cloud users. Each user can subscribe to a *template* rather than a service in traditional cloud computing systems. The dashboard provides a list of *templates* that each group of users can subscribe for billing tools to provide billing on resource usage of a *template*, and monitoring tools to provide monitoring on all subscribed *templates*. The *CCD* interacts with *FTaaS* to provide cloud services based on defined *templates*.

III. Cloud Vendor Dashboard (CVD)

The *CVD* component provides an interface to high-level users, such as system administrators and third-party users (with an administrator role). The *CVD* is isolated from regular users to provide a secure layer to cloud administrators who work on configuring, adding and editing cloud *templates*.

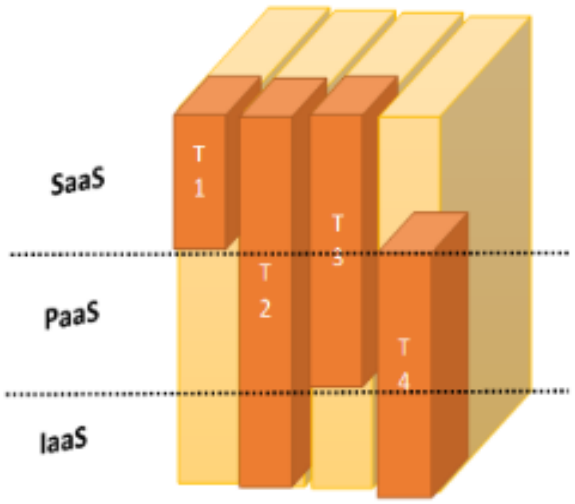


Figure 2.5. One snapshot of the dynamic BTaaS layer

IV. **User Governing Services (UGS)**

This service provides control and configuration of the *DTSL* for *Pricing*, *Billing and Subscription* services for each defined *template*. This layer sits on the *DTSL* because this service requires a list of users who are subscribed to *templates*. This service also interacts with the *Cloud Governing Services (CGS)* to enable high-level users to configure and control a cloud ecosystem.

V. **Cloud Governing Services (CGS)**

This service is accessible through *Cloud Vendor Dashboard* for administrative users. This service includes the following services: *Template Management Service (TMS)* which controls the *DTSL* to develop *FTaaS* and *BTaaS*. The *TMS* interacts with *BTaaS* and *Cloud Value-Added Services* layer to provide cloud services through a *template. Cloud Subscription Service* provides a management service for defining a different type of subscriptions as well as billing and pricing methods for each *template*. *Cloud Provisioning Service* provides a management service for resources and it provision elastic services based on *Cloud Subscription Service*. *Cloud Ecosystem Management Service* provides an integrated model of cloud interdependent components. *Quality of Services (QoS)* service provides a control management on overall performance of cloud services. *Monitoring Service* monitors cloud templates and the customer applications which run on the cloud. *Metering and Billing Services* provide a payment structure and access to one or multiple *templates*.

VI. **Virtualization Services (VS)**

This service layer provides a virtualization tools for storage, computing, and other resources. This service includes *Dispatcher*, *Storage and Programming API Tools*, and *Virtual Machine (VM) Services*, such as *Virtual Machine Monitors*.

2.5 Advantages of the Proposed Architecture

The *DCCSOA* has several advantages which are described as follows:

I. **Customizable architecture**

The dynamic component of the proposed architecture (*DTSL*) allows cloud vendors to modify and customize their cloud architecture on demand. This customization improves cloud architectural issues, such as *lack of usability of cloud computing* because a cloud vendor defines a new *template* that covers several services for enabling customers to have an integrated service. This offer will be more attractive for a variety group of users because a vendor is able to provide different customized services via different *templates*. For example, in traditional cloud computing systems, a telecom (Pal et al. 2011) user who

needs one or more network functions should find a cloud vendor who provides *IaaS* and subscribe to this service. However, a cloud vendor can define multiple services (e.g., a *VPN* service and a storage service) in a *template* for a group of users, such the telecom user.

II. **Flexibility and accessibility**

The *DTSL* gives more flexibility and accessibility to customers through a *template* that provides several services at the back-end of *templates* (*BTaaS*). As a result, cloud vendors are able to offer different *cloud templates* to their customers. Each *template* could be an integration of one or more services. For example, in Figure 2.5, a cloud vendor provides four different *templates*, and cloud users who work on template T_3 can interact with *PaaS* and *SaaS* layers simultaneously.

III. **Dynamic Abstraction**

The proposed architecture abstracts and encapsulates higher-level service layers from lower-level service layers by defining a *template* in *DTSL* that exposes lower-level services to advanced customers, and expose higher-level services to regular users or a customized service from both levels to a group of users. For example, in Figure 2.5, a vendor offers template T_4 to advanced customers who need service in *PaaS* and *IaaS* service layers. The reason for this exposure is to improve flexibility and accessibility for some customers who need access to different and multiple services.

The *DTSL* facilitates the customers' migration to the cloud and return back to the in-house *IT* department because a cloud vendor can provide a *template* at *DTSL* that has the similar features to in-house *IT* or other cloud vendors. For example, in Figure 2.5, customers who interact with T_4 can access to *IaaS* to setup an operating system, and use a cloud platform simultaneously.

IV. **Portability of applications and data in cloud**

The portability of both applications and data in cloud computing is another advantage of *DTSL* which is divided into *FTaaS* as front-end and *BTaaS* as back-end. As previously described, a lack of portability in cloud for both applications and data, that causes vendor lock-in issues, is a major issue in cloud computing systems. The *DCCSOA* enables different cloud vendors with heterogeneous infrastructures provide the similar *FTaaS* to their customers. The similar *FTaaS* allows customers to migrate data and applications to other vendors. The vendors can configure different *BTaaS* based on their specific infrastructures, such as hardware or platforms.

V. **Cloud Vendor Devolution**

Current existing cloud architectures do not support cloud vendor devolution that allows a partner of a cloud vendor to develop cloud configurations. The *DCCSOA* enables a cloud vendor to define a *template*, such as T_2 in Figure 2.5. The cloud vendors can provide full access, and give a devolution role to their partners who uses a *template* (e.g., T_4). The partners can provide new *templates* which are derived from the main *templates* (e.g., $T_{4.1}$ from T_4), to their customers. For security reasons, *DTSL manager*, *DTSL monitor* and *security monitor* control this group of users. The best advantage of this permission is that a cloud partner is able to develop the cloud computing system like a cloud vendor. For instance, a cloud vendor provides a *PaaS* as a *template* in *DTSL* to her partner. Cloud partners can offer a new service to their customers based on an integrated service of *PaaS* and other services.

VI. **Security**

The *DTSL* divided into *FTaaS* and *BTaaS*. This segmentation improves cloud security because customers have access to the *FTaaS* services layer and this layer is isolated from other value-added cloud services. This isolation makes the *DTSL* more secure. In addition, any data security and privacy methods can be implemented as a *template* in the *DTSL*. For instance, Chapter 7 describes a new template for electronic healthcare systems along with its implementation for maintaining data privacy.

VII. **Standardization**

One of the major issues in cloud computing is a lack of standardization because this problem causes vendor lock-in issue. However, *DCCSOA* provides a dynamic service layer (*DTSL*) to enable different vendors to offer the same front-end (*FTaaS*) service layer. When different cloud vendors provide the same *FTaaS* to their customers, the customers could transfer their data and applications to other vendors, or they can transfer data and applications to their private clouds through defining a similar *FTaaS*.

2.6 A Framework for Comparison of *DCCSOA* to Related Work

Several parameters are important to evaluate software architecture, such as quality attributes (Bianco et al. 2007) i.e., modifiability and system independent. We consider the quality attribute to provide a framework to compare the proposed architecture against related works.

In Table 2.1, we present a comparison of the proposed architecture (*DCCSOA*) to existing architectures, methods, and cloud tools. In this table, '×' denotes that the

literature did not provide information related to a feature of their platform, or they did not consider the feature. We use the following features in our comparison:

i) customization and standardization with minimal modification to the architecture and services;
ii) the capability of supporting interoperability;
iii) support new cloud-based services. In Table 2.1, each row represents a study or a product of a conceptual model, a cloud architecture, a cloud platform or a tool. Each column represents the following items: the level of customization that indicates the ease of customization with which vendors could customize their own architectures; the level of standardization indicates the level of modifications is needed to provide a standardized cloud computing system; and the last column represents different service capabilities that show which architecture, platform or tools could support future services with ease.

Low level *Customization* indicates customization at the *Implementation Level* because each application or product requires to be modified. For example, *MC* provides a solution to modify each service to provide a customize cloud computing system. *Medium Level* indicates customization at the *Conceptual Level and Architecture Level* (unadoptable) because both architecture and the existing applications are required to be modified. For example, *CRA* provides a new architecture without adopting new features with the existing architecture. *High Level* indicates customization at the *Architecture Level* with adopting new features with the existing architecture. For example, *CCIF* provides adoptable services through a uniform cloud interface. This level requires minimal modifications to achieve customization with standard model. The solutions of interest are high level customization because they provide customized services with minimal modifications to the existing architectures (conceptual level) and existing services (implementation level). *DCCSOA* provides an independent service (*TaaS*) to provide customization on existing services. *DCCSOA* is not required to modify the existing architecture or the existing services to achieve customization with minimal modifications. *DCCSOA* is only required to modify and adopt the *templates* of each service.

Low Level standardization represents the maximal modifications to major cloud services to provide a standard service between different cloud computing systems. For example, all components are required to be modified in *CCM* to provide a standardized cloud computing system. *High Level* indicates standardization with less modifications. For example, *CCIF* provides a solution for standardization through a uniform cloud interface. The solutions of interest are high level standardization because it does not require modifying the existing architecture or existing services to achieve standard cloud

Table 2.1 A comparison between different cloud architectures and cloud platforms

Cloud Architecture		Level of Customization	Level of Standardization	Level of Interoperability	*aaS Support
Conceptual	**MC [12]**	Low (at Product line level)	×	×	×
Architecture Level	**CCM [21]**	Low (in Implementation Level)	Low (if all vendors implement standard components)	Medium (via OO paradigm)	×
	SOCCA [15]	× (Medium)	×	×	×
	CCOA [16]	× (Medium)	×	×	×
	CRA [17]	Medium	Low (if all vendors implemented based on CRA)	×	×
	DCCSOA	High (via different Templates at Arch. level)	High (via TaaS)	High (via connection between services)	Yes
Applications and Open Source Tools	**OGF Open Cloud Computing Interface [18]**	×	Low (if all vendors modify their services based on OCCI specifications)	×	×
	Cloud Computing Interoperability Forum (CCIF)[1]	High	High (if all vendors implement Unified Cloud Interface)	Low (via Unified Cloud Interface)	×
	Deltacloud [19]	Medium (at API level)	High (via REST-based API)	×	×
	DMTF [2]	Medium (via Common Information Model)	High (via message exchange)	Medium (via message exchange)	×
	Open Cloud Consortium[3]	Medium (at API level)	High (via REST-based API)	Low (via Unified Cloud Interface)	×
	Open Cloud Computing Interface (OCCI) [20]	Medium (at API level)	High (via REST-based API)	Low (via Unified Cloud Interface)	×
Platform	**Open Stack [19]**	× (private cloud)	× (private cloud)	Medium	×

interface for different types of the existing services that cloud be implemented in different cloud vendor's systems with different architectures.

Low Level Interoperability indicates interoperability via an external interface that causes high traffic connection to external entity without an ability to control or modify the interface. For example, *CCIF* provides an external uniform interface that is disabled independently of a service and each service is required to connect to the interface to provide interoperability feature. *Medium Level* indicates interoperability through implementation because each component does not rely on an external interface but the correspondence method requires to modify major services. For example, *CCM* provides interoperability through object-oriented paradigm that does not require to connecting to an external interface and in this method each object requires to be modified to achieve interoperability. *High level* indicates interoperability with independent services and minimal modifications. The solutions of interest are high level interoperability solution that minimizes the modifications of the architecture and services to achieve interoperability. *DCCSOA* provides an independent service which is not relied on external interface or required modification of service.

The last column shows the capability of cloud-based services. Other related work (methods, platforms and architectures) did not consider this feature as a part of their proposed solution. *DCCSOA* allows a cloud vendor to define, deploy, customize and standardize new services via *FTaaS* and *BTaaS*. *DCCSOA* enables a cloud vendor to add new services by implement a heterogeneous service and adopt the service at the front-end layer (*FTaaS*) to provide a customizable and standardized service with minimal modifications and with ease. This comparison shows our proposed architecture (*DCCSOA*) allows vendors to define a dynamic, standardized and customizable cloud architecture with the capability of supporting interoperability. *DCCSOA* requires minimal modifications to the architecture and services with maximal the customization.

In addition to the framework in Table 2.1, we evaluate the proposed architecture based on *SOA* evaluation (Bianco et al. 2007). The evaluation is divided into the following topics: (1) Target Platform; (2) Synchronous versus Asynchronous Services; (3) Granularity of services; (4) Exception Handling and Fault Recovery; (5) *HTTPS* or Message-Level Security; (6) *XML* optimization; (7) Use of a registry of services; (8) Legacy Systems Integration; (9) Service Orchestration.

These major topics are divided into minor evaluation items as shown in Table 2.2. Icon "☺" in Table 2.2 shows the advantage of the selected parameter topic in *DCCSOA*. For instance, the proposed method in fine-grained services topic provides advantage in flexibility feature. More details about each parameter can be found in (Bianco et al. 2007). We consider the following general scenario to evaluate the proposed method:

Scenario SC_1: "User U_1 uses a platform as follow: P_1 runs on the top of Cloud$_1$ to provide service S_1, and U_1 is willing to transfer data and application to P_2 which is running on the top of $Cloud_2$ for the same service. When U_1 needs to transfer data and applications from P_1 to P_2, administrator of P_2 needs to define the same service on P_2. Both platforms (P_1 and P_2) are bound to the target cloud vendor services (S_1 and S_2)."

2.7 Summary of chapter

In this chapter, we proposed a new *Dynamic Cloud Computing Service-Oriented Architecture* (*DCCSOA*). The proposed architecture addresses the most existing cloud computing issues, such as data and applications migration between different clouds, transfer to cloud, or return back to in-house *IT*, data and applications lock-in issues, and a lack of standardization and customization. *DCCSOA* provides a dynamic and customizable service layer (*DTSL*). The *DTSL* provides simplicity these issues by defining a layer, *template*, with the same feature in *DTSL*. A *template* is divided into *front-end* (*FTaaS*) and *back-end* (*BTaaS*) layers. The defined *templates* can be customized by a cloud vendor for different groups of users. *DCCSOA* also allows different cloud vendors to provide the similar cloud services through a *template* that meets a standardization between different cloud computing systems. We discussed how the proposed architecture supports existing and future services by using the *DTSL* at *BTaaS* that can be configured to a specific cloud services. We evaluated the proposed method based on *SOA* evaluation. The result shows that the proposed architecture, *DCCSOA*, provides several advantages over existing cloud architectures and platforms, such as minimal modifications for providing standardization and customization. Chapter 7 describes a new template for electronic healthcare systems along with its implementation for maintaining data privacy.

Table 2.2. Evaluation parameters for each topic

9. Service Orchestration with using BPEL	8. Legacy Systems Integration	7. Use of a Registry of Services	6. XML Optimization	5. HTTPS or Message-Level Security	4. Exception Handling and Fault Recovery	3. Fine-grained services (versus coarse-grained services)	2.Asynchronous (versus Synchronous Services)	1.Target Platform	Topic
☺ Modifiability	Database Access	Service Management	The size of files	Embedding security	failure or resource exhaustion of an underlying component	Performance	Modification	Platform familiarity	1
Interoperability	Database Synchronization	Interface description	The cost	https encrypts	A formatting violation	Message rate	Blocking time	Run-time environment for Service User	2
Performance.	Direct API call	Security	Appropriate Parsing	Interoperability	Application business logic defects	☺ Flexibility	☺ Insert an ESB or other brokering software	Run-time environment for Service Provider	3
Cost	Web services wrappers	☺ Dynamic Service support	Validation	Service Access Authorization	Business rule failures	☺ Client centric	Overhead of receiving call responses	Development Environment	4
Reliability	ESB with adapters	History and Versioning	Compression	Interoperability of Security Standards	Intermittent failures	Overall cost	☺ Background Processing of service request	Network Infrastructure	5
		Life Cycle of services		Security in legacy components		Testability	☺ Scalability	Platforms Used by External Services	6
		Testing and quality		Identity management policy			☺ Independent operation		7
		Overhead costs		business logic Security			☺ Complex error		8

Chapter 3

Data Privacy Preservation in Cloud

As we described in Chapter 1 and Chapter 2, there are several challenges in cloud computing environment. One of them is "*Data Privacy*". The question is "*how we can protect users' data privacy?*" when users use cloud technology. This chapter focuses on developing a method to protect users' data privacy. We also consider mobile cloud computing. The next chapter describes how the proposed method in this chapter can be parallelized on GPU to provide better performance.

3.1 Introduction

Cloud computing paradigm refers to a set of virtual machines (VM) that provide computing and storage services through the Internet. Cloud computing uses virtualization technology to provide VMs on the top of distributed computing systems. Cloud computing provides several advantages over traditional in-house IT (Singhal et al. 2013), such as on-demand services, pay-per-use basis and elastic resources which have rapidly made cloud computing a popular technology in different fields, such as IT business, mobile computing systems and health systems (Adibi 2013, Rodrigues 2012). (Huang et al. 2011) defines Mobile Cloud Computing (MCC) is rooted from mobile computing and cloud computing paradigm that allows mobile users offload their data and process data on cloud computing. MCC provides a big data storage for mobile users with a lower cost and more ease of use. As of today, some popular MCC cloud vendors for storage service, such as Google and Dropbox, provide a free standard storage service with 2 GByte and 15 GByte capacity, respectively[33].

MCC provides an online massive storage for mobile users, but it aggravates the user data privacy issues because users have to trust third-parties (cloud vendors and their partners). For instance, in a recent study, (Landau 2014) reports some challenges toward privacy issues when users trust cloud vendors, and the vendor shares data with a third-

[33] Recorded on November 13, 2014 from https://drive.google.com and https://www.dropbox.com

party or other unauthorized users could have access to these data. In another study, (Kumar et al. 2010) suggest that not all MCC applications can save energy on mobile devices by offloading data. Finally, in an earlier study on cloud computing, (Ristenpart et al. 2009) show a VM can be a vulnerable component when users use cloud shared resources.

These examples of challenges in MCC show a mobile user requires an encryption method to protect their data privacy. One of encryption methods that can be considered as a secure method is Advanced Encryption Standard (AES) (Daemen 2002, Osvik 2010, Yoshikawa 2013). However, mobile devices have limited resources, such as limited power energy, low speed CPU and small capacity of RAM, and it is impossible to use AES encryption method for each file when offload/download is required for each file. Another solution for this challenge is light-weight security methods that provide a balance between maintaining energy efficiency and security. A light-weight security method is based on simple operations, such as permutation, rather than using expensive operations, such as secret key or public-key encryptions,

In this chapter, we propose a light-weight encryption method for mobile devices, such as smart phones, that uses permutation operation to protect data privacy. We investigate one of the popular image file formats as a case study, JPEG file formats because it is most world's widely used by smart phones for capturing pictures. We show that the proposed light-weight method provides a secure data privacy model based on chaos systems for JPEG file format in mobile devices. In addition, we evaluate the performance of the proposed method against other encryption methods, such as AES, and encryption on JPEG encoders, such as pixel and color encryptions. We investigate several attack scenarios against the proposed method.

The rest of the chapter is organized as follows: in Section 3.2, we briefly review background materials for this study. In Section 3.3, we present the proposed method and its requirements. In Section 3.4, we implement the proposed method and present its experimental results. We also present a statistical security model of the proposed method to show the level of the security. In Section 3.5, we investigate different scenario attacks against the proposed method. In Section 3.6, we review a comparison between the proposed method and existing methods. Finally, in Section 3.7, we conclude the proposed method which is described in this chapter.

3.2 Background

In the proposed method, we use JPEG file format as a case study. In this section, we briefly review the background of JPEG file format and its encoder/decoder requirements.

Each JPEG file has a header file that defines metadata, such as the canvas of a JPEG file with a specific dimension (width and height), resolution, camera information, GPS information, and compression information.

Each JPEG file (including the header and the content) consists of several segments and each segments beginning with a marker. A raw data of a JPEG file includes several markers (ITU 1993). For example, each JPEG file begins with "0xFF0xD8" marker that represents this binary is a type to image and it allows an image viewer application to decode the binary file and show the image. Each marker begins with '0xFF'. Table 3.1 describes some important markers.

We use these markers in our proposed method to retrieve different segments of a JPEG files.

A JPEG image encoder uses a lossy form of compression which is based on the discrete cosine transform (DCT) method (ITU 1993) to compress an image. A sequence byte of raw JPEG image file contains a multiple chunk of Minimum Codded Unit (MCU) as described in (ITU 1993). Each MCU block stores 4*4 pixels of an image.

Table 3.1. JPEG file format

Short Name	The description of marker	Bytes
SOI	Start of Image	0xFF, 0xD8
SOF0	Start of Frame (Baseline DCT)	0xFF, 0xC0
SOF1	Extended sequential DCT	0xFF, 0xC1
SOF2	Start of Frame (Progressive DCT)	0xFF, 0xC2
DHT	Define Huffman Table(s)	0xFF, 0xC4
DQT	Define Quantization Table(s)	0xFF, 0xDB
DRI	Define Restart Interval	0xFF, 0xDD
SOS	Start of Scan	0xFF, 0xDA
EOI	End of Image	0xFF, 0xD9

3.3 The Proposed Method

In this section, we present the proposed method to protect the privacy of data on MCC. As described in Section 3.1, since mobile devices have limited resources, we have to provide a method with minimum overhead to protect data privacy that runs on a mobile device to store and retrieve image files on MCC. In this chapter, we consider JPEG files as a case study because this format of photography by smart phones is popular at this time.

We assume the following requirements for the proposed method:

- The privacy model must satisfy a balance between computation overheads and maintaining the security.
- Unlike default MCC offloading methods that submit original files to MCC for encryption, the proposed method can be ran on mobile devices to provide data privacy, and then, the protected result will be stored on MCC.

The proposed method splits files into multiple files and uses a pseudo-random permutation to scramble chunks in each split file. The proposed method reads a JPEG file as a binary file rather than using a JPEG encoder/decoder to protect each pixel or the color of each pixel.

In this method, we have two phases to split files into multiple files and to recombine the files as follows:

- *Disassembling of an image* that splits an image file into multiple binary files, divide the original file into: (i) one file that contains the header of the original file, and (ii) multiple files that contain the content of the original file. The content of each split file consists of multiple chunks of original file. Chunks distribute through multiple files based on a *Pattern*, chunks in each file randomly scramble by using the chaos system. The output of this phase (split files) will be stored in MCC(s).
- *Assembly of split files* that recombine all split files to reorganize the original file. In this phase the following steps will be proceed: (i) read all scramble files from MCC(s); (ii) using the chaos system random arrays (which are used at the first phase) to reorder the chunks in each split file; and (iii) use the *Pattern* to reorganize the original files.

3.3.1 Disassembly Phase

We assume the proposed method requires to disassemble a series of images at the same time. The method divides each original file ($File_i$) into: (i) the header of the original file ($Header_i$), where i is the sequence number of original file in the file series that requires to be disassembled; (ii) the content of file that is divided into several chunks ($Chunk_{i,j}$) where j is the sequence number of the chunks of the original content file.

We divide the original JPEG file into header and the content because the header of the original file carries some important privacy information. This division also provides a complex method for recombining split files because it removes important JPEG markers from the original image file.

A split file is defined as follows:

$$File_i = Header_i + (\sum_{j=1}^{j=cmax} Chunk_{i,j}) \quad (1.)$$

where $cmax$ represents the maximum number of chunks in $File_i$ and is defined as follows:

$$cmax = \left\lceil \frac{Size_i}{Buffer} - HSize_i \right\rceil \quad (2.)$$

where $Size_i$represents the size of $File_i$ (Byte), $Buffer$ represents the size of chunks (Byte), and $HSize_i$ represent the size of the header of the original $File_i$ (Byte).

Figure 3.1, illustrates a general view of the proposed method that allows a mobile device to split an original JPEG file into header file and three multiple files. The split files are submitted to two MCCs. A user can configure his/her application to set: (i) the number of split files, (ii) the size of chunks, and (iii) the cloud user account(s) information to upload files on MCC(s).

We use the JPEG markers to split the header of the original file from the content and to find important JPEG markers. If someone has an access to split files to assemble

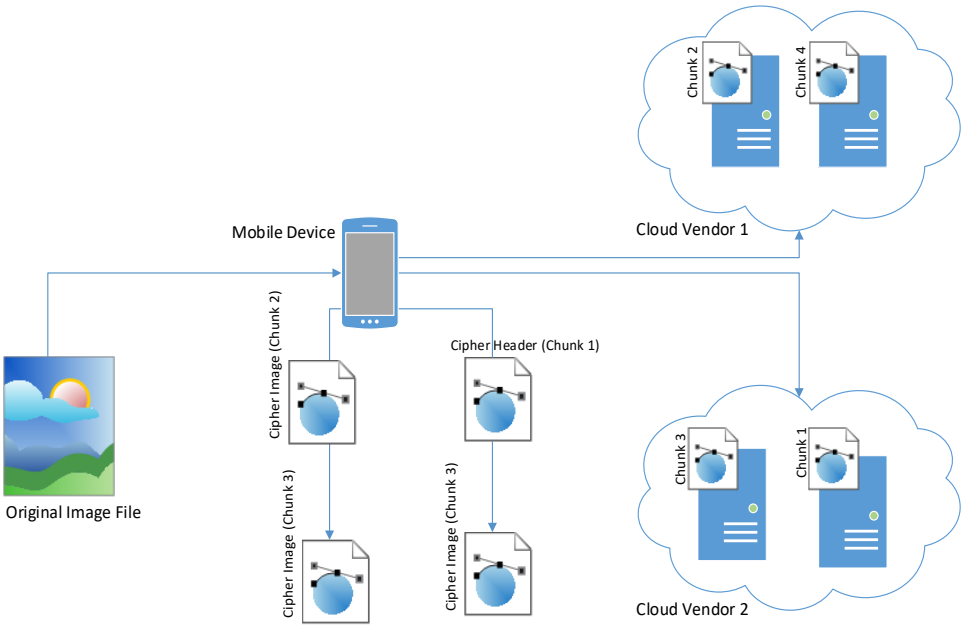


Figure 3.1. A general view of the proposed Method

different files without accessing to the header, or if the person creates a new header for a JPEG file, still he/she cannot simply retrieve the file. For example, Figure 3.2 shows two pictures (a) and (b), with the same size and the same resolution that took by a smart phone. The last right frame (c) shows the result of assembling the header of the first file and the content of the second file. As shown in this Figure 3.2, only the size and the resolution can be retrieved and still the assembler cannot retrieve the content of the image. To protect an image from a person who attempt to assemble a part of image by assembling files, first, we use a *pattern* to distribute chunks of the image to different files. The *pattern* can be defined as an input by a user to indicate, how to distribute a sequence of bytes in a split file. Second, we use chaos theory to randomly distribute each chunk in each split file.

