

ACCESS CONTROL AND SECURITY OF DATASETS  
BY USAGE TRACKING USING BLOCK CHAIN  
TECHNOLOGY

By

PIYUSH BHATT

Master of Science in Computer Science

Oklahoma State University

Stillwater, Oklahoma

2016

Submitted to the Faculty of the  
Graduate College of the  
Oklahoma State University  
in partial fulfillment of  
the requirements for  
the Degree of  
MASTER OF SCIENCE  
December, 2016

ACCESS CONTROL AND SECURITY OF DATASETS  
BY USAGE TRACKING USING BLOCK CHAIN  
TECHNOLOGY

Thesis Approved:

Dr. Johnson Thomas

Thesis Advisor

---

Committee Members:

Dr. Johnson Thomas

Dr. Christopher Crick

Dr. K.M.George

---

## ACKNOWLEDGEMENTS

I would like to express my deepest appreciation to all those who helped with me with creating and developing the concept of access control using block chains. A special gratitude to my thesis advisor, Dr. Johnson Thomas, whose contribution in stimulating suggestions and encouragement, helped me to coordinate my project. I would also like to thank Ashwin Thandapani Kumarasamy whose helpful insights and vast knowledge in the subject helped me achieve my goal.

Name: PIYUSH BHATT

Date of Degree: MAY, 2017

Title of Study: MASTER OF SCIENCE IN COMPUTER SCIENCE

Major Field: BIG DATA

Abstract: The thesis proposes a novel way to solve the threat of exposure of sensitive data from large datasets. Data is captured by organizations and converted into datasets. Two or more datasets may be combined to fetch critical or sensitive data which then can be misused for a variety of purposes. An unauthorized user may be able to access the data. This thesis proposes a blockchain approach to tracking data usage and will record details such as when and by whom the datasets are accessed. The Blockchain is used to record information about the user who accesses the datasets. The user name of the individual, name of the dataset accessed, the method using which the dataset is accessed (Command Line Interface or Map Reduce) and the command line operation performed (cat, copy, move, put) is recorded in each block. Each blockchain represents a dataset. Blockchains are very secure when it comes to storing sensitive information because the data inside a blockchain is immutable. The data inside each block is stored using hash values and each block is connected to the others using the hash value of the previous block. If an unauthorized modification is done to the data, the hash value would change, thus rendering all the following blocks invalid. This would alert the administrator that information has been modified. When a dataset is accessed, the username, the dataset name, the method of access (Command Line Interface or Map Reduce) and the command line operation (cat, copy, move, put) are captured using the data usage tracker and these values are used to create the blockchain. If an unauthorized modification is done, the blockchain validation process will identify the illegal access and report accordingly.

## TABLE OF CONTENTS

Chapter	Page
I. INTRODUCTION.....	1
Big Data .....	1
Block Chain .....	2
The Problem.....	4
Proposed Solution.....	4
My Contribution.....	4
Outline of the document.....	4
II. REVIEW OF LITERATURE.....	5
Access control of sensitive data in HDFS.....	5
Vigiles: FGAC for MR Systems.....	8
Access Control for Big Data using Data Content.....	9
An Access Scheme for Big Data processing.....	10
III. METHODOLOGY .....	12
Detailed problem Statement.....	12
Detailed solution .....	14
Algorithm.....	30
IV. Implementation.....	34
Findings and Results.....	41
V. Conclusion .....	59
REFERENCES .....	62

## LIST OF FIGURES

Figure	Page
1.....	17
2.....	17
3.....	18
4.....	18
5.....	21
6.....	22
7.....	23
8.....	23
9.....	24
10.....	25
11.....	26
12.....	27
13.....	28
14.....	29
15.....	35
16.....	42
17.....	42
18.....	43
19.....	43
20.....	44
21.....	45
22.....	45
23.....	46
24.....	46
25.....	47
26.....	47
27.....	48
28.....	48
29.....	49
30.....	50
31.....	51
32.....	51
33.....	52
34.....	53
35.....	54

36.....	54
37.....	55
38.....	55
39.....	56
40.....	56
41.....	57
42.....	57
43.....	58
44.....	58

## CHAPTER I

### INTRODUCTION

#### **1. BIG DATA**

Every day 2.5 quintillion bytes of data are created [1]. In 2011 1.8 zettabytes (or 1.8 trillion GBs) of data was created. In 2012 it reached 2.8 zettabytes and the IDC (International Data Corporation) now forecasts that 40 zettabytes of data will be generated (ZB) by 2020 [2]. Organizations with a large customer base like Facebook, Amazon or Google deal with huge amounts of data. This kind of data is called Big Data. With Big Data, the large amount of data cannot be processed using conventional database systems. Big Data is not only about the size of the data. It is also about how the data is being used, how the data is being transmitted or moved, how costly it is to maintain the data, how much time is consumed to process the data. Big Data is not one technology but an amalgamation of multiple different technologies. It uses tools and techniques to process and maintain data, effectively and efficiently. Industries like Banking, Health-care and Education use these technologies to maintain their data effectively. Big data technologies utilize data in a different way than conventional database systems. The data is compiled into structures called datasets and then these datasets are processed. For example Apache's Hadoop is one Big Data framework, which is widely used to deal with



big data. Its file system known as the Hadoop Distributed File System (HDFS) which is a distributed file system used to store large volumes of data. It provides reliability and efficiency. Some other frameworks which handle big data are: Storm [11], Hortonworks Data Platform [12], Apache Pig [13]. There are numerous such tools out there. Every tool has their own way to handle big data. Hadoop uses Map Reduce to process the data. Hadoop MapReduce [3] is a software framework for writing applications which process vast amounts of data (multi-terabyte data-sets) in-parallel on large clusters (thousands of nodes) of commodity hardware in a reliable, fault-tolerant manner. R is a programming language mostly used for Big Data Analytics. Big data analytics [4] examines large amounts of data to uncover hidden patterns, correlations and other insights. With today's technology, it's possible to analyze data and get answers from it almost immediately – an effort that's slower and less efficient with more traditional business intelligence solutions.

## **2. BLOCKCHAIN**

Blockchain is a Distributed Database. Blockchain was first used for Bitcoin [5]. Bitcoin is an organization which created a cryptocurrency with the same name (Bitcoin).

Cryptocurrency is analogous to digital currency. The Bitcoin protocol uses blockchain.

Blockchains are secure because they are built to prevent hacker attacks or unauthorized modification. There are three types of blockchains: Public, Private and Hybrid.

Organizations such as Bitcoin use a public blockchain. Everyone who has access to the Internet can look at the bitcoin transactions. A blockchain is kept private in an

organization which has sensitive data. A hybrid blockchain is one in which some parts of the blockchain are public while some parts are private. A block chain consists of blocks which consist of some kind of data. They are connected to each other using Hash values (Block header).

After some computations and modifications the items mentioned above (Version number, Hash of the previous block, Merkle Root, Time stamp and number of bits of the data) are concatenated together and converted into a single hash. The hash is a value generated by a Cryptographic Hashing Function [6]. If someone tries to modify even a single data entry in the block, the whole hash value will get corrupted. Thus it would be clear that an unauthorized change has been made. There are multiple nodes in a blockchain network. There might be hundreds of thousands of nodes present in the blockchain. If someone wishes to hack a blockchain, he will have to find all the nodes in the network and hack all of them at the same time. If he tries to access one node and modify the data, the values of the hash value of the block will change. If the block is modified, its hash value would change and thus it would be known that the block is now corrupt. The block from any of the other nodes, which are not corrupt could be used to replace the corrupt block. All the blocks are connected to each other using the hash value of the previous block. Hence if a change is made in any block, all the blocks following that block will be rendered useless. Hence the blockchain is secure and reliable.

### **3. THE PROBLEM:**

Data is captured by organizations and converted into datasets. Two or more datasets may be combined which may cause critical or sensitive data to be fetched from multiple locations. The data can then be misused for variety of purposes. An unauthorized user can access the data.

### **4. PROPOSED SOLUTION:**

A system is required to track usage of datasets. Every time a dataset is accessed, the usage of that data should be captured. This system should be secure so that no one can make any unauthorized changes to it. A blockchain will be created every time a dataset is introduced. Every dataset will have its own block chain. Every block in the dataset will have the user name of the user accessing the dataset, the name of the dataset, the type of access used and command line operation performed. Type of access can be using MapReduce (MR) or using the Command Line (CLI) and the command line operation performed could be copy, move or view. Since the block chain is secure and immune to any kind of cyber-attacks, the usage of the datasets would remain safe.

### **5. OUTLINE OF THE THESIS**

Chapter 2 consists of the review of previous work in similar areas. Chapter 3 describes the problem in detail and the proposed solution. Chapter 4 describes the implementation of the solution. Chapter 5 is the conclusion of the thesis work done.

## CHAPTER II

### LITERATURE REVIEW

Several other researches have been done in similar areas. This chapter explains how the work done by the other researchers is different from what has been done here.

#### 1. Access control of sensitive data in HDFS [7]:

The research presents a new security model which is different than the conventional key based federated system. Every federated user has a key (password), using which he can access data. For two parties to be able to access each other's data, a security token is used. This could be a RSA token. The TokenID is generated which provides the Token.

$$\text{TokenID} = \{ \text{OwnerID}, \text{RenewerID}, \text{issueDate}, \text{maxDate}, \text{sequenceNumber} \} \quad (1)$$

`OwnerID`: ID of the owner of the token.

`RenewerID`: ID of the individual who renews the token.

`issueDate`: Token issue date

`maxDate`: Token expiration date

`sequenceNumber`: The sequence number of the token in the list.

Every token has a validity which is usually one day or till the task is completed. The validity can be calculated using the `issueDate` and the `maxDate`. Access rights or limitations should be added to the `tokenId` so that a user cannot access any unauthorized data. Also the verification of the result generated should be present.

Equation (1) can be modified as:

$$\text{TokenID} = \{\text{ownerIDc}, \text{renewerId}, \text{issueDate}, \text{maxDate}, \text{sequenceNumber}, \text{outputCheck}\} \quad (2)$$

$$\text{ownerIDc} = \{\text{ownerId}, \text{accessLimit}, \text{KeyId}, \text{expirationDate}\} \quad (3)$$

The new parameters to generate `TokenID` has 2 new components. `OwnerIDc` and `outputCheck` as seen in (2). `OwnerIDc` describes the access rights allowed. It is generated using `accessLimit` as seen in (3). `accessLimit` provides permission to view the data for which the user has access. `OutputCheck` verifies the output generated. These two factors enhance the security of the data being accessed.

Let's assume a function  $G$  which comprises of:  $N$  (set of states),  $A$  (set of access rights),  $D$  (set of allowed resources in the Database),  $U$  (return result of the user query).

Function  $G$  has a fixed set of access rights, specifies the resources allowed in the database and the result of the query. This function acts as an access control mechanism. If a user tries to access any information which requires special access rights, or if he tries to access resources which are it is not permitted to access or if the result of the query does not

match the permitted result, the function G will be responsible for catching the malicious access.

Every authenticated user  $n_i \in N$  will have a token which will define a set of access rights  $a_i \in A$ . An unauthorized user (malicious user/hacker) will have no access rights. If an authorized user tries to access any data to which he doesn't have access, he will be blocked and reported. If the output doesn't match the access permissions he will be blocked. Two users can have some of the access rights in common. A user can have the same access rights for two database resources or his access rights may be different for the database resources. Every system or node has a super user who has all the access rights to all the resources. If a hacker tries to simulate a query as an authorized user he can be caught for multiple reasons. He might not know about the access rights or the queries do not match. If he is using a particular computer system, he will get locked out of that system. The security team might get hold of the query the hacker was trying to execute, flag it and prevent any similar future attacks. There might be internal hackers who try to access the data they do not have rights to. Their attempts to gain unauthorized access will be flagged.

There is a chance that sensitive information might be accessed without any trace of any attack. Even read access rights provided to a user will result in the exposure of the data.

