5-2015

Data Integrity Verification in Cloud Computing

Katanosh Morovat
University of Arkansas, Fayetteville

Data Integrity Verification in Cloud Computing

Data Integrity Verification in Cloud Computing

A thesis submitted in partial fulfillment
Of the requirements for the degree of
Master of Science in Computer Science

by

Katanosh Morovat
Isfahan University of Technology
Bachelor of Applied Mathematics in Computer Science, 1992
Azad University
Master of Software Engineering, 1998

May 2015
University Of Arkansas

This thesis is approved for Recommendation to the Graduate Council.

_____
Dr. Brajendra Panda
Thesis Director

_____
Dr. Gordon Beavers
Committee Member

_____
Dr. Wing-Ning Li
Committee Member

Abstract

Cloud computing is an architecture model which provides computing and storage capacity as a service over the internet. Cloud computing should provide secure services for users and owners of data as well. Cloud computing services are a completely internet-based technology where data are stored and maintained in the data center of a cloud provider. Lack of appropriate control over the data might incur several security issues. As a result, some data stored in the cloud must be protected at all times. These types of data are called *sensitive data*. Sensitive data is defined as data that must be protected against unwarranted disclosure. Generally, almost all personal information might be considered sensitive data. This research paper outlines how data owners determine which data should be considered sensitive data, how data owners are able to keep their data to be secured and trustable, and how data owners are able to verify integrity of their data in cloud computing. Finally this research provides several analyses to show the effectiveness of the data integrity verification method.

Acknowledgments

First and foremost, I have to thank my research supervisor, Dr. Brajendra Panda. Without his assistance and dedicated involvement in every step throughout the process, this paper would have never been complited. I would like to thank you very much for your support and understanding. I would also like to show gratitude to my committee, including Dr. Wing-Ning Li and Dr. Merwin G. Beavers for their lectures and academic advice during my study. My thanks and appreciations also go to Dr. Craig Thompson in his lectures and academic advice during my study and financial support in more than a year of this degree.

Most importantly, none of this could have happened without my family. I express my gratitude to my parents, my sister, and my brother for their continuous support, encouragement and belief in me. Last but not least, I thank the University of Arkansas for accepting me and offering me an excellent Master course of studies. This thesis is dedicated to you, my parents, my sister, and my brother who have made me who I am. Thank you for friendship, time, leadership, guidance, care, and primarily, thank you for your trust.

Table of Contents

List of Figures

1. Introduction

Cloud computing is a distributed architecture model that centralizes remote server resources on a scalable platform in order to provide on-demand computing resources and services. Due to users' requirements, deployment models of cloud computing can differ greatly. Four deployment models have been identified, such as Private Cloud, Community Cloud, Public Cloud, and Hybrid Cloud [1]. Furthermore, depending on the number of objects that are involved in cloud computing, three architecture models have been recognized. The first model includes the owner of the data and cloud service provider. Owners of data, users or clients, and cloud service providers identify the second model. Finally, several owners of data, users or clients, and cloud service providers are objects in the last model. Generally cloud service providers prepare three types of services such as Software as a Service (SaaS), Platform as a Service (PaaS) and Infrastructure as a Service (IaaS) [2, 5].

Since cloud computing offers a mass data storage, it must provide secure services as well. In fact, cloud security mechanism is effective if correct defensive implementations come to picture. Appropriate cloud security architecture should identify the issues that will be generated by security management.

Security management is the identification of an organization's assets (including information assets) followed by the documentation, development, and implementation of policies and procedures for protecting these assets [43]. Security management determines these issues with security controls which determine any drawbacks in the system and try to alleviate the effect of an attack. There are various types of security controls such as Deterrent Control, Preventative Controls, Corrective Control, and Detective Control [46]. On the other hand, the popularity of cloud computing is mainly due to the fact that many applications and data are moving into cloud

platforms. Consequently, a lack of security is the major issue for data owners who keep their data on cloud providers' servers. The major security aspects in cloud computing are confidentiality, integrity, authentication, authorization, non-repudiation and availability [47].

This thesis addresses the data integrity verification problem in cloud computing. Generally, almost all information systems keep various pieces of personal information like social security or bank account information. Intuitively, these types of information motivate others to know about, or in the worst case change, another person's personal information. So the first and foremost challenge is the issue of security, and outsourcing can make it harder to maintain data integrity and privacy. In short, we must determine data items which are likely to be stolen as sensitive data and try to keep them safe from any unauthorized updates. This document takes a closer look at a method to verify integrity of this data. In order to follow this method the data owner must define various metadata (or information about business data) and keep them as a cipher text. The data owner stores all business (real) data as plaintext in cloud service provider's computers, and metadata as cipher text in local computers or cloud service provider's computers. This method empowers data owners to recognize the updated data in case that data has been altered by malicious users.

2. Background

Cloud computing applies to the delivery of computing resources over the Internet; this gives data owners the opportunity to store their information and use their applications at another location, rather than keeping their data on their own computers. By outsourcing, organizations can concentrate on their specific tasks, and they are able to cut their main expenditures. Hence, database outsourcing is an important manifestation in this regard [6]. Furthermore the cloud computing model allows individual users and businesses to access information and computer resources from any location where a network connection is available. Cloud computing provides a shared pool of services and resources such as data storage space, computer processing power, and user applications.

The following definition of cloud computing has been developed by the U.S. National Institute of Standards and Technology (NIST):

"Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction". [34]

This cloud model encompasses three service models, four deployment models, and three architecture models. The Cloud service models include [1]: Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS). Cloud deployment models are typically referred to as [7]: Private cloud, Community cloud, Public cloud, and Hybrid cloud. Lastly, cloud architecture models are addressed as an architecture which each of which has a cloud service provider, include those with one data owner, one data owner with the addition of users or clients, and several data owners with users or clients [8].

Most recently, cloud computing has begun to offer groups of remote servers; the remote servers allow centralized data storage and online access to stored data (IaaS), and the amount of data increases massively. It can even reach the petabyte or exabyte scale, which is known as big data. Big data describes any voluminous amount of structured, semi-structured or unstructured data that has the potential to be mined for information. The main concentrate of cloud computing is maximizing the size of the shared data storage. It is therefore important to choose a model for storing massive data (such as XML, relational data or combination of both [9, 35]). In our model, we prefer to use the relational structure because it provides the highest level of performance. Besides, the schema is not volatile [10] in our research. Relational databases are robustly able to handle large volumes of structured data in a system. Database Management Systems can efficiently and reliably maintain large amounts of structured data. This data may be updated through transactions that guarantee the integrity of the database; the data can be extracted very rapidly when totally and properly configured [11].

Nowadays, cloud services have become both interesting and ubiquitous because they may reduce the cost and complexity of operating computers and networks. While there are benefits, there are also privacy and security concerns too. Data is stored in remote locations under the cloud service providers' policies and controls. In addition, cloud service providers often serve multiple customers simultaneously. Consequently, all of this risks exposure of information by breaking into the computers, either accidentally or deliberately. Security for the most part consists of maintaining data integrity and data privacy. Several studies have been done to protect data confidentiality [12]. Much of this work has been done defining the issue of safeguarding data privacy [13, 16]. Integrity, on the other hand, is a major concern for the research community [14, 15, 16, 17]. By integrity, we

mean that query results returned by the service provider must be correct, complete, and includes the most up-to-date data.

Mykletun et al [18] provide mechanisms to ensure data integrity and authenticity in outsourced data. In this work they suggest a secured and practical schema that authenticates the query replies. They suggest a technique that gives the data owner the power to authenticate the origin and verify the integrity of data returned by cloud service provider in response to a query. The type of query that has been considered is *SELECT* clause. They did not address queries that involve any kind of data aggregation such as *SUM* or *COUNT*. One of important issues of data integrity they address is the completeness of the query, which refers to the correct execution of the query over the entire targeted data. Data integrity examinations can be provided at different levels of granularity. For example, it can be done at the level of a table, a row, a column, or even an individual level of attribute value. Those authors believe that the optimum level of integrity is the row level. In order to justify their belief, they use message authentication codes, as they tend to be small and efficient to compute and verify. They show the applicability of popular digital schemes (e.g. RSA and DSA), besides a proposed schema presented by Boneh et al. [19] recently. They use digital signatures, eventually public key signature, with data encryption to provide integrity and aggregation techniques.

One related field is Private Information Retrieval (PIR) [20, 21], which has been mentioned in the cryptographic literature. This work focuses on privatively retrieving parts of data stored at remote storage, which is protected against data leakage. PIR techniques support a searching method which is based on either the physical location [20] of the data or keywords [22].

Hacigaumaus et al. [23] identify the problem of data integrity in the outsourced data model by using data encryption in tandem with manipulation detection codes to provide integrity.

Xie et al [26] proposes an approach to detect the integrity of outsourced data. They insert a small number of records into an outsourced database to audit the integrity by analyzing the inserted records in the query result. To analyze the result, the data owner must know what records had been inserted into the outsourced database, so the owner must keep a copy of the set of those inserted records. In order to increase the performance, this approach defines a deterministic function to describe the inserted data. The data owner only needs to maintain the definition of the function, rather than all of the inserted records.

Mukkamala et al. [27] present a mechanism for maintaining the data integrity in a cloud database-as-a-service. This approach is based on inserting several fake tuples into the database. They define generating functions to create fake tuples with a uniform distribution. Malicious users are not able to distinguish between inserted fake tuples and real tuples. Because of its efficiency, this method stores real data as plaintext.