(a) Source File #1 (b) Source File #2 (c) Header of file #1 with the content of file #2

Figure 3.2. Split the header of files and substitute header of file #1 with file #2

The proposed method uses two steps to disassemble an original JPEG file to a number of chunks, and to scramble randomly each chunk in each split file as follows:

3.3.2 Pattern

The proposed method divides each file into $cmax$ chunks (binary codes) and it distributes chunks to multiple split files in different order as will be discussed in the next phase. In this phase, the method use *pattern* that aims to distribute chunks to multiple split files. A *pattern* can be defined as a key by a user or it can be selected randomly as a predefined method. A user can define different *patterns* to provide different strategy for distribution. For example, Figure 3.3 shows two different patterns for disassembling a JPEG file. In this figure, the original file includes a header, and nine chunks of content. In $Pattern_A$, the proposed method reads two consecutive chunks from the original file and stores each chunk in one of two files. The first chunk stores in $File_{1,2}$ and the second chunk stores in $File_{1,3}$. At the end of reading $Source\ File_1$, $File_{1,2}$ contains $\{B_1, B_3, B_5, B_7\}$ and $File_{1,3}$ contains $\{B_2, B_4, B_6, B_8\}$. In Figure 3.4, $Pattern_B$ shows another pattern to store chunks. In this pattern, each four consecutive chunks of original file store in two files. The first and

the forth chunks store in $File_{2,2}$; the second and the third chunks store in $File_{2,3}$. At the end of reading $Source\ File_2$, $File_{2,2}$ contains$\{B_1, B_4, B_5, B_8, B_9\}$ and $File_{2,3}$ contains $\{B_2, B_3, B_6, B_7\}$. If someone has an access to all contents, still he/she cannot assemble files because he/she needs an access to the pattern as a key to assemble split files.

3.3.3 Scrambling of the content

We use the pseudo-random permutation (PRP) (Chakraborty 2006) with chaos theory (Stojanovski 2001) to scramble chunks in each split file. PRP uses chaos system which is defined as follows:

$$P_{k+1} = \mu P_k(1 - P_K) \qquad (3.)$$

where $P \in \{0,1\}$ and μ is a parameter of this equation.

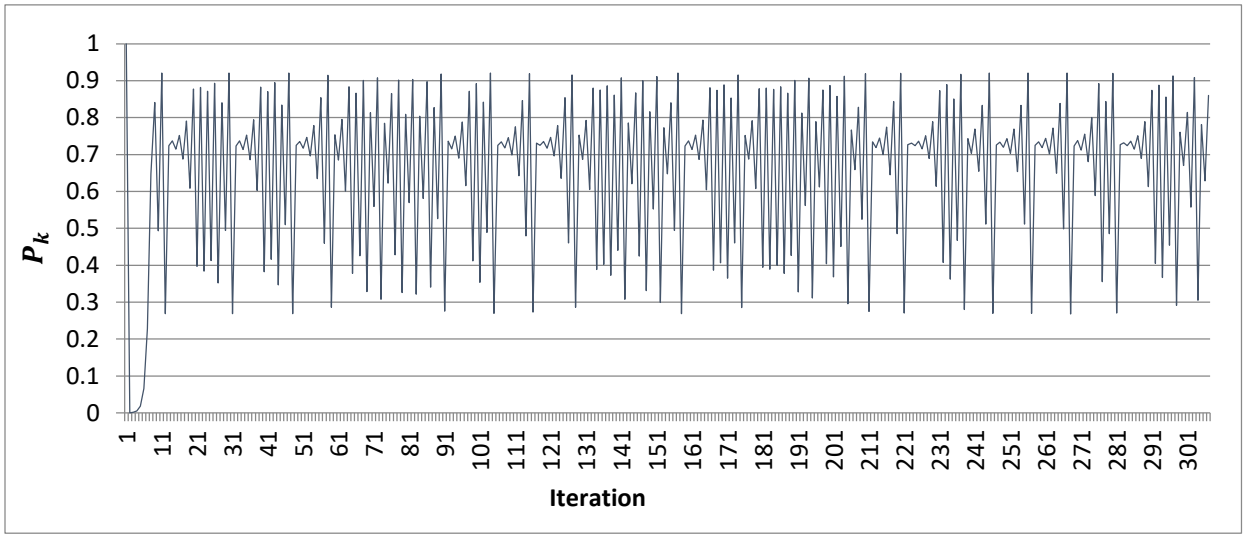


Figure 3.4. Chaos behavior of $\{P_k\}_{k=0}^{300}$

In the classic problem of the chaos system if selected μ is to be selected between $3.569945 \leq \mu \leq 4$, P can provide a complex chaos model. Figure 3.4, shows 300 iterations of a chaos behavior that describes in Equation (2) when $\mu = 3.684$.

The proposed method uses the following set to provide a non-convergent, non-periodic pseudo random numbers:

$$\{P_k\}_{k=0}^{\omega} \qquad (4.)$$

where $\omega = cmax$ by an initial value of $P_0 = 0.9999$

The proposed method uses the following equation to find the location of $Chunk_k$ in a split file:

$$Pos_k = P_k * cmax \tag{5.}$$

where Pos_k represents the position of $Chunk_k$ in each file.

Reading and writing a JPEG file as a binary adds more complexity to retrieve the image by unauthorized users. For example, let's assume that the original file has two consecutive bytes (i.e., 'FFD8' that indicates start of an image). First, we split these two consecutive bytes into two chunks (i.e., 'FF' and 'D8'). Second, distribute chunks in different location in a file (i.e., at 0 position and 2047 position); then a JPEG decoder cannot retrieve this file because the decoder cannot find JPEG makers. In this case, the

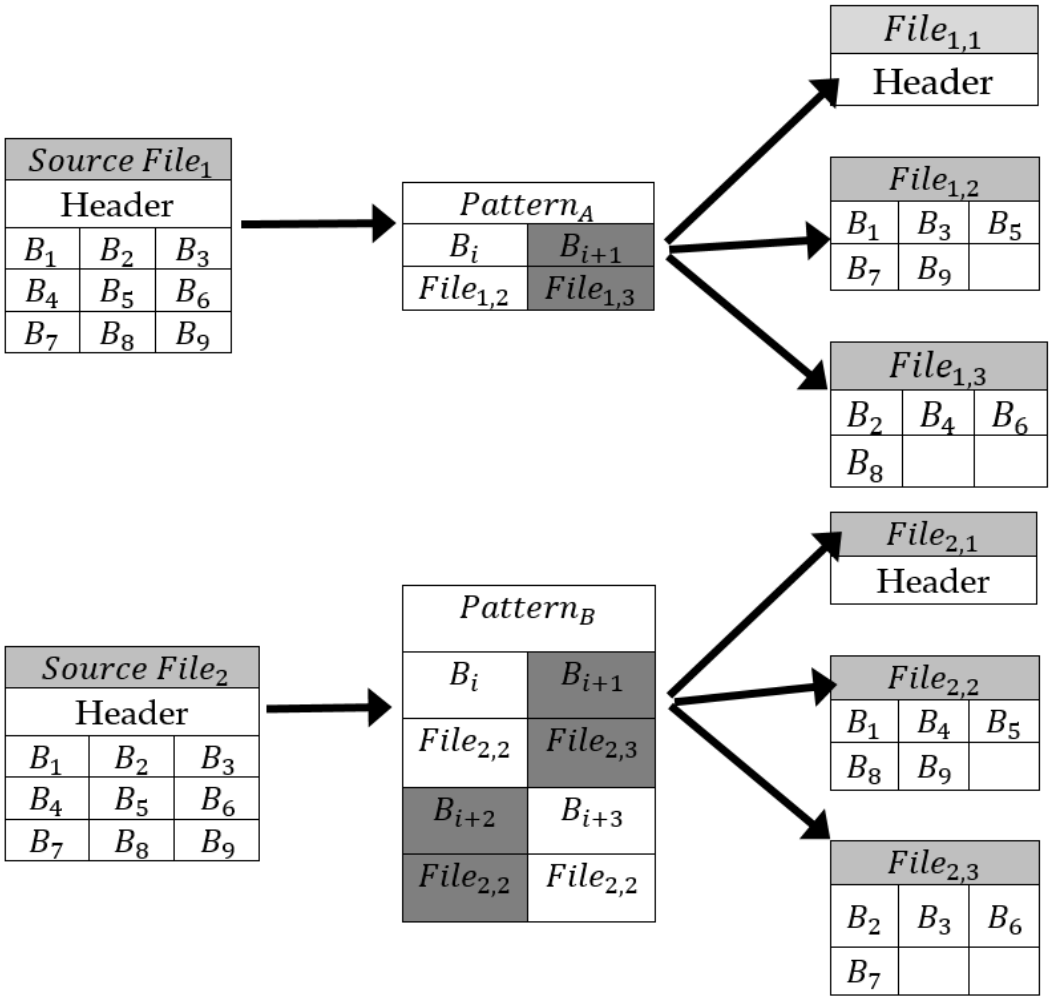


Figure 3.3. Two different patterns to store chunks in three files

JPEG decoder or an operating system cannot understand this binary file is a type of JPEG format.

The best option for selecting the buffer is: when Buffer $mod\ 2 = 1$ that splits two consecutive bytes into two chunks.

In the case of a collision between $Pos_k$and Pos_lin Equation 5, we develop a framework to find a new address. We use the following equations (6-11) to relocate a $Chunck$ from $Pos_k$to Pos_l in a file, which is called conflict remover. In these equations $\{Avail\}$ represents positions that are not used in $\{P_k\}$.

$$Pos_l = Avail_m \quad (6.)$$
$$If(k = \max) \Rightarrow \exists k \in \max[\{Avail_k\}]\ where\ m < k$$

$$Pos_l = Avail_n \quad (7.)$$
$$If(k = \min) \Rightarrow \exists n \in \min[\{Avail_n\}]\ where\ n > k$$

$$Pos_l = Avail_o \quad (8.)$$
$$If(k \neq min) \wedge \exists o \in \min[\{Avail_o\}] \text{ where } o < k$$
$$\Rightarrow \exists n \in \min[\{Avail_n\}]\ \ where\ o < k$$

$$Pos_l = Avail_p \quad (9.)$$
$$If(k \neq \min) \wedge \nexists p \in \min\left[\{Avail_p\}\right] where\ p < k$$
$$\Rightarrow \exists p \in \min\left[\{Avail_p\}\right]\ \ where\ p > k$$

$$Pos_l = Avail_q \quad (10.)$$
$$If(k \neq \max) \wedge \exists q \in \max\left[\{Avail_q\}\right] \text{ where } q > k$$
$$\Rightarrow \exists q \in \max\left[\{Avail_q\}\right] \text{ where } q > k$$

$$Pos_l = Avail_r \quad (11.)$$
$$If(k \neq \max) \wedge \nexists p \in \max[\{Avail_r\}]\ \ where\ r > k$$
$$\Rightarrow \exists r \in \max[\{Avail_r\}]\ \ where\ r < k$$

The procedure (Equations 6-11) extends the upper bound collisions and the lower bound collisions to upper and lower available addresses, respectively. If $Pos_{k-1} < Pos_k$ then, the procedure finds an upper level available position. If $Pos_{k-1} > Pos_k$ then, the procedure finds a lower level available position. Some exceptions are listed as follows: (i) k is the maximum address number: the procedure finds a maximum position of l where $k>l$; (ii) k is the minimum address number: the procedure finds a minimum position of l where $k<l$; (iii) there is no any available upper bound position: the procedure finds the maximum number of l where $k<l$; (iv) there is no any available lower bound position: the procedure finds the minimum number of l where $k>l$. Figure 3.5, shows the result of equations (6-11) when $\mu = 3.684$ and $P_0 = 0.9999$.

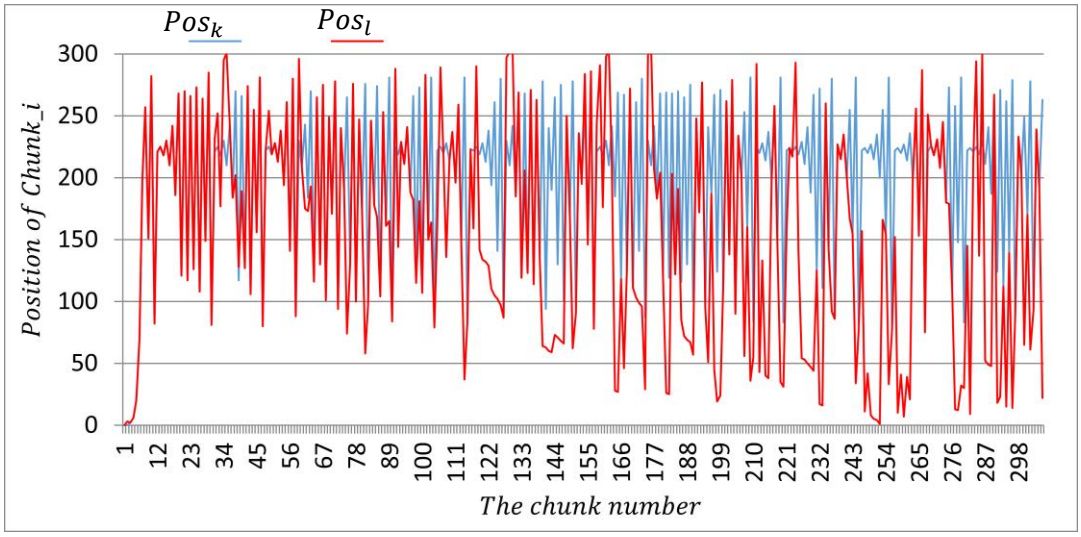


Figure 3.5. A comparison between Pos_k and its relocation (Pos_k)

3.3.4 Assembly a file

At the first step, the method reads each chunk from each split file based on selected pattern in Section 3.3.1, and then the proposed method uses the chaos model in Section 3.3.2 to relocate each chunk to the original location of JPEG file. At this phase the proposed method retrieves the address of each chunk by one the following procedure:

(i) use an array of PRP numbers which is used in assembling phase;

(ii) use μ and $P_0$to build the array of PRP numbers. The array can be defined by finding each P_k and its relocation (P_l), if there is a collision between P_k and P_l.

3.4 Evaluation of the proposed method

One of the approaches to evaluate the proposed method is implementation that represents experimental results, such as the response time in a load and performance

testing. Another approach is statistical model that describes a deviation between the original and the output. This section presents these two approaches to evaluate the proposed method.

3.4.1 Implementation

We use the proposed method to disassemble and store 21 JPEG files with different sizes that were taken from a smart phone. We compare the result of assembly phase against encryption methods and disassembly phase against decryption method.

3.4.2 Experimental Setup

In this experiment, we used 21 pictures with different qualities that were taken by a Samsung Galaxy III smart phone. The total size of our dataset was 24.9 Mbyte. We select different size of files to benchmark the proposed method with different input rates. Selected pictures' sizes are varying from 21 Kbyte to 2.86Mbyte with different dimensions (180*320 to 2448*3264 with 72dpi). In this experiment, the proposed method uses a buffer with the size of 4096 Byte. We use Rijndael cryptography with a key of 32 bit (minimum regular key size) and a buffer size of 4096 Byte, Initialization Vector (IV) of 16 Byte. The proposed method is implemented by Visual Studio 2013 in C#.Net programming language, with .Net framework 4.5. We used a well-known JPEG library, LibJPEG.Net package which is developed by BitMiracle, to evaluate the encryption on JPEG encoder in C#.Net programming language.

We assume our application have an access to a pre-defined $\{P_i\}$ to avoid the computation overhead.

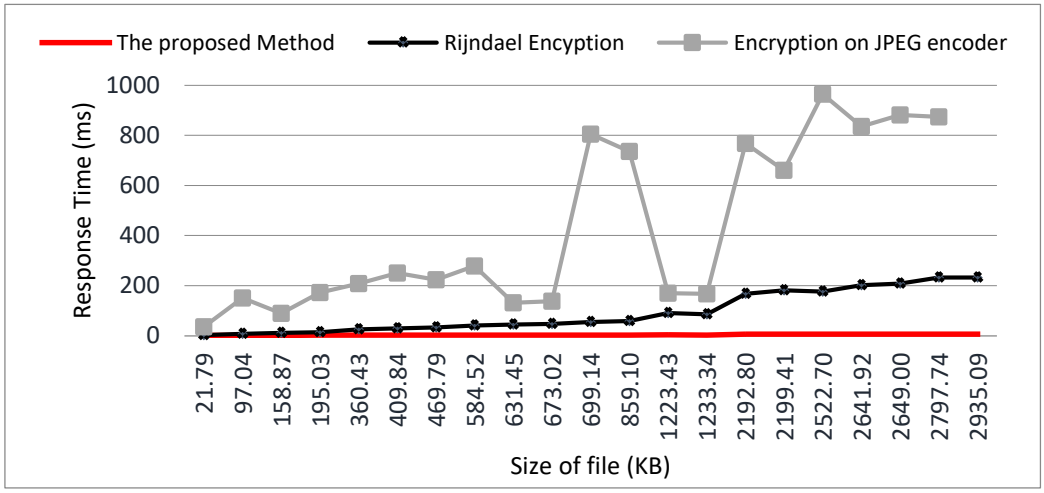
3.4.3 The result of the experiment

We implemented the proposed method with *pattern A* (according to the Figure 3.3) and AES encryption method to compare the response times. As shown in Figure 3.6, the size of files increases, the proposed method provides a flat response time but the response time for AES encryption increases linearly. As shown in this figure, encryption on JPEG encoder has more computation overhead over two other methods because it is required to read each pixel by a JPEG decoder, encrypt each pixel and use the JPEG encoder to write the result.

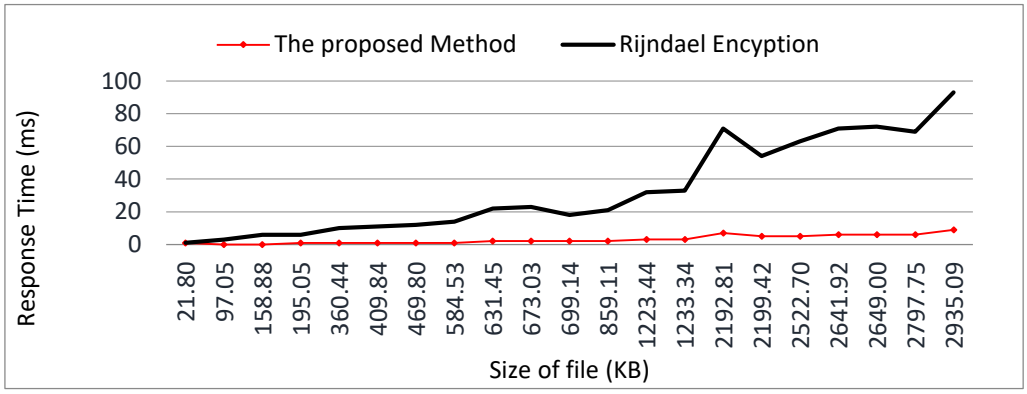
3.4.4 Statistical Model

Another approach to evaluate the proposed method is statistical model that presents a deviation of each chunk position between the position in original file and the position in scramble file.

To evaluate the proposed method, we select three different values of μ and we select the same other initial values as follows: $P_0 = .9999$, the maximum size of input file is 3057 Kbyte (the maximum size of our dataset images is described in the Section 3.4.2) and the size of each chunk (buffer size) is 10 Kbyte. Figure 3.7 shows a deviation of the original chunk position and the same chunk position in scrambled file with different parameters of (μ) values.



A. a comparison of disassembling files with pattern A and encryption methods for 21 JPEG pictures



B. a comparison of assembling files with pattern A and Decryption method for 21 JPEG pictures

Figure 3.6. Experimental results

In Figure 3.7, X-axis represents eight selected $Pos_k$and Pos_{k+1} and the Y-axis represents the deviation of position of Pos_k and Pos_{k+1} in scrambled file. As shown in this diagram, each two consecutive positions in a scramble file has different deviations.

The different values of μ provide different models of file scrambling. As shown in this figure, the deviation of each curve are different for different initial values of μ.

We also compare the position of each chunk in the original file and in the scrambled file. We use the configuration of Figure 3.7 and we assume the original file is scrambled in one file that includes the header and the content of the original file. Figure 3.8, shows the deviation of a chunk position from the original file to the new position in the scrambled file. As shown in this figure, each chunk is relocated to different locations in scrambled file and the value of each chunk is vary chunk to chunk, randomly.

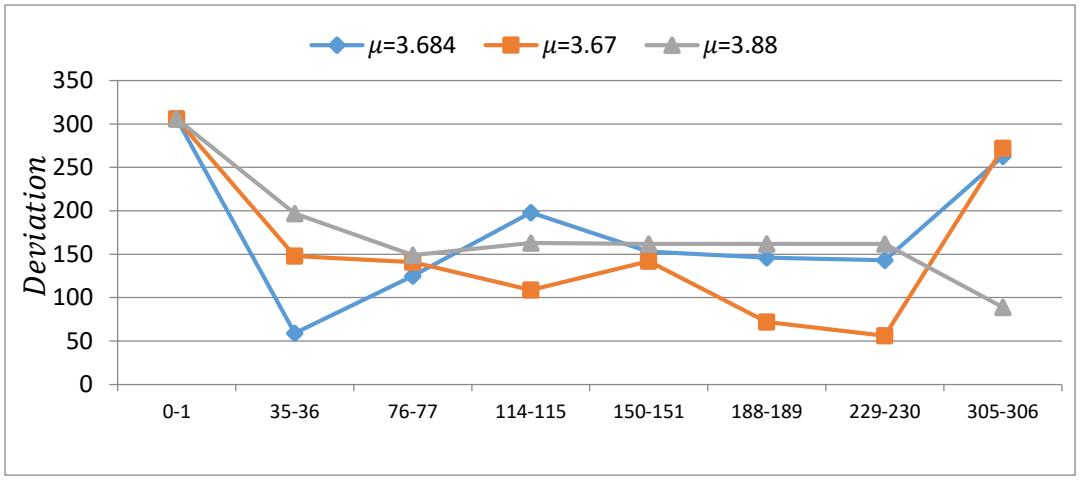


Figure 3.7. A deviation of chunks positions in scramble file with different parameter values

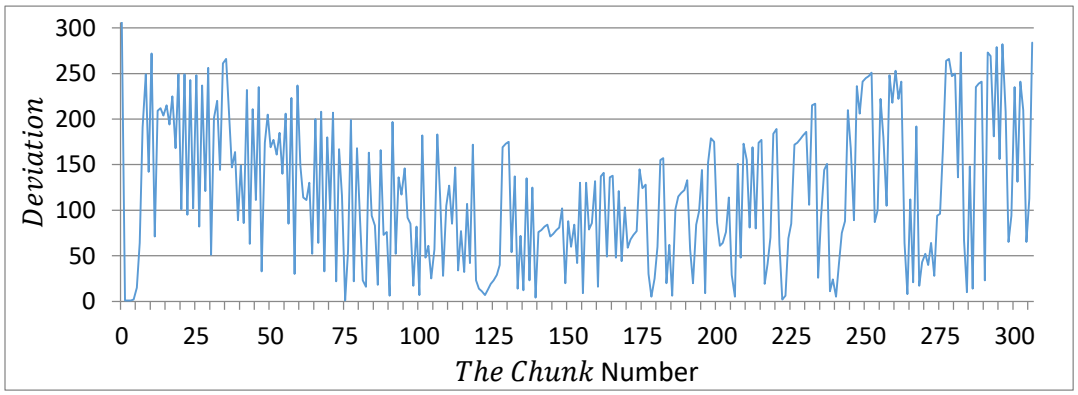


Figure 3.8. A statistical deviation of position in original file and scramble files

3.5 Security Attack Scenarios

In this section, we present different security attack scenarios that an attacker can implement against the proposed method.

An attacker requires assembling all split files and all chunks in each split file to retrieve an image. The proposed method provides a scrambled binary file that provides an obstacle for JPEG encoders to retrieve images because a JPEG encoder requires specific markers which are scrambled through split files.

We assume that the attacker wants to retrieve a JPEG file with a 2 MByte size and the attacker uses an Intel CPU i7 4770K with 127273 MIPS at 3.9 GHZ. The following scenarios can be implemented against the proposed method:

3.5.1 Scenario 1

Assumptions: the attacker who has access to all split files but dos not know $\{P_i\}$ and the size of each chunk

In this scenario, since the attacker does not have information of $\{P_i\}$, the attacker must run a brute-force attack to assemble split files and reorganize the scrambled chunks in split files. It requires minimum $O((n! - 1) - \partial)$, where ∂ is the number of similar bytes in all files and n is the size of file (byte). In this case, the attacker needs to try$2.229077716E + 9381$ permutation combinations to reconstruct the image ($\partial \approx 0$).

Using this scenario with this CPU configuration to retrieve an image is impossible and it is required 2.027100061111045012201E+9371 years for processing the computations.

3.5.2 Scenario 2

Assumptions: The attacker has access to all split files, knows the size of each chunks (10Kbyte) but does not know $\{P_i\}$.

In this scenario, the attacker needs to run a brute-force attack but the computation on size of the scrambled files (3070 Kbyte) is divided to the size of each chunk (10Kbyte). In this case, the attacker needs to try a minimum of $307! - 1$ permutation combinations to reconstruct scrambled file, needs an impossible computation (6.67732188350771659511 6E+621 years) to compute all permutation combinations.

3.5.3 Scenario 3

Assumptions: The attacker has access to all split files, knows the size of chunks (10Kbyte) but does not know the method is based on chaos system.

In this scenario, the attacker needs to use a brute-force attack against the proposed method that requires a computation with $O(\frac{n}{10}! - 1) - \partial)$, where ∂ is the number of

similar bytes in all files and n is the size of the original file (3070 Kbyte). In this case, the attacker needs $O(307!)$ computation.

3.5.4 Scenario 4

> *Assumptions*: The attacker has access to one file of the multiple split files.

In this scenario, since each two consecutives chunks stores in different clouds (i.e., using Pattern A in Figure 3.3), the attacker only could retrieve a part of an image by using a brute- force attack. We can estimate the probability of finding the size of each chunks as follows:

$$T = \sum_{j=1}^{j=size} \frac{Size!}{j} \tag{12.}$$

where *size* represents the size of file (byte).

The worst case of this scenario is described as follows:

$$T = \sum_{j=1}^{j=cmax} \frac{Size!}{j} \tag{13.}$$

The attacker needs to try all $T!$ permutation combination to reconstruct a partial of an image.

3.6 Related Works

Several studies (Lian et al. 2004, Podesser et al. 2002, Choo et al. 2007, Ye et al. 2010, Ra et al. 2013) have been conducted to image encryption. Unlike these studies that address encryption methods based on JPEG encoders, our proposed method provides a light-weight data privacy based on binary file. As described in Section 3.3, using JPEG encoders has computation overhead for mobile devices, such as smart phones. Our proposed method uses the binary file rather than using encrypt method on image pixels or color of a pixel.

(Podesser et al. 2002) proposed a selective encryption method for mobile devices to cipher a partial of an image. However, in this method, still a partial of image is visible to everyone. (Choo et al. 2007) proposed a light-weight method for real-time multimedia transmission. Although this method provides an efficient performance over AES encryption, the method still needs heavy computation on a smart phone to cipher an average 3 Mbyte JPEG image file in a real-time (see Section 3.5 for the comparison).

Unlike existing studies, our proposed method provides three steps to reconstruct a JPEG image file as follows:

i) ***Recognizing the type of the scrambled files*:** It is difficult to understand the type of file because the header of file (includes JPEG marker) splits from the content of file and all the chunks are scrambled in each file.

ii) ***Finding and assembling the scrambled files:*** Since the split files distributed through two or multiple clouds, it is impossible to reconstruct an image file completely.

iii) ***Reconstructing the original file from split scrambled files*:** Reconstructing split scrambled files requires heavy computations to retrieve partial or full image.

The proposed method hides JPEG markers from image decoders that does not allow JPEG encoders to retrieve the metadata of a scramble image. For example, Figure 3.9 shows: (a) an original image; (b) a scrambled JPEG file based on the proposed method (if we save the file with a JPG extension); (c) a cipher image based on JPEG encoder; (d) a cipher image based on AES. As shown in this figure, the proposed method and AES cannot retrieve the information of an image and the size of an image. However, metadata of an image can be retrieved for a cipher image based on a JPEG encoder. As described in Section 3.4, our proposed method provides a better performance over existing methods. The proposed method can be implemented for different applications in cloud computing systems such as (Bahrami et al. 2013) to collect information by a web crawler and maintain the privacy of the information in a cloud computing system. The method can be applied to eHealth systems (Rodrigues et al. 2013) to maintain data privacy.