## **2. Vigiles: Fine-grained Access Control for MapReduce Systems [8]:**

Vigiles is like a firewall which provides Fine Grained Access Control Predicates. User defined predicates are applied to each file and a user can access a file only after these predicates are applied. These predicates run Access Control Filters (ACF). Access Control Filters provide access based on the security policies and protocols. There are three types of ACF actions: reject (rejects access), grant (grants access) and modify (modifies the data). Vigiles acts as a middleware between users and the MapReduce system. It uses the same user name and password of the OS authenticating process. End users have no direct responsibility with Vigiles. Administrators create config files and customize ACFs. A user can communicate with Vigiles by using the computer screen (OS interface). Vigiles executes the job on behalf of the user and since it has all the file access parameters, it runs and provides the output. ACF generation is a three step process: Decompose, fetch and action. Unstructured text is broken into words. Sensitive words are found and created into tokens which are indexed. Then ACF actions are applied to the words. ACF injection is a process used in Vigiles. Three types of aspects are injected into these pointcuts: (1) initialization aspect is injected to `initialize()` method; (2) predicate aspect is injected to `nextKeyValue()` method; and (3) modification aspects are injected to `getCurrentKey()/getCurrentValue()` methods. Although Vigiles provides a way to identify sensitive data, it all depends on the

administrator. Administrator decides which data is sensitive. There's no way to track which user accesses which data.

### **3. Access Control for Big Data using Data Content [9]:**

The paper proposes a Content-based access control (CBAC) for big data. It is an additional access control scheme along with the existing Database access control rules. It is designed for areas where a certain amount of approximation regarding access control is applicable. A user can access a few more or less records than is assigned by the administrator.

A generic CBAC policy could be represented as:

$ACR = \{subject, object, action, f(u, d_i)\}$ , where  $u$  is the subject and  $d$  is the data object.

CBAC provides the users a special role which allows them to access a certain type of data for which they have the access. The role has predefined access to a type of data.

It uses a function:  $f(u, d_i) = \{true, false\}$ . Access to a record will be provided if the output of the function is true.

In this model, content similarity is compared with a preset threshold, and the user is granted access to all the records that pass the similarity. But if the threshold value is computed wrong, then it might allow the user to access more records than he was



allowed. This can be solved by implementing Top K Similarity. It can be implemented just like CBAC, as an additional component.

There's always the risk a user can access a record which he was not supposed to access and there would be no record of him accessing it since CBAC provides the access based on approximation.

#### **4. An Access Scheme for Big Data processing [10]:**

Big Data has no predefined scheme for database management. So Big Data access control requires additional access control capabilities. The following components are the parts of the new Access control scheme:

1. Security Agreement: It's an agreement between a Big Data source and the Master System which defines the security classes. Its purpose is to agree upon security classes by the Master System and Big Data sources.
2. Trust Cooperated Systems List (TCSL): This is a list of trusted Cooperated Systems which are recognized by the master system.
3. Master System Access Control Policy (MSP): This is a set of rules imposed by the Master System to enforce Access Control on Cooperated Systems.

4. Cooperated System Access Control Policy (CSP): This allows the Cooperated Systems to control the access to the distributed Big Data data/process by considering the processing capabilities and security requirements of the systems.

5. Federated Attribute Definitions (FAD): This lists the common attributes used by the Master System and Cooperated System, so that the Master System Policy and Cooperated System Policy can be composed using the common attributes in the FAD dictionary.

Someone can bypass the security classes or the security officer can modify the policies.

Human error can be a factor in this scheme.

There is very limited work on detecting malicious or unauthorized modifications to big data. The aspect of security when datasets are merged or joined has also not been addressed in previous works. None of the above approaches provide a proper and accurate mechanism for securing big data.

## CHAPTER III

### METHODOLOGY

#### **III.1. DETAILED PROBLEM STATEMENT**

Two or more datasets can be combined which may cause critical or sensitive data to be accessed which then can be misused for a variety of purposes. Even a single dataset carrying sensitive can be accessed for malicious purposes. For example one dataset has Candidate\_Id, Name, Date of Birth and Place of Birth of an individual. Another dataset might have the Person\_Id, Candidate\_Id, Date of Birth and Phone Number. The third dataset might have Person\_Id and SSN. If an unauthorized user is able to get all these 3 datasets, he can get the sensitive information about the person. A system is needed which would track the usage of these datasets; who is using what kind of datasets. If the same unauthorized user who is trying to access the sensitive information can cover his tracks in the system, then the system fails. So this system needs to be highly secure and quite reliable. There should be no loss or corruption of data for any reason. The system should have multiple backups on different locations like DR (Disaster Recovery) sites.

The solutions mentioned in chapter II which are provided by other researchers are good but they are dependent on human computer interaction. A user may not have access to modify the dataset but he may be able to read the data from the dataset which is a major concern. In some cases, the access rights are not accurately implemented. They could

provide a little more access or a little less access to data. A little less access would be a less of a concern than providing a little more access. Any kind of sensitive data could be accessed, accidentally or intentionally. The solution provided here is mostly automated. The data cannot be modified even if the individual has access rights. It can be deleted by the administrator but the report generated would be accessed by everyone and anyone can question why the dataset, blockchain or even a block was deleted. The blockchain is monitored after every fixed interval of time by an automated process. So any kind of unauthorized access (if possible) will be detected.

### **III.2. DETAILED PROBLEM SOLUTION**

In order to solve the problem stated above, a usage tracking system is created using block chain technology. There would be multiple nodes and each node will have the blockchains of all the datasets available. Once a block chain is created in any of the nodes, it would be synchronized to all the other nodes. So there would be no loss of data thus making it reliable.

Not only would the proposed scheme be able to detect changes to a block chain, it would also be able to detect any illegal accesses to data by comparing it with data access control rules. If an individual who does not have access rights, tries to access a dataset, the information would be recorded and monitored at regular intervals. As soon as a user accesses a dataset, his or her username, the dataset(s) he or she accessed, how he or she accessed the dataset(s) and what command line operation was performed will be recorded. If he does not have access rights to the dataset and if he or she still tries to access it, it would be detected.

The implementation of our proposed scheme is as follows:

A block consists of 5 kinds of data:

- Version Number
- Hash of the previous block(Block Header)
- Merkle Root
- Time stamp of creation of the block

- Number of bits of data inside the block. This is the size of data in the block in bits.

**1. Creation of Genesis block:** If an existing dataset is accessed for the first time, the dataset name, name of the user who accessed the dataset, the method using which the dataset was accessed (Command Line Interface or Map Reduce) and the command line operation performed (cat, copy, move) will be captured. If a new dataset is created in HDFS, name of the dataset created, name of the user who created the dataset, method using which the dataset was created (Command Line Interface or Map Reduce) and the command line operation using which the dataset was created (copy, move, put) will be captured. The values captured above would be hashed using SHA-256. This would give a 256 bit hash or 64 characters hash value. A version number would be assigned to the genesis block. This could be any random number of any random length. Every following block in the block chain would have a version number. There are many ways to calculate the version number of the next block. The version number can be incremented by a constant number or a customized algorithm could be used which would compute the version number. The number of bits of the data in the genesis block would be calculated. For example if the data in the blockchain is 10, then the binary encoded form of 10 in bits would be 00110001 00110000. Thus the size of the data in this case would be 16 bits. A timestamp would be entered for this block. The timestamp would be the creation time of the block. Once the hashed value of the data is calculated, the version number, the number of bits and the time stamp, will be concatenated and then the hash of this string

will be calculated. This will be the final hash of the genesis block. This hash value will be used in the next block.

**2. Creation of the first block:** Once the genesis block is created, it will provide a hash value for the next block. A version number will be calculated using a customized algorithm. It will take the version number of the previous block and generate a version number. A timestamp will be generated for this block. The username of the user accessing the dataset will be captured. The name of the dataset will be acquired. The method which was used to access the dataset will be fetched. It would be either using Map Reduce (MR) or the Command Line Interface (CLI). The command line operation with which the user accessed the dataset (view, copy, move, put) will be recorded as well. A Merkle root will be calculated using the Merkle tree. The Merkle root is calculated because it contains the amalgamated hash values of all the data which is present inside the block. If even a single data item is modified, the Merkle root will be changed because the hash value of the data is a part of the Merkle root.

**Steps to create a Merkle tree:**

1. Calculate  $H(\text{user name})$ : Hash value of the user name;  $H(\text{dataset name})$ : Hash value of the name of the dataset;  $H(\text{method})$ : Method of accessing the dataset;  $H(\text{operation})$  define. This order will be followed throughout the blockchain for consistency.

H(username)    H(operation)    H(dataset)    H(method)

2. Concatenate the hash of the first two items:

$H(\text{UNOP}) = H(\text{user name}) + H(\text{operation})$ . Point two arrows from  $H(\text{user name})$  and  $H(\text{operation})$  to  $H(\text{UNOP})$ .

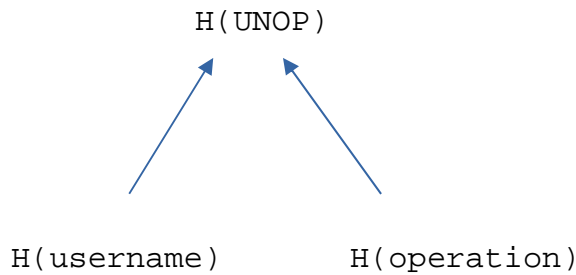


Figure 1: A node of Merkel Tree

3. Concatenate the hash of the next two items:

$H(\text{DSM}) = H(\text{Dataset name}) + H(\text{method})$ . Point two arrows from  $H(\text{dataset name})$  and  $H(\text{method})$  to  $H(\text{DSM})$ .

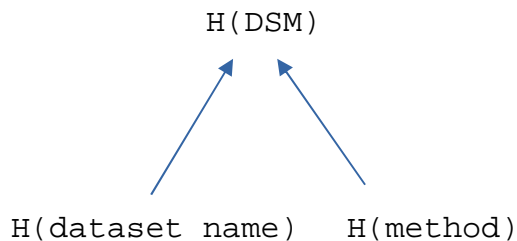


Figure 2: A node of the Merkle tree

4. Point two arrows from  $H(\text{UNOP})$  and  $H(\text{DSM})$  to  $H(\text{root})$ .



$$H(\text{root}) = H(\text{UNOP}) + H(\text{DSM}) .$$

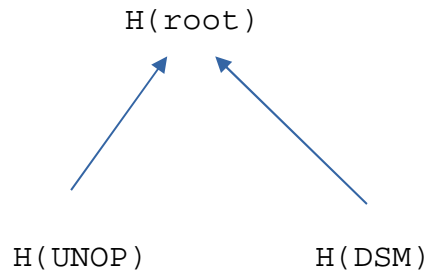


Figure 3: Combined nodes of the Merkle tree.

5.  $H(\text{root})$  is the Merkle root.

The following figure is the complete Merkle tree

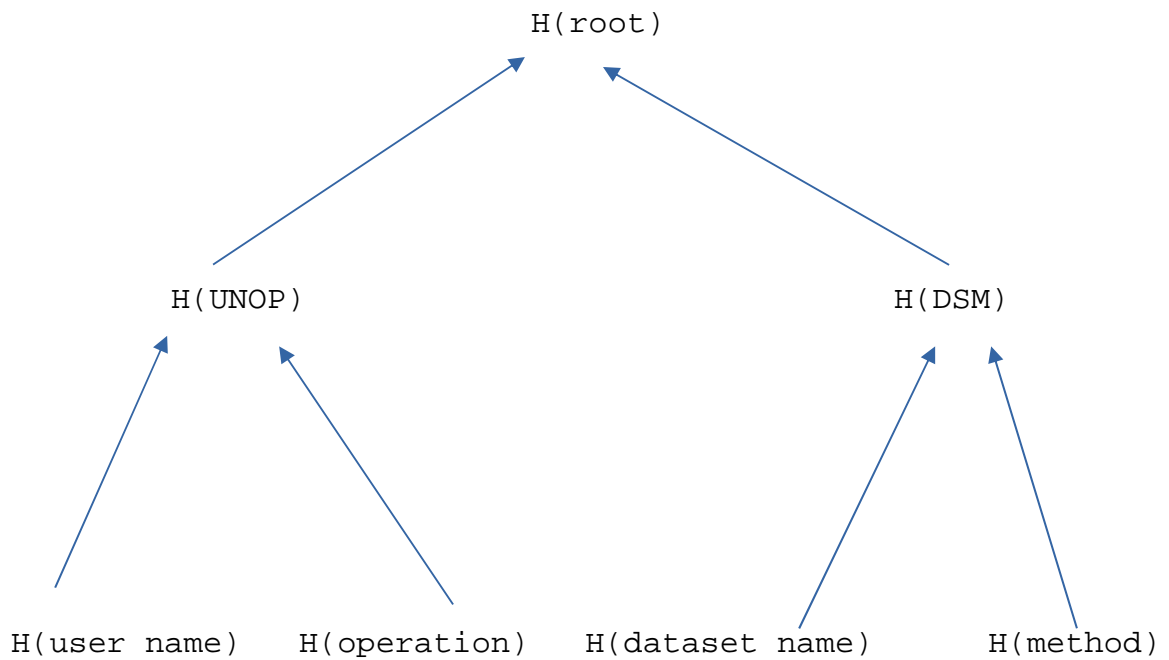


Figure 4: A complete Merkle tree

Once the Merkel root is calculated, the number of bits of the data is calculated. The total number of bits will be calculated for username, command line operation, dataset and the method.