Song et al. [24] propose a practical schema to search encrypted data; it requires a single server and has low computational complexity. In general, searching encrypted data is becoming an increasingly popular research topic. (See, for example, [24, 25].) However, all current schemas only support exact match queries whereby the server returns data matching either a physical address or a keyword.

Hacigaumaus et al. [12] explores how SQL queries can be executed over encrypted data and provides details of query processing and optimization techniques. In this strategy, a huge part of the process can be run on the client service provider without having to decrypt data. Decryption and a small part of the query processing are performed at the client site. Specifically, they support *range* searches and *joins* in addition to exact match queries.

The Proof of Retrievability, proposed by Juels and Kaliski [28], is a method to verify the data stored by a user in remote storage in the cloud that is not modified by the cloud. POR defines sentinels for very massive files. The main idea of sentinels definition is the cloud needs to access only a small portion of the file instead of accessing the entire file. Sravan and Saxena [29] offered a Schematic view of a proof of retrievability based on inserting random sentinels in the data file.

A Provable data possession method checks that a file, including a collection of *n* blocks, is retained by a cloud server provider. The data owner generates some metadata to store an information file locally, and then the file is sent to the server while the owner deletes the original copy of the file. The owner verifies the file ownership by using the "challenge response protocol". This technique is used by clients to check the integrity of the data. In this method, the client will compute the hash value for the entire file and a key. Clients have a collection of keys and hash values. Whenever the client wants to check the file, it releases one key and sends it to the server. The server must recompute the hash value, compare with the received key, and then return the result to client [30].

Feifei et al [17] proposes a method for query authentication. Query authentication has three important dimensions: correctness, completeness, and freshness. Correctness means that the client must be able to verify that the returned records have not been modified in an invalid way. Authenticating a set of data values is done by using the Merkle hash tree. The Merkle hash tree is a binary tree, where each leaf node contains the hash value of a data, and each internal node contains the hash value of the concatenation of its two children [31, 32]. Verification of data values is based on the fact that the hash value of the root of the tree is authentically published. To prove the authenticity of any data value, all clients have to do to verify data is to iteratively compute all the appropriate hashes up the tree, finally checking if the hash which has been computed for the

root matches the authentically published value. Freshness guarantees that queries are applied against the most up-to-date data, instead of just some version of the data in the past.

Min Xie et al [33] provided a practical solution on how to add freshness guarantees to provide integrity assurance.

3. Data Integrity Verification

In cloud computing, computing resources are shared between users; users can access resources such as memory and storage by saving information through the internet. Therefore, information should be kept safe from malicious users.

Information security can be perceived as three functions: access control, secure communications, and protection of private data [42]. In our research we mention an aspect of information security which protects private data and monitors all unauthorized users' access and modification, and for the most part, we focus on maintaining data integrity.

Before transferring data to cloud servers, several security issues should be mentioned, namely confidentiality, integrity, and availability. As we mentioned before, three aspects to protect data integrity in a cloud computing environment include correctness, completeness and freshness [45].

- Correctness-- verifies that all rows in a query result are generated from the original real data without being tampered.
- Completeness:--verifies that all rows in a query result are generated from the original real data include all information that we expect.
- Freshness:--verifies that queries are executed over the up-to-date real data.

In this research, two services of cloud computing, such as infrastructure and platform services, are used. For the architecture model, three objects, namely data owner, users or clients, and cloud service provider, are defined. For the deployment model, the public cloud is defined. The main focus of our research is defining a method to address how to unify private data (here referred to as real data) and how to supervise those in order to detect mischievous data updating, which is one of the major information security aspects. This is called verifying data integrity. The aforementioned data is stored in the cloud service provider's servers. This text addresses the

required steps to provide integrity of data in the cloud computing field and defines a method to examine that data in case real data has been destroyed by malicious users.

3.1 Sensitive Data Identification

The advent of the Internet and new technologies like cloud computing bring us quicker, easier, and even anonymous access to more data than ever before. Thus, people should be more aware of how to make conscious decisions for protecting their data (here referred to as sensitive data).

Sensitive data is any data which comprised with confidentiality, integrity, and availability. Sensitive data might have material effects on the privacy to which individuals are entitled [4, 3]. Data sensitivity is directly proportional to the subject of a compromise of the data with respect to these criteria. If the data is sensitive, it's likely to be protected by laws, regulations, or policies. Sensitive data needs the maximum level of protection. There are two types of sensitive data [36].

- **Regulated sensitive data** includes data protected under federal or state regulations.
- **Unregulated sensitive data** includes data that is not legally controlled, but still considered sensitive.

Protection of sensitive data might be required for legal or ethical reasons, for issues belonging to personal privacy, or for ownership considerations.

Data owners who create data, maintain data, and use metadata to check integrity of data must classify their data according to sensitivity that they define for data during storing, accessing, and altering that data. This document addresses verifying data integrity to recognize any changes of real data that might be done by users who do not have permission to change data, (here referred to as unauthorized users). Unauthorized disclosure could not only have critically adverse effects on the data owners' reputation, but in addition, they are less likely to be trusted in the future. Data owners must ensure that data being maintained or accessed should be kept secure. One vital

question is raised: what types of data should be secured and what types should not? One rule of thumb tells us that all personally identifiable data should be considered sensitive data. Examples of sensitive data may include, but not limited to [44]:

- Social Security Number
- Income tax records
- Date of birth
- Financial Information
- Medical health data
- Place of birth
- Bank account numbers
- Credit card numbers
- Drivers' license numbers
- Mother's maiden name
- Personal address

Depending on the type of a business, data owners are able to determine the degree of data sensibility, and then they are able to categorize data into two groups: sensitive data and non-sensitive data. Afterwards, data owners must define policies in order to keep data secure. Non-sensitive data may be effectively considered public data. Public data disclosures to the general public posture little or no danger to data owners.

This document addresses sensitive data, which must be defined and generated by the data owner. Sensitive data brings with it a massive amount of concern for data owner. In order to store data, including real data, metadata, and control data, a relational database structure is proposed. Hence, data owners are required determine the fields and tables that store the sensitive data items.

For instance, suppose a data owner requires using a service from a cloud service provider to store the real data of a payroll system which keeps salaries, wages, bonuses and deductions for all employees working for the data owner's company. There is no doubt that the payroll system maintains personal information, which is required to be kept in a safe and secure way. Data in the payroll system is stored on several tables in a relational database structure; here we look at some of them.

Figure 1 represents a table, as follows, including the employee's personal information such as Sequence number, First name, Middle name, Last name, Social security number, Employee identification number, Date of birth, Address, and Phone number.

Field	Type	Mandatory	Description
Sequence number	Number	Y	Created by Database management system
First name	Text	Y	
Middle name	Text	N	
Last name	Text	Y	
Social security number	Number	Y	
Employee identification number	Number	Y	Created by Payroll system
Date of birth	Date	Y	
Address	Text	N	
Phone	Number	N	

Figure 1: Employee Personal Information table

Based on the explanation of how to define the sensitive data that previously has been provided, it seems to be rational to define the social security number as sensitive data. Hence for each field that is tagged as a sensitive data item, the data owner must define several types of information, known as metadata, and keep them in a safe storage location. Later in this document we will explain a table which is used for storing the sensitive data items' information, called data dictionary. The data owner should add one row into the data dictionary table for each sensitive data item, including the name of the table and its column name which keeps the secure data. For

this example, the data owner stores “Employee_personal_ information”, which is a name of a table, and “social security number”, which is name of the column and the other required information, which is explained in the next section.

Another table keeps the financial data for each employee, such as Sequence number, Salary (wages), Bonus, Taxes, Effective date, and Personal information, which refers to the employee information in the Employee_personal_information table. Let us call that table “Employee_financial_information”; its structure can be viewed as follows:

Field	Type	Mandatory	Description
Sequence number	Number	Y	Created by Database management system
Salary (wages)	Number	Y	
Bonus	Number	N	
Taxes	Number	Y	
Effective date	Number	Y	
Personal Information	Number	Y	Foreign key refers to Employee personal information table

Figure 2: Employee Financial Information

Like the previous table, the data owner must determine which columns contain secure data. For this table, the data owner may define the field “salary” as sensitive data, which also agrees with the sensitive data definition. The data owner should add one row into the data dictionary table, containing the name of the table, “Employee_financial_information,” and its column name, which holds the secure data (Salary), as well as the other required information. A table in cloud servers can hold as many sensitive data items as necessary.

Besides the real data, in order to check the integrity of data, the data owner is required to define several tables to store information about the real data, like the data dictionary table and several data control tables, each of which corresponds to a single sensitive data item. Generally, data control tables store the information about the real data partitioning, as well as its hash value, which is addressed in this document later.

3.2 Data Dictionary

A data dictionary is a table that is defined by data owners and stored in the data owners' computers. Once data owners determine the sensitive data items for their business data, they can create one and put some information or metadata corresponding to sensitive data items into the data dictionary table. As a result, the data dictionary table keeps some descriptive metadata for each sensitive data item. It might be used as reference to find all the sensitive data items in the system. Consequently, we should keep this information in a safe location; the best location might be the data owner's location (local computers). The data dictionary table saves the following information:

- ID: A sequenced number which is generated by database management system.
- Name of table (Table_Name): The name of a table in the business database, including the sensitive data item as a field. Each table can have several sensitive data items.
- Name of sensitive data item (Column_Name): The name of a field of a table in a database which keeps real data or business data. The data owner is responsible for deciding whether or not a given field of a table should be considered sensitive or not. Recognizing which data should be considered sensitive data is a policy that was discussed in the previous section.
- Data type of the column (Type): The type of data of the column which keeps sensitive data on the real database; it might be Number, Text, or Date. The type of data is used for partitioning. For each column, the type of data is one of several parameters that can be used in splitting data into several groups. Furthermore, the type of data helps the data owner to define an appropriate function to split data into several partitions or intervals. In order to verify the integrity of data, we prefer to split data into several groups, because computing

a hash value of a small chunk of data is much easier and faster than computing a hash value of large one.