3.7 Summary of chapter

In this chapter, we proposed a new data privacy method to store JPEG files on multi cloud computing systems. Since the method uses less complexity, we have shown, the implemented method provided a cost effective solution for mobile devices that do not have enough energy for resources, such as CPU and RAM. The proposed method splits each file to multiple chunks, distribute each chunk to multiple split files, and scramble chunks based on chaos system. The proposed method provides low computation overheads and it can efficiently run on a smart phone. It restricts unauthorized users including cloud vendors and their partners to reconstruct a JPEG image file. We compared our proposed method against other encryption methods to demonstrate its performance superiority over existing methods. Furthermore, we investigated some important security attack scenarios against the proposed method to evaluate the level of security.

(a) (b) (c) (d)

Figure 3.9. (a) the original image; (b) a scrambled image based on the proposed method; (c) a cipher image based on JPEG encoder; (d) cipher image based on AES encryption

3.8 Acknowledgement

The implementation work of the application was supported by Microsoft Windows Azure through Windows Azure Educator Award.

Chapter 4

Parallel DPM for Mobile Cloud Users

The previous chapter describes DPM for mobile cloud users. In this chapter we extend the method by parallelizing it on GPU. Next chapters present different use case scenarios for deploying DPM on the top of DCCSOA.

4.1 Introduction

Cloud Computing and parallel computing paradigms introduce several advantages for processing heavy computation methods. Our study in this chapter is based on these two paradigms which are described in the following.

4.1.1 Cloud Computing

Cloud computing is an emerging technology that combines of distributed systems, and virtualization technology to offer new computing concepts. A cloud computing system shares all available resources with multiple users, and each user is billed base on pay-per-use model or similar payment model. The cloud enables users to increase or decrease their resources on-the-fly Cloud computing has been used in different disciplines, such as mobile computing, robotics when data is outsourced on the cloud. *Mobile Cloud Computing (MCC)* paradigm allows mobile users to outsource their data and applications to the cloud and *MCC* enables mobile users to run complex application on server-side. However, there is a tradeoff in *MCC* between processing faster of data and saving power resources, which is critical for mobile devices. In particular, using cloud computing on mobile devices is a challenge because not all applications are able to efficiently outsource data to the cloud as described in Chapter 3. For instance, if a user outsources data to the cloud by using an application but the application drains the battery due to periodically download/upload files, the application is not efficient to be used. The trade-off between power resource and computation speed are critical in *MCC*. Data privacy is another parameter when users outsource their data to *MCC*. Data security and network security methods can be used in order to protect user privacy in cloud

computing systems. However, using complex security methods on mobile devices raise resource limitation challenge. Therefore, not all complex security methods are able to protect user data privacy by providing a balance between resource power and computation speed.

4.1.2 Parallel Computing

Parallel computing has been popular since computing machine introduced. Parallel computing allows complex algorithms to be run in parallel in order to increase the computation speed.

Recently, the implementation of parallel computing algorithms has been transferring from massive server machines to small personal computers when open-forums and corporations have introduced new platforms that allows users to have efficient parallel computing on personal computers. For instance, *CUDA* platform uses *Graphics Processing Unit (GPU)* and it was introduced by *NVIDIA*. This platform enables a user or even a mobile user to process a procedure in parallel that needs intensive computation. The *GPU-based* computing paradigm of parallel computing allows a device to use one or multiple *GPUs* to perform heavy computations where each *GPU* consist of thousands of small and low speed cores. Regularly, each submitted task to a core is a repeated computing process that each instruction does not need heavy computation. *GPU-based* computation improves the performance of processing of complex methods by parallelizing small tasks of a complex method on each *GPU* core. In previous chapter, we developed a light-weight data privacy method (*DPM*) that uses a chaos system (Han et al. 2003) based on *Pseudo Random Permutation* (*PRP*) to scramble data. *DPM* uses pre-generated arrays that contain random addresses of chunks of data. The *DPM* provides a superior performance over other existing data security methods that needs heavy computation, such as *Advanced Encryption Standard (AES)*. We also developed several experiments on both traditional implementation of AES and DPM in Chapter 3. This chapter focuses on parallelization of DPM on GPU.

4.2 Threat Model

In this section, we are describing the specific threats that our proposed technique shall protect the privacy against.

If a mobile cloud user wishes to outsource her photos to a cloud, such as Google Drive, Dropbox, but she would not like to share the original content with the cloud vendor, she may encrypt the content and submit the encrypted photos to the cloud. However, encrypting each file may drain the battery power in short period time. Another option, she might use a *PRP* method to scramble the content of all photos based on bits. Then, submit the scramble data of the photos to the cloud. In this case, the *PRP* generator

should be secure and use a lengthy chunk of data (e.g., using a lengthy size of bits) to permute the original content and then submit it to the cloud. As another example, if a user submits a text file or a database file, she might use *PRP* to scramble the original content in order to submit a confusion content to the cloud. In the case of a database, a query also can be performed on a scrambled data without reconstructing the original content. The detail of implementation of DPM for cloud-based datasets is described in Chapter 8. Generally, our proposed method shall use available resources on a mobile device (e.g., cell phone) to scramble the original content. Our proposed method uses *GPU* on mobile device to parallelize the method in order to save device's power resources and process the method faster than device *CPU*. In addition, public analysis tools for data analysis from cloud vendors, or their third-party applications are not able to simply access the original user's photo, text file, or database. For instance, third-party application and bulk data analysis tools are not able to process users' data without reconstructing the original data from scrambled data. If users use a complex model of *PRP*, then reconstructing the original file would be more difficult for cloud vendors or their-third party partners.

In the previous chapters, we describe how a mobile device access to the shuffle addresses for different chunk sizes of an original file. However, in this chapter, we consider implementation of *DPM* on *GPU* to generate *PRP* numbers to permute the original content and outsource scrambled content to a cloud that does not allow bulk data analysis review user's content. The *GPU-based DPM* enables the process to be run on *GPU* core instead of *CPU* in order to improve *DPM* performance.

The rest of this chapter is organized as follows: the next section presents the motivation of the study and the major challenges to maintain data privacy while using cloud computing. Section 4.4 presents the related work. Section 4.5 presents the background of this study. Section 4.6 presents the proposed method on *GPU*. The experimental setup and its results are described in Section 4.7. The security analysis of DPM presents in Section 4.8 which shows the security assumptions and the level of security for the proposed method. Finally, Section 4.9 concludes the study.

4.3 Motivation

The following two options can be considered for shuffling data:

i) using an online *PRP* generator to produce shuffle addresses;
ii) using a set of pre-generated arrays of *PRP* (offline mode) as described in Chapter 3.

The first option causes an issue on a mobile device because the computation time of generating *PRP* is expensive on mobile devices. The second option is preferred because it removes additional costs for generating *PRP*. However, it uses mobile

device storage that could cause indirect issue. In this chapter, we consider the first option but we generate arrays of *PRP*s on-the-fly and the process distributes to multiple cores of a *GPU* in order to reduce computation time.

The important challenge which is our focus is data privacy for mobile users because when a user outsources data to the cloud, data privacy can be violated by the cloud vendor, the vendor's partners, hackers, malicious entities or even by other cloud users.

4.4 Related Work

To the best of our knowledge, no research has been published in the area of implementation of *PRP* on *GPU* because its nature of sequential when it stands against parallelism concept. However, some studies have been published for implementation of other random number generators on *GPU* or *FPGA*. For instance, (Thomas et al. 2009) compare the performance of three types of random number generators on *CPU*, *GPU* and *FPGA*. In this study, authors use an appropriate algorithm, such as the uniform, Gaussian, and exponential distribution for each hardware platform in order to have efficient power peaks and computations. This study shows that the performance of the different random-number generators relies on their platform. In this chapter, we consider *CUDA* platform on *GPU* in order to optimize the performance and power consumption which is not investigated in (Thomas et al. 2009). In another study, (Tsoi et al. 2003) implemented two different random number generators for embedded cryptographic applications on *FPGA*. The first is a true random number generator (*TRNG*) which is described by (Killmann et al. 2001) and it is based on oscillator phase noise, and the second is a bit serial implementation of a *Blum Shub (BBS)* which is described in (Blum et al. 1986). The study shows that *TRNG* is recommended for low-frequency-clock processors. Since *GPU* often consists of thousands of cores and with low speed and smaller than *CPU* cores, we consider this fact for designing small-scale generators in our study which is described in Section 4.5. This consideration makes *PRP* to be highly suited to the target platform. In the similar study by (Manssen et al. 2012), the authors evaluate different random number generators with different granularity. There are some studies on processing *AES* on *GPU* (Manavski et al 2007, Shao et al. 2010 and Li at al. 2012) or using the similar method for the security processing (Wang et al. 2009) on a *GPU*.

4.5 Background of the study

As described in Chapter 3, *DPM* splits an original file into several chunks. The method uses a pattern to split original file into multiple files when each file consists of random part of the original file. Then, DPM selects a set of bits (chunk) of each split file and finally it scrambles the content of each chunks by using a chaos system. The DPM

saves 72% battery power over AES encryption method because *DPM* can be run in *O(1)* time complexity for each chunks and it requires *O(n)* for n chunks.

Pseudorandom Permutation (*PRP*) is the key module of many methods, such as encryption and simulations as well as *DPM*. As previously described, *PRP* is defined as follows:

$$F\ is\ mapping\ \{0,1\}^n \times \{0,1\}^s \rightarrow \{0,1\}^n \tag{1}$$

$F\ is\ a\ PRP\ if$:

(i) $\forall K\ where\ \ K \in \{0,1\}^s$, F is a bijection of $\{0,1\}^n \rightarrow \{0,1\}^n$ (2)

$(ii)\ \forall K\ where\ \ K \in \{0,1\}^s\ \ F_K(x)$ is an efficient algorithm. (3)

$(iii)\ \ D: \Pr(D^{F_K}(1^n) = 1) - \Pr(D^{f_n}(1^n) = 1)| < \varepsilon(s)\ , where\ K \leftarrow \{0,1\}^s$ (4)

where $\Pr(.)$ is the probability of raising the input event.

As discussed in Equation (4), the *PRP* provides a uniform distribution between all generated elements of F. This property of *PRP* passes the important perfect secrecy parameter which was introduced on Shannon's theory (Shannon 1949) for encryption functions.

The main function for generating pseudo number is defined as follows:

$$F_{k+1} = \mu F_k(1 - F_K) \tag{5}$$

where $P \in \{0,1\}$ and μ is a parameter of this equation.

In the classic chaos system problem, if μ is selected between $3.569945 \leq \mu \leq 4$, and with an initial value of $F_0 = [0,1]$, F provides a complex chaos model. F uses a set ξ to provide a non-convergent, non-periodic pseudo random numbers (See Chapter 3 for detail):

$$\xi = \{P_k\}_{k=0}^{\omega} \tag{6}$$

where ω is maximum number of an original content.

The *DPM* splits the content of an original content to ω number of chunks. Then, it uses ξ to shuffle the content of chunks. We employed conflict-remover algorithm which

is described in Chapter 3 to provide a set of unique addresses for each chunks based on input parameter (μ) and selected pattern by a user.

4.6 The proposed method

The main challenges of *DPM* are:

(i) generating ξ (a set of addresses which is used to permute an original input chunk
(ii) Applying ξ to the original chunk in order to have permutated data.

Both processes need heavy computation and we can accelerate *DPM* by processing both on *GPU*. However, we face several challenges on both processes when we implement DPM in parallel. The following sub-sections explain these challenges as well as a possible solution for each.

4.6.1 Generating ξ

The original content which is an input to *DPM* comes from different sources and it depends on the type of application where *DPM* is employed. For instance, In Chapter 8, we employed *DPM* for a database system management system where the input is a set of database queries; in Chapter 3 employed *DPM* for protecting data privacy of *JPEG* files and each chunks is composed of one or multiple Minimum Coded Unit (MCU) blocks; and for healthcare electronic systems which is descried in Chapter 7 where data privacy plays a key role.

The nature of generating a set of ξ is a sequential process that stands against data parallelism. We consider $\mathcal{M}$ as a *2D-array* for generating *PRP* addresses in parallel that allows us to use each GPU core to generate different sets of ξ. Each GPU core is corresponding to partial part of ξ. We map the original input ($\mathcal{D}$) to a *2D-array* ($\mathcal{M}$) to maximize the usage of *GPU* cores. Then, we apply ξ to $\mathcal{M}$ in the next step. Figure 4.1 shows an example of mapping from the original input ($\mathcal{D}$) to a *2D-array* ($\mathcal{M}$). In this figure, $\mathcal{M} = \bigcup_{i=0}^{\delta} D^{\kappa}$ where δ is the maximum number of chunks for the original content/file and κ is the size of each chunk. ξ generates n set of ξs for m chunks.

Code 4.1 shows how each thread get same seed with different sequence numbers. RowCell and ColCell represent the number of rows and columns of $\mathcal{M}_{m,n}$, respectively. This configuration provides the best performance because each block of threads receives a unique initial seed and each block provides a unique set of ξ. During the initialization (*config* function) ColCell is considered as a parameter of application. Let's RowCell and ColCell be different size of $\mathcal{M}$. In our experiment (Section 4.7.1), we describe different configurations for the application in order to implement different size of $\mathcal{M}$.

Code 4.2 shows an implementation of generating of RowCell and ColCell of the *PRP* generator function in the *main()* function. In this code, *Line 1* allocates space for results on host. Space allocation for results on device is defined in *Line 2-4*. The ξ set is configured in *Line 4* and *PRP* generator is called in *Line 6*.

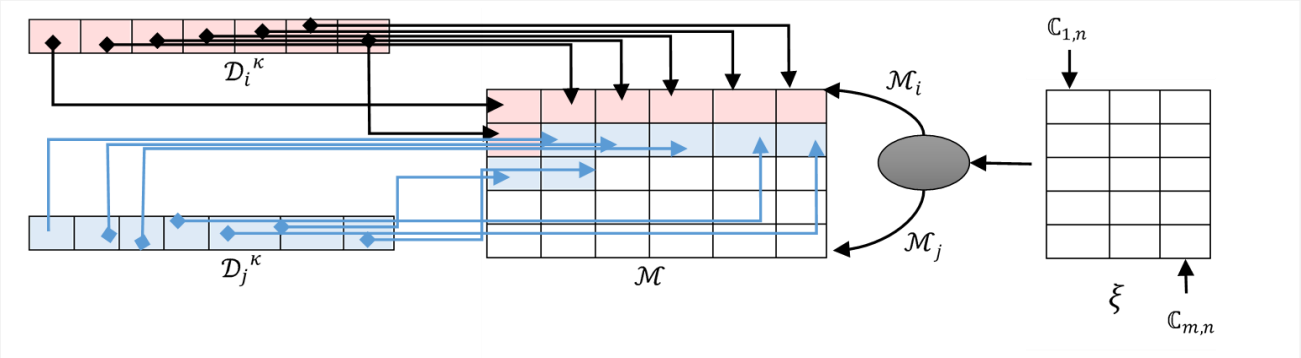


Figure 4.1. An example of mapping and exchanging process memory

```
// two parameters of the size of Sai (M: RowCell * ColCell)
const int RowCell = 128;
const int ColCell = 128;
__global__ void config (curandState *state)
{
  int id = threadIdx.x + blockIdx.x * ColCell;
  curand_init(1234, id, 0, &state[id]);
}
```

Code 4.1. The config function for initialization of a thread

```
1: hostResults = (unsigned int *)calloc(RowCell * ColCell, sizeof(int));
2: cudaMalloc((void **)&devResults, RowCell * ColCell *sizeof(unsigned int));
3: cudaMemset(devResults, 0, RowCell * ColCell *sizeof(unsigned int));
4: cudaMalloc((void **)&devStates, RowCell * ColCell * sizeof(curandState));
5: config << <RowCell, ColCell >> >(devStates);
6: DPM_PRP << <RowCell , ColCell >> >(devStates, Count, devResults);
```

Code 4.2. Main function for Calling the PRP generator

4.6.2 Appling ξ to $\mathcal{M}$

When random addresses have been generated in the previous section and it is stored in ξ, then different solutions are available for applying ξ to $\mathcal{M}$ as follows:

- The first option is transferring ξ to the host-memory and shuffle $\mathcal{M}$ on host-memory: This process needs $O(n)$ where n is the length of ξ. However, the computation on shuffling process of data on host-memory cannot be implemented in parallel on device.
- The second option, is transferring $\mathcal{M}$ to device memory, and shuffle $\mathcal{M}$ based on ξon device memory. Although this process still needs $O(n)$ computation time where n is the length of ξ. This process can be implemented in parallel on device that accelerate shuffling process of data.

The second option is the implementation of applying ξ to $\mathcal{M}$. In this case, there are two elements of $\mathcal{M}$ that needs to be exchanged when the *PRP* shows that an exchange is required between two κ-bits, $\mathcal{M}_i$where it is the original address and $\mathcal{M}_j$ where it is the destination address for the exchange. If we consider each element of $\mathcal{M}$ as a κ bit element, then minimum memory requirement for the implementation is κ. When *DPM* needs to exchange the content of two bits of $\mathcal{M}$ ($\mathcal{M}_i$ and $\mathcal{M}_j$) based on ξ, one of the following possibilities can raise: (i) $\mathcal{M}_i = \mathcal{M}_j$: In this case, if we assume that $\kappa = 1$ then no exchange is required because the content of both bits are the same and we can save the computation overhead of exchanging the content. If $\kappa > 1$ then the κ-bits of the content from $\mathcal{M}_i$ should be equal to all κ-bits of $\mathcal{M}_j$. (ii) $\mathcal{M}_i \neq \mathcal{M}_j$: The exchange is required. In this case $\mathcal{M}_i'$ and $\mathcal{M}_j'$ are the final value of the exchange process on $\mathcal{M}_i$ and $\mathcal{M}_j$ after exchange process, respectively. Table 4.1. summarizes these two different conditions.

Table 4.1. The summarize of exchange process

$\mathcal{M}_i$	$\mathcal{M}_j$	$\mathcal{M}_i'$	$\mathcal{M}_j'$	**Exchange is required**
0	0	0	0	N
0	1	1	0	Y
1	0	0	1	Y
1	1	1	1	N

We also optimize the implementation of shuffling process of the two different condition. The following rules are used in order to avoid additional host memory exchange computation:

$$\begin{aligned}\mathcal{M}_i' &= (\mathcal{M}_i \oplus \mathcal{M}_j) \oplus \mathcal{M}_i \\ &= (\mathcal{M}_i \oplus \mathcal{M}_i) \oplus \mathcal{M}_j \\ &= 0 \oplus \mathcal{M}_j \qquad (7)\end{aligned}$$

$$\begin{aligned}\mathcal{M}_j' &= (\mathcal{M}_i \oplus \mathcal{M}_j) \oplus \mathcal{M}_j \\ &= (\mathcal{M}_j \oplus \mathcal{M}_j) \oplus \mathcal{M}_i \\ &= 0 \oplus \mathcal{M}_i \qquad (8)\end{aligned}$$

Therefore, by using Equation 7 and 8, we do not need to run the exchange function in order to optimize the exchange function.

4.7 Evaluation of Proposed Method

4.7.1 Experimental setup

We implemented the proposed method on a PC with CPU i7-4790, x64 based processor, device memory of 12 GB and a *GeForce GT 720* GPU with 1,001,000 memory clock rate (kHz), 64 bits Memory Bus, 16.016000 GB/s Peak Memory Bandwidth (GB/s), and 16GB memory. We used *NVIDIA GPU Computing Toolkit v7.0* and we profiled (logging of *NVIDIA* functions) the executions of the proposed method with *NVIDIA profile* v7.0.

In order to record each step of execution, we used *NVIDIA* profiler in the implementation code where it requires to be started or stopped by the following code:

```
// start profiling of part of code
cudaProfilerStart();
// stop profiling of part of code
cudaProfilerStop();
```

By default, the first call of *CUDA API* starts the *profiler* (in this case *cudaGetDevice* initializes the profiler). An example of output of the profiler is shown in the List 4.1. In this example, the profiler shows that 10384 *API* calls (*CUDA API*) where 98.27% of time is taken in execution of *cudaMemcpy* which includes the following functions:

- *cudaMemcpyHostToHost*,
- *cudaMemcpyHostToDevice*,

- *cudaMemcpyDeviceToHost*, or *cudaMemcpyDeviceToDevice*.

```
==10384== NVPROF is profiling process 10384, command: calltester
==10384== Profiling application: calltester
==10384== Profiling result:
==10384== API calls:
```

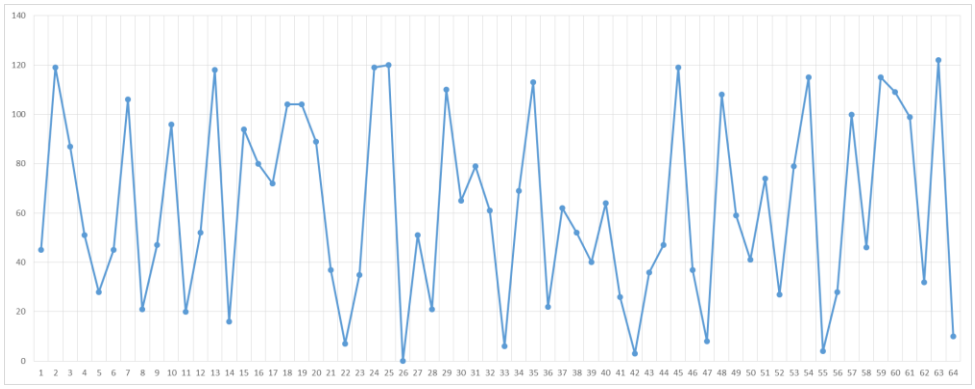
Time(%)	Time	Calls	Avg.	Min.	Max.	Name
98.27%	7.18901s	3	2.39s	1.39s	4.12s	cudaMemcpy
1.54%	112.41ms	2	56.20ms	435.92us	111.97ms	cudaMalloc
0.18%	13.240ms	385	34.38us	2.85us	11.74ms	cudaLaunch
0.00%	313.33us	83	3.77us	0ns	147.11us	cuDeviceGetAttribute

List 4.1. The result of "nvprof calltester"

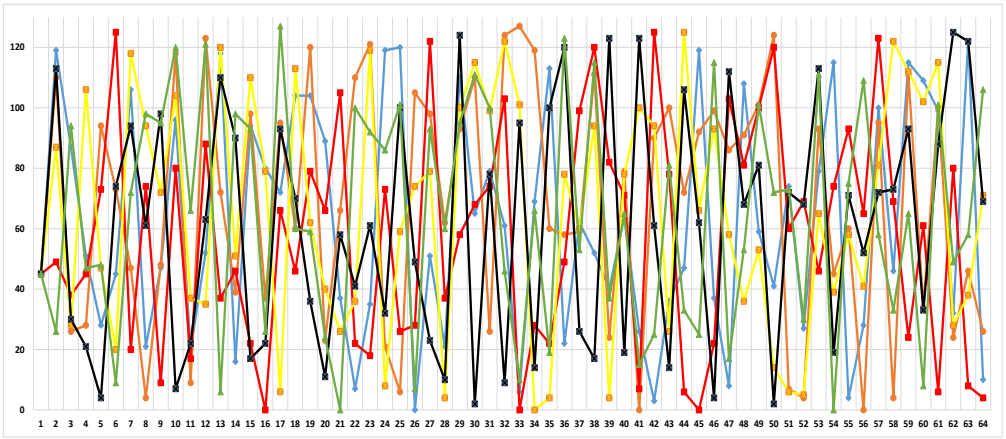
4.7.2 Experimental Results

First, we consider the generator of ξ sets, and the distribution values for each set over different iterations. These analysis allows us to evaluate the security of the method when the values have been generated with different *GPU*-cores. If there is a conflict between values of different sets, then it shows security issue in the proposed method. In the future, we plan to evaluate the generator with other statistical models, such as chi-square test of independence (McHugh et al. 2013).

Figure 4.2.a shows a set of ξ and Figure 4.2.b shows six sets of ξ with different initial values. In these figures, each point represents a value of ξ, *X-axis* represents the iteration number and *Y-axis* represents the value of *PRP* function. These figures illustrate the proposed method generates different sets of ξ when each one does not introduce any conflict to other values of other ξ sets. Figure 4.3 shows the distribution variances of ξ where each set of ξ values is illustrated in Figure 4.2. As shown in Figure 4.3, the values of each set is uniformly distributed through the range of ξ values. As clearly shown in Figure 4.2.b and 4.3.b there is not any spot pattern or a cluster pattern that helps an attacker to estimate values by knowing values (partial values) of one or different ξ sets.



(a) The result of a set of ξ

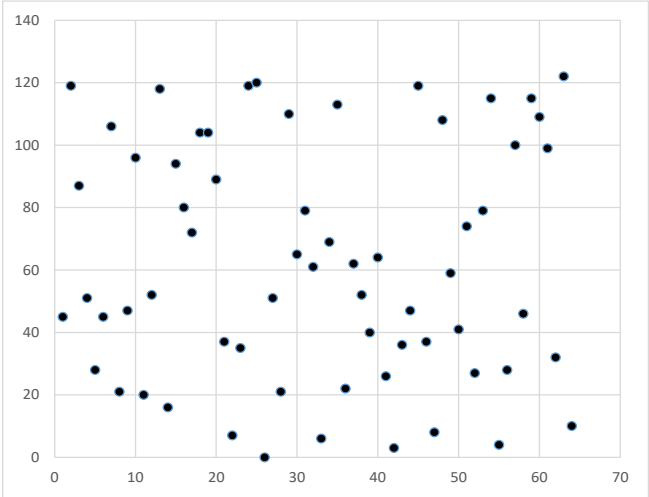


(b) The result of six sets of ξ with different initial values

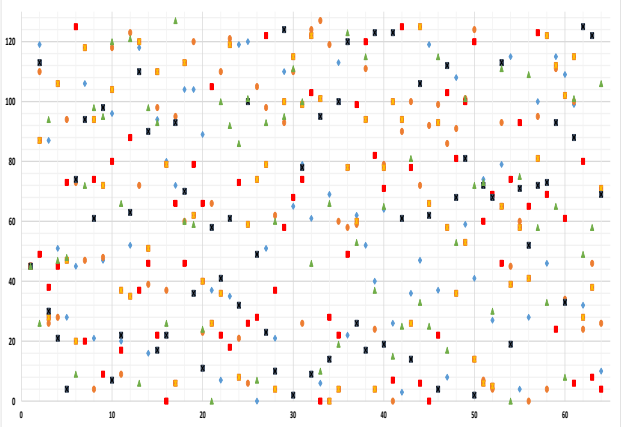
Figure 4.2. The evaluation results of one and six sets of ξ with different initial values

Overlap of different values from different sets of ξ is one of the security challenge for the proposed method. Each ξ provides a permutation model. An overlap between different value of ξ sets, allows an attacker to understand a permutation model by knowing one or multiple permutation models. Another challenge is finding a pattern between different subsets of ξ. As shown in Figure 4.3, different curves do not show similar patterns in any curves even for one set of ξ.