Total items in a block

- Hash value of the genesis block.
- Version number
- Time stamp value
- Merkel root value (The hashed value of H(root) from Figure 4)
- Number of bits of the data.

The version number, timestamp and number of bits will be converted into hexadecimal format. Each hexadecimal number will be of 8 characters. If it is shorter than 8 characters it will be padded with 0s in the beginning.

Total items in a block:

- Hash value of the genesis block.
- Version number (hex)
- Time stamp value (hex)
- Merkel root value
- Number of bits of the data (hex)

Next convert all the items into Little Endian format. For example 000003EB would be converted into EB030000. The characters are replaced in pairs from the end of the string

to the front of the string. So initially 'EB' was at the end and it is now brought to the front of the String. Similarly for all the other pairs as well.

$E(HG)$  : Little Endian Format of hash value of genesis block.

$E(HEXVN)$  : Little Endian format of hexadecimal value of version number

$E(HEXTS)$  : Little Endian format of hexadecimal value of time stamp

$E(MRV)$  : Little Endian format of Merkel root.

$E(HEXNB)$  : Little Endian format of hexadecimal value of number of bits

The hash value of this block will be calculated by concatenating the above 5 values and hashing:

$H(CB)$  = Hash value of current block

$H(CB) = H(E(HG) + E(HEXVN) + E(HEXTS) + E(MRV) + E(HEXNB))$

This would give the final hash value of the current block.

Following the same procedure for all the blocks, a block chain is completed.

As mentioned earlier, every dataset has a block chain; hence there will be multiple block chains. Every time a new dataset is introduced, a new block chain is created

If a user wishes to access dataset (A) and dataset (B) as shown in figure 5 a new block chain would be formed by merging the 2 block chains and creating a whole new block chain.

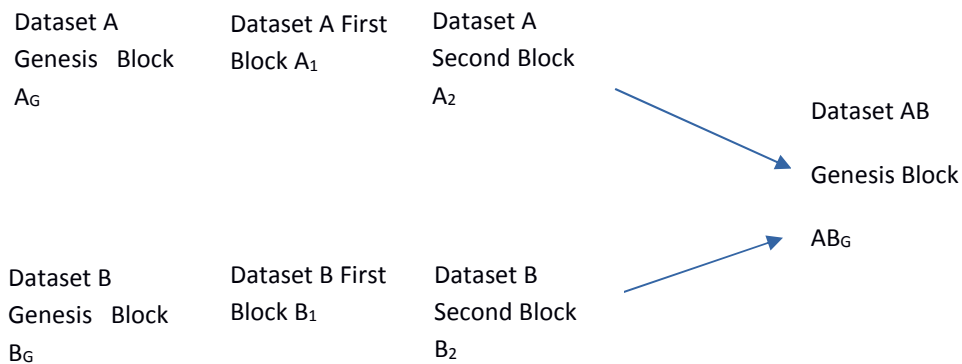


Figure 5: Merging of 2 blockchains to create a new block chain

The resulting dataset (AB) as shown in the figure is the amalgamation of dataset A and dataset B. The newly formed block is the genesis block of the block of the new chain AB and the following blocks would be AB<sub>1</sub>, AB<sub>2</sub>, ..., AB<sub>n</sub>. The genesis block would be created by capturing the name of the dataset. The name of the dataset would be created by concatenating the name of the 2 datasets which are combined. Name of the user who accessed the 2 datasets together will be captured. Map Reduce is the only method allows the user to access 2 datasets at the same time. The command line operation would be MRAccess. These 4 values would be used to create the Merkle root. The version number

of the new genesis block  $AB_G$  would be computed by creating a similar customized algorithm which would take the input as version numbers of  $A_2$  and  $B_2$ . The timestamp when the 2 datasets were accessed will be captured. The number of bits of data (the username, new dataset name, method of access and the command line operation) will be calculated.

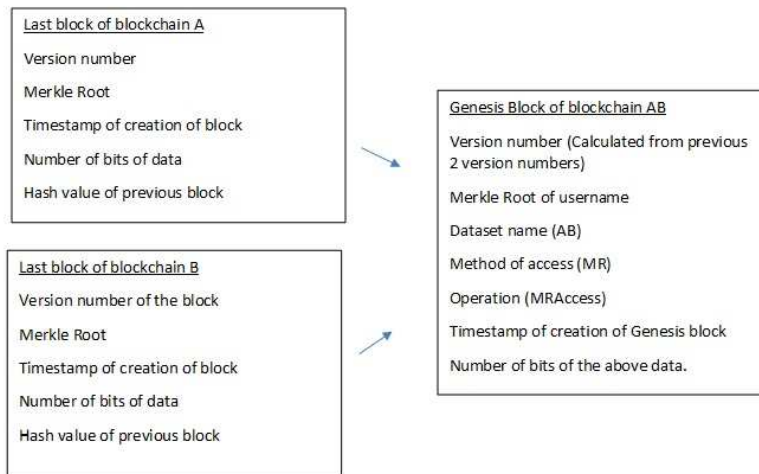


Figure 6: Creation of Genesis Block when 2 blockchains are combined.

If a new dataset C is introduced, and all three are merged, then the new blockchain would be ABC.

**Deletion of datasets:** If a dataset is deleted from Hadoop File System (HDFS), the blockchain representing that dataset would be deleted as well.

Dataset A was accessed to create the blockchain A. The blockchain had a genesis block  $A_G$  and  $A_1, A_2, \dots$  were the subsequent blocks.

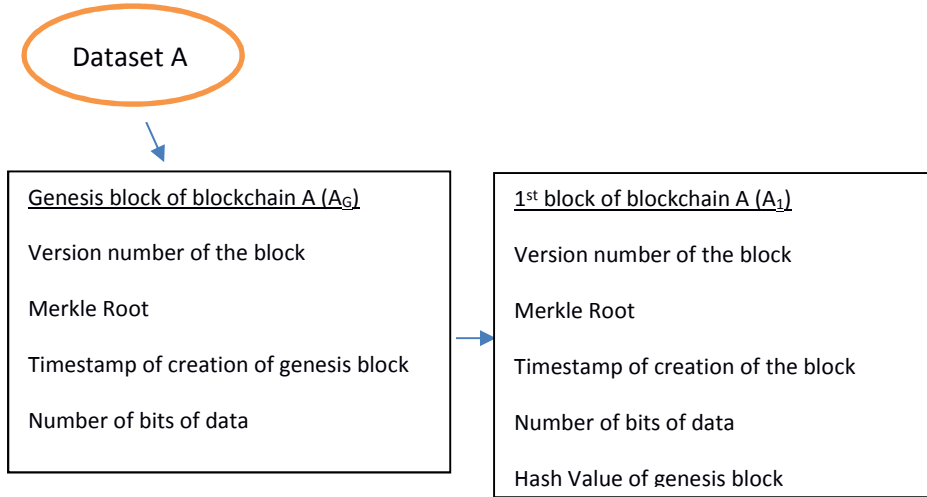


Figure 7: Creation of blockchain when dataset is accessed.

When we delete the dataset A from HDFS, the blockchain A will be deleted automatically.

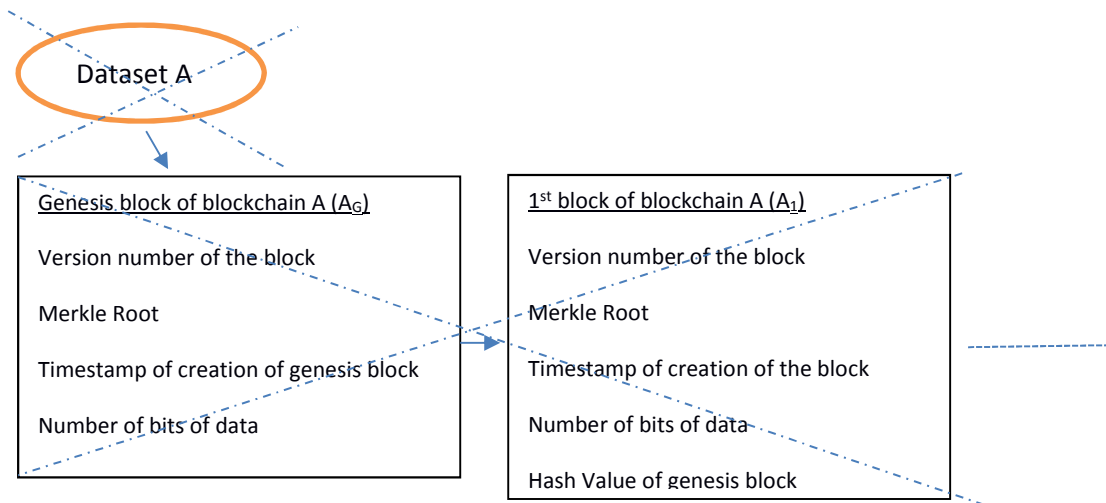


Figure 8: Deletion of blockchain when dataset is deleted.

In case of merged blockchains:

For example if from figure 5, dataset B is deleted, the new genesis block  $AB_G$  would simply become a part of block chain A. The Genesis block of AB remains the same.

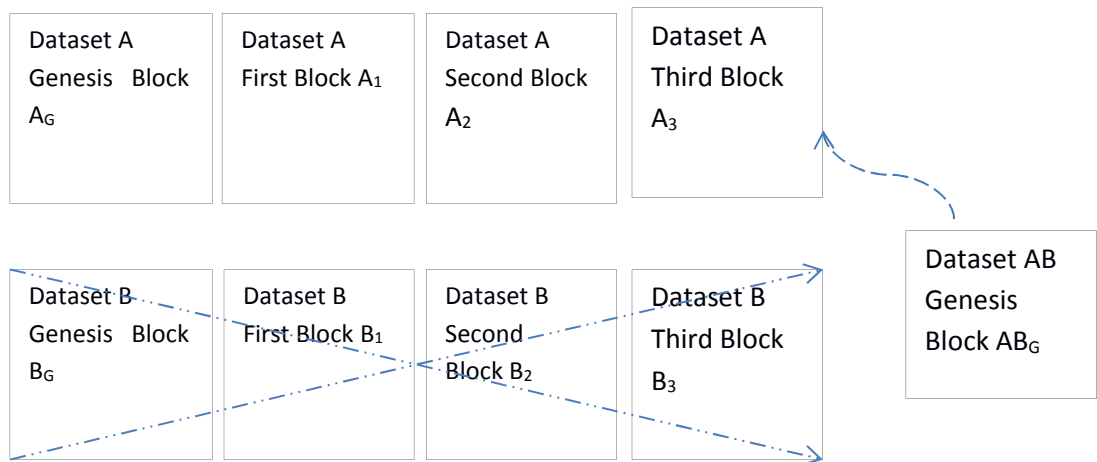


Figure 9: Deletion of a Dataset

**Deletion of Block:** If an unauthorized user tries to modify any block, he will change the hash value of the block and thus will be caught. But now the block will be useless. So the block will be deleted and the block with the correct data will be updated by retrieving the block from the backup nodes. Hence data will not be lost. This is why the blockchain is secure and reliable.

The dataset A was accessed to create the blockchain A. It has the genesis block ( $A_G$ ) and the subsequent blocks  $A_1, A_2 \dots$ . The entire blockchain is replicated in the other node with the same values. The value (Merkle Root) in block  $A_1$  in Node 1 has been modified by an unauthorized user thus rendering it useless. While the blockchain A in Node 2 which is a backup node has not been modified and has correct values.