- Minimum and maximum value (LBound, UBound): The smallest value and the greatest value that can be accepted in the column that keeps real data. Regarding the column's data type and its column's criteria, each column has a domain which shows the range of acceptable values. The data owner can recognize the maximum and minimum value by looking at the domain and the history of data and then define the lower bound and upper bound of the domain. Determining the minimum and maximum for each column that has been marked as sensitive data is vital for partitioning.
- Partitioning: In order to maintain the integrity of data, the data owner splits data into several groups; this process is called partitioning. A partition is a division of real data of one column (or its constituent real data) into distinct, independent parts. Data partitioning is normally done for manageability or increased performance; it helps reduce the total cost of data integrity checking.
- Function for defining the partition method (P_formula): This field stores a formula to compute partitions. There are two types of partitioning, equal width and equal depth. The former method divides the data into several intervals (k) of equal size. The width of intervals calculated as follows:

$$W = (\max - \min)/k \max \min$$

The latter method divides the data into several groups (k) which each group contains approximately same number of values. For this method, the general way of determining *k* is by looking at the data and its diversity and trying to define different intervals or groups.

One of the easiest ways is to count the entire data and split it into a number of groups (k). The depth of interval calculated as follows:

$$D = (\text{numbers of items})/k$$

Hence there are equal numbers of data in each group.

Due to diversity of data in a column, the data owner can choose one of two methods for making partitions, although for simplicity we use the equal width method in our research. Besides, the data owner is able to change the formula based on their requirements, and he can define different formulas for various columns as well.

- Number of partitions (K): The number of partitions. Again regarding diversity of data in a column and the number of data which put in a partition, the data owner can make a decision about the number of partitions and the size of a single partition. The number of partitions has a profound impact on both the performance and the degree of data integrity checking available.
- A hash function (H_formula): Might be one of the several well-known hash functions which will be chosen by the data owner. Using a one-way hash function as an encryption method is a wise choice. Although all real data is stored as plaintext in the data owner's computer, data items in each partition are encrypted using a hash function and might be stored in the cloud. By using H_formula we generate data, which is the concatenation of all data items in that partition, calculate a cryptographic hash function, and then store the generated cipher text into data control table, the structure of which will be explained later.
- Start position of substring (start_position): This field holds the start position of a substring in the original string. Due to increasing the performance costs of computing hash value for one partition, using part of data rather than data as a whole might be helpful. Viewing the

original data as a string, the substring could be a part of the original string or could be the same as the original string, depending on the data owner's preference. Using a substring in hash computation, we might lose some of the subtlety of data integrity verification, however.

- Length of substring (substring_length): This number represents the length of the substring that participates in the hash value computation. In fact, the smaller substring length leads to higher performance when creating cipher text for a partition. On the other hand by defining this policy, we might sacrifice accuracy in case of a data attack. Again depending on the data owner’s decision, the size of substring might be less than or equal to the size of the original string.
- Hash value (Hash_value): This stores the hash value of all of the data in one column (here referred as hash_partition) of the data control table, which corresponds to the column that kept sensitive data. We compute a hash value for each partition and store in data control table and then compute a hash value for all of the hashed data generated by last step. As part of data integrity verification, it is necessary to ensure that the data control tables are kept in all partitions and the corresponding hash values are stored in a secure way. In the data control section we will explain the structure of this table more. We should mention that if we store data control on the data owner’s computers, this field remains empty.
- Data integrity verification time (Data_Integrity_chk_type): This is defined by data owner and shows the frequency of checking data integrity during the data modification process.
- Cycle of the data integrity verification (Data_Integrity_chk): This is defined by the data owner and shows the frequency of data integrity verification.

- Rollback command interval time (Rollback_interval_time): This keeps the time of reading the command file and executes all commands over the real data.

The following table shows the structure of Data dictionary.

Field Name	Type	Opt	Description
ID	N	N	Primary key, sequence number. Generated by DBMS
Table_Name	chr(50)	N	Table name of the real data
Column_Name	chr(30)	N	Column that is stored under secure policy
Type	chr(20)	Y	Type of column that is kept in hash value
LBound	chr(8)	N	Lower bound value of domain of the corresponding column
UBound	chr(8)	N	Upper bound value of domain of the corresponding column
Partitioning	byte	N	Type of partitioning method – Equal-width(1), Equal-depth(2)
P_formula	chr(100)	N	Formula to make partition
K	N	N	Number of partitions
H_formula	chr(100)	N	Formula to make hash value of all records in control table
Start_position	N	N	Start position of substring in string
Substring_length	N	N	Substring length
Hash_value	chr(64)	N	Keeps hash value of entire data of a column in a data control table, if data control has been stored on cloud server
Data_Integrity_chk_type	chr(1)	N	Data integrity checking time. It could be daily(“D”), weekly(“W”), monthly(“M”), or yearly(“Y”)
Data_Integrity_chk	N	N	Cycle of data integrity checking
Rollback_interval_time	N	N	Time of roll back the commands

Figure 3: Data Dictionary Structure

In order to more clearly understand, let us suppose the data owner determines that the social security number column of the Employee_personal_information table and the salary column of the Employee_financial_information table are both sensitive data items. The data owner must then put two records into data dictionary table, each of which keeps information for each of sensitive data items. The following two records show information that the data owner stores into the data dictionary table.

Record no. 1

Field Name	Value
ID	1
Table_Name	Employee personal information
Column_Name	Social security number
Type	Number
LBound	0000000000
UBound	9999999999
Partitioning	1 (“Equal width”)
P_formula	(max – min)/k
K	9
H_formula	SHA512
Start_position	1
Substring_length	9
Hash_value	2fba5541fda58c643524cb629cb310674d029d7dd688e974f9c0d95299c228fa3531d06a29a69b6715ad4ec074d2bb50393fe5b4e7d2de5bc83b10ac7d3114ff
Data_Integrity_chk_type	“M”
Data_Integrity_chk	1
Rollback_interval_time	Hourly

Figure 4: A Record of Data Dictionary Table

Record no. 2

Field Name	Value
ID	2
Table_Name	Employee financial information
Column_Name	Salary
Type	Number
LBound	50,000
UBound	1,000,000
Partitioning	1 (“Equal width”)
P_formula	(max – min)/k
K	9
H_formula	SHA512
Start_position	1
Substring_length	6
Hash_value	8d969eef6ecad3c29a3a629280e686cf0fa3531d06a29a69b6715ad4ec074d2bb50393fe5b4e7d2c3f5d5a86aff3ca12020c923adc6c924cb629cb310674d02e
Hash_value	8d969eef6ecad3c29a3a629280e686cf0fa3531d06a29a69b6715ad4ec074d2bb50393fe5b4e7d2c3f5d5a86aff3ca12020c923adc6c924cb629cb310674d02e
Data_Integrity_chk_type	“W”
Data_Integrity_chk	1
Rolleback_interval_time	Hourly

Figure 5: A Record of Data Dictionary Table

3.3 Data Control

Data control tables are created by the data owner to maintain the integrity of real data, which has been stored in cloud service provider’s servers. Since the integrity of sensitive data is our concern, no matter whether we store them in the cloud service provider’s servers or the data owner’s computer, we should store each partition as cipher text in tables. The data owner defines several tables, which are known as data control tables, each of which is used for storing one sensitive data item. Data is added into the data control tables as long as the data owner creates and adds business data into tables on the business relational database. In order to put data into data control tables, the data owner splits all of the real data of a column into several partitions, and for each partition, computes a hash value of that partition, and then stores into the data control table,

which has been created to store the information of that sensitive data item. If we store data control tables on cloud servers, we need to generate new data, which is concatenation of all computed hash values in data control table, and then store that into the data dictionary table, which has been created in order to store the metadata of that sensitive data item. This means for each sensitive data item the data owner creates a data control table, so the number of sensitive data items is the same as the number of data control tables. The following figure shows the structure of a data control table.

Field Name	Type	Opt	Description
ID	N	N	Generated by DBMS
Hash_value	String	N	Hash value corresponding to the partition

Figure 6: Data Control Table Structure

A data control table keeps the following information such as:

- ID: A sequential number which is generated by database management system. It also presents the sequence number of partitions.
- Hash_value: The hash value of one partition, which consists of several real data items.

For more clarification, we should explain what a partition is and how we create one. There are two approaches for partitioning data, equal width and equal depth. Regarding the data of the column which has been defined as sensitive data, a data owner can choose one of the two methods and keep that in the data dictionary table, as well as the method that was used for partitioning. The first method divides the data into several intervals of equal size in each partition, and the second method divides the data into several partitions which each group contains approximately same number of values. For more simplicity in our research we only use the first method, equal width of data partitioning. In order to make partitions, the data owner looks at the range of the valid data that was defined in the criteria of the column, which is a field of a table in cloud servers and keeps real data, and then by using the definition of equal width method, partitions will be generated. The

ranges of data (minimum and maximum value) that are acceptable in a column of a table in cloud servers are stored in the data dictionary table; the data dictionary table keeps these values in two fields, LBound and UBound. As previously shown, the former holds the lower bound of the valid data and the latter holds the upper bound of the valid data for the corresponding column.