Second, we considered the performance of the proposed method by profiling the behavior of *DPM* for: (i) generating different sets of ξ on different size of a 2D-array; and (ii) permutation process. As shown in Figure 4.4, we assessed the performance of generating ξ values with the following sizes: 32*64, 64*64, 64*128, 128*128. X-axis represents the size of input. Y-axis in Figure 4.4.an illustrates the number of calls. Y-axis in Figures 4.4.b-d represent the computation time (*milliseconds*). Figure 4.4.e shows the behavior of *CudaMemcpy* where it is responsible for transferring data from CPU to *GPU*. As shown in this evaluation, the blue curve indicates that it is increased linearly. As a



(a) A set of ξ



(b) Six sets of ξ

Figure 4.3. Uniform distributions of ξ

result, the proposed method is capable to increase the size of input data with minimal transferring cost between *CPU* and *GPU*.

The evaluation results show that the 2D-array with the size of 128*128 provides better performance over other input sizes. However, the energy consumption is another parameter that can be assessed in order to provide an overall result for this performance evaluation. This evaluation also provides an overall view of the performance of the proposed method.

4.8 Security Analysis

In this section, we describe the security assumption and the level of security for the proposed method.

Let $SC(\mathcal{M})$ be the scramble function of *DPM* on n-core *GPU*. Perfect secrecy as described in Shannon theory is the probability of two different encrypted messages and in our study, $SC(\mathcal{M}_i)$ and $SC(\mathcal{M}_j)$, which is defined as follows:

$$\forall m_0, m_1 \in M \; |m_0| = |m_1| \; and \; c \in C \qquad (9)$$

$$(\Pr[SC(\xi_i, m_0) = c]) = (\Pr[SC(\xi_j, m_1) = c]) \qquad (10)$$

where ξ_i and ξ_j are defined as different sets of *PRP* with different initial values, μ and P_0 is defined in Equation 5. M is a set of all original messages and C consists of permutated messages based on a set of ξ values.

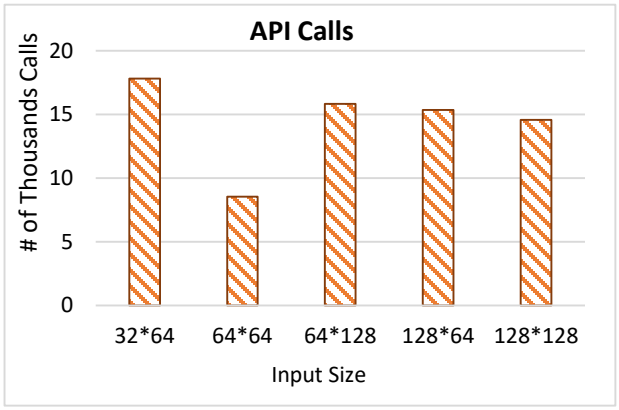
Lemma 1: *DPM has perfect secrecy.*

To proof Lemma 1, we must proof the following sub-lemmas, *Lemma 1.1* and *Lemma 1.2* as follows:

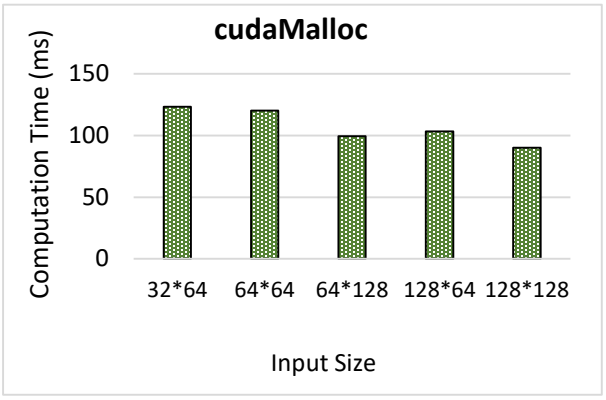
Lemma 1.1: By a given *c* (scrambled data), the adversary cannot learn about m_i and m_j (two different original messages). Therefore, we must generate different outputs for all different inputs.

Proof: Each separate original content in *DPM* should be scrambled with different sets of ξ to avoid similarity between $SC(\xi, m_i)$ and $SC(\xi, m_j)$. Each set of ξ is generated by $\mathbb{C}_m$GPU-core independently. Each core uses different initial values to generates different ξ sets without any conflict with other ξ sets, or with minimal partial conflict to other sets.

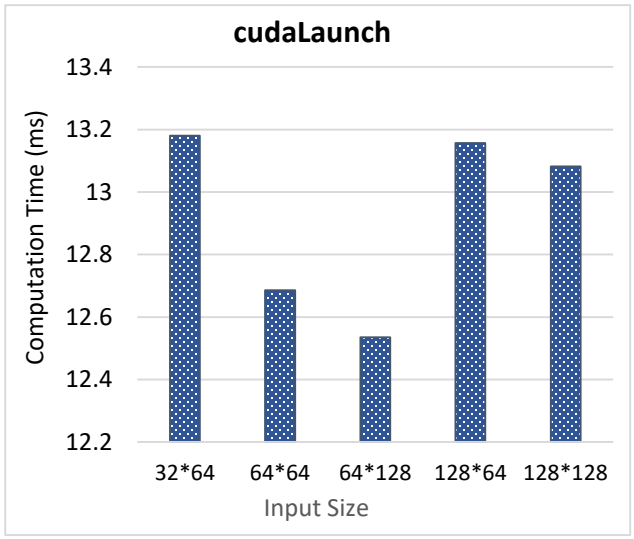
$$\forall File_i, c: Pr_\zeta[SC(\zeta, Content_i) = c] = \frac{\#\zeta \in \mathrm{Z} \text{ such that } \mathrm{SC}(\zeta, Content_i) = c}{|\mathrm{Z}|}$$


API Calls
# of Thousands Calls
20
15
10
5
0
32*64
64*64
64*128
128*64
128*128
Input Size

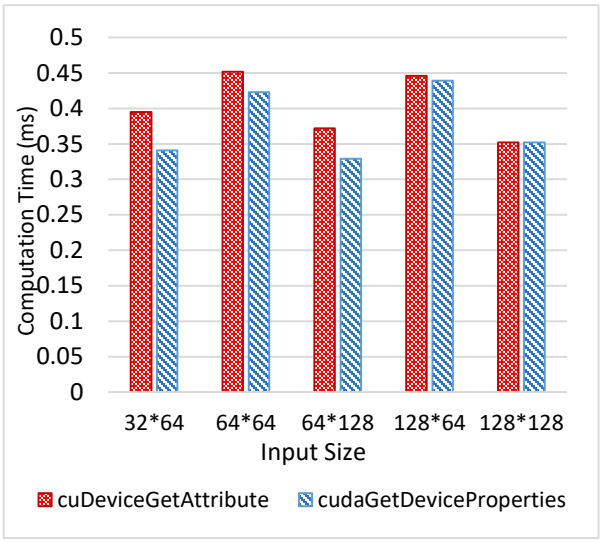
(a)


cudaMalloc
Computation Time (ms)
150
100
50
0
32*64
64*64
64*128
128*64
128*128
Input Size

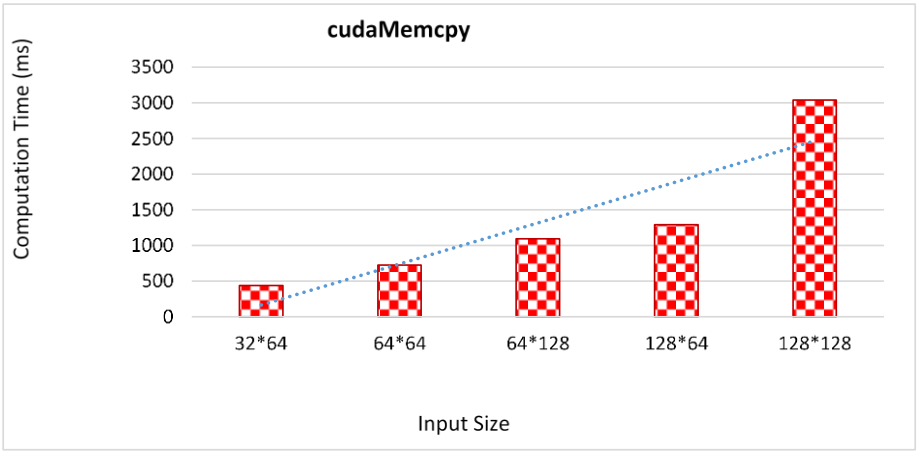
(b)


cudaLaunch
Computation Time (ms)
13.4
13.2
13
12.8
12.6
12.4
12.2
32*64
64*64
64*128
128*64
128*128
Input Size

(c)


Computation Time (ms)
0.5
0.45
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
32*64
64*64
64*128
128*64
128*128
Input Size
cuDeviceGetAttribute
cudaGetDeviceProperties

(d)


cudaMemcpy
Computation Time (ms)
3500
3000
2500
2000
1500
1000
500
0
32*64
64*64
64*128
128*64
128*128
Input Size

(e)

assumption), then an attacker by accessing to the scrambled content is not able to learn about $m_i$and m_j, if and only if the attacker cannot learn about sequence of ξ values which means the attacker should not have knowledge of parameters of ξ generator. As our evaluation of generated *PRP* shows in Figure 4.2 and 4.3, then the attacker is not able to learn about m_i and m_j by accessing to c. ■

Lemma 1.2: The ξ generator has perfect secrecy for all *GPU* cores.

Proof: The *PRP* must provide a uniform distribution for all entries of *n bits* as follows:

$$P: U \to [0,1] \ \ such\ that \ \sum_{x \in U} P(x) = 1$$

where $U = \{0,1\}^n$.

$$\forall\, x \in U: P(x) = \frac{1}{|U|}$$

Since each GPU-core generates a unique set of ξ values, then the probability of all ξ sets are equal and $P(x) = \frac{1}{|U|}$ for each GPU-cores satisfied the generator condition of perfect secrecy.

■

4.9 Summary of Chapter

Cloud computing offers new opportunities to users to efficiently outsource data and applications. Data privacy is one of the major issues in cloud computing systems. In Chapter 3, we introduced a light-weight data privacy method (*DPM*) that allows users to protect their data before submitting original file to the cloud. *Graphic Process Units (GPU)* allows parallel processes to be run efficiently. *GPU* kernel is able to process computationally intensive tasks on client side by using a *GPU* platform, such as *NVIDIA CUDA Toolkit*.

In this chapter, we introduced a solution to mobile cloud users to accelerate *DPM* on multicore *GPUs*. This study shows that *DPM* can be implemented securely and efficiently on multiple independent *GPU*-cores. The proposed method protects users' data privacy by processing independent pseudo-random number generator on each core when it is complying with perfect secrecy requirements. We evaluated the proposed method by performing rigorous assessments on performance and the security. On performance side, we ran different number of parallel processes in order to assess the computation time on each input size. We implemented the proposed method when it is being parallelized on a 2D-array of parallel processes where each thread block assigned by different initial values to generate different and unique pseudo-random numbers. The generated

numbered are used for permutation of an original file. On security side, we considered the security assumption of the method and we assessed the result of pseudo-random numbers, distribution of this random numbers and perfect security assessments to analysis the security of the proposed method on multiple *GPU* cores.

Chapter 5

Cloud-Assisted IoT based on DCCSOA

This chapter extends DCCSOA which is describe in Chapter 2 for supporting *Internet-of-Things (IoT)*. The next chapter describes how DPM can be implemented on the proposed architecture in this chapter.

5.1 Introduction

This chapter describes a convergence of two recent and popular paradigms that include cloud computing and the *Internet-of-Things (IoT)*. These two paradigm defines a new cost-effective model for a network of devices that are connected through the Internet. In this section, we define these two paradigms, and advantages of each as follows when we plan to collect a large amount of data and process the big data. The processing big data efficiently on the cloud environment is an opportunity to use a variety of IoT devices to transfer raw data to intelligent data.

5.1.1 Cloud Computing Paradigm

As we described in Chapter 1, cloud computing uses virtualization technology to provide virtual services to users. Virtualization technology introduced since operating systems were introduced in 1960s. Although the virtualization technology is not a new paradigm, the method of using the virtualization technology is new, and it provides virtual services to multitenant users in order to offer cost-effective *IT* services to cloud users. The virtual services include storage, and processing units on the cloud. The virtual services are shared among all cloud users.

The cloud enables users to increase or decrease the capacity of virtual storages, and the number of virtual processing machines on-demand. The cloud vendor bills cloud users based on a pay-per-use basis model. This dynamic resource allocation allows users to support any number of user's requests without paying additional cost to maintain a large number of processing machines, and a massive amount of data. Cloud computing has become popular to both startup businesses, and corporations because startup

businesses could start their businesses with low investment, and the corporations could save maintenance costs when the number of user's requests is decreased.

Since cloud vendors maintain all storages and processing units, the cloud users do not need to pay for additional maintenance costs, or even recovery costs in a disaster situation. The cloud vendors also deploy global backup servers to prevent users' data loss.

5.1.2 Internet-of-Things (IoT) Paradigm

The *Internet-of-Things (IoT)* (Gubbi 2013) paradigm is another opportunity for businesses to have a network of small computing sensor devices for sensing the environment, such as reading current temperature of the environment, predicating maintenance, and recently self-driving car's *IoT* sensors. Usually *IoT* devices use *Internet-Protocol (IP)* for connecting to the Internet, and it allows *IoT* devices to send data to the servers directly. *IP* allows each device to connect to the Internet without any local server, or a complicated *IT* infrastructure. The *IoT* devices are able to send their sensors results to any server around the world. The *IoT* devices use *IPsec* protocol (Doraswamy 2003) in order to transfer securely their data to the server(s). However, *IoT* devices have limited resources, such as limited power resource, low-speed *CPU*, and the small size of storage.

5.1.3 Convergence of IoT and the Cloud (Cloud-Assisted IoT)

The convergence of cloud and *IoT* devices, *cloud-assisted IoT*, enables *IoT* devices to outsource their data to a cloud computing systems. The cloud allows users to process the outsourced data with multiple virtual *CPU*s. Currently, several cloud vendors offer particular platforms, and cloud services for *IoT* devices. For instance, recently Microsoft introduced the *Azure IoT Suite* which provides a platform for developing *IoT* devices on *Microsoft Cloud Azure*.

Another advantages of *cloud-assisted IoT* is providing a hub between all *IoT* devices from around the world to collaborate with each other while using the cloud as the main connection.

Finally, big data analytic tools, as summarized in Chapter 1, is another advantages of the cloud that offers analytics services on collected raw data from *IoT* devices. Some of these tools are open-source, and are freely available to users to process complex queries on raw big data which are collected from IoT devices. Some of these analytic tools and services have been summarized in Chapter 1.

The rest of the chapter is organized as follows: the next section describes the architecture issue of *cloud-assisted IoT* as one of the major challenge that faces users when they want to use different cloud platforms. Section 5.3 describes a cloud architecture

based on *Service-Oriented Architecture (SOA)* (Perrey 2003) and *DCCSOA* that allows different cloud vendors to define a generic and dynamic platform in order to facilitate user's transformation for data, applications and IoT devices. Section 5.4 explains some big data characterization and how DCCSOA can support these characterization of big data. Section 5.5 describes advantages of the proposed architecture, such as standardization between heterogeneous cloud platforms by using the proposed architecture, security of the proposed architecture and it reviews a case study of implementation of data security on the proposed architecture. *Section 5.5* reviews related works, and finally, *Section 5.6* concludes this chapter.

5.2 Challenges in Cloud-Assisted IoT

Currently, we do not have an acceptable standard between cloud computing systems as we describe in Chapter 2, and even between major key cloud vendors, such as Microsoft, and Amazon. So, each cloud vendor offers its own platform to the users. The provided services differ from a vendor to another because each of them uses their unique set of inputs, outputs and processes to provide cloud services. These different services from different cloud vendors, offer heterogeneous cloud services to users. Although the heterogeneous cloud vendors accept major programming languages, transferring data and applications from one vendor to another, or returning back applications and data to the user's in-house *IT* department is difficult because each cloud vendor uses its own platform, or uses its own *Application Programming Interface (API)*. If a user wishes to transfer data and applications to another platform, he/she needs to modify the codes, and sometimes the user needs to redevelop the applications, or databases. The dependency of *IoT* devices to a particular cloud platform causes several sub-issues as follows:

i) flexibility issue that does not allow *IoT* devices to properly use all functions when the devices are moved to another platform;

ii) customization issue that happens when a vendor is not able to customize a cloud platform for supporting different users' requests;

iii) Security issue: if a cloud platform provides specific security protocols to *IoT* devices, the security of *IoT* devices can be compromised when the devices are transferred to another platform. This issue originates when the security of an *IoT* device relies on the particular cloud platform. In this case, the application on the cloud-side (server-side) needs to be redeveloped in order to protect the *IoT* device. For instance, if an IoT devices uses a handshake protocol in order to submit its data to the cloud and it is transferred to another vendor, the handshake protocol needs to be redesign for the new cloud vendor.

5.3 A DCCSOA-based Architecture for Cloud-Assisted IoT

As a result of lack of standardization between cloud vendors, each cloud vendor offers their own cloud architecture which means we have heterogeneous systems for *IoT* devices. For instance, if a user uses an *IoT* device that relies on *IBM* cloud, it is difficult, or even impossible to use the device with other cloud vendors, such as *Microsoft Azure*. Sometimes, it requires a redevelopment of database and applications in order to transfer *IoT* devices from one vendor to another.

Since we have heterogeneous cloud platforms, it is not cost-effective to modify all cloud platforms to provide standard services between different cloud platforms. We proposed a dynamic cloud computing architecture (*DCCSOA)* in Chapter 2 based on *Service-Oriented Architecture (SOA)*. Figure 5.1 illustrates how the *DCCSOA* facilitate deployment of multiple *IoT* devices on the top of a cloud vendor. In this figure, each component is connected to others as a service. The SOA's feature allows the architecture to implement on the top of heterogeneous cloud platforms without requiring additional modification on each cloud platform. Each cloud vendor with different platforms is able to define different *Templates* on the top of their cloud services. A *template* is an interface that interacts with one or more cloud services. A cloud vendor is able to define different *templates* in order to customize their services to a variety of users. *DCCSOA* provides a flexible cloud architecture that supports both the cloud vendor, and the vendor' users. On the user's side, the users can freely transfer data, and applications from one vendor to another with minimal modification costs while a network of *IoT* devices uses *DCCSOA*' unique interface on the top of heterogeneous cloud computing systems. On the vendor's side, cloud vendors are able to customize their platforms based on *DCCSOA* with minimal modifications in order to provide a unique and standard service to the users. On both sides (client's side and vendor's side), the level of modification is a key parameter of the provided architecture.

On one hand, less modification on user's side *(IoT)* device means more standardization between heterogeneous cloud platforms and more independent cloud services, and on the other hand, less modification on the architecture means offering a cost effective model that supports a variety of user's requests with minimal costs for customization.

The proposed architecture for *cloud-assisted IoT* is divided into three layers as follows:

i. **Cloud vendor:** This layer shows the implementation of a cloud vendor. Different cloud vendors offer heterogeneous cloud architecture in this level. Each cloud vendor offers a variety of cloud services which are called value-added services. For instance, *Infrastructure-as-a-Service (IaaS)* provides virtual infrastructures, such as virtual storage and virtual *CPU* to users, *Platform-as-a-Service (PaaS)* offers a

development and deployment environment to implement applications, and *Software-as-a-Service (SaaS)* that provides preinstalled applications on the cloud. Recently, a variety of cloud services also have been deployed on the cloud computing (see Chapter 2). A summary of different cloud services can be found in Chapter 1.

In this level, *DCCSOA* allows any type of cloud services with heterogeneous platforms with different interfaces to be offered by a cloud vendor.

ii. **Cloud-assisted IoT:** This layer provides a generic interface on the top of heterogeneous cloud platforms to client-side. The *Dynamic Template Service Layer (DTSL)* is divided into two sub-layer as follows:

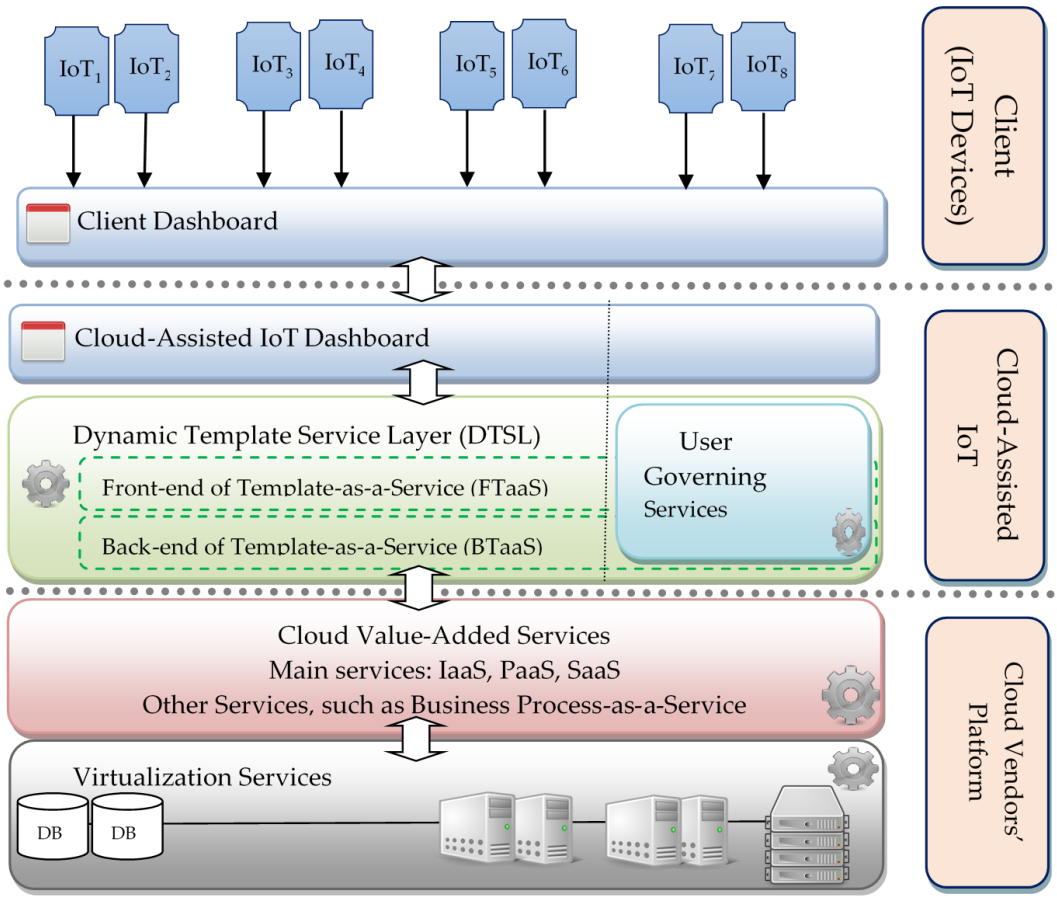


Figure 5.1. The Proposed Architecture of *DCCSOA* for *Cloud-Assisted IoT*

- *Front-end-Template-as-a-Service (FTaaS)* that provide a generic interface for cloud services. Clients are able to access this layer though a platform, or even as a preinstalled application on the cloud;
- *Back-end-Template-as-a-Service (BTaaS)* that binds each service at *FTaaS* as a generic interface to a particular cloud-value added service. A *template* (*T*) defines a generic interface for a service s (K_s) based on the required input/output of the given service s as follows:

$$T(K_s) = \sum_{i=1}^{m} input_{k,i} + \sum_{j=1}^{n} output_{k,j} \quad (1)$$

where $K \in \{P\}$ and P is a set of different cloud platforms which are heterogeneous, and each given service, s has m number of input parameters and n number of output parameters.

A cloud vendor defines dynamically several different *template* on-demand at *DTSL*. Each *template* integrates with one or multiple value-added cloud services. Cloud vendors can set up, configure and provide different *templates* to their customers based on different value-added service layers in a cloud computing system.

iii. **Client:** end-users' applications that include software and *IoT* devices are located in this layer. If an *IoT* vendor offers cloud-assisted storages, and computing units, the vendor can use the *dashboard* component (as illustrated in Figure 5.1) to have collaboration/interacts with cloud vendor on one side, and their customer on the other side. An *IoT vendor* who uses cloud vendor infrastructure to provide *IoT* services, can use heterogeneous cloud vendors while using a generic interface (a defined *template*) at the top of each cloud vendor.

The following equation defines a service at $FTaaS$ which is required to pass a satisfaction function $\mathcal{S}$ to propose a generic service on the top of heterogeneous cloud vendors. The detail of the services is described in Chapter 7.

$$\exists\, s \in FTaaS \mid \mathcal{S}at(s) \quad (2)$$

where *s* is a service at $FTaaS$ and $\mathcal{S}at$ is a satisfaction function which is defined as follows:

$$Sat(s): \mathcal{R} \rightarrow \mathcal{O} \quad (3)$$

where $\mathcal{R}$ is a finite set of requirements of *r*, and $\mathcal{O}$ is a finite set of corresponding output for each requirement in $\mathcal{R}$.

The generic service (UI) can be defined as follows:

$$UI(s) \rightarrow Sat(s_1)\text{^}Sat(s_1)\text{^}Sat(s_k) \quad (4)$$

A client is able access to a *template* without concerning about the location of the data source and other related configuration. The client accesses to the data source in this example through $FTaaS$ which is defined as a web service.

DCCSOA allows users to interact with a *template* at *FTaaS* that provides cloud services. The user does not need to have any knowledge of cloud-value-added services because they are implemented at *BTaaS*. This independency would be a great opportunity for IoT users who use different IoT devices and it allows them to freely transfer their data and applications from one vendor to another.

5.4 Big data processing on DCCSOA for cloud-assisted IoT

IoT offers a new set of tools, sensors and devices that can be controlled through the Internet. In the era of big data, collecting data, processing and analysis on data are the key points. Moving to IoT in the cloud environment allows us to achieve all these key points when each small tiny computing machine (IoT devices) generates daily reports of human activities. Collecting a large amount of data can be accomplished in the cloud environment by employing cloud-based databases (i.e., NoSQL databases) when users only pay per usage. In addition, there are several analysis tools that provides analysis service on big data by using a pool of computing machines in the cloud environment. In order to transfer bulk raw data of IoT devices to valuable data, we can process the content by using a large number of processor where the number of machines can be increased/decreased on demand. Using intelligent computation tools also raises interest for cloud-based systems because computation on big data causes several limitations on traditional computing machines. The question is how a variety of heterogeneous *IoT* devices able to use a unique cloud computing environment to efficiently collect big data and intelligently process the big data.

In this study, we consider each IoT device sends data to cloud vendors. The data is processed through FTaaS and the corresponding BTaaS, and then it is submitted to the cloud. Big data is defined by four characteristics - volume, velocity, variety and veracity which are described in Chapter 1. We describe the requirements of each of these characteristics as well as our recommended solutions to maintain these characteristics based on the proposed cloud-assisted IoT architecture.

5.4.1 Volume

Big data is characterized by extremely high volume data which is indicated the by the size of data. Regularly, each IoT device generates a small portion of big data; however, periodically generating small data and collecting all data from a large number of devices, we will have a large volume of data.

Processing big data on cloud computing server might be a challenge; however, different tools allow a cloud vendor to store and process this big data. For instance, different NoSQL cloud-based databases allows users to store structured and unstructured data on the cloud without concerning about speed of data retrieval.

5.4.2 Velocity

Velocity indicates the speed of data processing in term of response time. The response time could be a batch, real-time or stream response-time. When we consider the velocity only for a portion of data which is generated by an IoT device in a short period of time, it is not a challenge but if we consider a long-term data collection from a large number of IoT devices, it is a challenge for response time of processing data in a timely manner. In this case, sometimes it requires a cloud computing system to support real-time response. The real-time response is required a High-Performance Computing (HPC) system. In Chapter 4, we discuss a similar challenge for mobile users who need data privacy. We use Graphics Processing Unit (GPU) rather than CPU that allows 1000s of threads run a portion of the task when each thread is processed on a GPU-core. In this case, the proposed architecture is able to efficiently process a request by parallelizing the task and splitting the task among to multiple GPU-cores. In order to offer this type of service (i.e., a real-time system), we may bind a BTaaS to a set of GPUs and provide a function as an interface at FTaaS. Therefore, any task is submitted to the function at FTaaS will be able to processed in real-time at its corresponding BTaaS when the task actually runs on thousands of GPU-cores. Some similar studies for implementing parallel tasks on the cloud can be found in (Suttisirikul 2012 and Oikawa 2012). These systems can be connected to BTaaS to perform HPC on cloud computing.