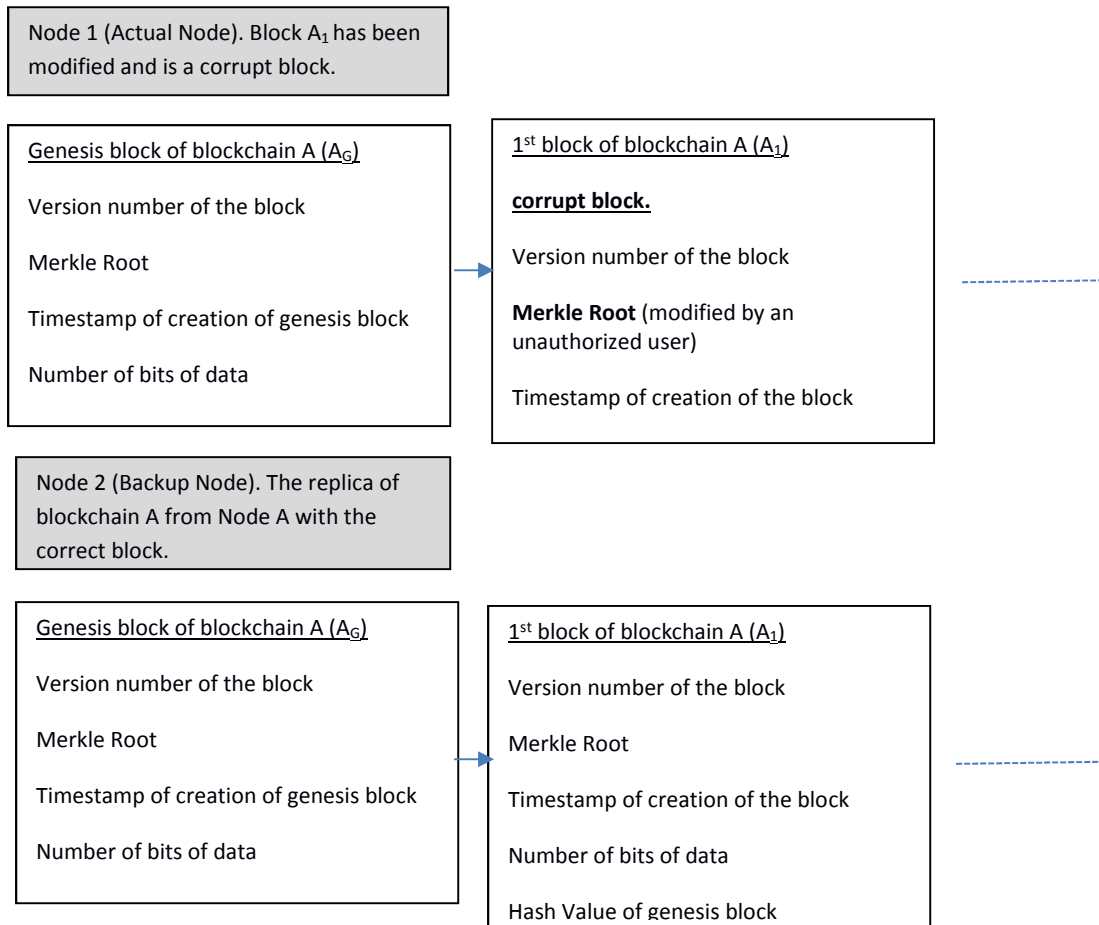


Figure 10: Corrupt blockchain in Node 1 and correct blockchain in backup node



When the blockchain validation process will be executed, the corrupt block will be identified. The corrupt block will be deleted and the blockchain will be updated using the correct blockchain in Node 2.

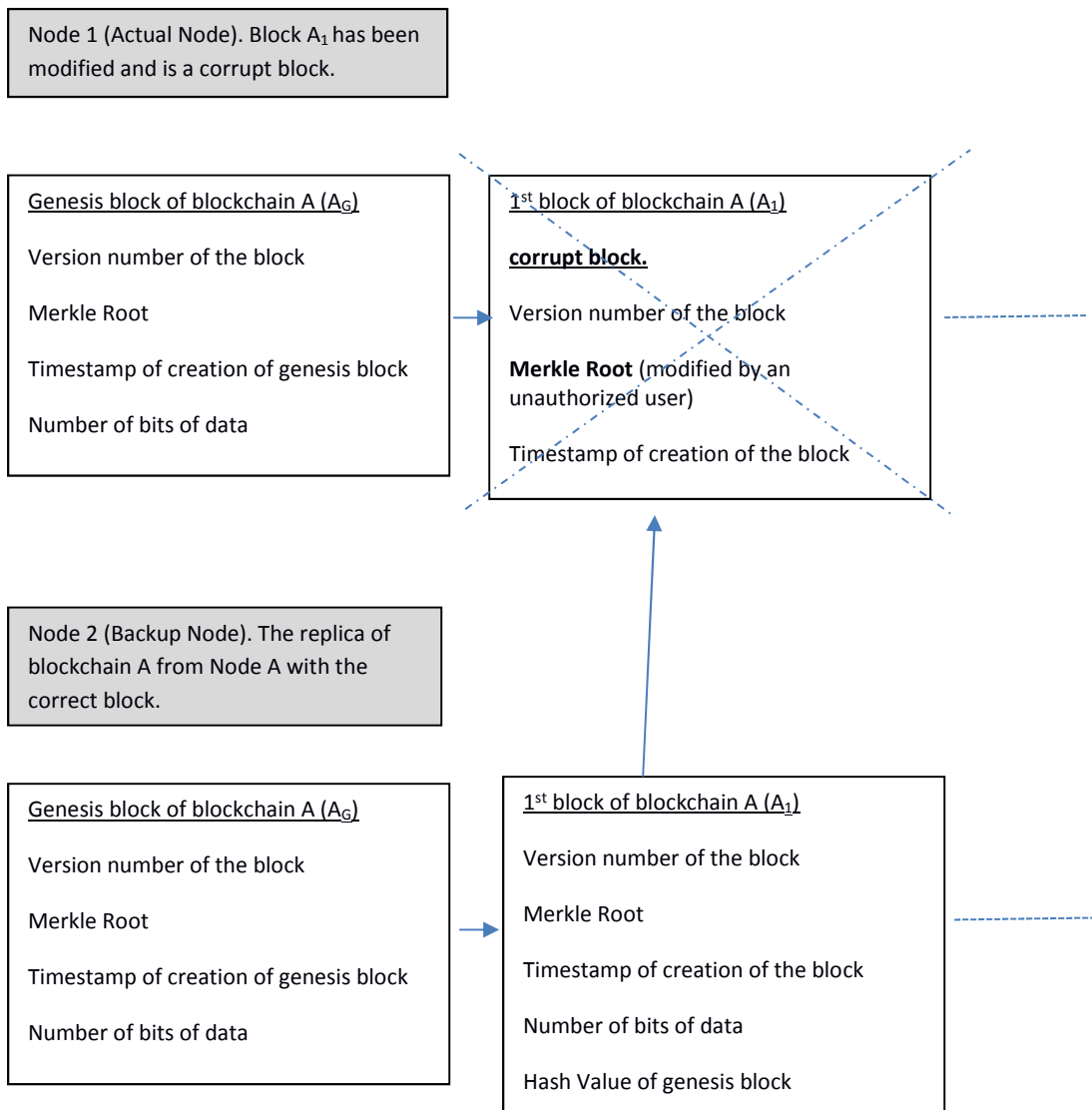


Figure 11: Backup node replacing corrupt blockchain.

**Deletion of dataset and restore:** If the administrator decides to delete a dataset and then decides to insert the same dataset again with the same name, a new block chain will be created. It will not be linked back to the existing blockchain.

Dataset A was created/accessed to create the blockchain A with genesis block ( $A_G$ ) and subsequent blocks  $A_1, A_2 \dots$

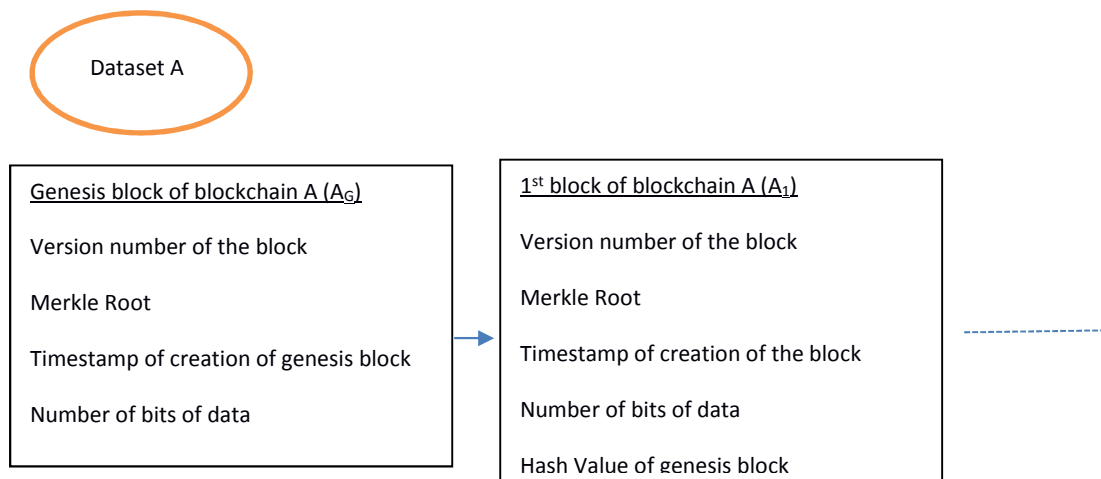


Figure 12: Dataset creating the blockchain.

The administrator decides that dataset A is of no use anymore and thus deletes it. Since the dataset A is deleted, as explained previously, the blockchain A will be deleted automatically.

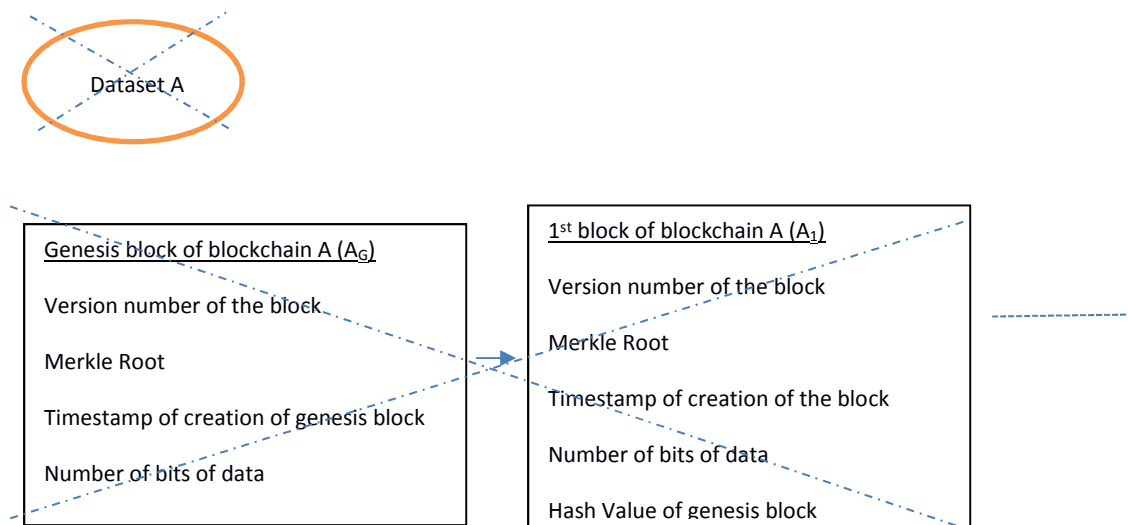


Figure 13: Deletion of dataset results in deletion of blockchain.

The administrator realizes that dataset A was important or dataset has to be used again, he inserts dataset A. Once he creates dataset A again, a fresh blockchain A.