Let us look at the following example: Suppose our data ranges between 50 and 100, so a set of data might look like this: {50, 99, 70, 80, 90, 85, 88, 82, 95, 98, 55, 100}. We would like to split data in two groups; using equal width and by using the following formula: ((100 – 50)/2). We have two sets, the first set includes items like *x*, where *x* is between 50 and 75, 50 <= x < 75, so the set becomes: {50, 55, 70}. The second set includes items which are greater than or equal to 75 and less than or equal 100--that is: {80, 82, 85, 88, 90, 95, 98, 99, 100}; both sets have the same width. Using the equal depth method, however, the data is separated differently. Because there are 12 items in the main set, the partitions look like {50, 55, 70, 80, 82, 85}, and {88, 90, 95, 98, 99, 100}; both sets have the same number of data items—6.

The running time of both methods in creating partitions and checking the data integrity process is the same, although the running time of the equal depth method might be higher than the equal width method in the data modification process. Although database maintenance for keeping data integrity is the main concern of this document, performance plays an important role in our research. Consequently, for gaining better performance and making a balance between security and performance, we use the equal width method. Besides, we must combine the hash value of all the data in the column that holds the hash value of the data in a partition and then store the corresponding record in the data dictionary table.

Now let us look at an example to show how the data owner generates the data for a data control table. Suppose that the data owner is going to add information for the Salary column. First of all,

the data owner must create several partitions for this column. These partitions can be generated based on several parameters which have been defined in data dictionary such as Type of data, minimum and maximum value of data for Salary (LBound, UBound), number of partitions (K), method for partitioning (partitioning), and the function for computing partitions (P_formula).

While adding record into data control, the data owner must compute the hash value for each partition and edit the Hash value column.

It could be like the following table:

Id	Hash value	Description
1	c6d1b480d7e4b2888c1fa3e6f6ade6060621a83e89fb12809ae0055cac12bed1d1d7eaa33b7a12df1e42fce481125382376dd6ad52cf3c7eae9780ded1c22589	50,000 – 155,000
2	9e36e4f63b1dda97f9d466584d163d7523d8782901f052ca075120e5637beacac507647e7eb8991eb3873d640c4c54e5ff559d2f0920385d0af4e8d84595665c	156,000 – 206,000
3	735d535fd6198720702dbdbfc9ae34cab679f962bac4e0cada9d3e9cc0cbefa6047797438d1966abd6d85a5e32c65b1ef1c330bcbb19ef40ac8bcd5ac658c4a8	207,000 – 366,000
4	920590a60901e1f0e3b5d74a7cf70debb00f819f29796b77ef43d9f0ddf85033799e6806845748426797026ee1bba7526052788f4ffcd3251fd6139022401894	367,000 – 472,000
5	fb31d057badc6d1628ea5c04bddd7bf7136f99ed6a7df7267c0b7a099c98bb20e7b3bb3c3a13464fc393c23880392a3f981c59dee0f4ba085feac433dcbe30df	473,000 – 578,000
6	e92b9a98328f65a7d9cdabc43fb657181223e1aff703c421e9bc28133c794e10dc2f471dc1f49879d790e475007597e14d51f2dbd77d98d97fbb1447ee6a57b5	579,000 – 683,000
7	d8170dc56099aa94eff3a3a35132904139375ba6330554f58bc88d2176563e90f53760efb88b8f12f7d3e92787b3c7531c3956794c14efaeee54dcd0f501c371	684,000 – 788,000
8	48df8b265f003f4a19afbd64ee0daa0aa0759f9a06fd4e52ee20f7a8be6e044f7d8655908b62e9d5426fba693cdc5e5ea952cbe421e352d50782263846eb2b1e	789,000 – 895,000
9	232c6ae8694e6ab2a2ec742cfff61f1109f6da5d8cd8b552e8fabb5732d5a1d0f6a863e3a1b4d6505aacd3c8c8eed20e6a01b17742f5dafc587a30c4faefc7b9	896,000 – 1,000,000

Figure 7: Example of Data Control Table

For example the hash value for record with rows:

- Id=1 shows the result of concatenating of all data in the Salary which its value is greater than or equal to ‘50,000’ and less than ‘155,000’; and then computing the hash function

3.4 Cryptographic Hash Function

Cryptographic hash functions play an essential role in modern cryptography, while the conventional hash function is more commonly used in non-cryptographic applications. In both cases, larger domains are mapped to a smaller range, inferring that collisions might occur [41]. Hash functions take a message as input and produce an output referred to as hash-result, hash-value, or simply, a hash. A hash function h(m) is a message digest; in some sense, the message is condensed [37]. This is one hash property called compression. A hash function maps from a domain to a smaller range, typically many-to-one, like:

$h : X \rightarrow Y$, $|X| > |Y|$. This function should be easy to compute; this is the second property of hash function.

At a high level, a hash function might be divided into two classes: un-keyed hash functions, which accept a single input parameter, and keyed hash functions, which accept two distinct inputs, a message and a secret key. The hash function we address for creating hash values is an un-keyed hash function.

Generally hash functions are used to check data integrity. To improve performance, hash functions accept messages of any length as input and generate a fixed length output. A cryptographic hash function must have the following properties [38]:

- A hash function should be one-way. Having an output prepared by the hash function h, it might be computationally infeasible to find an input m which hashes to that output. This property is called pre-image resistance.

- A hash function should also be second pre-image resistant. Giving an input m_1, it might be computationally infeasible to find another input m_2 with $m_1 \neq m_2$ having $h(m_1) = h(m_2)$.
- A hash function should be strongly collision resistant. Having two different inputs $m_1 \neq m_2$, it might be computationally infeasible to have $h(m_1) = h(m_2)$. Consider the following hash function:

  h: {0,1}* → {0,1}n.

  Randomly generating k outputs, the probability of having a collision depends on n. To have an insignificant probability of collision, we should choose n to be sufficiently large so that the value of *k* will be infeasibly large.

A hash function is a cryptographic hash function if it is collision resistant; collision resistant hash functions can be built from collision resistant compression functions in using Merkle-Damgrad construction [39].

There are two widely used methods of cryptographic hash functions: the message digest (MD) family and the secure hash algorithm (SHA) [40]. Any version of MD like MD5 uses the Merkle-Damgard construction. SHA-1 is a NIST standard and it also uses Merkle-Damgard construction. Newer SHA's have been included in the standard such as: SHA-256, SHA-384, and SHA-512. In order to generate a hash value, we use SHA-512 cryptographic hash function in our research.

3.5 Data Generation

The first step of verifying data integrity is to generate data, including business real data and control data. In the rest of the document, we refer to this step as the initializing step. This step starts with determining which data items are sensitive, which we discussed previously. This part also mentions which data items should be defined as sensitive. After determining the sensitive data items, we should continue the process by adding one tuple into the data dictionary table. We

previously presented the structure of the data dictionary and the information that we keep in this table; this table is fundamentally important for checking integration of real data. Consequently, we should keep this table in the local machine to be ensured that malicious users do not have access to this table. In order to have better performance, we would prefer to keep information as plain text in that table.

After transferring all real business data to the cloud service provider and receiving confirmation that the data was inserted properly in the relational database in the cloud, the data owner must create data control tables. If those tables are stored in the data owner's servers, inserting real data into the tables in the relational database and inserting the information into the data control tables are done simultaneously, and then the real data will be transferred to the cloud servers. As we explained before, data control tables store information about sensitive data items. There is a one-to-one relation between sensitive data items and data control tables; it means each data control table keeps information about one sensitive data item. In fact, a data control table maintains information on each group of data stored in a column that keeps sensitive data. In the data control section we described how the data owner splits data into several partitions for each sensitive data item, computes the hash value for each partition, and then stores the data control table corresponding to that sensitive data item. We can either store all data control tables in the data owner's computers, or store them on another cloud service provider's servers. For both choices we can use the data dictionary table and real data, which have been scattered among several tables in a relational database, to create all data control tables. Afterwards, we should transfer the real data to cloud servers and eliminate all real data from local servers. If data control tables are stored on cloud servers, we should transfer data control tables to cloud servers and then delete them from the data owner's computer.

The algorithm is used to generate the required data has been defined in Algorithm1, which is presented later.

3.6 Data Modification

Generally, three commands that modify data in a table in a relational database are insertion, deletion, and update. Once clients perform one of the listed commands, the command is sent to data owner's computer. After doing some pre-processing in advance, the real data will be updated. If the command tries to update sensitive data items in real data, one or more data control tables must be updated too.

Data modification is a process that is responsible for updating data, including real data and control data. We split this process into two phases. First, the command is received, parsed, and used to update the real data. Then the data is verified, which includes real data and data control, and the data control tables (and the data dictionary table, if necessary) are updated.

The data modification process is performed when authorized clients run queries (also known as commands) to alter the real data in the table in the business relational database, which is located on cloud servers. If a client tries to update sensitive data stored in a business relational database on cloud servers, the data control table(s) must also be updated. In some circumstances the data dictionary table might need to be updated as well. If non-sensitive data is updated, the table in the business relational database on cloud servers will be updated and the data modification process will be done. During the data modification process and in case of updating the sensitive data items (here referred as the first phase), before updating a data control table and the data dictionary table, we should check data (both real data and data control) in order to verify that data is trustable and it doesn't consist of maligned information. Checking data is a part of the second phase of the data modification process.