5.4.3 Variety

Variety represents heterogeneity/diversity in data which is collected from different IoT devices. Another challenge is analyzing of the variety of data including structured, unstructured and semi-structured data. We have a variety of IoT devices and they might generate a set of video files, text files, JSON-based output files that includes structured data such as database, semi-structured data such as text and even unstructured data, such as multimedia (e.g., voices, images, videos). When we are using a cloud computing, it is capable of processing of a variety of formats. Some cloud-based tools that allow users to

analyze a variety of data, such as Talend (see Chapter 1) which is a data integration, data management, enterprise application integration and big data software tool. The similar cloud-based tools are available to solve the variety issue when each tool can be bound to BTaaS and the functionalities of each application can be exposed to the users at FTaaS. For instance, if Talend is bound to BTaaS, it can provide data integration at *DTSL* layer but each function or multiple functions of data integration can be implemented in FTaaS layer. Flexibility in defining services through defining multiple FTaaS allows cloud vendor to support a variety of data type for IoT devices.

5.4.4 Veracity

Veracity is the level of accuracy of data. For example, a sensor that generates data may provide a wrong value (e.g., an IoT device which reports inaccurate temperature). The proposed architecture is able to verify the accuracy of collected data when minimum of two similar IoT devices are located in the same environment and they are connected to an FTaaS (e.g., two IoT devices that measure the temperature are connected to FTaaS). The architecture allows the veracity of data to be reviewed on the edge (which is also the purpose of Fog Computing (Bonomi 2014) at FTaaS. In this scenario, if the value of collected data from two sensors are not the same, then an intelligent application might review the history of each device to see which device provided the correct result, or we can cross check the received data at FTaaS. Reviewing data at FTaaS allows us to remove additional overheads on the back-end of cloud computing system. Therefore, the veracity of data can be accomplished without entering to the back-end (BTaaS). In this example, the accurate data can be collected at BTaaS and can be stored in the cloud.

5.5 Advantages of DCCSOA for cloud-assisted IoT

The proposed architecture provides several advantages to the cloud vendor, *IoT* vendors and their end-users that include standardization, customization and security which are described as follows.

5.5.1 Standardization

One of the major issues in cloud computing is a lack of standardization between different cloud platforms and as we discussed it causes vendor lock-in issue that does not allows *IoT* devices, and their data and applications freely transfer to another cloud vendor. *DCCSOA* provides a dynamic service layer (*DTSL*) to enable different vendors to offer a generic platform through defining the same *FTaaS* in different cloud platforms. When different cloud vendors provide the same *FTaaS* to their customers, the customers are able to transfer their data and applications from one vendor to another, or even they can transfer data and applications to their own *IT* department. It also allows *IoT* devices to use all functions while transferring from one vendor to another.

In addition, the standardization between different cloud platforms in *DCCSOA* can be extended to heterogeneous *IoT* devices by using *a generic template* that enables a cloud platform to interact with different *IoT* devices where each *IoT* device has specific I/O features.

Adding standardization between cloud platforms enables portability features for users' applications and data in the cloud. For example, Figure 5.2 illustrates how three different group of IoT users use two heterogeneous clouds. Users are able to subscribe for different *templates* (T_1, T_2, and/or T_3). Users interact with *templates* from *FTaaS* layer of *DTSL*, and each *template* binds to one or more cloud service layers in any cloud platform. For instance, *IoT Customer Group 1* uses *template* T_1 to access SaaS layer which is provided by the first cloud vendor, $SaaS_a$. As illustrated in Figure 5.2, T_3 provides different *PaaS*s from both vendors, $PaaS_a$ and $PaaS_b$.

The *templates* enable the vendor to have flexibility for offering a generic cloud service. Although defined *templates* in Figure 5.2 bind to the similar cloud services from both vendors, each template can be bound to different services from different cloud vendors. For instance, T_1 can be bound to $SaaS_a$ and $PaaS_b$.

In addition, the number of *templates* can be added/reduced on-demand. If a cloud vendor has an attractive service that brings more users, then the vendor can define a new *template* that covers the new user group needs for the new service.

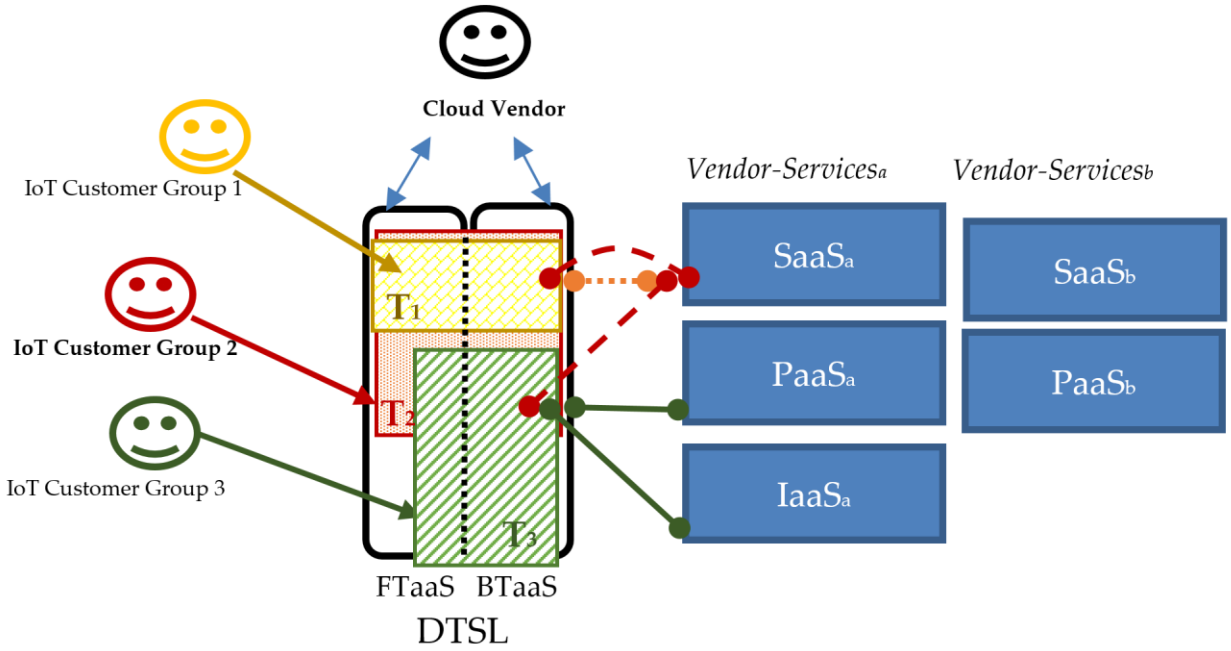


Figure 5.2. One snapshot of DTSL and its interaction with two heterogeneous cloud platforms

5.5.2 Customization of architecture

The customization is one of the major issue of cloud architectures which is described in Chapter 2. For instance, a third-party who design services for cloud vendor could not customize defined services. This issue causes other related issues such as the *lack of usability of cloud computing (*IDC Enterprise Panel 2013) that emphasizes *integration issue* in cloud computing systems.

The proposed architecture includes a dynamic layer (*DTSL*) that allows cloud vendors to customize their cloud architecture on-demand because a vendor is able to define a template for a particular service. Although the *template* binds to a particular service at *BTaaS* layer, it provides a generic and customized service at *FTaaS* layer. When a cloud vendor defines a new *template* that interacts with several cloud services, it enables users to have an *integrated-service* from different cloud services. This *integrated-service* cannot be only customized by cloud vendors or their partners, it also can be provided as a generic service on the top of heterogeneous cloud platforms. In addition, offering an *integrated-service* could attract a variety group of users. For instance, if a cloud vendor offers a message service that allows different applications to contact each other, and a *virtual private network* (*VPN*), the cloud vendor can integrate these services and customize them into a *template*. In this case, a user is able to subscribe to the *template* and only subscribe to the *template*.

Customization of cloud services can be completed at different levels as follows:

- *Low level Customization* that customize services at the *Implementation Level* where cloud developer implements a system. This level of customization is not a cost-effective practical method for both cloud vendors and users because each individual application needs to be modified.
- *High Level Customization* customizes the cloud services at the *Architecture Level* that adopts new and customizes existing cloud services. If the architecture implements on the top of existing architecture, it allows a cloud vendor to customize the service with minimal modifications. It also allows existing applications to be used without any modifications. Since DCCSOA allows a cloud vendor to modify a *template*, the existing applications can be run without modification and a cloud vendor only needs to modify the *template*.

As a result, the customization of cloud services is a practical model if both cloud vendors and users do not need to have extensive modifications. DTSL removes the additional modification costs by using *templates*.

5.5.3 Data Security

Data security emphasizes authentication for accessing to data and applications and it protects data against an adversary from internal/external access to data. Data privacy is another challenge in cloud computing that emphasizes who are authorized to use the outsourced data in the cloud. Both data security and privacy schemes can be implemented as a *FTaaS*. More details on data privacy is described in the following section. The implementation of the security schemes as a *FTaaS* allows vendors, or users to use standard security schemes on the top of heterogeneous cloud computing platforms without additional computation overhead.

5.6 Summary of Chapter

In the era of big data, using cloud computing gives several advantages to users to not only collect a large volume of data from *Internet-of-Things (IoT)* but also process data efficiently through a variety of tools. The combination of cloud computing and the *Internet-of-Things (IoT)* paradigms allows users: to sense environment through *IoT* devices, to outsource data directly from *IoT* devices to the cloud, and compute a massive amount of raw data on the cloud.

Although the cloud-assisted *IoT* are providing a cost effective model for users, we do not have an acceptable standard among cloud vendors for supporting different *IoT* devices that causes heterogeneous cloud platforms. This cause lock-in issue for users that does not allow users to freely transfer their data and applications from one vendor to another, or even returning back to their *IT* department. In this chapter, we proposed a dynamic cloud computing architecture which is designed based on *Service-Oriented Architecture* (*SOA*) that allows cloud vendors to offer a generic interface *(FTaaS)* to their users that supports heterogeneous cloud platforms at its corresponding *BTaaS*. The *SOA*-based feature of the architecture allows cloud vendor to define a generic interface with minimal modifications on their platforms in order to avoid additional costs.

The proposed architecture uses a dynamic layer *(DTSL)* to offer vendor services and it is divided into *front-end layer* (FTaaS), and the *back-end layer (BTaaS)* which binds each generic service to a particular service in a cloud computing system. The proposed architecture allows heterogeneous platforms to provide generic and standard services to the users with minimal modifications on the vendor side.

The proposed architecture provides independency and customization ability to cloud vendors as well as for the users. The proposed architecture can protect data users' data privacy through a light-weight data privacy method. We also describe the definition of big data and what are the key requirements for supporting big data including collecting a massive volume of data, and velocity of processing of the big data in the cloud.

To the best of our knowledge, we could not find standardization and customization on the cloud for IoT devices. However, a limited study has been conducted on cloud architecture, service customization and standardization between clouds. In this section, we review the most major studies.

Chapter 6

DPM for Cloud-Assisted IoT

In the previous chapter, we describe how Cloud-assisted IoT may use DCCSOA (see Chapter 2) to provide flexible services, dynamic feature to the users. As we described, the second goal of our study is maintaining users' data privacy. The questions that aims to answer in this chapter is: "*How the DCCSOA-based architecture for cloud-assisted IoT can maintain users' data privacy?*" This chapter described a convergence of DPM (Chapter 3) and Cloud-assisted IoT architecture based on DCCSOA (Chapter 5) to answer the question.

6.1 Introduction

The Internet of Things (*IoT*) (Gubbi 2013) paradigm provides a connected network of smart devices through the Internet. The devices include sensors, actuators, cameras and Radio Frequency IDentification (*RFID*) devices to record current environmental condition. The goal of the IoT is making smart decisions based on archived sensed data and the current situation of the environment. IoT is used for different environments, such as healthcare systems, smart homes, smart cities, smart cars and more recently, self-driving cars. The network of these smart devices that comprise of sensors and other smart technologies working in tandem and communicating efficiently are creating a new world of operation called the IoTs.

Massive amounts of data need to be collected from IoT objects to achieve smart decision making, but IoT objects are not capable of collecting massive amounts of data in their storage and hence they have to outsource their data. As described in Chapter 3, Mobile Cloud Computing (*MCC*) (Dinh 2013) provides an efficient platform for IoT devices to outsource their data directly through the Internet. *MCC* allows *IoT* objects:

(i) to read data from sensors and archive data in *MCC*;

(ii) to process data on cloud computing systems; and (iii) to retrieve or download data by other devices from everywhere. In addition, *MCC* paradigm allows users to use pay-per-use basis and offers the ability to upgrade the size of resources on-demand.

Tiny computing machines, *IoT devices*, have two challenges- resource limitation and data privacy issue, which are described, in the following sub-sections.

6.1.1 Resource Limitation

First, all mobile devices have limited resources, such as battery power, CPU speed and storage capacity. When IoT devices outsource their data to *MCC*, they must monitor *MCC* network transactions for tracking the battery usage. According to (Kumar et al. 2010) not all mobile applications are useful when the applications use *MCC*. For example, an application that reads/writes content from/to *MCC* uses more battery than an application that uses *MCC* to only process data on *MCC*. The second application which only processes data on *MCC*, does not have additional network traffic costs.

The IoT devices are smart compact devices designed for efficiency and portability and ease of use. All these features lead the designer of these devices to think smaller and compact devices, which in turn leads to limited resources to accomplish full blown operations. In addition, IoT devices have limited storage capacity, generally less than 256 KByte. This size of storage adds an obstacle for most of the encryption methods to run efficiently on IoT devices.

6.1.2 Data Privacy

The second challenge with respect to the IoTs is data privacy because when an IoT device outsources its data to the *MCC*, data privacy can be violated by the cloud vendor, the vendor's partners, hackers, malicious entities or even by other cloud users.

IoT devices require a light-weight data privacy method while using *MCC* to save their resources, such as power consumption, and to maintain data privacy. In Chapter 3, we presented a Data Privacy Method (*DPM*) for *MCC* applications on mobile devices with respect to limited battery power. *DPM* splits a file into several files, and each file is divided into several chunks. The method scrambles chunk of each split file randomly by using a chaos system (Kocarev 2001). The encryption/decryption method saves 72% battery power over AES encryption method because *DPM* can be run in *O(1)* time complexity and it does not add a significant overhead to the mobile device.

This chapter introduces the customization of *DPM* for *IoT* devices. However, as discussed previously, the biggest challenge to customizing *DPM* for *IoT* devices is their limitation of storage capacity because scrambling a small size of data is not secure. We

propose a data privacy scheme in Section 6.6 that provides a secure method based on *DPM* for *IoT* devices.

The rest of the chapter is organized as follows: the next section reviews motivation of the study and the major challenges *IoT* devices face to maintain data privacy while using *MCC*. Section 6.3 presents the motivation of this study. Section 6.4 explains some limitation for IoT devices. Section 6.5 describes the related work for this chapter. Section 6.6 presents the proposed data privacy scheme *DPS* for IoT devices. Section 6.7 presents the experimental setup. Its results in Contiki simulation tools is described in Section 6.8. Finally, Section 6.9 presents a summary of this chapter.

6.2 Data Privacy for IoT Devices

On one hand, we need a method to protect data privacy for *IoT* devices before outsourcing the original data to *MCC*, and on the other hand, *IoT* devices need a simple and efficient method to avoid battery drainage in a short period of time. Traditional security methods can be run on regular computing machines, such as a laptop to protect data privacy. However, running traditional encryption methods on *IoT* devices is not efficient because it causes the battery drainage in a short period of time. Sometimes, it is impossible to run a heavy encryption method on *IoT* devices (Bogdanov 2007) because the devices have limited storage capacity and the encryption methods need several rounds to encrypt a plaintext. For instance, *AES* encryption needs 10 rounds to encrypt a plaintext (Bogdanov 2007), and each round needs a heavy *CPU* computation and a storage capacity to buffer data to complete the encryption process.

In addition, an encryption method needs to be run in preemptive mode because of *CPU* speed limitation of an *IoT* device. However, the device must read its sensors in a short period of time regularly, and it cannot stop sensor reading in order to process an encryption method.

6.3 Motivation

With the advancements in technology and communication the use of IoT devices to enhance and improve living has created a plethora of devices. The major issue for mobile and *IoT* devices to run an encryption method is the time complexity of the method. The advantage of *DPM* for IoT devices is its time complexity for both running fast on *IoT* device with limited resources, and heavy computations against attacker.

The *DPM* disassembles a file in *O(1)* time complexity but in an attack scenario, an adversary needs *O(n!)* time complexity to assemble a file. In Chapter 3 we explain by increasing the size of *n*, the complexity will be increased exponentially. As illustrated in Figure 6.1, the worst case of time complexity is *O(n!)*, and the next order is $O(2^n)$. When

a heavy computation is required to retrieve the original data, it creates an obstacle against an adversary to use unauthorized data. Practically, a method with $O(2^n)$ can provide more than 100 years of computation when the size of n increases to more than 256.

Another advantage of *DPM* is the simplicity of the scrambling method that needs *O(1)* time complexity when we have an array of pseudo-random numbers. This advantage allows the method to run on a low-speed *CPU*. In *DPM*, a set of random numbers based on a chaos system can be generated, and the method uses this set to scramble chunks of original data.

This chapter describes a method with minimum time complexity to maintain data privacy and to provide maximum time complexity (i.e., $O(2^n)$) against an adversary to retrieve an original file.

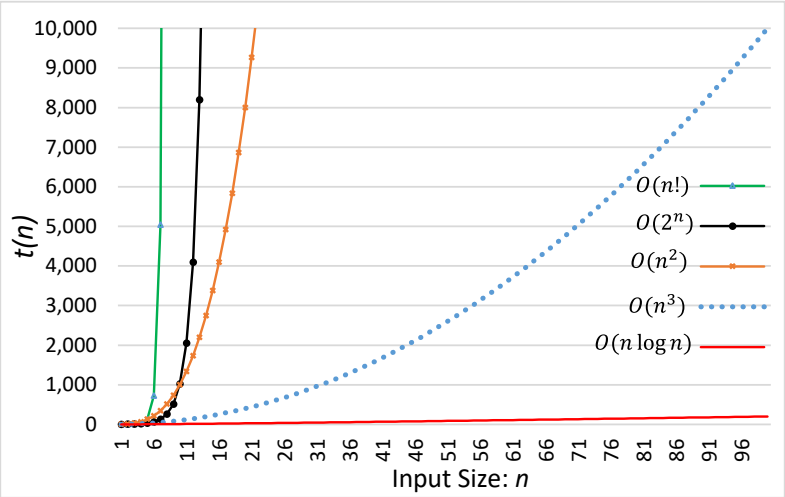


Figure 6.1. A comparison of algorithm time complexity

Cloud computing paradigm provides a cost effective storage for connected *IoT* devices and it allows each device to outsource its data to the cloud computing servers. In addition, the data can be accessible by any device from everywhere.

Some of *IoT* devices may carry sensitive data that requires privacy protection, such as health care systems or the department of defense systems. In addition, there are legal regulations to consider, such as the U.S. Health Insurance Portability and Accounting Act (*HIPPA*) that does not allow a mobile device to outsource the original data to a third-party.

6.4 IoT devices and their limitation

Currently, most of the *IoT* devices use Wireless Sensor Networks (*WSNs*). Internet Engineering Task Force (IETF)[34] developed IPv6 in *WSN* through *6LoWPAN* protocol that provides Routing Protocols for Low power and Lossy networks (*RPL*) (Zhang 2014). The *6LoWPAN* protocol allows millions of smart objects using *IPv6* to connect directly to the Internet. *IoT* devices generate small amount of data but during a lengthy period of time, a set of smart devices produce a large data set. We assume each smart devices uses *IPv6* for connecting to a mobile cloud computing to outsource its data.

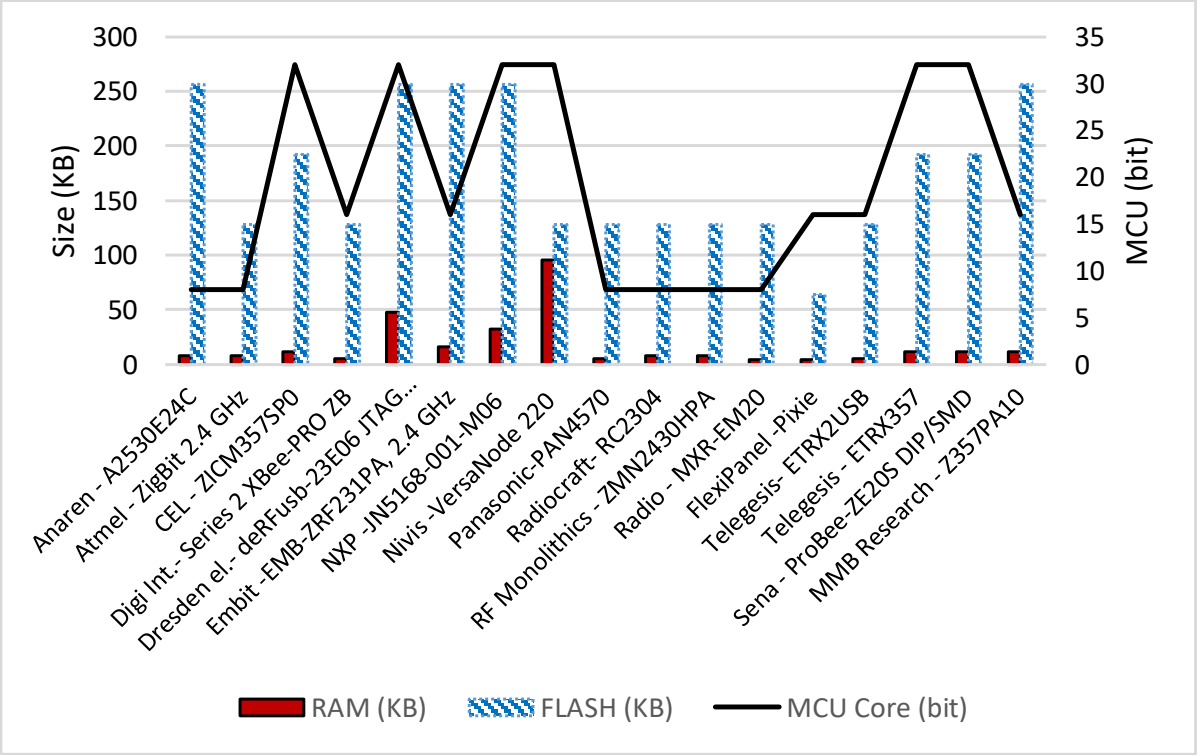


Figure 6.2. A comparison of resource capability of 6LoWPAN modules

One of the advantages of *IoT* technology is a network of sensors that uses the *6LoWPAN* protocol. Although the protocol provides secure communications through *IPSec,* the protocol is defined for devices with limited resources and it is almost impossible to encrypt generated data in an efficient way (Bogdanov et al. 2007) on *IoT* device. Figure 6.2 summarizes major *6LoWPAN* modules' specifications. *X-axis* represents the name of *IoT* modules and their manufacturers. *Y-axis* represents modules' resources (*RAM*, Flash and CPU core bits) and *Z-axis* represents the name of modules' resources that includes *RAM*, Flash and CPU core bits. As shown in this Figure 6.2, the

[34] https://www.ietf.org/

maximum size of *RAM* and Flash is less than 96KByte and 512Kbyte, respectively. It is not efficient and it is sometimes impossible to run traditional encryption methods on these tiny computing machines with these capacities. *AES* encryption method is a round-based method that needs to run for each plaintext with a key length of 128 bits to 256 bits. Practically, the module cannot pass the requirements to run *AES* encryption (Bogdanov et al. 2007).

6.5 Related Works

(Ayuso et al. 2009) presents an encryption method that splits the plain text to encrypt data but most of our target devices as shown in Figure 6.2 have a low-speed *CPU* with 8-bit core. (Marin 2013) also represents a method for *IoT* devices to run *AES* encryption by splitting the blocks. This method requires processing time to finish the encryption process. It needs to stop reading other processes from sensors, and it is almost impossible for an *IoT* device to stop sensing the target environment or to stop transferring the generated data while their *CPU* is busy to run encryption method. Several other studies such as (Ayuso et al. 2009 and Marin 2013) have been developed for encryption on *IoT* devices but each method can be run on a specific device, i.e., (Marin 2013) is developed for 16-bits devices on 6LoWPAN. However, the proposed scheme in this chapter can be run on any *IoT* device with considering how to extend the size of n.

As related works indicate, we need cloud computing to outsource data to use the cloud's benefits while also considering that encrypting the generated data with traditional security methods, such as *AES*, is not efficient. We are interested in submitting to the cloud is not efficient for *IoT* devices because the target devices have limited resources as shown in Figure 6.2.

In addition, as previously discussed, some devices must not submit original (unencrypted) data to a cloud computing system when they need to maintain *HIPPA* privacy act requirement. This requirement forces all *IoT* devices in e-health care systems to encrypt data before submitting their data to a cloud computing system as a third-party.

As we describe in Chapter 3, a light-weight Data Privacy Method *(DPM)* can be run on mobile devices. The method splits an original file to several chunks and the method employs chaos system to generate random numbers based on pseudo-random permutation (PRP) to scramble chunks in a file. A set of PRP numbers, P, is used to scramble chunks of original data. An attacker needs a heavy computation to retrieve the original file. For instance, if an attacker uses a brute-force algorithm to retrieve the original data from 256 chunks, the time complexity will be $O(2^{256})$. In theory, it requires 3×10^{51} years of computation with a supercomputer that can run 10^{18} combinations per second.

6.6 The Proposed data privacy Scheme for IoT Devices

A general view of the proposed Data Privacy Scheme (*DPS*) for *IoT* devices is illustrated in Figure 6.3. The scheme allows a smart object to submit its scrambled data to *MCC* without exposing its original data to the cloud vendor. In Figure 6.2, *DPS* runs on an *IoT* device. *DPS* uses μ and P_0 as keys to generate or to retrieve an array of *PRP* numbers. These initial values can be stored in a local server, *KeyDB* because needs not be stored in the cloud to ensure data privacy. The size of *KeyDB* is small and it can also be encrypted in the local server.

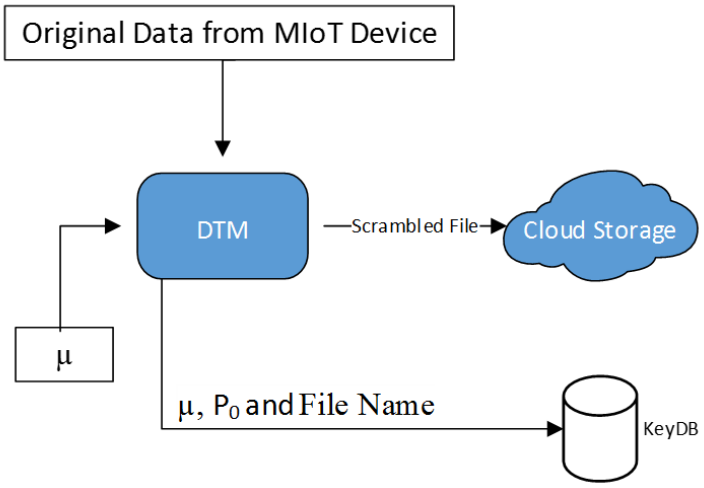


Figure 6.3. A General view of the proposed data privacy method for MIoT

As described in Figure 6.2, our *IoT* target devices have limited Flash and RAM capacity and each device generates a small data size. If we scramble generated data each time, the number of n will be too small and an attacker can retrieve the original data in a short period of time. For example, if we only scramble 16 chunks and submit it to the cloud, an attacker can retrieve the original data in $O(2^{16})$ time complexity. In this case, the attacker needs to try 65,536 permutation combinations to retrieve the original data that takes only several minutes to compute it on a *PC*. We can increase n by submitting partial data to the cloud and buffering the rest in the buffer. We scramble a full-buffer before submitting the content of buffer to the *MCC*.