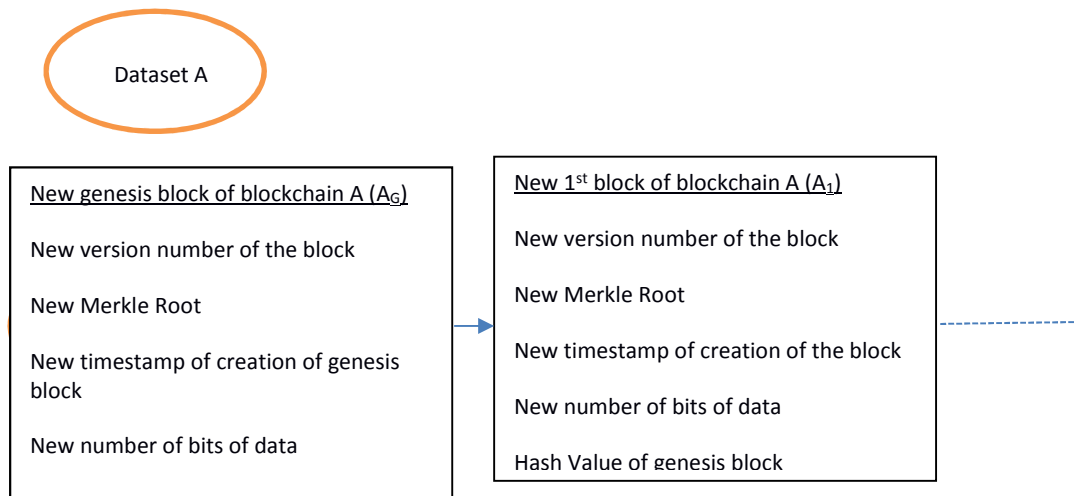


Figure 14: Reintroduction of dataset results in anew blockchain

The blockchain cannot be modified. Once a block has been created, it cannot be modified legally. But if someone tries to change the values in a block in the block chain in an unauthorized fashion, it would change the hash value. The access rights to delete the blockchain lies with the administrator. If he deletes a blockchain or a block, his actions would be documented in a log file with the timestamp and his username. If someone else tried to delete the block or blockchain, it would be detected because after every fixed interval an automated batch process will be executed which will compare the blockchains of one node to the other. If any unknown discrepancy happens, the blockchain would notify the administrator. Algorithm 5 below explains this process.

## Algorithms for Blockchain Data Usage Tracker

### Algorithm 1 merkle\_root\_calc

**Input:** dataset\_name, operation, user\_name, method\_Of\_Access;

**output:** merkle\_root;

1: Compute Hash (dataset\_name and operation) = Hash (Hash (dataset\_name) +Hash (operation));

2: Compute Hash (user\_name and method\_Of\_access) =Hash (Hash (user\_name) +Hash (method\_Of\_Access));

3: Compute merkle\_root = Hash ((dataset\_name and operation) and (user\_name and method\_Of\_access)) = Hash (Hash (dataset\_name and operation) +Hash (user\_name and method\_Of\_access));

**Description:** This algorithm explains the calculation of Merkle Root. The hash of dataset name and the hash of command line operation are concatenated and hashed again. The hash of username and the hash of method of access are concatenated and hashed again. The two hash values created are concatenated and hashed again. The output is Merkle Root.

## **Algorithm 2** block\_creation

**Input:** version\_number, timestamp, merkle\_root, prev\_block\_header, data\_size, dataset\_name, operation, user\_name, method\_of\_access;

**Output:** block\_header;

1: Compute merkle\_root = merkle\_root\_calc (dataset\_name, operation, user\_name, method\_of\_access);

2: Compute block\_header=Hash (LittleEndian (Hex (version\_number)) LittleEndian (Hex (timestamp)) +LittleEndian (Hex (data\_size)) +merkle\_root + prev\_block\_header);

if (blockType==Genesis block)

prev\_block\_header=0

3: insert to database= Encrypted (dataset\_name, operation, user\_name, method\_of\_access), LittleEndian (Hex (version\_number), LittleEndian (Hex (timestamp)), LittleEndian (Hex (data\_size), merkle\_root, block\_header);

**Description:** This algorithm explains the creation of blocks in a blockchain. Dataset name, username, command line operation and method of access are used to calculate the Merkle root. When a block is being created, a version number is calculated. A timestamp of creation is captured. The number of bits of data (data used for calculating Merkle Root) is calculated. The version number, the timestamp and the number of bits are converted into hexadecimal format and then converted into Little Endian format. Finally these three values are concatenated and hashed. If a genesis block is being created, the

hash value created is concatenated with the Merkle root. This provides the blockheader of the genesis block. For the subsequent blocks the Merkle root, the hash value of version number, timestamp and number of bits and the blockheader of previous block are concatenated to create the block header of the current block.

**Algorithm 3** combine\_blockchains

**Input:** dataset\_name (1 to n), operations, version\_number, data\_size, timestamp, user\_name, method\_of\_access, prev\_block\_header (1 to n);

**output:** block\_header;

1: merkle\_root= merkle\_root\_calc ((dataset\_name1+dataset\_name2+....+ dataset\_namen), operations, user\_name, method\_of\_access);

2: block\_header = Hash(LittleEndian(Hex(version\_number))+LittleEndian(Hex(data\_size)) + LittleEndian (Hex(timestamp) + merkle\_root + Hash(prev\_block\_header1 +prev\_block\_header2 +.....prev\_block\_header<sub>n</sub>)));

**Description:** If two datasets are accessed at the same time using Map Reduce, then the blockchains of the two datasets are combined to create a third blockchain. The Merkle root is recomputed based on the two separate blockchains, the version number is calculated based on the two blockchains. Timestamp and the number of bits are calculated.

**Algorithm 4** delete\_dataset

**input:** dataset\_name1, dataset\_name2, dataset\_name12;

1: Dataset\_name12 = dataset\_name1+dataset\_name2;

2: if dataset\_name2 present == dataset\_name2 blockchain

    then do nothing

    else dataset\_name2 deleted

    dataset\_name12 = dataset1;

**Description:** If a dataset in Hadoop File System (HDFS) is deleted, the blockchain of that dataset will be deleted as well.

**Algorithm 5** validation\_blockchain

**input(from database):** prev\_block\_header,

LittleEndian(Hex(version\_number),LittleEndian(Hex(timestamp)),LittleEndian(Hex(data\_size)),merkle\_root, block\_header;

**output:** boolean\_flag;

1: if (Hash (prev\_block\_header+LittleEndian (Hex (version\_number)

    +LittleEndian (Hex (timestamp) +LittleEndian (Hex (data\_size)



```
+merkle_root)) ==block_header) {boolean_flag=true} else {boolean_flag=false};
```

```
if (boolean_flag = false) {delete_block}
```

**Description:** The blockchains are validated at regular intervals. The values are recomputed and if an anomaly is detected in any block of any blockchain, it is reported to the administrator in the form of a log file with the block details. The corrupt block is deleted.

## CHAPTER IV

### **Implementation**

Usage tracker is a program which tracks users who access datasets in the Hadoop file system. The implementation of Blockchain Data Usage Tracker begins with fetching the name of the dataset accessed, name of the user who accessed the dataset, the method using which the dataset was accessed (Command Line Interface or Map Reduce) and the command line operation using which the dataset was accessed (cat, copy, move, put) .In order to extract the values related to the dataset, snippets of extra code was added to the Hadoop source code in order to fetch the values mentioned above and pass them as arguments to the Usage Tracker. The username, the dataset name, the method used to access the dataset (Command Line Interface) and the command line operation performed to access the dataset (cat, copy, move, put) is being captured in the Hadoop source code.

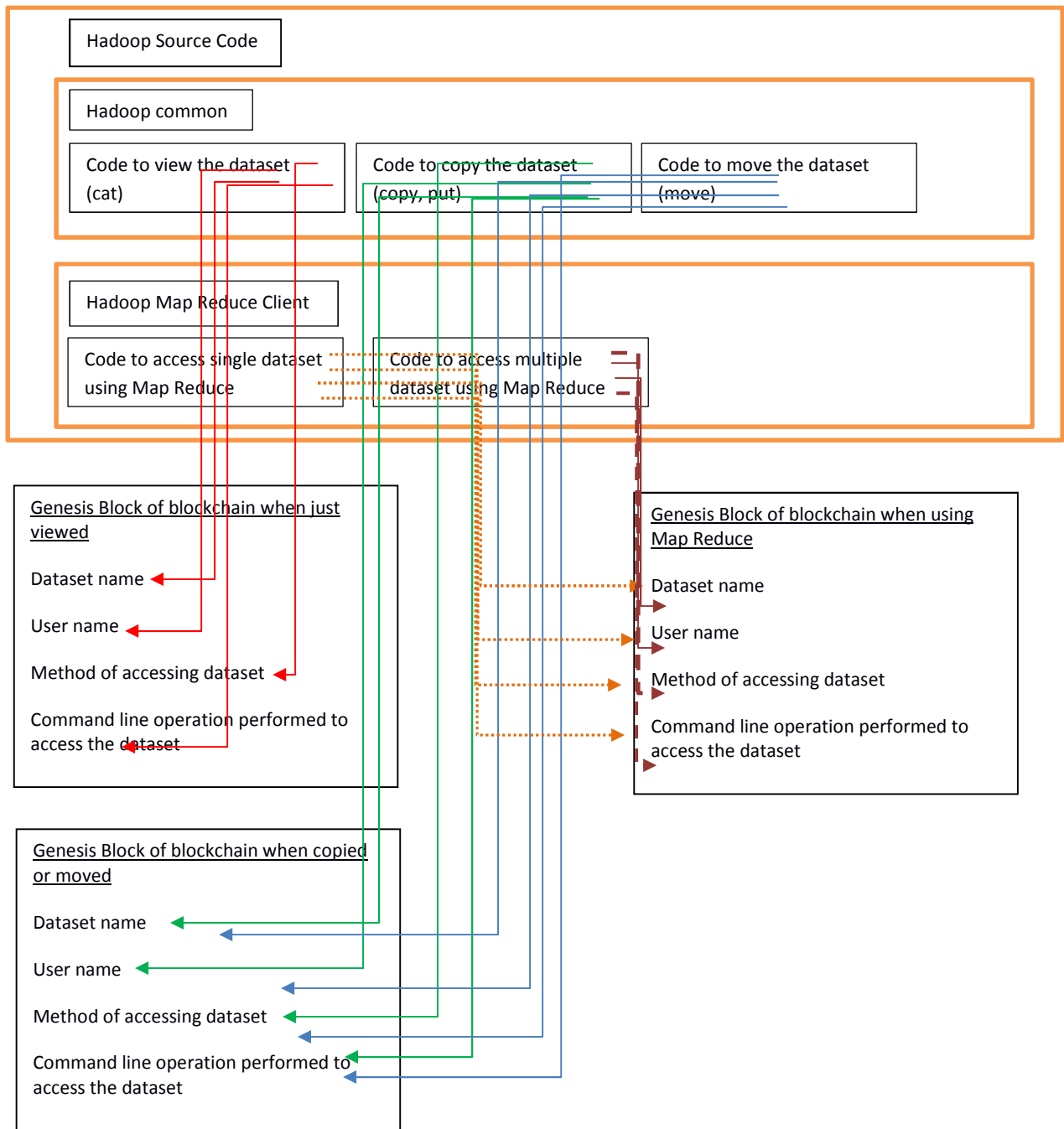


Figure 15: Blockchain Data Usage Tracker with Hadoop Source code origins

The following CLI (Command Line Interface) operations were recorded:

1. `copyFromLocal`: This operation copies the data files from the local system to the Hadoop file system. This operation is responsible for the creation of the Genesis block in the blockchain because it creates a new dataset.
2. `moveFromLocal`: This operation is again responsible for creation of the Genesis block. It copies the files from local system to the Hadoop file system and deletes the original file in the local system.
3. `cat`: This operation is used to read the content of the dataset. Its primary purpose is creation of blocks after the Genesis block. Every time a dataset is read using this command, a new block is added to the blockchain.
4. `put`: This operation is same as `copyFromLocal`.
5. `copyToLocal`: This operation is the reverse of `copyFromLocal`. It copies the dataset from the Hadoop file system to local file system. It is responsible for adding blocks to the blockchain after the genesis block.
6. `get`: It is same as `copyToLocal`.

The dataset accessed information using Map Reduce is recorded as well. Data for single dataset access and multiple dataset access is recorded. If multiple datasets are accessed and the datasets already have blockchains, this operation would amalgamate the two blockchains to create a third blockchain.