In our model, we apply the command on the real data first. For better performance, we postpone the checking data phase, known as rollback command process, until the end of a day or a specific time (here referred as rollback command time interval). Using this process, we read the command, parse it, and then apply the command on the real data. Then at the specific time, we perform the rollback command process to ensure the data, including real data and control data, has been updated only by authorized user. During the data modification, the command is applied to the real data as long as the query parser process is parsing the received command and storing extracted information from the command into log files.

The query parser is a process that reads the command, tokenizes the command in order to extract several vital objects from the command, such as the type of the command (*INSERT, UPDATE, DELETE*), the name of the table and its columns which are expected to be updated, the column's value, and a conditional part (*WHERE* clause). If this query tries to alter data in the column of a table which is defined as sensitive data in the data dictionary, this process returns a list of several objects which have been mentioned before; otherwise it returns an empty list. If the list is not empty, it means the table and its columns being updated have been defined as sensitive in the data dictionary table.

While the command is run on the real data and updated the real data, we must save numerous data in two log tables, namely command table and command_detail table. The command table might include several fields as follows:

- Sequential number: Created by database management system. The primary key of the table.
- Command type: The type of command. The value might be like "1" representing *INSERTION*,"2" representing *DELETION*, and "3" representing *UPDATE*.
- Command: The command which has been requested by client.

- Table name: The name of the table which has been mentioned in the command.
- Date & Time: The date and time when the command was added to the log files.
- Status: The result of a roll backed command. When the command has been inserted to this table its value is “I” –“Inserted”. If the command is rolled back successfully, this value will be changed to “A” – “Applied”. If it cannot be done, this value will be changed to “R” – “Error”, and if the whole process is done successfully, this value will be “D” – “Done”

The following table shows the various commands:

Field Name	Type	Opt	
Sequential Number	N	N	
Command Type	String	N	
Command	String	N	
Table Name	String	N	
Date & Time	Date	N	
Status	Char	N	• “I” –“Inserted • “A” – “Applied” • “R” – “Error” • “D” – “Done”

Figure 8: Command Table Structure

The command_detail table, as follows, keeps the detail information of command:

- Sequential Number: Created by database management system and is the primary key of the table.
- Record number: The sequential number of a record in the table that keeps real data. This record is involved in the data modification process. This sequential number referees the command that was run on real data.
- Column name: The name of the column which has been mentioned in the command and keeps sensitive data.
- New value: The final value of the data item after the update.
- New Partition: The partition that includes the new value for the column of the specified table in the command.
- Old value: The final value of the data item before it was updated.
- Old Partition: The partition that includes the old value.
- Status: The status of this partition in the real data. The value could be “C” – “Correct”, or “E” – “Error”.

The following shows the structure of command_detail table:

Field Name	Type	Opt	
Sequential Number	N	N	
Record Number	String	N	
Column Name	String	N	
New Value	String	Y	
New Partition	N	N	
Old Value	String	Y	
New Partition	N	N	
Status	Char	N	• "C" –"Correct" • "A" – "Error"

Figure 9: Command Detail Table Structure

After parsing the command, the command will be applied to the real data. The data modification process continues to update the data control table in the event that the command tries to update at least one of the sensitive data items. In other words, if the query parser process returns a non-empty list, the data control table(s) corresponding to the column(s) of the table that keeps sensitive data must be updated. If we keep a hash value of that control table in the data dictionary table, we must update the data dictionary table also. Before updating the data control table(s), we have to ensure the validity of data, including real data and data control. We must check data because they might be altered by an unauthorized user. If we save the data control tables, we must guarantee the information of the data control table, which corresponds to the column containing sensitive data and involved in updating data process. Furthermore, we must confirm the validity of data in the real data table. The process of verifying real data is done by the update verification process; this process includes the rollback command and update control data process.

The rollback command process reads all commands saved in the log command table. It reads the last received command first and rolls back the data in the corresponding table in order to retrieve data in the table to an earlier state. This process will be done after reading all records in the command table whose status is “I”. This process may not be able to roll back a command if that data has been changed by an unauthorized user. In this case, the partition including tampered data will be recognized and its status in command_detail table will be changed to “E” (“Error”) which helps for the later data recovery process. After running the rollback command process, data will be in the state before applying the all commands of the last rollback command time interval.

After processing all commands in the command table, we should process the status of all partitions in the command_detail table. This process checks the status of each partition. This means that if two different commands alter the same partition and the status of the partition altered by one command is “C” and status of the same partition altered by the another command is “E”, we must change the status of the partition affected by both commands to “E”, because in the recovery process we restore the last backup, which has not been applied by any commands in the last update. Consequently we must change the status of the command in the command table to “R”.

The update control data process will start after the rollback command process completed its task. This process starts with reading command table if this process finds a command whose status is “R”. This means all partitions involved in updating real data by using this command will be a candidate for data recovery. On the other hand, knowing the partitions that are involved in the rollback command process, the update control data process computes the hash value of the partition of the retrieved data and compares it with the corresponding hash value kept in the data control table. If both hash values are the same, it shows that the data is trustable, so we compute hash value of the partition, including recently updated real data, and replace it with a corresponding hash

value in the data control table. Then we should change the status of the command in command table to “D”. Otherwise we cannot trust the real data and we must recover that partition.

During the recovery process we must restore that partition of data from the last backup and apply all the commands stored in the log command table related to the last rollback command time interval.

In the case of updating real data and the corresponding data control table, this process completes the following task. For simplifying, based on the command type (insertion, deletion, and update) we show three examples that demonstrate the output list generated by the query parser, the rollback command process, and the updating of the data control table.

Suppose that a client sends a command to insert a record into “employe_ personal _information” table. The command could be structured like the following: “INSERT INTO employee_personal_information (Emp_ssn, First_name, Last_name, Emp_id, ‘Birth_date’) VALUES (‘999999999’, ’John’, ‘Smith’, ‘202020’, ‘01012000’)”. First the query parser reads the command, tokenizes it, and then stores the extracted information into command table and command_detail table like as follows:

Sequence number	Command type	Command type	Table name	Date & Time	Status
1	INSERT INTO employee_personal_information (Emp_ssn, First_name, Last_name, Emp_id, 'Birth_date') VALUES ('999999999', 'John', 'Smith', '202020', '01012000')	1	employee_ personal_information	03/01/2015 : 10:00	I

Figure 10: Command Table Example

Sequence number	Rec #	Field name	New field value	Old field value	New partition #	Old partition #	Reference to Command table	Status
1	20	Emp_ssn	999999999		9		1	

Figure 11: Command_detail table Example

If several columns which hold sensitive data have been mentioned by the command, the command_detail must have a record for each of them. In this example, the client tries to update a table which holds a sensitive data item, so besides updating that table, the data control table which keeps the information of "Emp_ssn" column must also be updated as well. Insert commands may address columns either explicitly or implicitly. Let's look at a general insertion command. If columns appear in an insertion command, the command addresses columns explicitly. In contrast, an insertion command may prepare without explicitly showing columns. In the latter case, regarding the values that have been appeared in *VALUES* clause, corresponding columns can be recognized, and then they will get new values. If the command addresses several columns which all or some of them keep sensitive data, several corresponding data control tables must be updated also.

Next we should compute the partition. For computing the partition which is updated by the insertion command, we should use LBound, UBound, Cell#, and the partitioning method (Equal-

width) for that table. This information has been stored in the data dictionary table. Determining the partition is possible, and it will be done using the following formula:

Partition_group = UBound-LBound/ Cell#,

Partition# = new data/Partition_group. The last formula returns the ID of the partition.

In case of a delete command, suppose that a client sends a command in order to delete a record from "employe_ personal _information" table. The command could be structured as follows: "DELETE FROM employee_personal_information WHERE Last_name LIKE 'Smith'". Like the previous example, the query parser reads the command, tokenizes it, and then stores the extracted information into a command table and command_detail table like as follows:

Sequence number	Command type	Command type	Table name	Date & Time	Status
2	DELETE FROM employee_personal_information WHERE Last_name LIKE 'Smith'	3	employee_ personal_in formation	03/01/2015 : 11:30	I

Figure 12: Command Table Example

Sequence number	Rec #	Field name	New field value	Old field value	New partition #	Old partition #	Reference to Command table	Status
2	10	Emp _ssn		909090909	9	9	2	

Figure 13: Command_detail Table Example

If we look at the general statement of a delete command, we notice that the list of columns has not been presented explicitly. By using the data dictionary table, determining which columns which hold sensitive data is possible. In this example, the "employee_personal_information" table has several columns, but only one of them is defined in the data dictionary table ("Social security number"). For running this command, by searching in the "employee_personal_information" table we have the value of the field "Social security number", and by using the formula shown in the

insertion example, the partition ID will be determined. It is obvious if several columns, which keep sensitive data, have been mentioned by the command or the delete command removes several records from the table kept real data, the command_detail must have several records.