In addition, we consider each bit as a chunk of the original data that allows us to increase the size of n rather than considering Bytes or Kbytes as a chunk.

Algorithm 6.1. shows the distribution of generated data from a sensor where D_i is a bit of each generated sequence bytes from the module's sensor. The algorithm decides where D_i must be stored based on $\{\psi_i\}$ where $\psi_i$is a set of PRP numbers. If ε_i=1, D_i will be buffered in *B*. Otherwise, D_i will be stored in the next available slot of *S*. The algorithm scrambles *S* based on ψ_j where $\psi_j$is another set of PRP numbers. Finally, the algorithm submits the scrambled data to the *MCC*.

In Algorithm 6.1, we consider two sets, B and S, to split generated sequence of bits. These two sets add complexity by increasing the size of n. S represents the selected bits that must be submitted to the *MCC* and B represents the selected bits that must be buffered in *RAM*. If the size of *RAM* and buffer defines as $\mathbb{C}_{RAM}$ and $\mathbb{C}_D$, respectively, the buffer will be full when $\mathbb{C}_D = \mathbb{C}_{RAM}$.

Algorithm 6.1. Distribution Algorithm for Scrambling $\{D\}$

```
01  While (1)
02  {
03    While (ℂ_D ! = ℂ_RAM)
04          If (ε_i==1)
05              {B} ← D_i
06          Else
07              {S} ← D_i
08    If  (ℂ_D == ℂ_RAM)
09       {
10          {S} ← SC_ψi{B}
11          STREAM_{S}
12          {B} ← {}
13       }
14  }
```

Generated data by an IoT device, *D*, defines a set of bits for each clock-cycle in which the device reads its sensors. After a time period, T, the device reads the sensors and it generates a new sequence of bits, D. We assume that the device generates *k* set of *n-bits* and it defines as follows:

$$D = \bigcup_{i=0}^{k} \{0,1\}^n \tag{2}$$

As previously described, *DPS* splits *D* into two sections as follows:

$$D = \sum_{i=0}^{k} B_i + \sum_{i=0}^{k} S_i \tag{3}$$

where B_i stores in buffer and S_i submits to the *MCC*.

DPS define two *PRP* based on Equation 1, ψ_i and ψ_j as shown in Equation 4 and 6. ψ_i generates decision bits for D_i whether D_i must be stored in buffer, B or D_i must be stored in S to submit to the *MCC*. ψ_j is used for scrambling the buffer when B is full.

$$\psi_i = \{\varepsilon_0, \varepsilon_1, \dots, \varepsilon_{\mathbb{C}_{RAM}}\} \tag{4}$$

where $\varepsilon_i \in \{0,1\}$ and it is defined as follows:

$$\varepsilon_i = \begin{cases} 0 \; if \; Pk \; mod \; 2 = 0 \\ 1 \; if \; P_K \; mod \; 2 \neq 0 \end{cases} \tag{5}$$

where P_k can be generated from Equation 1 with the initial values of μ_i and P_j.

DPS uses B to buffer partial bits based on ψ_i, then, *DPS* scrambles S based on ψ_j, and finally, the scrambled data, $SC_{\psi_i}\{B\}$ is submitted to the *MCC*.

DPS adds a set of bits (D_i) to S, if (ε_i=1) until $\mathbb{C}_D < \mathbb{C}_{RAM}$. If ($\mathbb{C}_D = \mathbb{C}_{RAM}$) *DPS* uses ψ_j to scramble B where ψ_j is defined as follows:

$$\psi_j = \{\varepsilon_0, \varepsilon_1, \dots, \varepsilon_{\mathbb{C}_{RAM}}\} \tag{6}$$

where $\varepsilon_i = P_l$ for i=0.. $\mathbb{C}_{RAM}$. P_l can be generated from Equation 1 with initial values of μ_j and P_j.

Finally, scrambled data, $SC_{\psi_j}\{S\}$, is transferred to S for submitting to the cloud.

The algorithm increases the complexity of retrieving the original data from scrambled data, $\mathbb{E}_D$by using B and S. The complexity of the algorithm for retrieving the original data is defined as follows:

$$O(\mathfrak{D}(\mathbb{E}_D)) = O(2^{\mathbb{C}_{RAM}{}^2}) \tag{7}$$

where $\mathfrak{D}(\mathbb{E}_D)$ is the complexity of retrieving the original data from scrambled data, $\mathbb{E}_D$.

If we consider maximum size of *RAM* in Figure 6.1, the time complexity of retrieving an original data from its scrambled data is $O(2^{96})$. However, we increase the size of n

from 96 to 96^2 that increases the complexity of retrieving the original data from the scrambled data by $O(2^{9216})$ when we consider 1Kbyte as an input.

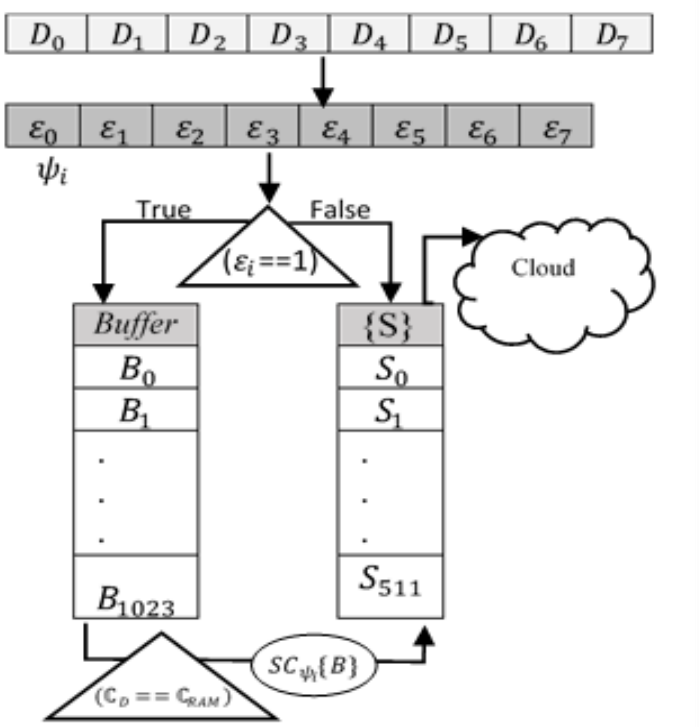


Figure 6.4. An example of submitting 8-bit generated data from IoT device with 128KB buffer, and 64 KB stream data to cloud

Moreover, the complexity can be increased to $O(2^{9437184})$ when we consider each bit as a chunk. In this case, the scheme scrambles an array of 9,437,184 bits which requires several hundred years of computation to retrieve the original data.

An example of the transferring of a generated 8-bit length data from the sensors to *MCC* is shown in Figure 6.4. In this example, 8 bits generated from the sensors as follows:

$$D = \{D_0, D_1, \ldots, D_8\}. \quad (8)$$

The scheme uses ψ_i in Equation 6 to generate ε_i and uses ε_i to decide the location of D_i. The scheme scrambles B based on ψ_j when 1024th-bit added to B and then it clears the buffer. In this case, if we assume ψ_i decides each two consecutive bits to be stored in B and S, randomly, B can be completed after 256th reading data from the sensors (k=256). In this case, an attacker needs to run a brute-force algorithm with $O(2^{2048})$ time complexity to retrieve an original data from scrambled data.

6.7 Experimental Setup

We conducted an experiment for the proposed method by a network of sensors on Cooja simulator with Contiki Ver. 2.7 operating system (Dunkels 2004), which is developed in Java. In this experiment, we use Sky Mote to emulate Tmote Sky mote (a wireless sensor module) in UDP Sink and UDP Sender. An UDP Sink is able to read sensors, to transfer its data, and to transfer data from one node to another node (works as a router).

An UDP Sender is able only to read the sensors and to transfer its data to the UPD Sink.

Each node is connected to the Internet and obtained an IPv6 address by using 6LoWPAN protocol. All nodes are able to transfer their data to the *MCC*. We ran the experiment with 12 nodes for two hours.

6.8 Experimental Results

In this experiment, we were interested in the result of the differential of power consumption when a node runs the proposed data privacy scheme and when it does not. We ran the first-half of the experiment (the first hour) without the proposed scheme and then we ran the proposed scheme for the second–half of the experiment (the second hour). As we previously discussed in Section 6.2, we expected the proposed scheme to run the algorithm in *O(1)* which means the algorithms must not introduce additional power consumption during the experiment.

We assume the following initial values for ψ_i:

$P_0 = 0.999 \quad \mu = 3.67$

Figure 6.5 shows the repetition rate of ϵ_i in ψ_i. *X-axis* represents the value of ϵ_i and *Y-axis* represents the number of consecutive bits with the same value. Each "×" shows the value of ϵ_i where *i=0* to *i=151*. As discussed previously, the scheme based on ϵ_i decides whether $D_i$must be stored in *B* or *S*. The figure shows that the repetition curve does not follow any pattern and an attacker cannot predicate the next decision bit. For instance, $\epsilon_0 = 1$ but the value of $\epsilon_1$to ϵ_9 is 0. We consider the following initial values for ψ_j:

$P_0 = 0.999 \qquad \mu = 3.684 \qquad |B| = 306$ bits

Figure 6.6 shows the experimental results. Figure 6.6.a shows power consumption during the two-hour experiment and the result indicates that the power consumption of the nodes does not change dramatically. Figure 6.6.a shows the experimental result for 8 different nodes. Nodes 2, 3, 4 and 8 did *not* use the proposed scheme during the experiment and nodes 12, 13, 18 and 23 use the proposed scheme during the second-hour

of experiment. Figure 6.6.b shows that the average power consumption which is not changed when we run the proposed scheme.

6.9 Summary of Chapter

In this chapter, we considered Mobile Cloud Computing (*MCC*) as a solution for outsourcing the generated data from *IoT* devices. There are two obstacles for *IoT* devices to use *MCC*. First, submitting the original data to the *MCC,* exposes users' data privacy to the cloud vendor and the vendor's partners. Second, data encryption for mobile users, in particular for tiny mobile devices, such as *IoT*, is not practical because they have limited resources, such as storage capacity less than 256 *Kbyte*. In this chapter, we presented a scheme that allows *IoT* devices to maintain their data privacy while each device outsources its data to MCC directly. We implemented the proposed scheme on one of the popular simulation tools by simulating *Tmote Sky* which uses *IPv6* and *6LoWPAN* protocol. We simulated a network of *Tmote Sky* on Contiki for two hours. The experimental results show that process of the proposed scheme on these modules does not introduce additional power consumption overhead.

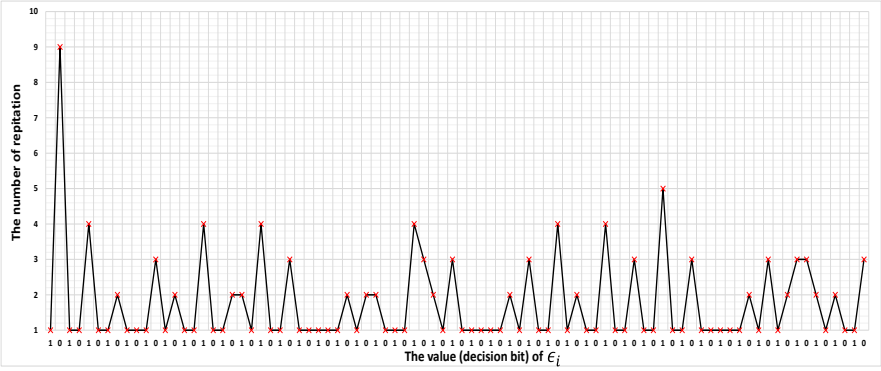
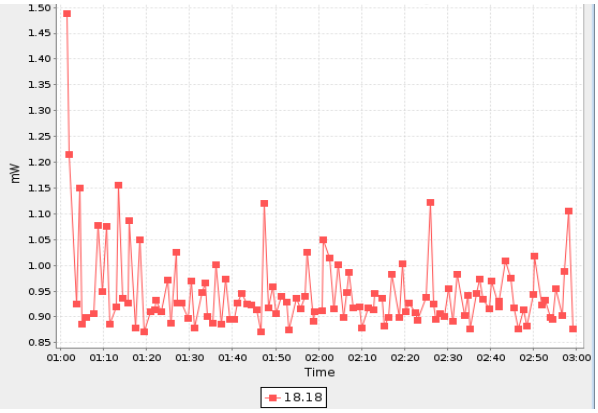
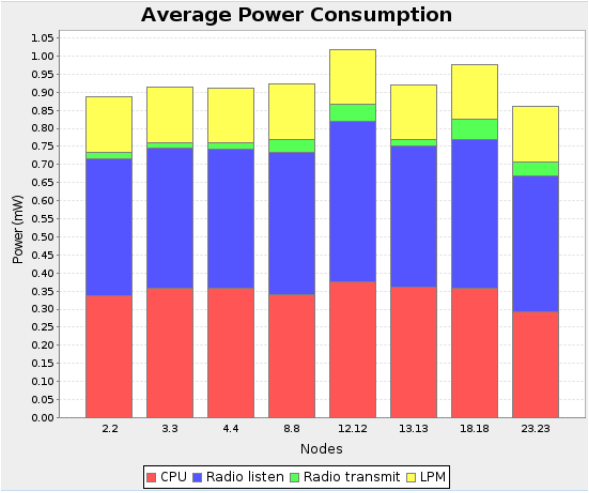


Figure 6.5. The repetition rate for the first 152 values of ϵ_i in ψ_i



(a) Historical Power Consumption for one node



(b) Average power consumption for 8 nodes in one partition

Figure 6.6. An experimental result for 8 Sky-mote nodes

Chapter 7

An EHR Platform based on DCCSOA

Electronic Health Record (EHR) systems collect and process sensitive patients' health data. In order to allows an EHR system to be deployable on heterogeneous cloud vendors and protect patients' data, we describe a novel EHR cloud-based platform.

The proposed platform is composed of two components: i) DCCSOA which is explained in Chapter 2 that allows cloud vendors to define a variety of services. ii) DPM which is described in Chapter 3 and it allows the proposed platform to protect data privacy of patients.

7.1 Introduction

Cloud computing provides a cost effective model through pay-per-use that allows each individual or healthcare business (Rodrigues 2009) to start a cloud based service with minimum investment. The cloud has several major issues which are described in Chapter 2. Let us describe these issue for EHR systems as follows.

7.1.1 Migration of EHR systems

Data and application migration is one of the major issues when users decide to transfer their data and applications from an IT department to a cloud computing system or from one cloud computing to another. Migration may cause several sub-issues, such as data security issue. For instance, a user who used a regular application based on a specific Application Programming Interface (*API*) could have some issue when the application transfers to a cloud computing system that needs to redefine or modify the security functions of the API in order to use the cloud. Each cloud computing system offer own services to

7.1.2 Data Security of EHR systems

Data security refers to accessibility of stored data to only authorized users, and network security refers to accessibility of transfer of data between two authorized users through

a network. Since cloud computing uses the Internet as part of its infrastructure, stored data on a cloud is vulnerable to both a breach in data and network security.

7.1.3 Data Privacy of EHR systems

Users have to outsource their data to an untrusted cloud vendor (e.g., public cloud vendors) in order to use the cloud computing benefits. In addition of data and network hack issues in cloud computing, data privacy could be violated by other users, malicious applications or even the cloud vendor when users share their data with a cloud vendor. Data privacy becomes one of the major challenges in outsourcing data to the cloud. Data encryption methods allow users to avoid exposing the original data to the cloud vendors. However, encryption for each single original data is not cost effective or feasible for some machines, such as mobile devices. For example, some mobile devices in EHR systems have limited resources, such as *CPU*, *RAM* and battery power.

7.2 Background

In Chapter 2, we proposed a dynamic cloud computing architecture based on Service-Oriented Architecture (*DCCSOA*). The architecture provides a new layer, *Template-as-a-Service (TaaS),* on the top of a cloud computing system that allows a cloud vendor to standardize its cloud services by defining *TaaS* services. *TaaS* is divided into two sub-layers: *front-end (FTaaS)* that allows different cloud vendors to define a generic and standard cloud service, and *back-end (BTaaS)* that allows a cloud vendor to bind a defined generic cloud service, *FTaaS*, to its cloud computing system. In other words, *DCCSOA* enables different cloud vendors to standardize their services through a uniform interface at *FTaaS* that allows users to transfer their data and applications from one vendor to another.

In this chapter, we use *DCCSOA* to provide a *template, TaaS,* for EHR system. A *template* allows an EHR system to use heterogeneous cloud computing systems. It provides flexibility, customizability and standardization for EHR services that needs to be run on the cloud computing.

As previously discussed, the data security and data privacy are two major issues in cloud computing system for EHR systems. We use a light-weight data privacy method (*DPM*) which is described in Chapter 3 that allows clients to scramble the original data on the client side before submitting to the cloud, and AES encryption method on the proposed platform. We evaluate the performance of implemented platform while clients use the methods. Our contribution in this chapter are as follows:

- Propose a platform for EHR system based on *DCCSOA*.

- Introduce an *eHealth template* for the proposed platform that provides a uniform interface for EHR systems to interact with heterogeneous cloud computing systems.
- Conduct an experiment through *DPM* and AES on the proposed platform to evaluate the performance and scalability of the proposed platform.

The rest of the chapter is organized as follows: In the next section, we introduce the proposed platform based on *DCCSOA* and the implementation of the proposed platform. We compare the behavior of *DPM* against AES on the proposed platform for a massive healthcare dataset in Section 7.5. We review related work in Section 7.6, and finally, we conclude our study in Section 7.7.

7.3 The proposed EHR platform

We consider *DCCSOA* as the main architecture for the proposed cloud platform. We define an *eHealth Template,* (T_{eH}), for EHR systems which is divided into the front-end, $FTaaS_{eH}$, and the back-end, $BTaaS_{eH}$.

$FTaaS_{eH}$ provides a generic and a uniform interface with standard services. $BTaaS_{eH}$ binds specific cloud value-added services to the uniform service interfaces at $FTaaS_{eH}$.

Figure 7.1 illustrates a general view of cloud stacks for the proposed platform. A client (end-user) accesses a generic and a uniform cloud service interfaces through an

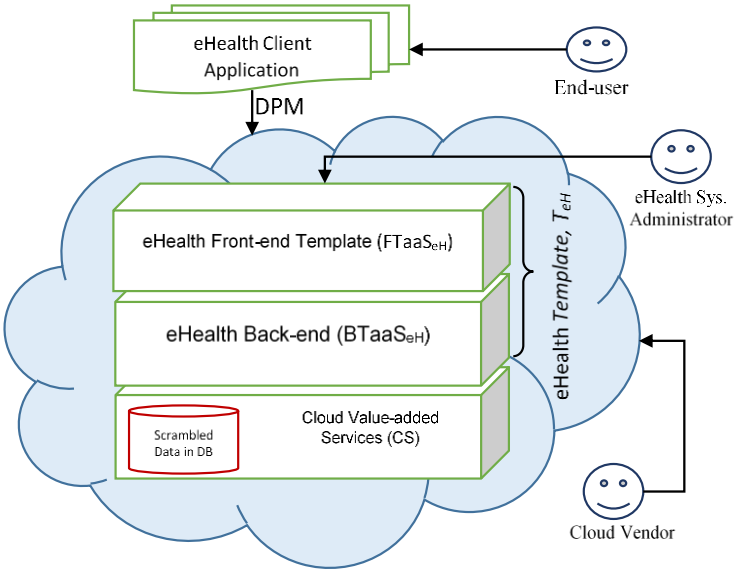


Figure 7.1. A view of EHR template with implementation of DPM and its connection to cloud value-added services

eHealth Client Application. The proposed platform can be simply transferred from a vendor V_1 to another V_2 by using the same $FTaaS_{eH}$ in another cloud but with different $BTaaS_{eH}$.

$FTaaS_{eH}$ is a dynamic layer, and it allows cloud vendors to customize their cloud services as *a template*. First, cloud vendors bind defined generic and uniform services at $FTaaS_{eH}$ to their value-added services through $BTaaS_{eH}$. As shown in Equation 1 each service at $FTaaS_{eH}$ must pass a satisfaction function $\mathcal{S}$ to propose a uniform service interface.

$$\exists\, s \in FTaaS_{eH} \mid \mathcal{S}at(s) \tag{1}$$

where s is a service at $FTaaS_{eH}$ and $\mathcal{S}at$ is a satisfaction function which is defined as follows:

$$Sat(s): \mathcal{R} \rightarrow \mathcal{O} \tag{2}$$

where $\mathcal{R}$ is a finite set of requirements of r, and $\mathcal{O}$ is a finite set of corresponding output for each requirement in $\mathcal{R}$.

The uniform service interface, UI, can be defined as follows:

$$UI(s) \rightarrow Sat(s_1)\text{^}Sat(s_1)\text{^}Sat(s_k) \tag{3}$$

Code 7.1 which is described also in Chapter 5 shows an example of how a client accesses *FTaaS* through a uniform data access layer with an abstraction on a cloud service (database access in this case). In this code, a client loads a web service, *FTaaS_Service_Ref*, for accessing services on the proposed platform. Then, the client requests a data access by calling *GetDataList* procedure from the web service, and finally, it retrieves the result on an object, *DataGridView*.

Code 7.1. Data Access at client side through FTaaS

```
FTaaS_Service_Ref.Service1Client FTS =new TaaS_Service_Ref.Service1Client();
DataSet ds = FTS.GetDataList();
DataGridView.DataSource = ds.Tables[0];
DataGridView.DataBind();
```

On one hand, defined services at *FTaaS* are dynamic, and the services can be customized by a cloud vendor to provide different type of services to the clients. Cloud

vendors bind services from *BTaaS* to their value-added cloud services that facilitates a service accessibility on heterogeneous cloud services for an EHR system. On the other hand, an EHR application, and its data can be transferred to another cloud vendor with minimal modifications at the client side. In addition, providing a generic and a uniform service is important for mobile health care devices because software modification for these devices can be expensive, and sometime requires hardware modifications.

7.4 Experimental Setup

We implemented the proposed platform through a case study based on a defined *template* for an EHR system. The proposed platform provides a generic data access at *FTaaS* to end-users for accessing to an *Electronic Medical Record (EMR)*. We implemented two methods on the proposed platform to protect patients' data privacy - one is a light-weight data privacy method (*DPM*) which is described in Chapter 3 and another method is AES encryption (Harrison 2008). These methods allow us to assess the performance of the proposed platform.

We consider the following scenario for the implementation of the proposed platform.

*"A client requests a data access to an Electronic Medical Record (EMR) which is implemented as a web service at FTaaS. FTaaS provides a generic, and a uniform function to the client. The request will be submitted from FTaaS to the BTaaS. Each retrieved response is processed through two user-data protection methods, DPM and AES encryption. BTaaS is implemented by Windows Communication Foundation (WCF) (*Resnick 2008*), and it is bounded to a SQL database. We ran different queries at this level, and uses data protection methods to evaluate the performance of the proposed platform. BTaaS' responses sent to the client at FTaaS by a web service."*

We implemented the proposed platform that includes an *eHealth template*. The *template* at the *FTaaS* enables end-users to interact with data access layer without considering the source of data. In the proposed platform is *FTaaS* and *BTaaS* are implemented as a web service, a *Windows Communication Foundation (WCF)* service, respectively. The services can be easily customized at *BTaaS* to adapt with heterogeneous cloud computing systems or traditional IT systems.

We used an Artificial Large Medical Dataset[35] as our *EMR* database that contains records of 100,000 *patients*, 361,760 *admissions*, 107,535,387 *lab observations*, and with the size of 12,494,912 KB (~12.2 GB). We ran 31 different queries on the largest table, *lab observations*. Each query retrieved different numbers of fields with different size. We ran *DPM* and *AES Encryption* at *BTaaS* to protect patients' data privacy on each retrieved

[35] http://www.emrbots.org retrieved on July 12, 2015

field. It allows us to assess the performance of the methods on the proposed platform by monitoring the computation time of the methods for each retrieved field from database.

The processed queries in this experiment are based on *Select Distinct Top* in *TSQL* language that retrieves data from 6 fields to 30,000 fields with the total queries' result size from 180 Byte to 911 Mbyte.

In this chapter, we are interested in evaluation of both *quantity parameters* and *quality parameters* in the proposed platform.

The *quantity parameters* include the following parameters:

Performance: We consider different workloads to evaluate the performance of a given method on the proposed platform and its performance when the size of workload is increased.

Scalability: A scalable service allows the service to provide the same performance when the number of transactions is increased.

The *quality parameters* include the following parameters:

Customization: The higher level of this parameter allows a cloud vendor to customize provided *services* with minimum modifications.

Independence of services: The higher level of this parameter allows the administrator to freely transfer an *EHR* system to another cloud vendor or bring it back to a traditional IT department with minimal service modifications.

Standardization of service: The higher level of this parameter allows an *EHR* system to interact with heterogeneous cloud services with minimal modifications.

7.5 Experimental Results

Figure 7.2 illustrates the experimental results for the evaluation of the quantity parameters on the proposed platform for an *EHR* system. We ran 31 different queries on the *EMR* database. Each submitted query from *FTaaS* is processed on the proposed platform to retrieve data from database at *BTaaS*. The platform is retrieved the response of each query and ran *DPM* and *AES* encryption on each retrieved field (result) from *BTaaS*. Figure 7.2.a shows the performance of the implemented methods on the proposed platform.

We expect that *DPM* provide a better performance over *AES* as described in (Harrison 2007) as well as on the proposed platform. Figure 7.2.a compares the performance of *DPM* and *AES* encryption on the proposed platform. This figure shows

that *DPM* provides a better performance over *AES* encryption for all query results as we expected.

Figure 7.2.b illustrates the performance of *DPM* and *AES* encryption for different size of an input string while the methods are not performed on the proposed platform. We considered each input string as a Unicode character with a size of 16 bits each. X-axis represents the size of input string, and Y-axis represents its response time (millisecond). In our experiment, we assumed that *DPM* does not need to generate a set of *PRP* by accessing to predefined arrays that described in Chapter 3.

Figures 7.2.a and 7.2.b show that the performance of processing of *DPM* and *AES* on the proposed platform (Figure 7.2.a) is not different from a single string (in Figure 7.2.b).

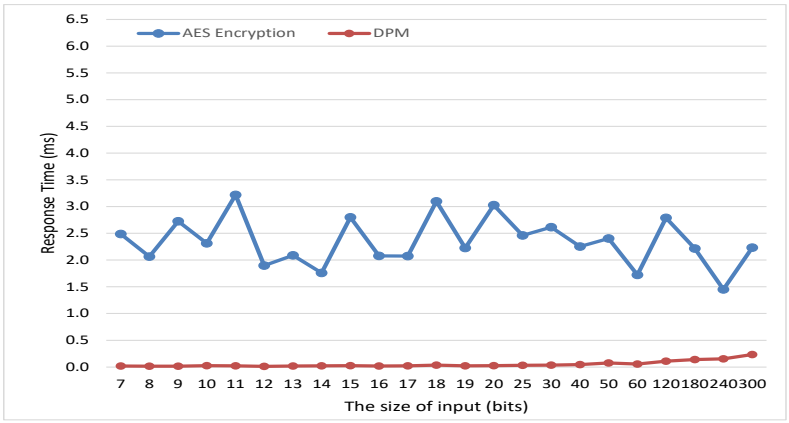
Another parameter which can be evaluated is quality parameters that includes *service independency* and a *service standardization*.