The program to create the blockchain and blocks is being called from the Hadoop source code using common daemons after the required values are extracted. A daemon is a computer program that runs as a background process, executing tasks on a predefined schedule or in response to particular events, or in response to requests for information or services from other programs. In order to use the daemon, the Apache Common Daemon libraries are used. In this application, the daemon is called from Hadoop source code by executing a shell script. This shell script has the location of Common Daemon libraries, java class path, location of the Blockchain program (jar file) location, the location of log files. It also has the commands for Start, Stop and Restart. In order to start the daemon, script “start” command has to be executed. Similarly once the daemon task is performed, script “stop” is executed which will terminate the daemon. The values which are fetched from the source code are passed as arguments along with the daemon.

As soon as the information from the Hadoop source code is available, the daemon is called and it starts the creation of the block while the Hadoop process is continuing in the background.

The blockchain creation process starts by fetching the dataset related values from the daemon. Once the program has all the values it needs, these values are encrypted using an encryption key and an encryption algorithm and the same values are hashed using a hashing algorithm. The encryption key is created by the individual who has access to the

plain text values. The hashed values are used to create the Merkle root by concatenating them and hashing them again.

The current timestamp, the number of bits of information and the version number using a customized algorithm are calculated. The above mentioned three values are converted into hexadecimal format. The output (hexadecimal values) are then converted into little endian format and finally the output is encrypted using an encryption algorithm.

The hexadecimal-little endian-encrypted values of version number, time stamp and number of bits are concatenated along with the Merkle root and hashed to create the blockheader of the genesis block of the blockchain. In the subsequent blocks along with the above mentioned concatenation, the blockheader of the previous block is concatenated and then hashed to create the blockheader which will be used in the following blocks.

Validation of blockchain is done at regular intervals. A Cron Job is being using which starts the process at every regular intervals. It re-computes the values of each block and compares these values to the previous block. If there is a mismatch in any of the values, it is recorded and a report is generated which specifies which values in which block were modified and the decrypted values of the Username, dataset name, the method of access and the operation performed.

The blockchain data usage tracker was implemented and tested on a standalone machine first with 1 namenode and 1 datanode single cluster. The system had 3 GB RAM and 250GB hard drive. Hadoop File System (HDFS) version 2.7.3 along with Hive 2.1.0 (Hiveserver2) were used. The blockchain data usage tracker at this point was tested using 10 datasets with sizes varying between 1000 and 10000 lines of data. Each blockchain had number of blocks varying between 10 and 15.

Once testing on a standalone machine was completed, the blockchain data usage tracker was implemented and tested on Amazon Web Service EC<sub>2</sub> cloud platform. 2 Hadoop clusters were used. One was the main cluster and the other was the backup cluster. Each Hadoop cluster had 4 separate instances with 8GB hard drive and 4GB RAM. 1 instance was used as namenode and 3 other instances were used as datanodes. Hadoop 2.7.3 and Hive 2.1.0 (Hiveserver2) was used. In Amazon Web Service EC<sub>2</sub> the blockchain data usage tracker was tested with 30 different datasets, their sizes varying between 10000 and 100000 lines of data. The number of blocks created varied between 30-50 blocks.

## **Findings and Results**

The data usage tracker works as expected. When an existing dataset is accessed for the first time, a new blockchain is created. Every time the dataset is accessed again a new block is added to the blockchain.

The average execution time for creation of Genesis block/Blockchain creation in the AWS EC2 cluster with 4GB RAM and 8GB hard drive system is 4000 milliseconds. 2 processes (HDFS and Hiveserver2) were running simultaneously. The sample for this result is 25 executions. The size of datasets varied between 10000-100000 lines.

If a dataset is deleted, its corresponding blockchain is deleted as well. The average execution time for deletion of a blockchain in the AWS EC2 cluster with 4GB RAM and 8GB hard drive system is 1000 milliseconds. The sample for this result is 10 executions. The size of datasets varied between 10000-100000 lines.

At every regular intervals the blockchain is validated for validity. The values are recomputed with the previous values and if there is an anomaly, it is recorded in the report.

The average execution time for validation of 16 blockchains with an average of 40 blocks in each blockchain in the AWS EC2 cluster with 4GB RAM and 8GB hard drive system is 19500 milliseconds.



## Screenshots:

```
ubuntu@ip-172-31-24-71: ~
ubuntu@ip-172-31-24-71:~$ hdfs dfs -ls /blockchain
ubuntu@ip-172-31-24-71:~$
```

Figure 16: No blockchains initially

This screenshot portrays that there are no blockchains initially. The hdfs blockchain folder is empty.

```
File Edit View Search Terminal Help
ubuntu@ip-172-31-24-71:~$ hdfs dfs -ls /user/ubuntu
Found 2 items
drwxr-xr-x - ubuntu supergroup          0 2017-12-10 23:24 /user/ubuntu/multipleInput
drwxr-xr-x - ubuntu supergroup          0 2017-12-10 23:25 /user/ubuntu/wordcount
ubuntu@ip-172-31-24-71:~$
```

Figure 17: No datasets initially

This screenshot portrays that initially there are no datasets in the system. That is, there are no files in the Hadoop file system.

```

ubuntu@ip-172-31-24-71: ~
ubuntu@ip-172-31-24-71:~$ hdfs dfs -copyFromLocal /home/ubuntu/inputs/input1000 /user/ubuntu
ubuntu@ip-172-31-24-71:~$ hdfs dfs -ls /blockchain
Found 2 items
drwxr-xr-x - ubuntu supergroup      0 2017-12-10 23:37 /blockchain/input1000_blockc
-rw-r--r--  1 ubuntu supergroup     17 2017-12-10 23:37 /blockchain/list
ubuntu@ip-172-31-24-71:~$ hdfs dfs -ls /blockchain/input1000_blockc
Found 1 items
-rw-r--r--  1 ubuntu supergroup     460 2017-12-10 23:37 /blockchain/input1000_blockc/input1000_blockc
ubuntu@ip-172-31-24-71:~$

```

Figure 18: Create first dataset and blockchain

This screenshot portrays that a new Hadoop file system file (dataset) was created (input1000) by copying it from a local system. Once the dataset (input1000) was created, the blockchain related to that was created as well. The blockchain is created by appending the name of the dataset with “**\_blockc**” and putting it in a folder with the same name. In this case a folder named “**input1000\_blockc**” was created in the blockchain folder and inside the input1000\_blockc, a Hadoop file with the name “**input1000\_blockc**” was created. The file has the genesis block with size 460 Bytes. The contents of a blockchain would be shown in another screenshot.

```

ubuntu@ip-172-31-24-71: ~
ubuntu@ip-172-31-24-71:~$ hdfs dfs -copyFromLocal /home/ubuntu/inputs/input10000 /user/ubuntu/multipleInput
ubuntu@ip-172-31-24-71:~$ hdfs dfs -copyFromLocal /home/ubuntu/inputs/input12000 /user/ubuntu/multipleInput
ubuntu@ip-172-31-24-71:~$ hdfs dfs -ls /blockchain
Found 4 items
drwxr-xr-x - ubuntu supergroup      0 2017-12-10 23:39 /blockchain/input10000_blockc
drwxr-xr-x - ubuntu supergroup      0 2017-12-10 23:37 /blockchain/input1000_blockc
drwxr-xr-x - ubuntu supergroup      0 2017-12-10 23:39 /blockchain/input12000_blockc
-rw-r--r--  1 ubuntu supergroup     53 2017-12-10 23:39 /blockchain/list
ubuntu@ip-172-31-24-71:~$

```

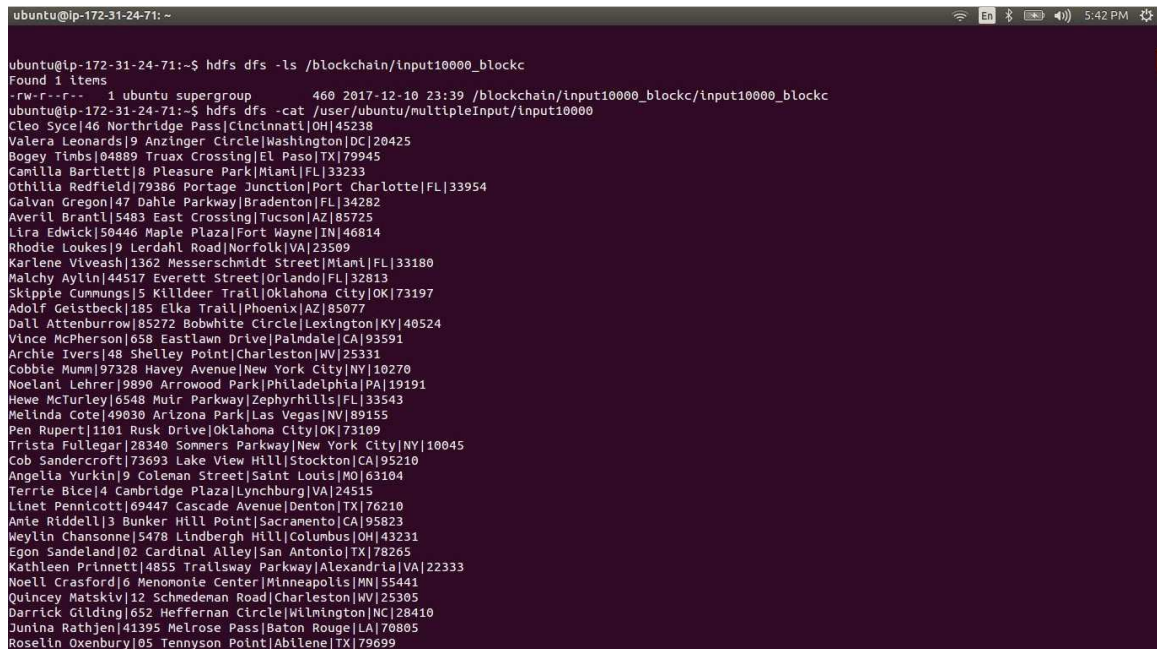
```

ubuntu@ip-172-31-24-71: ~
ubuntu@ip-172-31-24-71:~/MR$ hdfs dfs -ls /user/ubuntu/multipleInput
Found 2 items
-rw-r--r--  3 ubuntu supergroup    524730 2017-12-10 23:39 /user/ubuntu/multipleInput/input10000
-rw-r--r--  3 ubuntu supergroup    629676 2017-12-10 23:39 /user/ubuntu/multipleInput/input12000
ubuntu@ip-172-31-24-71:~/MR$

```

Figure 19: Inserting more files in hdfs

The screenshot portrays more files being added to Hadoop file system by copying them from local. These files are being added to a folder called multipleInput which later will be used for Map Reduce purposes. The 2 new files (input10000 and input12000) creates two new blockchains named: **“input10000\_blockc and input12000\_blockc”**.



```
ubuntu@ip-172-31-24-71: ~  
ubuntu@ip-172-31-24-71:~$ hdfs dfs -ls /blockchain/input10000_blockc  
Found 1 items  
-rw-r--r-- 1 ubuntu supergroup 460 2017-12-10 23:39 /blockchain/input10000_blockc/input10000_blockc  
ubuntu@ip-172-31-24-71:~$ hdfs dfs -cat /user/ubuntu/multipleInput/input10000  
Cleo Syce|46 Northridge Pass|Cincinnati|OH|45238  
Valera Leonards|9 Anzinger Circle|Washington|DC|20425  
Bogey Timbs|04889 Truax Crossing|El Paso|TX|79945  
Camilla Bartlett|8 Pleasure Park|Miami|FL|33233  
Othilia Redfield|79386 Portage Junction|Port Charlotte|FL|33954  
Galvan Gregon|47 Dahle Parkway|Bradenton|FL|34282  
Averil Brantl|5483 East Crossing|Tucson|AZ|85725  
Lira Edwick|50446 Maple Plaza|Fort Wayne|IN|46814  
Rhodie Loukes|9 Lerdahl Road|Norfolk|VA|23509  
Karlene Viveash|1362 Messerschmidt Street|Miami|FL|33180  
Malchy Aylin|44517 Everett Street|Orlando|FL|32813  
Skiptie Cummings|5 Killdeer Trail|Oklahoma City|OK|73197  
Adolf Geistbeck|185 Elka Trail|Phoenix|AZ|85077  
Dall Attenburrow|85272 Bobwhite Circle|Lexington|KY|40524  
Vince McPherson|658 Eastlawn Drive|Palmdale|CA|93591  
Archie Ivers|48 Shelley Point|Charleston|WV|25331  
Cobbie Mumm|97328 Havey Avenue|New York City|NY|10270  
Noelani Lehrer|9890 Arrowood Park|Philadelphia|PA|19191  
Hewe McTurley|6548 Mutr Parkway|Zephyrhills|FL|33543  
Melinda Cote|49030 Arizona Park|Las Vegas|NV|89155  
Pen Rupert|1108 Rusk Drive|Oklahoma City|OK|73109  
Trista Fullegar|28340 Sommers Parkway|New York City|NY|10045  
Cob Sandercroft|73693 Lake View Hill|Stockton|CA|95210  
Angelia Yurkin|9 Coleman Street|Saint Louis|MO|63104  
Terrie Bice|4 Cambridge Plaza|Lynchburg|VA|24515  
Linet Pennicott|69447 Cascade Avenue|Denton|TX|76210  
Amie Riddell|3 Bunker Hill Point|Sacramento|CA|95823  
Weylin Chansonne|5478 Lindbergh Hill|Columbus|OH|43231  
Egon Sandeland|02 Cardinal Alley|San Antonio|TX|78265  
Kathleen Prinnett|4855 Trailsway Parkway|Alexandria|VA|22333  
Noell Crasford|6 Menomone Center|Minneapolis|MN|55441  
Quincey Matskiv|12 Schmedenan Road|Charleston|WV|25305  
Darrick Gilding|652 Heffernan Circle|Wilmington|NC|28410  
Junina Rathjen|41395 Melrose Pass|Baton Rouge|LA|70805  
Roselin Oxenbury|05 Tennyson Point|Abilene|TX|79699
```