In case of an update command, suppose that a client sends a command in order to update record(s) from "employe_ personal _information" table. The command could be structured as follows: "UPDATE employee_personal_information SET Emp_ssn='808080808' WHERE Emp_id = '101010'". Like the previous example, the query parser reads the command, tokenizes it, and then stores the extracted information into a command table and command_detail table, as follows:

Sequence number	Command type	Command type	Table name	Date & Time	Status
3	UPDATE employee_personal_information SET Emp_ssn='808080808' WHERE Emp_id = '101010'	2	employee_ personal_in formation	03/01/2015 : 11:45	I

Figure 14: Command Table Example

Sequence number	Rec #	Field name	New field value	New partition#	Old field value	Old partition #	Reference to Command table	Status
3	15	Emp _ssn	808080808	9	508080802	5	2	

Figure 15: Command_detail Table Example

At the end of the rollback command time interval, we require updating data control table(s), and we will probably update the data dictionary table as well. The rollback command process is responsible for that task. This process reads the command table from the last update until the first one which occurred during the last rollback command time interval. Let us look at the functionality of this process for three different types of command.

In case of insertion command, by using the data of “Table name”, “Record#”, and “Partition#” in the command and command_detail table, the rollback command process tries to find that record in the table, which has been updated by the insertion command. If this process finds the record, deletes the record, and then extracts all records included in that partition and saves into a temporary storage. This process changes the status of the current command to “A”. This means the current command has been rolled back successfully. If the process does not find that record in the table, it shows this record has been removed by an unauthorized user and this process cannot delete the record. So this process changes the status of this record in the command table to ‘R’. This means that partition has been candidate for later recovery.

Similarly, in case of a deletion command, by using the data of “Table name” in the command table and one or several “Record#” and “Partition#” in the command_detail table, the rollback command process tries to find record(s) in the table that have been deleted by the deletion command. If this process does not find the record(s), this record is inserted into the table using the command and command_detail tables. Then all records included in the specified partition are extracted and saved into temporary storage. This process changes the status of the command to “A”. This means this command has been rolled back successfully. Any other way, it shows that the record has been inserted by an unauthorized user. This process changes the status of this record in the command table to ‘R’ in such circumstances. This means that this partition is a candidate for later recovery.

In the case of an update command, like previously, this process tries to find all records that exist in the command_detail table corresponding to this update command. If each one exists in the table that was mentioned in the update command, and each column carries the value equals to value, which has been given in the update command as a new value, the process updates the

specified record in the reverse direction. Then the process extracts all records included in the specified partition and saves them into temporary storage. The process then changes the status of the command to “A”. Any other way it shows the record has been updated by an unauthorized user and this process cannot finish this task successfully. This process changes the status of this record in the command table to ‘R’ in this case. This means this partition is a candidate for later recovery.

The rollback command process will be finished by checking all records from the command_detail table. This process makes a decision about the status of a partition based on the policy that was mentioned previously. After termination of the rollback command process, we require the related data control table(s) to be updated. Before updating data control table, however, we need to confirm that the real data is trustable. To do so, we must read all records with no “R” status from command_detail table, and compute a hash value of that partition from the temporary storage and compare it to the corresponding hash value stored in the data control table. If both hash values are the same, we change the status of the partition to “D”, indicating that the process completed successfully. Then we compute hash value of that partition from the permanent table and replace it with the corresponding partition in the data control table. For the rest of the records in command_detail whose status is not equal to ”D”, we must restore the real data related to the damaged partition from the last backup, apply those commands whose status is not equal to “D”, and then compute the hash value of the partition and update the corresponding record in data control table.

The algorithm for this process is provided in Algorithm 2 below.

3.7 Integrity Verification

The data stored in the cloud service provider's servers should be checked from time to time**.** If any unauthorized users changed the data in the database, the data integrity verification process can be used to inform the data owner about the spoiled data and try to recover it.

The frequency of the data integrity check is entered in the data dictionary table by the data owner. According to the sensitivity of the data and the rate of updating data, the frequency will be determined from daily to yearly. Besides the time period, the data owner should create a time scheduler for each sensitive data item and add the time of checking the integrity of data into this table. This scheduler represents the date of data integrity check for each sensitive data item and the result of performing the check as well. This table can be viewed as a history of the data integrity verification system, and the results of process are also reflected in the table.

Verification of the real data is done in several steps. This process starts with finding all tables and their columns which must be checked at the time. After searching in the time scheduler table, the data owner can extract the table(s) which must be investigated at the time. Afterwards, by using the data dictionary table, the data owner is able to determine the column(s) of that table(s). The process of checking data integrity must be continued for each column. Using the data dictionary table, the data owner must get the information corresponding to that table and its columns mentioned in the data integrity scheduler. So the data owner requires extracting information from the data dictionary in order to create partitions and compute the hash value. If the data owner keeps data control tables in the local servers, data in data control tables are trusted. Otherwise, we need to be sure about the quality of information in data control tables. In order to achieve this goal, we can compare the hash value stored in the data dictionary table with the concatenation of all hash vales stored in the data control table. If they match, we can trust data in data control table. If the

data control table has not been tampered with, the process will be continued by computing a hash value for each partition of the real data and comparing it with the corresponding hash value stored into the data control table. This means by using the data dictionary table and by knowing the number of partitions, and lower bound and upper bound of the entire data in the column, the data owner must compute all partitions from data, compute the hash value of real data for each partition, and compare it to the corresponding partition's hash value saved in the data control table. If any difference is found, the data owner can determine the group or set of records that have been tempered with. To fix this issue, the data owner must restore the backup of the real data previously stored by cloud service provider. Using that backup, the data owner must update the data which is held in a partition or partitions that have been tampered. In addition, there may exist other tables that are related to a damaged table. Those should be checked as well and in case of any damage, that related data must be restored too. To clarify this process, assume several records which show the time of data integrity checking for Salary in Employee_financial_information were added into data integrity scheduler by data owner.

The follow table shows some record of the scheduler table.

ID	ID_dataDictionary	Check_date	Situation	Description
1	2	12-22-14	Done	Trustable data
2	2	12-29-14	Done	Trustable data
3	2	01-05-15	Done	Unreliable data
4	2	01-12-15	Scheduled	
5	2	01-19-15		
6	2	01-26-15		

Figure 16: Scheduler Table Example

The process will be started by reading from the data integrity scheduler table. Records whose Check_date's values match the date will be selected. Then situation will be changed to 'Scheduled'. The data integrity verification process starts by selecting records whose situation is

‘Scheduled’. Those situations will be changed to ‘Running’, and the process is started. If data had been tampered with and the backup had been restored, the description filed will be changed to “Unreliable data”.

The algorithm for this process is provided in Algorithm 3 below.

4. Evaluation

To validate our solution, we implemented three parts of the data integrity verification process: data generation, data modification, and data verification. We only define "salary" as a sensitive data item, so we create one data control table for that field. For the first part we randomly generate two hundred thousand salaries having values in a range between twenty thousand and two hundred thousand. That data is saved in a file. In order to determine the performance of our solution, we carried out the three processes for different size of data. This means each process was run three times, first for one hundred thousand data values, next for a hundred and fifty thousand, and the last for two hundred thousand data values. By using several different widths, we divide data into ten sets. In the first set we scatter the entire data into ten partitions where all partitions have equal width. For the second set we scatter the whole data into twenty partitions with equal width and so on. Then for each set, we compute the hash value of each partition.

Figure 17 gives information about the partitioning data and computed hash values. For each data size, this linear graph shows the running time of scattering data into ten sets, each having different width, and computing of hash values of data for each set by using the SHA512 cryptographic hash function. As can be seen from the graph, execution time generates a curve depending on the size of a partition. When the number of partitions becomes very large, the curve levels off. There is a sharp decrease in the case where we generate twenty partitions compared to ten partitions. If we divide the data into fifty and sixty partitions and the size of data is one hundred thousand, the graph shows an unexpected change. It may be that there are several partitions without data, and the number of partitions including data is less than the case of defining fifty partitions. It also might be the size of the data if having sixty partitions is greater than the size of data in fifty partitions. We can see an increase for eighty and ninety partitions when we have two hundred

thousand data points also. At first sight, it seems that using equal depth gives us better performance than using equal width. This may not be true. For the data modification process, once a new record is inserted or deleted, all of the data must be repartitioned again, which is exceptionally time consuming. Generally, as we know, the data modification rate is remarkably high in a real business system, so using the equal depth method will have a negative impact on the performance of the system. Overall, there is a strong downward trend in the time of dispensing data into ten sets in which each set has different number of partitions; the greater number of partitions, the less execution time we have. Providing a balance of data size for partitions, we could achieve better performance.

We should mention that before partitioning the data, it must be sorted. If we don't sort it, when we want to fetch the data that belongs to a specific partition, we might get different ordering which would cause it to have a different hash value. This situation should be avoided. It is necessary whenever the data of one partition is brought into the main memory, as the order of data has not been changed.