As described in Code 7.1, a client can access the platform by using the provided generic service. Since the service is independent of the cloud value-added services at the *BTaaS*, it allows users to interact with the cloud services without concerning about its requirements or type of output of a service. For instance, an application at client side in Scenario 1 retrieves data without understanding the type of database, and the location of the database. The service at *FTaaS* can be bind to any kind of services at *BTaaS*.

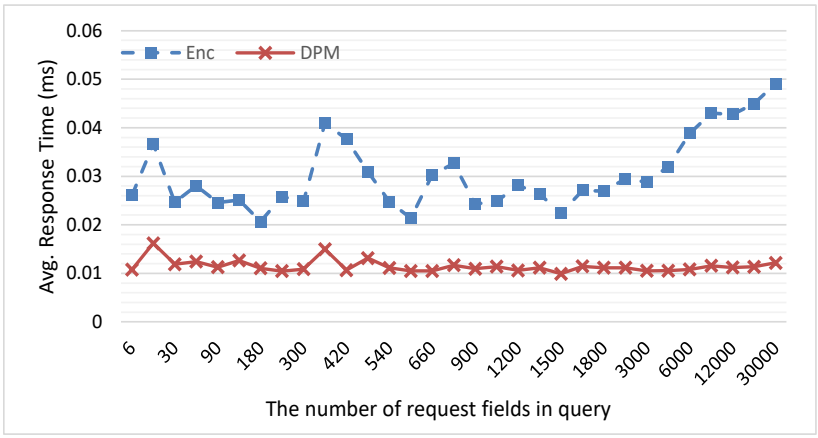
Different cloud vendors are able to define the similar services at *FTaaS* in Scenario I that allows an *EHR* system use different cloud standardized services.

7.6 Related Work

Several cloud-based services and platforms have been developed for *EHR* systems. For instance, (Fan et al. 2011) developed a platform which is used from capturing health care data for processing on the cloud computing. The platform relies on its architecture, and the authors did not describe how the proposed platform can be implemented for different architectures or how it can customize services for heterogeneous clouds. As discussed previously, a dynamic and a customizable cloud platform allows administrators to implement, and to transfer an EHR system to different cloud computing systems. There is also a vendor lock-in issue as described in Chapter 2, if a platform's services rely on a specific cloud architecture. In another study, (Lounis et al. 2012) developed a secure cloud architecture which is only focused on wireless sensor networks, and the study has limited work on the architecture. The study does not discuss the architecture features, such as service modifications or dynamic services. (Magableh et al. 2013) proposed a dynamic rule-based approach without considering the cloud environment. Finally, (Hoang et al. 2010) focus on mobile users features in their proposed



(a) a comparision between the performance of DPM and AES for a single Unicode string with different sizes



(b) a comparision between the performance of DPM and AES on the proposed platform

Figure 7.2. Experimental Results: a comparision between the performance of DPM and AES on the proposed platform

architecture, and the study does not discuss the overall of the architecture. In our study in this chapter, we proposed a dynamic platform for *EHR* system, and we showed how the proposed platform implements a dynamic service at *FTaaS*.

7.7 Summary of chapter

In this chapter, we proposed a dynamic cloud platform for an *EHR* system based on a cloud *SOA* architecture, *DCCSOA*. The proposed platform can be run on the top of heterogeneous cloud computing systems that allows a cloud vendor to customize and

standardize services with minimal modifications. The platform uses a *template* layer which is divided into *FTaaS* that allows cloud vendors to define a standard, generic, and uniform service, and *BTaaS* that allows defined services at *BTaaS* to bind to the cloud vendor value-added services. In addition, we implemented a data access scenario on the proposed platform with two different methods to evaluate its performance. The first method is a light-weight data privacy method (*DPM*), and the second is *AES* encryption method. The evaluation shows that the platform is scalable and the methods which are ran on the platform have not introduce additional overheads.

Chapter 8

Data Privacy Preservation for Cloud-based Databases

This chapter aims to use DPM which is described in Chapter 3, to provide users' data privacy in cloud-based databases. Although the proposed method can be deployed on traditional SQL databases, we focus only on NoSQL databases in this chapter.

8.1 Introduction

As we described in Chapter 3, most of the time, cloud vendors are not fully trusted by the users, and are vulnerable to users' data privacy violation by the cloud vendor. Users have several options to use the cloud. First, the users may employ a hybrid-cloud (Li et al. 2013) that allows them to outsource sensitive data to their private storage, and uses a public cloud for their non-sensitive data. This option may not be a practical solution due to the complexity of the system integration (Li et al. 2013) and network security issues. Another option is to encrypt user data before outsourcing it to an untrusted cloud vendor. However, most well-known encryption methods, such as *AES* (Daemen et al. 2013) are expensive because they increase computation time due to encryption/decryption of data during query processing. The third option is light-weight data security methods that secure data based on some conditions which are discussed in Section 8.2. In this chapter, we are interested in this option that allows users to protect their outsourced data with minimal computation overheads. The final option, is outsourcing data without considering users' data privacy.

Several studies (Denning et al. 1986, Popa et al. 2011, Osborn 2011 and Laur et al. 2013) have been conducted to secure a database with different encryption schemas. Although an encrypted database causes additional computation overheads to run queries, it enables users to protect their outsourced data, in particular sensitive information. In this chapter, we assume that users are willing to protect their outsourced database on an untrusted cloud vendor. We assume that the vendor must not be able to

access the database, and users may be able to access the database with minimal computation overheads.

Our primary contributions in this chapter are as follows:

- We propose an efficient light-weight schema that includes several components and algorithm, to securely outsource data to an untrusted cloud;
- We implement, and assess the performance of the proposed schema, and compares the performance of the light-weight data privacy method to a well-known encryption method, *AES* (Daemen et al. 2013);
- We analyze the security level of the proposed schema.

The rest of the chapter is organized as follows. The next section introduces some background related this study. Section 8.3, introduces the proposed schema, and its various components. Section 8.4 presents a security analysis of the proposed schema. The experimental setup of the implementation of the proposed schema, and the experimental results are discussed in Section 8.5, and Section 8.6, respectively. The related work is discussed in Section 8.7, and finally, Section 8.8 concludes this chapter.

8.2 Background

In Chapter 3, we proposed a light-weight data privacy method (*DPM*) that scrambles chunk of data based on a chaos system. The *DPM* uses the following equation in a chaos system that generates sets of distributed random numbers.

$$\psi_i: P_{k+1} = \mu P_k(1 - P_K) \qquad (1)$$

where $P \in \{0,1\}$ and μ are two initial parameters of this equation, and i is the index of each set of ψ.

In another words, ψ provides a set of numbers that does not allow an adversary who knows P_l to predict the future numbers, P_m where $m > l$. The content of each chunk (a set of bits or bytes) of an original data (input message) can be scrambled based on ith set of scrambled addresses in ψ_i which relocates the content of the original data. ψ_i generates repeated numbers, and *DPM* uses an algorithm to remove collision in addresses (see Chapter 3), and to cover all addresses of a given chunk of data.

The advantage of *DPM* is its time complexity. On one hand, a user scrambles a chunk of data with $O(1)$ time complexity, and on the other hand, an adversary needs $O(2^n)$

computation time to retrieve the original data from scrambled data when he does not know the initial parameters, where n is the size of each chunk.

8.2.1 Security parameters of DPM

DPM scramble the content of an original bit to avoid adding computation overhead, and it has the following two important security parameters.

I. The size of chunks

The size of each chunk, n is important to *DPM* to provide a sufficient level of security. For instance, *DPM* can be secured with $n > 120$ based on current computational capabilities. If an adversary runs an exhaustive search on the scrambled data, he needs to perform $O(2^{120})$ computational steps to retrieve the original data. In implementation work of the proposed schema which is described in Section 8.5, we consider each bit as an input that allows us to increases the size of n. If we consider a field of a record as an input, it could be small enough to retrieve the original data fast. We can combine multiple field(s) of a record as a chunk of the original data, and we can consider bits of the chunks as an input of *DPM* in order to increase the size of n. For instance, a Unicode character in *Microsoft SQL Server* has 2 *Bytes*, and for an adversary to perform an exhaustive search over a truly scrambled field (see the next parameter) with 20 characters' length requires $O(2^n)$ computation steps, where $n = 10\ chars * 2\ Bytes * 8\ bits$.

II. The number of repeated initial parameters

DPM needs to run with different initial parameters for each chunk of data (message) in order to be secure. The proof of this claim is given in Section 8.4.

We can generate different set of ψ for each original data but it adds additional computation overheads. We can precompute ψ offline, and store them on a database in order to eliminate online computation overheads. A detail of implementation of these parameters is discussed in the next section.

8.3 The proposed DPM-based Schema for cloud databases

The proposed schema stores scrambled data with minimal computation overheads in the database. The database is accessible only by the owner of the database, who has a key. In case of database compromise as whole, or access to database by authorize or unauthorized users without a key, the data on the database cannot be retrieved. The cloud vendors also cannot access the database because only the owner has the key that can reconstruct the scrambled data.

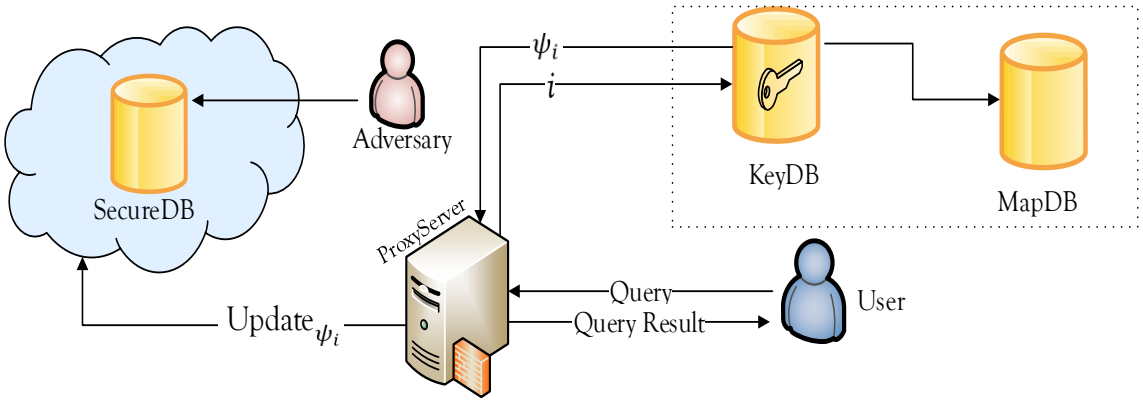


Figure 8.1. The proposed DPM-based Schema for cloud databases

The proposed schema for a cloud-based database is illustrated in Figure 8.1. Each submitted query from a user will go through the *proxy server* in order to scramble data prior to running the query operation (*insert, update or select*) on the database (*SecureDB*). The scrambled data is stored in *SecureDB*. The *proxy server* uses *MapDB* to access different *set of* ψ which is defined in Equation 1. We can remove *MapDB* by adding a ψ generator function that produces several sets of ψ. The *proxy server* uses *KeyDB* to store a user's keys for a record in *SecureDB*.

The main components of the schema are as follows.

SecureDB: This database stores scrambled data. Authorized and unauthorized users including cloud vendor administrators are not able to retrieve the original data from this database without knowing the keys that are stored in *KeyDB*. Only submitted transactions from *Proxy Server* which has access to *KeyDB,* is able to retrieve the original data. Even if this database is compromised on the cloud, an internal and an external adversary cannot retrieve the original data.

KeyDB: This database stores an index to ψ which is located in *MapDB*, for each record in *SecureDB*. This database is updated with an *insert/update* operation, and it is used for reconstructing a record of *SecureDB* by providing ψ for the corresponding record. The *KeyDB* can be used locally in order to protect *SecureDB* from an untrusted cloud vendor.

MapDB: This is an optional database that collects a set of predefined ψ in order to avoid adding runtime computation overhead for generating ψ with different initial parameters. For instance, Table 7.1 shows a definition of *Customer's* dataset with 5 fields, fields' types, and the size of each field (Bytes). If we consider the combination of all fields as an input

to the scramble process, we need 2,272 bits (284 Bytes) to be scrambled for this table. The join of all fields as an input increases computation time against an adversary to retrieve the original data from scrambled data. In this example, *MapDB* stores different shuffle addresses from the first bit to 2,272 by defining different initial values of μ and P_0 which is discussed previously in Equation 1. The *proxy server* uses one record of *MapDB* (shuffle addresses) to scramble and insert data to *SecureDB.* Then, the proxy server stores the record number of inserted data and its correspondence shuffle addresses from *MapDB* into *KeyDB,* that allows the *proxy server* to retrieve data later by using this information. *MapDB* can be used for several *SecureDBs*, on multiple clouds because this database can protect each *SecureDB* against an adversary from each cloud.

Table 8.1. The definition of a customer dataset

Customer Key	Name	Address	National Key	Phone
Integer	nchar(10)	nchar(64)	Integer	nchar(64)
4 Bytes	20 Bytes	128 Bytes	4 Bytes	128 Bytes

MapDB can be updated periodically in an offline mode (similar to database indexing) in order to remove online computation overheads. For instance, the database can be updated with adding sets of ψ based on the number of used ψs as a threshold parameter.

Proxy Server: This server allows a user to retrieve, update, or insert data to *SecureDB*. It runs *DPM* on each submitted user's query. Each user's operation, such as *Insert*, *Update* or *Select*, needs to be submitted to the *Proxy Server*. If a new record needs to be added to the database, *proxy server* assigns an index of a ψ to the record, and then, it scrambles the record based on the assigned ψ, and finally the index is stored in *KeyDB* for future record retrieval.

Algorithm 8.1, shows the insert procedure in the proposed schema that uses a user's key and the input record to insert data into *SecureDB*.

Algorithm 8.1. Insert procedure

1: i = ***NewKey*** *(Key)*

2: ψ_i =***Map(i)***

3: *NewScrambledRec* =$\boldsymbol{Sc_{\psi_i}}(input)$

4: *Rec#* = ***Insert(****NewScrambledRec****)***

5: ***UpdateKeyDB(****i, Table, Rec#****)***

First, the procedure stores an index of ψ in i, and it stores ith set of shuffle addresses from *MapDB* in ψ_i *(step 2)*. Second, it scrambles the user's input record (*step 3*), and it inserts the scrambled data into *SecureDB* (*step 4*), and stores the record number in *Rec#*. Finally, it updates *KeyDB* with i (the corresponding ψ of the record), the record number, and the name of table.

The *proxy server* uses the record number, and its corresponding ψ from *MapDB* to reconstruct a record when it needs to retrieve or update a record.

8.4 Security Analysis

A schema has a perfect secrecy, if it can pass the following conditions.

i) The adversary cannot learn about two scrambled records, r_i and r_j when he knows a scrambled data, $\mathfrak{s}$;
ii) The chaos system generator has perfect secrecy.

For the first condition, each record of a table of *SecureDB* in the proposed schema needs to be scrambled with different initial parameters, in order to avoid similarity between scrambled records as follows.

$$\exists i \in \psi_i \; and \; \exists k \in SecureDB \mid SC(\psi_i, r_k) = \mathfrak{s}_i \tag{2}$$

$$\exists j \in \psi_j \; and \; \exists l \in SecureDB \mid SC(\psi_j, r_l) = \mathfrak{s}_j \tag{3}$$

$$\forall i, j \in \psi \; such \; that \; \mathfrak{s}_i \neq \mathfrak{s}_j \; where \; i \neq j \tag{4}$$

where *SC* is the scramble function, ψ_i *and* ψ_j are two different sets of shuffle addresses, and r_k and r_l are two different kth and lth records of *SecureDB*.

$$\forall \mathrm{i}, \mathrm{s}: \Pr[SC(\psi_i, r_k) = \mathrm{s}] = \frac{\#r \in \mathrm{Z} \text{ such that } \mathrm{SC}(\psi_i, r_k) = \mathrm{s}}{|\mathrm{Z}|} \tag{5}$$

where r is a record in *SecureDB*.

In other words, the proposed schema uses different $\psi's$ which are defined with different initial parameters to prevent an adversary from learning about two original records by knowing their scrambled data.

■

For the second condition, ψ must provide a uniform distribution of addresses in ψ_i for all entries of n bits as follows:

$$P: U \to [0,1] \ \ such\ that \ \sum_{x \in U} P(x) = 1 \tag{6}$$

where $U = \{0,1\}^n$.

$$\forall\, x \in U: P(x) = \frac{1}{|U|} \tag{7}$$

In this case, the generator must produce different addresses with a uniform probability. As previously discussed in Section 8.2, the generator provides scrambled addresses in each ψ, which is stored in *MapDB*. *DPM* uses a set of shuffle addresses in ψ to scramble data. If *DPM* provides the same probability for each scrambled address in ψ, it must show the difference between the original addresses, and the scrambled addresses are not the same, and *DPM* must not show any relation between addresses. The Figure 8.2 illustrates a statistical model of the first 100 differences between the original addresses and the scrambled addresses in Equation 1 with the initial parameters of $P_0 = 0.999$, $\mu = 3.684$ for the length of 921 bits (n). I Figure 8.2, *X-axis* represents the address of the original bit and *Y-axis* represents the difference between the original address and the final address in the scrambled bits. The result shows that *DPM* scrambles data with a uniform distribution with different differences that does not allow an adversary to find a pattern between scrambled addresses.

■

In addition, more security analysis has been conducted against *DPM* which is discussed in evaluation section of Chapter 3.

As shown in Figure 8.2, there is no pattern between scrambled addresses that allows an adversary to predict the addresses. For instance, if an adversary knows the first bit moves to 13th bit when it is scrambled, still he cannot predict that the second bit moves to 48th address, or 3rd bit moves to 180th bit.

8.5 Experimental Setup

We conducted an experiment based on the proposed schema. We used *TPC-H* (Council 2008) which is a standard database benchmark with the scale of 1 GB. We ran different queries on *Customer* dataset. Each submitted query went through the *proxy server* that ran

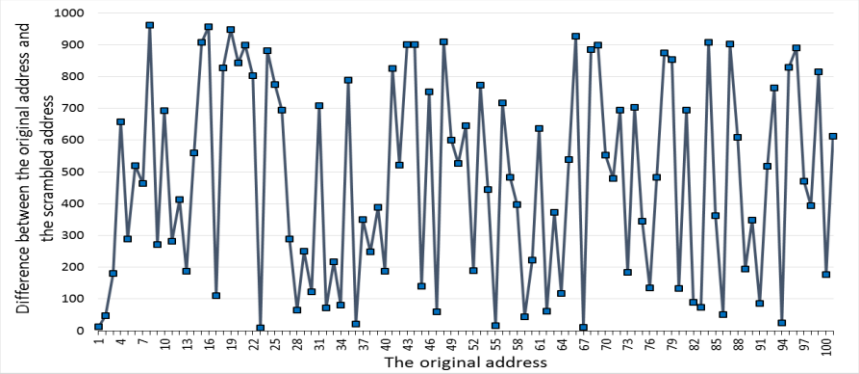


Figure 8.2. The difference between the original address and the scrambled address

DPM and *AES* encryption separately in order to compare the performance of both methods on the proposed schema. We use *ADO.Net* (Lerman 2010) at client side to retrieve and bind data from the database. *DPM* and *AES* encryption were implemented as a class (Lerman 2010) and written in C#.Net version 4.5, and executed on a PC with CPU Intel Core i7 with 8 GB RAM.

8.6 Experimental Results

Figures 8.3 and Figure 8.4 show the experimental results for the performance of the security methods (*AES* and *DPM*) on the proposed schema, and Figure 8.5 shows the data binding latency for different range of queries' responses.

In Figure 8.3, *X-axis* represents the number of the fields which were requested by a user's query, and *Y-axis* represents the total response time (millisecond) of *AES* encryption, and *DPM* on the proposed schema. Figure 8.3.a shows the total response time for 22 queries with a small query range from 9 fields to 9,000 fields with the increase rate

of 450 fields for each next query. Figure 8.3.b shows the total response time for 9 queries with a larger query range from 9 fields to 81,000 fields with the increase rate of 9,000 fields for the next query. As shown in these figures, *DPM* provides superior performance over *AES* encryption. In particular, the response time difference between *AES* and *DPM* increases for the larger queries. Figure 8.4 shows the response time difference between *AES* and *DPM* for the query range of 9 fields to 81,000 fields. In this figure, *X-axis* represents the number of requested fields for a given query, and *Y-axis* represents the performance difference between *AES* and *DPM*. For instance, as shown in this figure, *DPM* saves 2,909 milliseconds (~3 seconds) computation time for a database management system (*DBMS*) over *AES* for a query with a request of 54,000 fields.

In another evaluation, we considered data binding latency which assesses the response time of data binding (retrieving data from the query's results to the client objects*)*. Figure 8.5 shows a comparison of data binding latency with different range of queries. In this figure, the client's objects need additional computation to fetch data that causes an additional computation overhead for the first query. The results show that *DPM* not only provides better performance on computation time as described in Figure 8.3, it also provides an efficient computation time for data binding.

In addition, some studies on databases, such as *CryptDB* (Popa et al. 2011) show that queries can be executed over encrypted database without decryption. Our proposed method in this study can be used in *CryptDB* in order to reduce *AES* encryption overheads.

8.7 Related Works

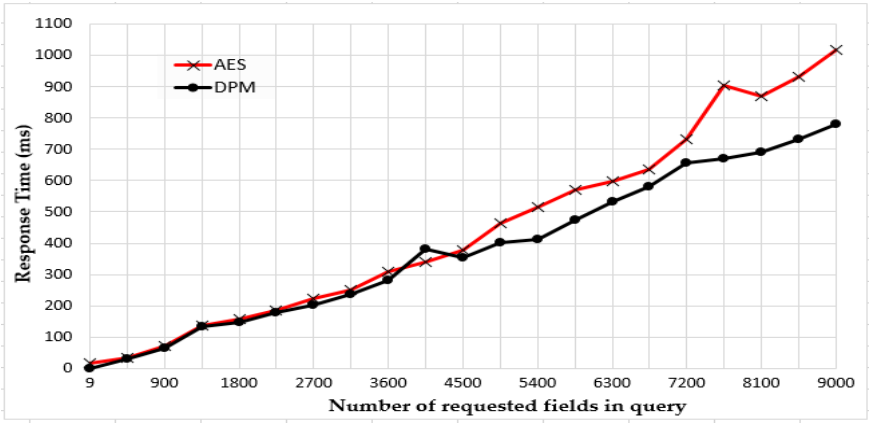
To the best of our knowledge, early a limited number of studies have been conducted on data privacy for cloud-based databases. Most of the studies consider encryption methods, or role-based data access methods on *DBMS* side, but any database security method that runs on a server side cannot protect users' data privacy.

In an early study, (Denning at al. 1986) proposed a theoretical multilevel database security which provides the basic idea of role-based access (*RBAC*) control in database. Later, (Jonscher et al. 1994) focus on the security of individual queries which cannot be implemented for all queries. (Osborn et al. 2001) developed an integration of systems where access control is represented by role graphs. The Osborn's security system needs several computation overheads that includes collecting the role of each user, the relation of roles based on a graph, the integration of the graphs, and an algorithm that needs to be run on all transactions. In addition, a graph-based algorithm needs heavy computation, which is not practical for large databases. One of the popular recent study is by (Popa et al. 2011) *CryptDB* which considers users' data privacy, but the database is

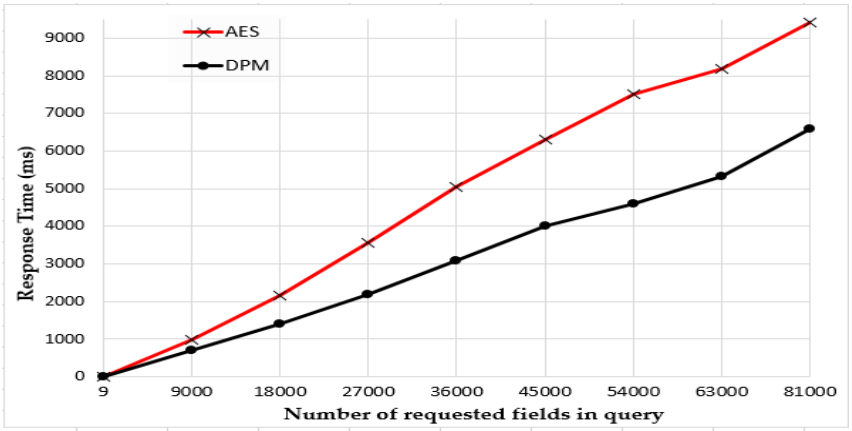
implemented based on *RSA* and *AES* encryption. *CryptDB* uses a proxy server to encrypt or decrypt each user's query. Database likes *CryptDB* can be extended by using *DPM* in order to remove additional computation overheads of *AES*.

8.8 Chapter summary

Users are facing several challenges when they must outsource their data to a cloud computing system. First challenge in cloud computing is data privacy because any entity from the cloud vendor's side can violate users' data privacy. Second challenge is data security because cloud computing is a form of the Internet-based services that need users to access their data through an untrusted and public network. A cloud-based database can be compromised by authorized cloud vendor users, or unauthorized users. In this chapter, we introduced a schema that consists of several components for cloud-based databases that protect users' data privacy. In the case of a compromised database, the data can be only accessible to users who have the key. Although the schema can be implemented by any encryption method, it uses a light-weight data privacy method (*DPM*) that allows users to efficiently protect each record inserted into the database. We conducted several experiments to evaluate the performance of the proposed schema while using *DPM* and *AES* encryption. The experimental results show that the proposed schema provides efficient response when *DPM* is employed. In addition, we analyze the security of *DPM* and the level of users' data protection.



(a)



(b)

Figure 8.3. A comparison between AES encryption and DPM on NoSQL databases

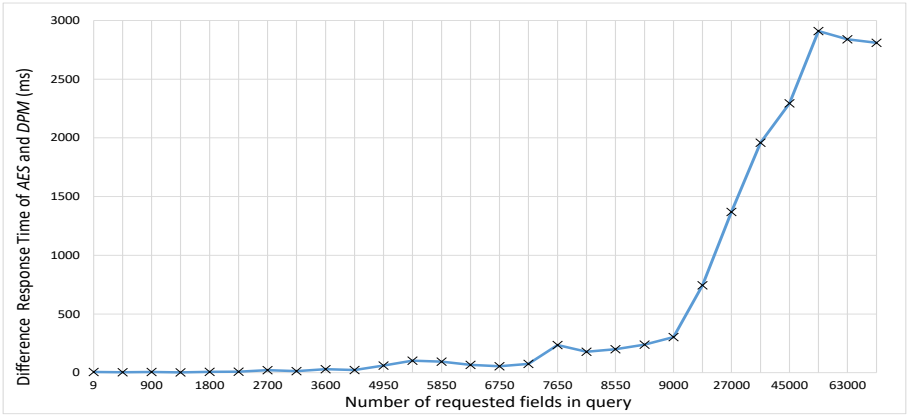


Figure 8.4. The response time difference between *AES* and *DPM*

Chapter 9

Summary and Conclusions

Cloud computing is a trending technology now. In order to use the advantages of the cloud, users need to outsource their data and applications to a cloud vendor which plays as a third-party. Outsourcing data to a third-party adds several challenges to the users, such as transferring data and application from one vendor to another, transferring data and application from a vendor to the in-house IT department, and users' data privacy. This thesis answers these questions.

Chapter 1 introduces cloud computing architecture, and its services. We explained different layers of a cloud computing system. We summarized a variety of cloud-based tools based on the service layer for handling big data.