Figure 20: Viewing a dataset

The screenshot portrays that the blockchain **“input10000\_blockc”** has the genesis block of size 460 Bytes. Cat command is executed for the dataset input10000. This would display the contents of the dataset and create a new block to the blockchain.

```
ubuntu@ip-172-31-24-71: ~  
ubuntu@ip-172-31-24-71:~$ hdfs dfs -ls /blockchain/input10000_blockc  
Found 1 items  
-rw-r--r-- 1 ubuntu supergroup 920 2017-12-10 23:42 /blockchain/input10000_blockc/input10000_blockc  
ubuntu@ip-172-31-24-71:~$
```

Figure 21: Second block added to blockchain

The size of blockchain “**input10000\_blockc**” increased to 920 bytes with second block being of 460 Bytes.

```
ubuntu@ip-172-31-24-71: ~/MR  
ubuntu@ip-172-31-24-71:~/MR$ ls  
MWordCount.java SWordCount.java  
ubuntu@ip-172-31-24-71:~/MR$
```

Figure 22: Map Reduce programs

This screenshot portrays that there are two map reduce programs.

MWordCount.java takes multiple input files, counts the words in all of them, and returns a single output.

SWordCount.java takes single input file, counts the words and returns a single output.

```

ubuntu@ip-172-31-24-71: ~/MR
ubuntu@ip-172-31-24-71:~/MR$ export HADOOP_CLASSPATH=/usr/lib/jvm/java-8-openjdk-and64/lib/tools.jar
ubuntu@ip-172-31-24-71:~/MR$ hadoop com.sun.tools.javac.Main SWordCount.java
Note: SWordCount.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
ubuntu@ip-172-31-24-71:~/MR$ jar cf swc.jar SWordCount*.class
ubuntu@ip-172-31-24-71:~/MR$ hdfs dfs -copyFromLocal /home/ubuntu/inputs/input5000 /user/ubuntu
ubuntu@ip-172-31-24-71:~/MR$ hadoop jar swc.jar SWordCount /user/ubuntu/input5000 /user/ubuntu/wordcount/output
17/12/10 23:50:03 INFO client.RMProxy: Connecting to ResourceManager at ec2-18-218-1-144.us-east-2.compute.amazonaws.com/172.31.24.71:8032
17/12/10 23:50:04 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement the Tool interface and execute your a
pplication with ToolRunner to remedy this.
17/12/10 23:50:04 INFO Input.FileInputFormat: Total input paths to process : 1
17/12/10 23:50:04 INFO mapreduce.JobSubmitter: number of splits:1
17/12/10 23:50:04 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1512947981511_0007
17/12/10 23:50:04 INFO Impl.YarnClientImpl: Submitted application application_1512947981511_0007
17/12/10 23:50:04 INFO mapreduce.Job: The url to track the job: http://ec2-18-218-1-144.us-east-2.compute.amazonaws.com:8088/proxy/application_15129479
81511_0007/
17/12/10 23:50:04 INFO mapreduce.Job: Running job: job_1512947981511_0007
17/12/10 23:50:11 INFO mapreduce.Job: Job job_1512947981511_0007 running in uber mode : false
17/12/10 23:50:11 INFO mapreduce.Job: map 0% reduce 0%
17/12/10 23:50:15 INFO mapreduce.Job: map 100% reduce 0%
17/12/10 23:50:21 INFO mapreduce.Job: map 100% reduce 100%
17/12/10 23:50:21 INFO mapreduce.Job: Job job_1512947981511_0007 completed successfully
17/12/10 23:50:21 INFO mapreduce.Job: Counters: 49
File System Counters
  FILE: Number of bytes read=66259
  FILE: Number of bytes written=372603
  FILE: Number of read operations=0
  FILE: Number of large read operations=0
  FILE: Number of write operations=0
  HDFS: Number of bytes read=262512
  HDFS: Number of bytes written=52619
  HDFS: Number of read operations=0
  HDFS: Number of large read operations=0
  HDFS: Number of write operations=2
Job Counters
  Launched map tasks=1
  Launched reduce tasks=1
  Data-local map tasks=1
  Total time spent by all maps in occupied slots (ms)=34848

```

Figure 23: Map Reduce for single input

The screenshot portrays that the Map reduce program SWordCount.java is compiled. A new dataset named “input5000” is added to Hadoop and Map Reduce is performed on it.

```

ubuntu@ip-172-31-24-71: ~/MR
ubuntu@ip-172-31-24-71:~/MR$ hdfs dfs -ls /blockchain
ind 5 items
-r-xr-xr-x - ubuntu supergroup 0 2017-12-10 23:39 /blockchain/input10000_blockc
-r-xr-xr-x - ubuntu supergroup 0 2017-12-10 23:37 /blockchain/input1000_blockc
-r-xr-xr-x - ubuntu supergroup 0 2017-12-10 23:39 /blockchain/input12000_blockc
-r-xr-xr-x - ubuntu supergroup 0 2017-12-10 23:49 /blockchain/inputs5000_blockc
-r--r-- 1 ubuntu supergroup 70 2017-12-10 23:49 /blockchain/list
ubuntu@ip-172-31-24-71:~/MR$

```

Figure 24: New blockchain created

The screenshot portrays that a new blockchain named “input5000\_blockc” is created after the dataset “input5000” was accessed using Map Reduce.

```

ubuntu@ip-172-31-24-71: ~/MR
ubuntu@ip-172-31-24-71:~/MRS export HADOOP_CLASSPATH=/usr/lib/jvm/java-8-openjdk-amd64/lib/tools.jar
ubuntu@ip-172-31-24-71:~/MRS hadoop com.sun.tools.javac.Main MWordCount.java
ubuntu@ip-172-31-24-71:~/MRS jar cf mwc.jar MWordCount*.class
ubuntu@ip-172-31-24-71:~/MRS hadoop jar mwc.jar MWordCount /user/ubuntu/multipleInput /user/ubuntu/wordcount/multiOutput
17/12/10 23:54:03 INFO client.RMProxy: Connecting to ResourceManager at ec2-18-218-1-144.us-east-2.compute.amazonaws.com/172.31.24.71:8032
17/12/10 23:54:04 INFO input.FileInputFormat: Total input paths to process : 2
17/12/10 23:54:04 INFO input.CombineFileInputFormat: DEBUG: Terminated node allocation with : CompletedNodes: 3, size left: 1154406
17/12/10 23:54:04 INFO mapreduce.JobSubmitter: number of splits:1
17/12/10 23:54:04 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1512947981511_0009
17/12/10 23:54:04 INFO impl.YarnClientImpl: Submitted application application_1512947981511_0009
17/12/10 23:54:04 INFO mapreduce.Job: The url to track the job: http://ec2-18-218-1-144.us-east-2.compute.amazonaws.com:8088/proxy/application_1512947981511_0009/
17/12/10 23:54:04 INFO mapreduce.Job: Running job: job_1512947981511_0009
17/12/10 23:54:10 INFO mapreduce.Job: Job job_1512947981511_0009 running in uber mode : false
17/12/10 23:54:10 INFO mapreduce.Job: map 0% reduce 0%

```

Figure 25: Map Reduce on Multiple inputs

The screenshot portrays that Map Reduce program MWordCount.java was executed with input files in multipleInput folder: input10000 and input12000 (Figure 19). The program counts the words of both the files and provides a single output file.

```

ubuntu@ip-172-31-24-71: ~/MR
ubuntu@ip-172-31-24-71:~/MRS$ hdfs dfs -ls /blockchain
Found 6 items
drwxr-xr-x - ubuntu supergroup          0 2017-12-10 23:39 /blockchain/input10000_blockc
drwxr-xr-x - ubuntu supergroup          0 2017-12-10 23:54 /blockchain/input10000_input12000_blockc
drwxr-xr-x - ubuntu supergroup          0 2017-12-10 23:37 /blockchain/input1000_blockc
drwxr-xr-x - ubuntu supergroup          0 2017-12-10 23:39 /blockchain/input12000_blockc
drwxr-xr-x - ubuntu supergroup          0 2017-12-10 23:49 /blockchain/input5000_blockc
-rw-r--r--  1 ubuntu supergroup        99 2017-12-10 23:54 /blockchain/list
ubuntu@ip-172-31-24-71:~/MRS$

```

Figure 26: A blockchain with multiple datasets

The screenshot portrays that when multiple files were accessed using Map Reduce, a blockchain was created with the names of the 2 files separated by “\_”. In this case, Map Reduce accessed two datasets: input10000 and input12000. The resulting blockchain created is **input10000\_input12000\_blockc**.



```

ubuntu@ip-172-31-24-71: ~/MR
ubuntu@ip-172-31-24-71:~/MR$ hdfs dfs -rm /user/ubuntu/multipleInput/input12000
17/12/10 23:56:23 INFO fs.TrashPolicyDefault: Namenode trash configuration: Deletion interval = 0 minutes, Emptier interval = 0 minutes.
Deleted /user/ubuntu/multipleInput/input12000
ubuntu@ip-172-31-24-71:~/MR$ hdfs dfs -ls /blockchain
Found 4 items
drwxr-xr-x - ubuntu supergroup      0 2017-12-10 23:39 /blockchain/input10000_blockc
drwxr-xr-x - ubuntu supergroup      0 2017-12-10 23:37 /blockchain/input1000_blockc
drwxr-xr-x - ubuntu supergroup      0 2017-12-10 23:49 /blockchain/input5000_blockc
-rw-r--r--  1 ubuntu supergroup     99 2017-12-10 23:54 /blockchain/list
ubuntu@ip-172-31-24-71:~/MR$ hdfs dfs -ls /blockchain/input10000_blockc
Found 1 items
-rw-r--r--  1 ubuntu supergroup     920 2017-12-10 23:42 /blockchain/input10000_blockc/input10000_blockc