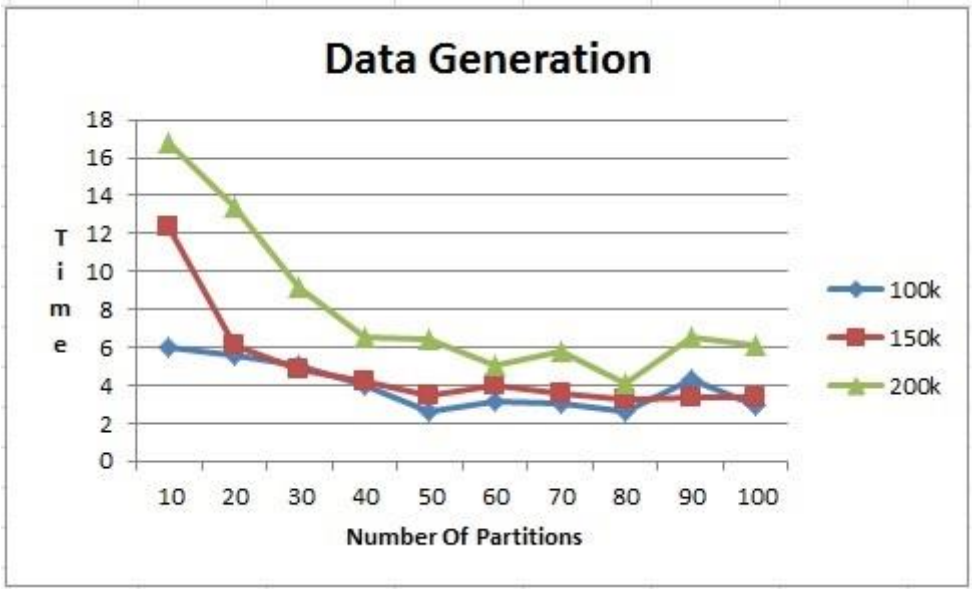

Figure 17: Data Generation Graph

The implementation of the data manipulation starts with generating random data between the aforementioned ranges. Then this process adds the generated data into the dataset and updates the corresponding partition's hash value with the new one. Figure 18 represents the execution time of inserting one datum. Again the graph execution time generates a curve that depends on the size of a partition. This graph shows that the greater number of partitions, the less execution time we have.

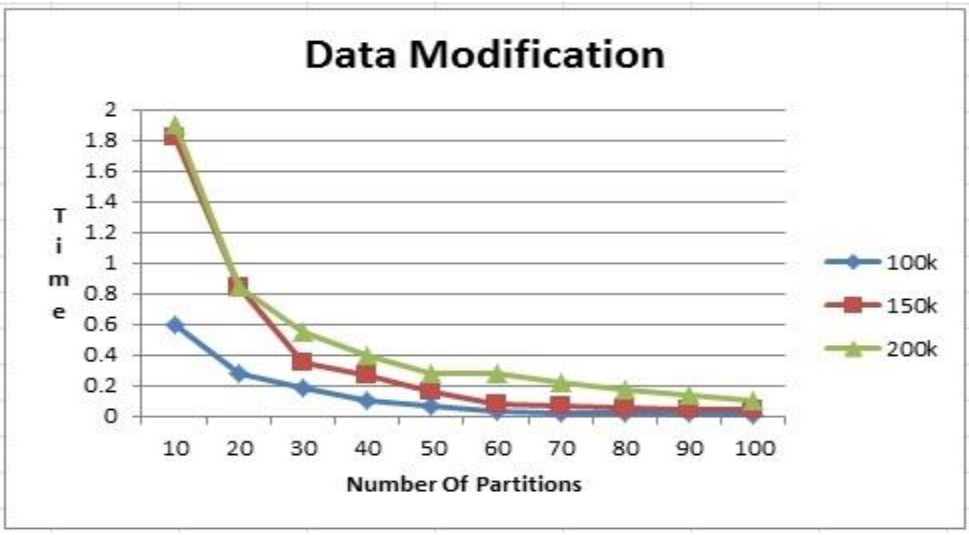

Figure 18: Data Modification Graph

The third test is to analyze whether the data integrity verification is correct.. For each of the sets of data, we compute a hash value of each partition and compare it to the hash value of that partition we saved before. Again we see a curve in execution time that depends on the size of partitions. Figure 19 depicts this behavior.

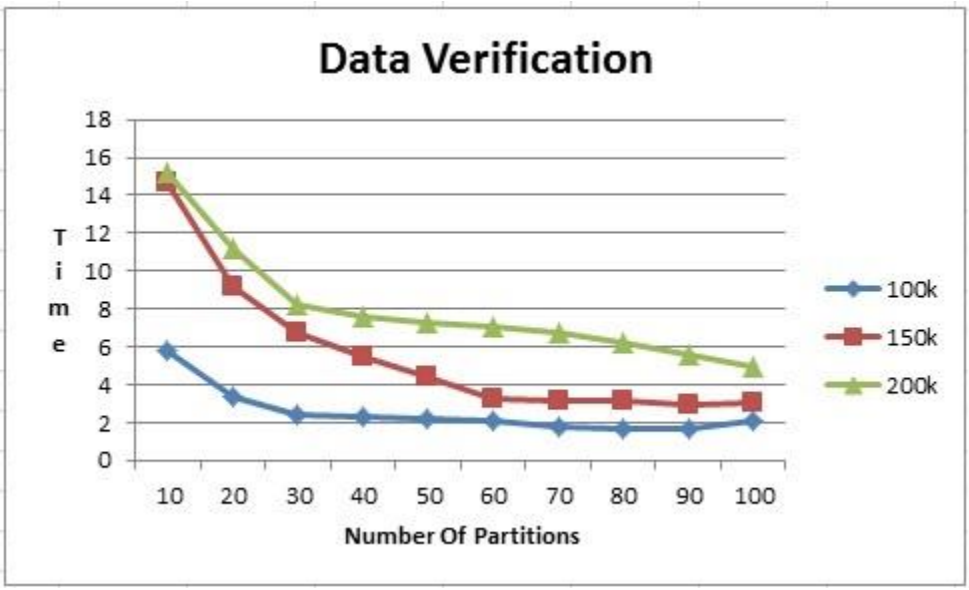

Figure 19: Data Verification Graph

5. Algorithms

Algorithm 1: Data Generation

Comments: following algorithm is the main process of generating data control tables

DataGeneration() {

1: Do while !EOF (data dictionary) {

2: Read data from data dictionary

3: Call Create_DataControlTable (A record from data dictionary)

4: }

Comments: following algorithm reads records of a column which keeps the real data, then sorts the data and splits data into several partitions.

Create_DataControlTable (Data dictionary record) {

1: Extract the table_name, column_name, LBound, UBound, and Cell# from data dictionary table to calculate the partition

2: Sort column_name in table_name

3: Partition_size = (Ubound – Lbound) / Cell#

4: For l=1 to Cell# { //Create partitions for the first level

5: Ubound = Partition_size+Lbound

// Read data from column which keeps sensitive data
6: Partition = Read column_name from table_name where Lbound<=column_name.data<=Ubound

// make substring to compute hash value
7: Compute hash value of all column_name.data in Partition

8: Insert the calculated hash value into the data control table

9: Lbound = Ubound +1

10: }

Algorithm 2: Data Modification

Comments: following algorithm is the main process of data modification process
DataModification() {

1: Command_Table = Query_Parser (Command)

2: Run the command on relational database

3: If (! Empty(Command_Table)) {

4: Rollback_Command(Command_Table)

5: Data_Recovery(Command_Table)

6: Modify_DataControl(Command_Table)

7: }

Comments: following algorithm parses the query, extracts important information, and stores into log tables
Query_Parser(command) {

1: Token = Read command, and tokenize

2: if (Seek Token.table_name in the data dictionary) {

3: Select Token.command_type{

4: Case: “Insert” {

5: if (Seek Token.table_name and Token.column in the data dictionary)

6: Command_type = 1

7: Status = ‘I’

8: Insert into Command values (Command_type, Command, Table_name, Status)

9: do while (Columns) {

10: Insert into Command_detail

11: }

12: }

13: }

14: Case: “Update” {

```
15:     if (Seek Token.table_name and Token.column in the data dictionary)        {
16:             Command_type = 2
17:             Status = 'I'
18:             Insert into Command values (Command_type, Command, Table_name, Status)
19:             Insert into Command_detail
20:     }
21: }
22: Case: "Delete" {
23:     Command_type = 3
24:     Status = 'I'
25:     Insert into Command values (Command_type, Command, Table_name, Status)
26:     do while (Columns) {
27:             Insert into Command_detail
28:     }
29: }
30: }
31: }
```

Comments: following algorithm returns the database to previous state, indicating the state before applying the current set of commands

Rollback_Command(Command_Table) {

```
1: Read from Command, Command_detail ordered by Command.Sequence_number  Desc
2: If (Retrieve real date to the earlier state) {
3:     Change Command.Status = 'A'
4: }
5: Else {
6:     Change Command.Status = 'R'
```

7: }

8: }

Comments: following algorithm recovers damaged data from the last backup
Recovery_Data(Command_Table) {

1: rec = Read from Command, Command_detail where Command.Status = ‘R’

2: Do while (!rec) {

3: Partition = generate a list of partitions which has been damaged

4:}

5: Do while (!Partition) {

6: Restore real data from the last safe backup

7: }

8: Do while (!rec) {

9: Apply rec.Command on the real data

10: Change rec.Command.Status = ‘A’

11: }

12: }

Comments: following algorithm updates the data control table
Modify_DataControl(Command_Table) {

1: Partition = Read from Command, Command_detail where Status = ‘A’

2: Do while (!Partition) {

3: Compute hash value of partition

4: Update the data control

5: }

6: }

Algorithm 3: Data Integrity Verification

Comments: following algorithm verify the integrity of the real data
DataIntegrityVerification() {

1: Scheduler_rec = Read records from Scheduler table which date is Todaye

2: Do While !eof (Scheduler_rec) {

3: Extract the table_name, column_name, LBound, UBound, and Cell# from data dictionary table to calculate the partition

4: Sort column_name in table_name

5: Partition_size = (Ubound – Lbound) / Cell#

6: For l=1 to Cell# { //Create partitions for the first level

7: Ubound = Partition_size+Lbound

// Read data from column which keeps sensitive data
8: Partition = Read column_name from table_name where Lbound<=column_name.data<=Ubound

// make substring to compute hash value
9: Compute hash value of all column_name.data in Partition

10: if (! Compare the new hash value to that one stored in data control table) {

Partition = generate a list of partitions which has been damaged

11: }

12: }

13: Do while (!Partition) {

14: Restore real data from the last safe backup

15: }

16: }

6. Conclusion

We have presented a new security model to verify the data integrity for an outsourced database. This method defines some control data to recognize data updates in case that the data is altered by an unauthorized user. To achieve faster query response time, we store the real data in a relational database as plaintext on a remote server. In order to verify integrity of the real data that is integrated, we generate metadata (kept as cipher text) and a table which stores information about all sensitive data items as plaintext. Using simulation, we proved that our method is efficient in that when the number of partitions increase, there is a decrease in processing time.