In Chapter 2, we proposed a *Dynamic Cloud Computing Service-Oriented Architecture* (*DCCSOA*). In this chapter, we explained the most existing issues in the cloud, such as migration issue between different clouds and in house IT department, return back data from cloud to in-house IT, data and vendor lock-in issues, a lack of standardization and customization, and data privacy preservation.

The first goal of this study was to improve the architecture-level issues of the cloud. The second was to preserve users' data privacy, and the third goal deal with different use cases of the two improvements.

First, we introduced *DCCSOA* in Chapter 2 which yields a dynamic and customizable service layer (*DTSL*). The *DTSL* mitigates the architecture issues by defining a *template* which is divided into *front-end* (*FTaaS)* and *back-end* (*BTaaS*) layers. The defined *templates* can be customized by a cloud vendor for different groups of users. *DCCSOA* also allows different cloud vendors to offer unified cloud services through a *template*. Two cloud vendors are able to define the same *FTaaS* because each *template* at *FTaaS* is independent

of native cloud services. Each *template* maps *FTaaS* services to *BTaaS* and each *BTaaS* is bound to native cloud services.

We evaluated the *DCCSOA* based on a well-known Service-Oriented Architecture evaluation method. The result showed that *DCCSOA* provides several advantages over existing cloud architectures and platforms, such as minimal modifications requirement for providing standardization and customization.

As a future work, we plan to extend *DCCSOA* performance evaluation on *DTSL*, such as scalability of templates and data security protection at *DTSL*.

The second goal of this study was data privacy preservation in the cloud computing environment. Data privacy is one of the key challenges for cloud users because users must outsource their data to cloud in order to use the advantages of cloud computing. Outsourcing data to cloud computing or performing computation on the data raises a challenge for user that *how the cloud vendor is preserving users' data privacy against anyone from inside the cloud including the third-parties of cloud vendors*. In Chapter 3, we presented a light-weight Data Privacy Method (DPM) that makes an obstacle for an attacker from inside or outside the cloud to access users' data. The method is a light-weight that allows users to deploy it on a mobile device, such as a cellphone. DPM can be deployed on client-side as discussed in Chapter 3, as a server-side which is explained in Chapter 7, or as a middle box which is described in Chapter 8.

By considering *DCCSOA* as the host cloud architecture, *DPM* can be run at *BTaaS* that allows a cloud vendor to preserve users' data privacy.

We developed different scenarios for the proof of concept and for the third goal of this study, which are described as follows.

In order to efficiently and securely process *DPM* for preserving users' data privacy, we deployed a parallelization model of *DPM* which is explained in Chapter 4. We used *CUDA*, a *GPU-based* platform, which was introduced by *NVIDIA*. The parallel computing model of *DPM* allows a device to use one or multiple *GPUs* to perform heavy computations where each *GPU-core* consists of thousands of small and low speed cores. Each submitted task of *DPM* to a core is required small computation. In this chapter, we discussed both security level of *parallel DPM* and the performance of each *GPU function*.

Chapter 5 describes an extension of *DCCSOA* for cloud-assisted IoT devices. In Chapter 6, we presented a scheme that facilitates *IoT* devices to maintain their data privacy while each device outsources its data to mobile cloud computing by using *DPM*. We implemented the proposed scheme on an *IoT* simulation tool to investigate the power consumption and CPU computation overhead when the method is deployed on an IoT

device. The experimental results show that the proposed scheme does not introduce additional power consumption overhead.

In the future, we plan to extend the result of data privacy preservation of cloud-assisted IoT in Chapter 5 and Chapter 6, by implementing DPM on different IoT devices.

Chapter 7 describes an *eHealth template* at *BTaaS* for preserving data privacy in an electronic healthcare system on *DCCSOA*. The defined *template* allows different cloud vendors to define the same cloud environment for *Electronic Health Record (EHR)* systems. In addition, we performed an evaluation by deploying *DPM* on the proposed architecture.

We plan to extend the EHR platform by adding more capabilities, such as computing DPM on GPU for EHR systems.

Finally, *DPM* can be used as a proxy server (middle box) as explained in Chapter 8 to preserve users' data privacy when user wishes to outsource data to a cloud database.

As a future work for Chapter 8, we plan to extend the schema with a zero knowledge paradigm that allows users to run queries on scrambled dsata without reconstructing data from database. It will remove additional overheads on the database management system, and it will allow users to protect their data privacy efficiently.

Bibliography

Accorsi, Rafael. "Business process as a service: Chances for remote auditing."Computer Software and Applications Conference Workshops (COMPSACW), 2011 IEEE 35th Annual. IEEE, 2011.

Adibi, S., Nilmini Wickramasinghe, and C. Chan. "CCmH: The Cloud Computing Paradigm for Mobile Health (mHealth)" The International Journal of Soft Computing and Software Engineering, 3.3 (2013): 403-410.

Ayuso, Jesús, et al. "Optimization of Public Key Cryptography (RSA and ECC) for 16-bits Devices based on 6LoWPAN." 1st Int. Workshop on the Security of the Internet of Things, Tokyo, Japan. 2010.

Bahga, Arshdeep, and Vijay K. Madisetti, "Rapid Prototyping of Multitier Cloud-Based Services and Systems", Computer 46.11 (2013): 76-83.

Bahrami, Mehdi, and Singhal, Mukesh. "The role of cloud computing architecture in big data." Information granularity, big data, and computational intelligence. Springer International Publishing, 2015. 275-295.

Bahrami, Mehdi, and Singhal, Mukesh. "DCCSOA: A Dynamic Cloud Computing Service-Oriented Architecture." Information Reuse and Integration (IRI), 2015 IEEE International Conference on. IEEE, 2015.

Bahrami, Mehdi, Singhal, Mukesh, "A Light-Weight Permutation based Method for Data Privacy in Mobile Cloud Computing." Mobile Cloud Computing, Services, and Engineering (MobileCloud), 2015 3rd IEEE International Conference on. IEEE, 2015.

Bahrami, Mehdi, Singhal, Mukesh and Zixuan Zhuang, "A Cloud-based Web Crawler Architecture" in 2015 18th Int. Conf. Intelligence in Next Generation Networks: Innovations in Services, Networks and Clouds (ICIN 2015), Paris, France, IEEE, 2015.

Bahrami, Mehdi, Mukesh Singhal, "A dynamic cloud computing platform for eHealth systems." 2015 17th International Conference on E-health Networking, Application & Services (IEEE HealthCom). IEEE, 2015.

Bahrami, Mehdi. "An Evaluation of Security and Privacy Threats for Cloud-based Applications." Procedia Computer Science 62 (2015): 17-18.

Bahrami, Mehdi, Li, Dong, and Singhal, Mukesh, Kundu, Ashish "An Efficient Parallel Implementation of a Light-weight Data Privacy Method for Mobile Cloud Users" IEEE/ACM SC'16 – DataCloud Workshop, Utah IEEE, 2016.

Bahrami, Mehdi, Khan, Arshia, & Singhal, M. "An Energy Efficient Data Privacy Scheme for IoT Devices in Mobile Cloud Computing" (IEEE MS 2016), San Francisco, IEEE 2016.

Bahrami, Mehdi, and Mukesh Singhal. "CloudPDB: A light-weight data privacy schema for cloud-based databases." 2016 International Conference on Computing, Networking and Communications (ICNC). IEEE, 2016.

Bahrami, Mehdi, Singhal, Mukesh "A Dynamic Cloud Computing Architecture for Cloud-Assisted Internet-of-Things in the era of Big Data", Big Data and Computational Intelligence in Networking, Taylor & Francis LLC, CRC Press, 2016.

Bargiela, Andrzej, and Witold Pedrycz. Granular computing: an introduction. Springer, 2003.

Barry M. Leiner, et al. "A brief history of the internet", SIGCOMM Comput. Commun. Rev. 39, 5 (October 2009), 22-31, 2009.

Berner, Eta S. Clinical Decision Support Systems. Springer Science+ Business Media, LLC, 2007.

Bessis, Nik, et al. "The big picture, from grids and clouds to crowds: a data collective computational intelligence case proposal for managing disasters." P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), 2010 International Conference on. IEEE, 2010.

Bianco, Philip, Rick Kotermanski, and Paulo F. Merson. "Evaluating a service-oriented architecture", SEI, Carnegie Mellon University, 2007.

Bist, Meenakshi, Manoj Wariya, and Amit Agarwal. "Comparing delta, open stack and Xen Cloud Platforms: A survey on open source IaaS", Advance Computing Conference (IACC), 2013 IEEE 3rd International. IEEE, 2013.

Blum, Lenore, Manuel Blum, and Mike Shub. "A simple unpredictable pseudo-random number generator." SIAM Journal on computing 15.2 (1986): 364-383.

Bogdanov, Andrey, et al. PRESENT: An ultra-lightweight block cipher. Springer Berlin Heidelberg, 2007.

Bonomi, Flavio, et al. "Fog computing: A platform for internet of things and analytics." Big Data and Internet of Things: A Roadmap for Smart Environments. Springer International Publishing, 2014. 169-186.

Buscema, Massimo, et al. "Auto-Contractive Maps: an artificial adaptive system for data mining. An application to Alzheimer disease" Current Alzheimer Research 5.5 (2008): 481-498.

Candea, George, Stefan Bucur, and Cristian Zamfir. "Automated software testing as a service." Proceedings of the 1st ACM symposium on Cloud computing. ACM, 2010.

Cartier C, Paynetitle T (2001-07-30). "Optical Carrier levels (OCx)". Retrieved 01-24-2014.

Chakraborty, Debrup, and Palash Sarkar. "A new mode of encryption providing a tweakable strong pseudo-random permutation" Fast Software Encryption. Springer Berlin Heidelberg, 2006.

Chen, Yinong, Zhihui Du, and Marcos García-Acosta, "Robot as a service in cloud computing", Service Oriented System Engineering (SOSE), 2010 Fifth IEEE International Symposium on. IEEE, 2010.

Choo, Euijin, et al. "SRMT: A lightweight encryption scheme for secure real-time multimedia transmission." Multimedia and Ubiquitous Engineering, 2007. MUE'07. International Conference on. IEEE, 2007.

Council, Transaction Processing Performance. "TPC-H benchmark specification." Published at http://www. tcp. org/hspec. html (2008).

Cudré-Mauroux, Philippe, et al. "A demonstration of SciDB: a science-oriented DBMS." Proceedings of the VLDB Endowment 2.2 (2009): 1534-1537.

Curino, Carlo, et al. "Relational cloud: A database-as-a-service for the cloud", 2011.

Daemen, Joan, and Vincent Rijmen, "The design of Rijndael: AES-the advanced encryption standard", Springer, 2002.

Daemen, Joan, and Vincent Rijmen. The design of Rijndael: AES-the advanced encryption standard. Springer Science & Business Media, 2013.

Davenport T H, Dyche J (2013), Big Data in Big Companies, SAS

Denning, Dorothy E., et al. "Views for multilevel database security." Security and Privacy, 1986 IEEE Symposium on. IEEE, 1986.

Digital Compression and Coding of Continuous-Tone Still- Images Requirements and Guidelines, T.81, ITU retrived on 11/11/2014 at http://www.w3.org/Graphics/JPEG/itu-t81.pdf

Dinh, Hoang T., et al. "A survey of mobile cloud computing: architecture, applications, and approaches." Wireless communications and mobile computing 13.18 (2013): 1587-1611.

Doelitzscher, Frank, et al. "Private cloud for collaboration and e-Learning services: from IaaS to SaaS." Computing 91.1 (2011): 23-42.

Doraswamy, Naganand, and Dan Harkins. IPSec: the new security standard for the Internet, intranets, and virtual private networks. Prentice Hall Professional, 2003.

Dunkels, Adam, Bjorn Gronvall, and Thiemo Voigt. "*Contiki-a lightweight and flexible operating system for tiny networked sensors*" Local Computer Networks, 2004. 29th Annual IEEE International Conference on. IEEE, 2004

Fairhurst, Paul. "Big data and HR analytics." IES Perspectives on HR 2014 (2014): 7.

Fan, Lu, et al. "DACAR platform for eHealth services cloud." Cloud Computing (CLOUD), 2011 IEEE International Conference on. IEEE, 2011.

Foster, Ian, and Steven Tuecke. "Describing the elephant: The different faces of IT as service." Queue 3.6 (2005): 26-29.

Gewin, V. "The New Networking Nexus", Nature, vol.451, no.7181, pp. 1024-1025, 2008.

Gharajedaghi, Jamshid, "Systems thinking: Managing chaos and complexity: A platform for designing business architecture", Elsevier, 2011.

Grossman, Robert L., et al. "An overview of the open science data cloud" Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing. ACM, 2010.

Gubbi, Jayavardhana, et al. "Internet of Things (IoT): A vision, architectural elements, and future directions" Future Generation Computer Systems 29.7 (2013): 1645-1660.

Han, Zhang, et al. "A new image encryption algorithm based on chaos system." Robotics, intelligent systems and signal processing, 2003. Proceedings. 2003 IEEE international conference on. Vol. 2. IEEE, 2003.

Hanna, Margo. "Data mining in the e-learning domain" Campus-wide information systems 21.1 (2004): 29-34.

Harrison, Owen, and John Waldron, "AES encryption implementation and analysis on commodity graphics processing units", Springer Berlin Heidelberg, 2007.

Hoang, Doan B., and Lingfeng Chen. "Mobile cloud for assistive healthcare (MoCAsH)" Services Computing Conference (APSCC), 2010 IEEE Asia-Pacific. IEEE, 2010

Howe, Doug, et al. "Big data: The future of biocuration." Nature 455.7209 (2008): 47-50.

Hu, Bo, et al. "A CCRA Based Mass Customization Development for Cloud Services", Services Computing (SCC), IEEE International Conference on. 2013.

Huang, Song, Shucai Xiao, and Wu-chun Feng. "On the energy efficiency of graphics processing units for scientific computing." Parallel & Distributed Processing, IPDPS 2009. International Symposium on. IEEE, 2009.

Huang, Dijiang. "Mobile cloud computing" IEEE COMSOC Multimedia Communications Technical Committee (MMTC) E-Letter 6.10 (2011): 27-31.

Hunger, Jens. Business Intelligence as a Service. GRIN Verlag, 2010.

IDC Enterprise Panel, 3Q09, retrieved on October 13, 2013 at http://blogs.idc.com/ie/?p=730

Itani, Wassim, Ayman Kayssi, and Ali Chehab. "Privacy as a service: Privacy-aware data storage and processing in cloud computing architectures." Dependable, Autonomic and Secure Computing, 2009. DASC'09. Eighth IEEE International Conference on. IEEE, 2009.

Jacob, Adam "The Pathologies of Big Data", Communication of the ACM, Vol.52, No. 8, pp.36-44, 2009.

Jonscher, Dirk, and Klaus R. Dittrich. "An approach for building secure database federations." Proceedings of the 20th International Conference on Very Large Data Bases. Morgan Kaufmann Publishers Inc., 1994.

Josette Rigsby, Studies Confirm Big Data as Key Business Priority, Growth Driver, retrieved on Jan 21, 2014 at http://siliconangle.com/blog/2012/07/13/studies-confirm-big-data-as-key-business-priority-growth-driver

Juve, Gideon, E., Vahi, K., Mehta, G., Berriman, B., Berman, B. P., & Maechling, P. "Scientific workflow applications on Amazon EC2." E-Science Workshops, 2009 5th IEEE International Conference on. IEEE, 2009.

Katz, Jonathan; Lindell, Yehuda (2007). Introduction to Modern Cryptography: Principles and Protocols. Chapman and Hall/CRC.

Kaufman, Cynthia C. "Getting Past Capitalism: History, Vision, Hope", Rowman & Littlefield, 2012.

Kelly, Jeff "Big Data in the Aviation Industry", Wikibon, Sep 16, 203, retrieved on March 18, 2014 at: http://wikibon.org/wiki/v/Big_Data_in_the_Aviation_Industry

Killmann, W., Schindler, W.: AIS 31: Functionality Classes and Evaluation Methodology for True (Physical) Random Number Generators, version 3.1, Bundesamt für Sicherheit in der Informationstechnik (BSI), Bonn (2001)

Kocarev, Ljupčo. "Chaos-based cryptography: a brief overview." Circuits and Systems Magazine, IEEE 1.3 (2001): 6-21.

Kumar, Karthik, and Yung-Hsiang Lu. "Cloud computing for mobile users: Can offloading computation save energy?" Computer 43.4 (2010): 51-56.

Landau, Susan. "Highlights from Making Sense of Snowden, Part II: What's Significant in the NSA Revelations" Security & Privacy, IEEE 12.1 (2014): 62-64.

Laur, Sven, Riivo Talviste, and Jan Willemson. "From oblivious AES to efficient and secure database join in the multiparty setting." Applied Cryptography and Network Security. Springer Berlin Heidelberg, 2013.

Lerman, J. Programming Entity Framework: Building Data Centric Apps with the ADO. NET Entity Framework. " O'Reilly Media, 2010.

Li, Qinjian, et al. "Implementation and analysis of AES encryption on GPU." High Performance Computing and Communication & 2012 IEEE 9th International Conference on Embedded Software and Systems (HPCC-ICESS), 2012 IEEE 14th International Conference on. IEEE, 2012.

Li, Qing, et al. "Applications integration in a hybrid cloud computing environment: modelling and platform." Enterprise Information Systems 7.3 (2013): 237-271.

Lian, Shiguo, Jinsheng Sun, and Zhiquan Wang. "A novel image encryption scheme based-on JPEG encoding." Information Visualisation, 2004. IV 2004. Proceedings. Eighth International Conference on. IEEE, 2004.

Liu, Fang, et al. "NIST cloud computing reference architecture." NIST special publication 500 (2011): 292.

Lounis, Ahmed, et al. "Secure and scalable cloud-based architecture for e-health wireless sensor networks." Computer communications and networks (ICCCN), 2012 21st international conference on. IEEE, 2012.

Magableh, Basel, and Michela Bertolotto, "A Dynamic Rule-based Approach for Self-adaptive Map Personalisation Services", International Journal of Soft Computing and Software Engineering, vol.3. no.3, 104, March 2013.

Manavski, Svetlin. "CUDA compatible GPU as an efficient hardware accelerator for AES cryptography." Signal Processing and Communications, 2007. ICSPC 2007. IEEE 2007.

Manssen, Markus, Martin Weigel, and Alexander K. Hartmann. "Random number generators for massively parallel simulations on GPU." The European Physical Journal Special Topics 210.1 (2012): 53-71.

Manyika, James, et al. "Big data: The next frontier for innovation, competition, and productivity." (2011).

Marin, Leandro, Antonio Jara, and Antonio Skarmeta Gomez, "Shifting primes: Optimizing elliptic curve cryptography for 16-bit devices without hardware multiplier." Mathematical and Computer Modelling 58.5 (2013): 1155-1174.

Marx, Vivien. "Biology: The big challenges of big data." Nature 498.7453 (2013): 255-260.

Matheson, David, and James E. Matheson, "The Smart Organization: Creating Value through Strategic", Rand D. Harvard Business Press, 1998.

McAfee, Andrew, and Erik Brynjolfsson. "Big data: the management revolution." Harvard business review 90.10 (2012): 60-66.

McHugh, Mary L. "The chi-square test of independence." Biochemia Medica 23.2 (2013): 143-149.

Metsch, Thijs, and Andy Edmonds. "Open Cloud Computing Interface–Infrastructure", no. GFD-R in The Open Grid Forum Document Series, Open Cloud Computing Interface (OCCI) Working Group, Muncie (IN). 2010.

Moreno-Vozmediano, Rafael, et.al. "Key challenges in cloud computing: Enabling the future internet of services", Internet Computing, IEEE 17.4 (2013): 18-25, 2013.

Oikawa, Minoru, et al. "DS-CUDA: a middleware to use many GPUs in the cloud environment." High Performance Computing, Networking, Storage and Analysis (SCC), 2012 SC Companion:. IEEE, 2012.

Osborn, Sylvia. "Database security integration using role-based access control." Data and Application Security. Springer US, 2001.

Osvik, Dag Arne, et al. "Fast software AES encryption" Fast Software Encryption. Springer Berlin Heidelberg, 2010.

Pal, Subhankar, and Tirthankar Pal. "TSaaS—Customized telecom app hosting on cloud" Internet Multimedia Systems Architecture and Application (IMSAA), 2011 IEEE 5th International Conference on. IEEE, 2011.

Pedrycz, W., Granular Computing: Analysis and Design of Intelligent Systems, CRC Press/Francis Taylor, Boca Raton, 2013

Perrey, Randall, and Mark Lycett. "Service-oriented architecture." Applications and the Internet Workshops, 2003. Proceedings. 2003 Symposium on. IEEE, 2003.

Pine, B. Joseph., "Mass customization: the new frontier in business competition", Harvard Business Press, 1999.

Podesser, Martina, Hans-Peter Schmidt, and Andreas Uhl. "Selective bitplane encryption for secure transmission of image data in mobile environments." Proceedings of the 5th IEEE Nordic Signal Processing Symposium (NORSIG'02). 2002.

Popa, Raluca Ada, et al. "CryptDB: protecting confidentiality with encrypted query processing." Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles. ACM, 2011.

Ra, Moo-Ryong, Ramesh Govindan, and Antonio Ortega. "P3: Toward Privacy-Preserving Photo Sharing" NSDI. 2013.

Resnick, Steve, Richard Crane, and Chris Bowen, "Essential windows communication foundation: for .Net framework 3.5", Addison-Wesley Professional, 2008.

Ristenpart, Thomas, et al. "Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds" Proceedings of the 16th ACM conference on Computer and communications security. ACM, 2009

Rittinghouse, John W., and James F. Ransome. Cloud computing: implementation, management, and security. CRC press, 2009.

Rodrigues, Joel JPC, ed. "Health Information Systems: Concepts, Methodologies, Tools, and Applications", Vol. 1. IGI Global, 2009.

Rodrigues, Joel JPC, et al. "Analysis of the security and privacy requirements of cloud-based Electronic Health Records Systems" Journal of medical Internet research 15.8 (2013).

Rodrigues, Joel JPC, et al. "Distributed media-aware flow scheduling in cloud computing environment" Computer Communications 35.15 (2012): 1819-1827.

Sasikala, P. "Research challenges and potential green technological applications in cloud computing", International Journal of Cloud Computing 2.1 (2013).

Schonfeld, Erick, Google Processing 20,000 Terabytes A Day, And Growing, retrieved on Jan 21, 2014 at http://techcrunch.com/2008/01/09/google-processing-20000-terabytes-a-day-and-growing/

Shayan, J., Azarnik, A., et al.,"Identifying Benefits and risks associated with utilizing cloud computing", International Journal of Soft Computing and Software Engineering, Vol. 3, No. 3, pp. 416-421, 2013.

Shannon, C.E. "Communication Theory of Secrecy Systems", Bell System Tech. J., Vol. 28, 1949, pp. 656-715.

Shao, Fei, Zinan Chang, and Yi Zhang. "AES encryption algorithm based on the high performance computing of GPU." Communication Software and Networks, 2010. ICCSN'10. Second International Conference on. IEEE, 2010.

Siegel, Carolyn F. "Introducing marketing students to business intelligence using project-based learning on the world wide web." Journal of Marketing Education 22.2 (2000): 90-98.

Singhal, Mukesh, "A Client-centric Approach to Interoperable Clouds", International Journal of Soft Computing and Software Engineering, Vol. 3, No. 3, pp. 3-4, 2013.

Singhal, Mukesh, Santosh Chandrasekhar, Gail-Joon Ahn, Elisa Bertino, Ram Krishnan, Ravi Sandhu and Ge Tingjian, "Collaboration in Multi-Cloud Systems: Framework and Security Issues", IEEE Computer, Vol 46, No 2, February 2013, pp. 76-84.

Stanik, Alexander, Matthias Hovestadt, and Odej Kao. "Hardware as a Service (HaaS): The completion of the cloud stack." Computing Technology and Information Management (ICCM), 2012 8th International Conference on. Vol. 2. IEEE, 2012.

Stojanovski, Toni, and Ljupco Kocarev. "Chaos-based random number generators-part I: analysis [cryptography]." IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications 48.3 (2001): 281-288.

Suttisirikul, Kiatchumpol, and Putchong Uthayopas. "Accelerating the cloud backup using gpu based data deduplication." Parallel and Distributed Systems (ICPADS), 2012 IEEE 18th International Conference on. IEEE, 2012.

Tan, Wei, et al. "Social-Network-Sourced Big Data Analytics" Internet Computing, IEEE 17.5 (2013): 62-69.

Thomas, David Barrie, Lee Howes, and Wayne Luk. "A comparison of CPUs, GPUs, FPGAs, and massively parallel processor arrays for random number generation." Proceedings of the ACM/SIGDA international symposium on Field programmable gate arrays. ACM, 2009.

Truong, Hong-Linh, and Schahram Dustdar. "On analyzing and specifying concerns for data as a service." Services Computing Conference, 2009. APSCC 2009. IEEE Asia-Pacific. IEEE, 2009.

Tsai, Wei-Tek, Xin Sun, and Janaka Balasooriya, "Service-oriented cloud computing architecture", Information Technology: New Generations (ITNG), 2010 Seventh International Conference on. IEEE, 2010.

Tsai, Wei-Tek, et al. "SimSaaS: simulation software-as-a-service", Proceedings of the 44th Annual Simulation Symposium. Society for Computer Simulation International, 2011.

Tsoi, Kuen Hung, K. H. Leung, and Philip Heng Wai Leong. "Compact FPGA-based true and pseudo random number generators." Field-Programmable Custom Computing Machines, FCCM 2003. 11th Annual IEEE Symposium on. IEEE, 2003.

Wang, Wei, et al. "Accelerating fully homomorphic encryption using GPU." High Performance Extreme Computing (HPEC), 2012 IEEE Conference on. IEEE, 2012.

Wei-Tek Tsai, Wu Li, Hessam Sarjoughian, and Qihong Shao. 2011. SimSaaS: simulation software-as-a-service. In Proceedings of the 44th Annual Simulation Symposium (ANSS '11). Society for Computer Simulation International, San Diego, CA, USA, 77-86.

Wilson, Lori A. "Survey on Big Data gathers input from materials community" MRS Bulletin 38.09 (2013): 751-753.

Xu, Meng, et al. "Cloud computing boosts business intelligence of telecommunication industry." Cloud Computing. Springer Berlin Heidelberg, 2009. 224-231.

Ye, Guodong. "Image scrambling encryption algorithm of pixel bit based on chaos map." Pattern Recognition Letters 31.5 (2010): 347-354.

Yoshikawa, Masaya, and Hikaru Goto. "Security Verification Simulator for Fault Analysis Attacks", International Journal of Soft Computing and Software Engineering, vol.3, no.3, 71, March 2013.

Young, Mark “Automotive innovation: big data driving the changes”, retrieved Jan 26,2014 at http://www.thebigdatainsightgroup.com/site/article/automotive-innovation-big-data-driving-changes

Zhang, Liang-Jie, and Qun Zhou, "CCOA: Cloud computing open architecture", Web Services, ICWS 2009. IEEE International Conference on. IEEE, 2009.

Zhang, Tao, and Xianfeng Li. "Evaluating and analyzing the performance of RPL in contiki." Proc. of the first int. workshop on Mobile sensing, computing and communication. ACM, 2014.

Zibin Zheng; Jieming Zhu; Lyu, M.R., "Service-Generated Big Data and Big Data-as-a-Service: An Overview," Big Data (BigData Congress), 2013 IEEE International Congress on, vol., no., pp.403,410, June 27 2013-July 2 2013

Zorrilla, Marta, and Diego García-Saiz. "A service oriented architecture to provide data mining services for non-expert data miners." Decision Support Systems 55.1 (2013): 399-411.