7. References

[1] "Security and IBM Global Technology Services,". Technical White Paper high availability in cloud computing environments.

[2] "IBM Cloud Computing". IBM. Retrieved 22 July 2011.

[3] Virginia's Community College, "Sensitive data definition,", June 2009.

[4] California State Polytechnic University, Pomona (2012, July 25). Available: "https://ehelp.wiki.csupomona.edu/Information_Security:_Defining_Sensitive_Data".

[5] Gerard Conway, Edward Curry, "Managing Cloud Computing: A Life Cycle Approach," Innovation Value Institute, National University of Ireland, Maynooth, and 2Digital Enterprise Research Institute, National University of Ireland, Galway.

[6] Hakan HacigÄumÄus, Sharad Mehrotra, and Balakrishna R. Iyer. "Providing database as a service,". In ICDE, pages 29. IEEE Computer Society, 2002.

[7] "Introduction to Cloud Computing", "www.priv.gc.co".

[8] Einar Mykletun, Maithili Narasimha, Gene Tsudik, "Authentication and Integrity in Outsourced Databases", Computer Science Department School of Information and Computer Science, University of California, Irvine.

[9] Matthias Nicola, "XML versus Relational Database Performance", The XML database blog, IBM Developer works, March 17, 2011.

[10] IBM, "Comparing XML and relational storage: A best practices guide", Storage best practice, March 2005.

[11] Frank Font (2005, Sep 11). Available: "http://www.room4me.com/index.php" .

[12] Hakan HacigÄumÄus, Balakrishna R. Iyer, Chen Li, and Sharad Mehrotra. "Executing SQL over encrypted data in the database service provider model". In SIGMOD, 2002.

[13] Rakesh Agrawal, Jerry Kiernan, Ramakrishnan Srikant, and Yirong Xu. "Order-preserving encryption for numeric data". In SIGMOD, 2004.

[14] Einar Mykletun, Maithili Narasimha, and Gene Tsudik. "Authentication and integrity in outsourced databases". In NDSS. The Internet Society, 2004.

[15] HweeHwa Pang, Arpit Jain, Krithi Ramamritham, and Kian-Lee Tan. “Verifying completeness of relational query results in data publishing”. In Fatma Ä Ozcan, editor, SIGMOD Conference, pages 407{418. ACM, 2005.

[16] Radu Sion. “Query execution assurance for outsourced databases”. In Klemens BÄohm, Christian S. Jensen, Laura M. Haas, Martin L. Kersten, Per-ºAke Larson,and Beng Chin Ooi, editors, VLDB, pages 601{612. ACM, 2005.

[17] Feifei Li, Marios Hadjieleftheriou, George Kollios, and Leonid Reyzin. “Dynamic authenticated index structures for outsourced databases”. In Surajit Chaudhuri, Vagelis Hristidis, and Neoklis Polyzotis, editors, SIGMOD Conference, pages 121{132. ACM, 2006.

[18] Einar Mykletun, Maithili Narasimha, Gene Tsudik, “Authentication and Integrity in Outsourced Databases”, Computer Science Department School of Information and Computer Science, University of California, Irvine.

[19] D. Boneh, C. Gentry, B. Lynn, and H. Shacham, “Aggregate and Veri_ably Encrypted Signatures from Bilinear Maps in Advances in Cryptology”. EUROCRYPT '2003 (E. Biham, ed.), Lecture Notes in Computer Science, International Association for Cryptologic Research, Springer-Verlag, Berlin Germany, 2003.

[20] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan, “Private Information Retrieval,” Journal of ACM, vol. 45, pp. 965.981, Nov. 1998.

[21] Y. Gertner, Y. Ishai, E. Kushilevitz, and T. Malkin, “Protecting Data Privacy in Private Information Retrieval Schemes,” in 30th Annual Symposium on Theory of Computing (STOC), (Dallas, TX, USA), ACM Press, 1998.

[22] B. Chor, N. Gilboa, and M. Naor, “Private Information Retrieval by Keywords,” Tech. Rep. TR CS0917, Department of Computer Science, Technion, 1997.

[23] Hakan HacigÄumÄus, B. Iyer, and S. Mehrotra, “Encrypted Database Integrity in Database Service Provider Model,” in International Workshop on Certi_cation and Security in Eservices (CSES'02 IFIP WCC), 2002.

[24] D. Song, D. Wagner, and A. Perrig, “Practical Techniques for Searches on Encrypted Data,” in 2000 IEEE Symposium on Security and Privacy, May 2000.

[25] E.-J. Goh, “Secure Indexes for Efficient Searching on Encrypted Compressed Data,” Cryptology ePrint Archive, Report 2003/216, 2003. http://eprint.iacr.org/2003/216/.

[26] Min Xie, Haixun Wang, Jian Yin, Xiaofeng Meng, "Integrity Auditing of Outsourced Data" ,Renmin University of China Beijing 100872, China, IBM T. J. Watson Research Center, Hawthorne, NY 10532, USA.

[27] Puya Ghazizadeh, Ravi Mukkamala,Stephan Olariu "Data Integrity Evaluation in Cloud Database-as-a-Service," Department of Computer Science, Old Dominion University, Norfolk, Virginia.

[28] A. Juels and B.S. Kaliski, Jr., "Pors: proofs of retrievability for large files," in CCS '07: Proceedings of the 14th ACM conference on Computer and communications security.

[29] Saxena, Sravan Kumar and Ashutosh,"Data Integrity Proofs in Cloud Storages", IEEE 2011.

[30] Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song. "Provable Data Possession at Untrusted Stores. "ACM Conf. Computer and Comm. Security (CCS '07), 598-609, 2007."

[31] Georg Becker, "Merkle Signature Schemes, Merkle Trees and Their Cryptanalysis," August 2008.

[32] Phil Bagwell, "Ideal Hash Trees," [1-1] Igor Tatarinov, Stratis Viglas, Kevin Beyer, Jatavel Shanmugasundaram, Eugene Shekita, Chun Zhang, "Storing and Queryring Ordered XML Using a Relational Database System", University of Wisconsin, IBM Almaden Research Center.

[33] Min Xie, Haixun Wang, Jian Yin, Xiaofeng Meng, "Providing Freshness Guarantees for Outsourced Databases" ,Renmin University of China, IBM T. J. Watson Research Center, Hawthorne, NY 10532, USA.

[34] "Introduction to Cloud Computing,". Office of the privacy of commissioner of Canada.

[35] Igor Tatarinov, Stratis Viglas, Kevin Beyer, Jatavel Shanmugasundaram, Eugene Shekita, Chun Zhang, "Storing and Queryring Ordered XML Using a Relational Database System", University of Wisconsin, IBM Almaden Research Center.

[36] Information and Technology Services, University of Michigan. Available: "http://safecomputing.umich.edu/protect-um-data/examples.php".

[37] Susan Landau, "Find Me a Hash," Notices of the American Mathematical Society, 53(3), March 2006, 330 – 332.

[38] Lars R. Knudsen,"SMASH - A Cryptographic Hash Function," Department of Mathematics, Technical University of Denmark.

[39] Georg Becker, "Merkle Signature Schemes, Merkle Trees and Their Cryptanalysis," 18.07.08.

[40] Richard Spillman, "Classical and Contemporary Cryptology," Prentice Hall, 2005.

[41] A. Menezes, P. van Oorschot, and S. Vanstone, "Applied Cryptography," CRC Press, 1996.

[42] Nedhal A. Al-Saiyd, Nada Sail, "Data Integrity In Cloud Computing Security," Journal of Theoretical and Applied Information Technology, 31st December 2013. Vol. 58 No.3.

[43] Wikipedia, Security Management (2015, Feb 26). Available: "http://en.wikipedia.org/wiki/Security_management".

[44] Colby College Information System and Data Security, "Definition of Sensitive Information".

[45] W. Wei, T. Yu, R. Xue, "iBigTable: Practical Data Integrity for BigTable in Public Cloud", North Carolina State University. North Carolina, United States, State Key Lab. Information Security, Institute Of Information Engineering Chinese Academy Of Science, China.

[46] Wikipedia (2015, Feb 17). Available: "http://en.wikipedia.org/wiki/Security_controls".

[47] Vitthal Raut, Suhasini Itkar, Department Computer Engineering, PES Modern College of Engineering, Pune, India "A Survey on Data Integrity of Cloud Storage in Cloud Computing", International Journal of Advance Foundation and Research in Computer (IJAFRC) Volume 1, Issue 2, Feb 2014. ISSN 2348 – 4853.