```

Figure 27: Delete dataset

The screenshot portrays that the dataset input12000 was deleted. Along with input12000, the corresponding blockchain (**input12000\_blockc**) was deleted as well. There was another blockchain was created using input12000, i.e. “**input10000\_input12000\_blockc**”. It was deleted as well and the content was copied to the remaining blockchain “**input10000\_blockc**”.

```

ubuntu@ip-172-31-24-71: ~
ubuntu@ip-172-31-24-71:~$ hdfs dfs -cat /user/ubuntu/input5000
Cleo Syce|46 Northridge Pass|Cincinnati|OH|45238
Valera Leonards|9 Anzinger Circle|Washington|DC|20425
Bogey Timbs|04889 Truax Crossing|El Paso|TX|79945
Camilla Bartlett|8 Pleasure Park|Miami|FL|33233
Othilia Redfield|79386 Portage Junction|Port Charlotte|FL|33954
Galvan Gregon|47 Dahle Parkway|Bradenton|FL|34282
Averil Brantl|5483 East Crossing|Tucson|AZ|85725
Lira Edwick|50446 Maple Plaza|Fort Wayne|IN|46814
Rhodie Loukes|9 Lerdahl Road|Norfolk|VA|23509
Karlene Viveash|1362 Messerschmidt Street|Miami|FL|33180
Malchy Aylin|44517 Everett Street|Orlando|FL|32813
Skippie Cummings|5 Kildeer Trail|Oklahoma City|OK|73197
Adolf Geistbeck|185 Elka Trail|Phoenix|AZ|85077
Dall Attenburrrow|85272 Bobwhite Circle|Lexington|KY|40524
Vince McPherson|658 Eastlawn Drive|Palmdale|CA|93591
Archie Ivers|48 Shelley Point|Charleston|WV|25331
Cobbie Mumm|97328 Havey Avenue|New York City|NY|10270
Noelani Lehrer|9890 Arrowood Park|Philadelphia|PA|19191
Hewe McTurley|6548 Muir Parkway|Zephyrhills|FL|33543
Melinda Cote|49030 Arizona Park|Las Vegas|NV|89155
Pen Rupert|1101 Rusk Drive|Oklahoma City|OK|73109
Trista Fullagar|28340 Sommers Parkway|New York City|NY|10045
Cob Sandercroft|73693 Lake View Hill|Stockton|CA|95210
Angelia Yurkin|9 Coleman Street|Saint Louis|MO|63104
Terrie Bice|4 Cambridge Plaza|Lynchburg|VA|24515
Linnet Pennicott|69447 Cascade Avenue|Denton|TX|76210
Amie Riddell|3 Bunker Hill Point|Sacramento|CA|95823
Weylin Chansonne|5478 Lindbergh Hill|Columbus|OH|43231
Egon Sandeland|02 Cardinal Alley|San Antonio|TX|78265
Kathleen Prinnett|4855 Trailway Parkway|Alexandria|VA|22333
Noell Crasford|6 Menomonie Center|Minneapolis|MN|55441
Quincey Matsklv|12 Schmedeman Road|Charleston|WV|25385
Darrick Gliding|652 Heffernan Circle|Wilmington|NC|28410
Junina Rathjen|41395 Malrose Pass|Baton Rouge|LA|70805
Roselin Oxenbury|05 Tennyson Point|Abilene|TX|79699
Crichton Mahon|940 American Street|El Paso|TX|88519

```

Figure 28: Viewing the dataset

The screenshot portrays that dataset input5000 was viewed. This would add another block to the blockchain.

```
ubuntu@ip-172-31-24-71: ~  
ubuntu@ip-172-31-24-71:~$ hdfs dfs -ls /blockchain/input5000_blockc  
Found 1 items  
-rw-r--r--  1 ubuntu supergroup    1380 2017-12-11 00:00 /blockchain/input5000_blockc/input5000_blockc  
ubuntu@ip-172-31-24-71:~$
```

Figure 29: New block added to blockchain

The screenshot portrays that when the dataset input5000 was accessed, a new block was added to the blockchain. The blockchain “**input5000\_blockc**” now has 3 blocks.

The next few steps are performed to attack the blockchain, so that it can be modified which is an unauthorized access. Since the system has all the access and all the privileges, it can perform any actions to modify the data without leaving much evidence.



```
1, JUIBLJf0qNXU9nr/s1AUzQ==, xsfGfTi6kCVKhWcSRPea0kcS/SPJwT62L/4yF83qpE=, HTydcGt5M03jtrSPAJQUG==, ltPq5qll+r/Ub+wg9cBycrljDF3wQo
+ItkDXzM4TA3vfoGfe0k69sccsfKEuL+A6ucnqghyYQhj+w5L9tpYx0kcS/SPJwT62L/4yF83qpE=, 8spGh7l4yP4VU4+jB0j0y7VI2ghMfvhpUbyPcNJFFLfbqbank6YTLfNGexPn9tszYGxc/
UFaWiDLJnERIAxwRekcS/SPJwT62L/4yF83qpE=, ePveZL+ZSTu0Ip3PnQ0khw==, pG+/TxE
+sN3cE41jMlLSzw==, QkhyGIIn0F55rzTXnSkGLrTGpvmIQqgDIIWUmDsU5G0=, eSHnoJASvCLXlFngTAHzg==, Y0weY1V90zGgb6Iom41M9A==
2, 6dARV8ThwRRXHG0aw8nzQ==, xAokZktaEwGX9CJz+EzU0kcS/SPJwT62L/4yF83qpE=, BN+oN1hwu3tCtwL7S0z0Bw==, 98wxsW0Cuzthv/
cZQbWnB7zJR0Kduke1Qdwqh3NRA8B0Cj0Ic2Cud7nedTjGuk0Lch5nh0B6PIzwxDAW550kcS/SPJwT62L/4yF83qpE=, YnzAg15q5fBGZLZ5+LUoA1U
+laFhrIkmpaug55IBusXJGU606MavAGWbyFqk100u03tN0buwtMleWCVctgFLokcS/SPJwT62L/4yF83qpE=, ePveZL+ZSTu0Ip3PnQ0khw==, pG+/TxE
+sN3cE41jMlLSzw==, QkhyGIIn0F55rzTXnSkGLrTGpvmIQqgDIIWUmDsU5G0=, InqSgYDLZNL8jDBZjn7zg==, SPfqVlW661lUaX5t25utLA==
3, xEF5J0QTJHglTEG0NBj3rA==, qJULbsxhLlS84yb51kbnpekcS/SPJwT62L/4yF83qpE=, HTydcGt5M03jtrSPAJQUG==, ltPq5qll+r/Ub+wg9cBycrljDF3wQo
+ItkDXzM4TA3vfoGfe0k69sccsfKEuL+A6ucnqghyYQhj+w5L9tpYx0kcS/SPJwT62L/4yF83qpE=, j1EXCPqMs7Tl3uItYndPE/7eJ0ub7jvQJREJekU0LPjndjXhRN08Q+Uwdb
+k8QEhkx18ZD0llhAvFmZf4mME+0kcS/SPJwT62L/4yF83qpE=, ePveZL+ZSTu0Ip3PnQ0khw==, pG+/TxE
+sN3cE41jMlLSzw==, QkhyGIIn0F55rzTXnSkGLrTGpvmIQqgDIIWUmDsU5G0=, eSHnoJASvCLXlFngTAHzg==, txMQoLZtnM4C4nfjSuzcWw==
```

Figure 30: Blockchain data

The blockchain “**input5000\_blockc**” was copied to the local file system. This blockchain has 3 blocks. The fields in the blocks are separated by “,”. There are 10 fields in each block. The field marked by red will be modified.

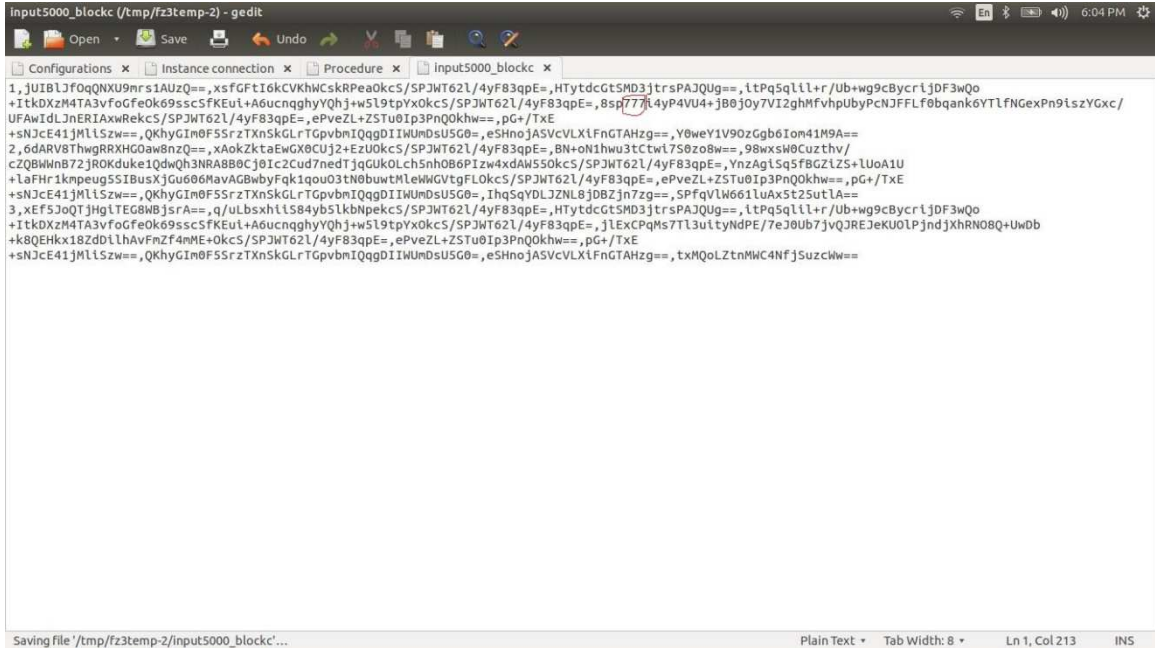


Figure 31: Modified block

A part of the genesis block for blockchain “**input5000\_blockc**” has been modified.

Originally (Figure 30) it was “+w7” and it was modified to “777”.

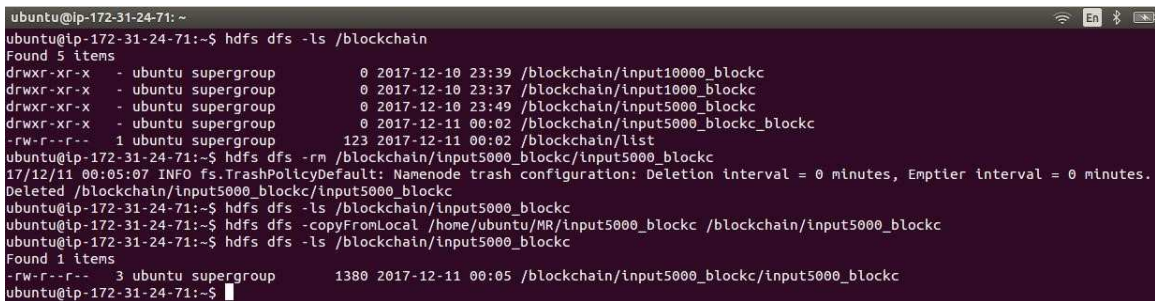
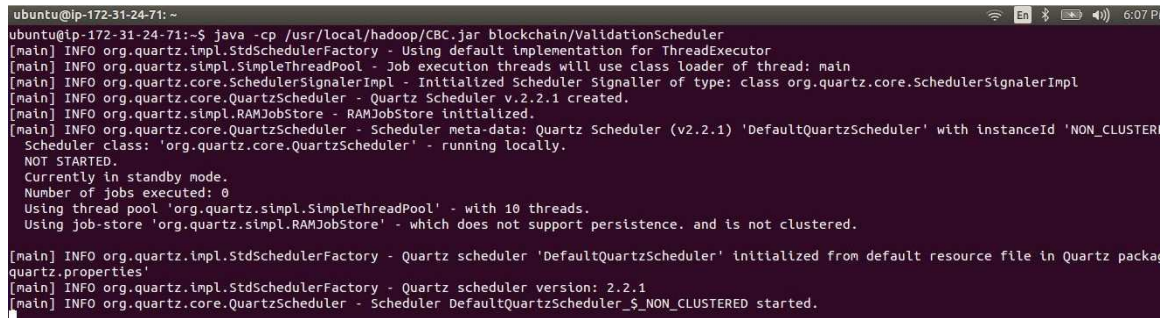


Figure 32: Changing the blockchain

The blockchain “**input5000\_blockc**” is modified by the following steps:

1. The blockchain file (**input5000\_blockc**) was copied to the local file system (figure 30).
2. It was edited in the local file system and some of the values were modified (figure 31).
3. The blockchain file (**input5000\_blockc**) in Hadoop file system was deleted and replaced with the modified file from local file system using **copyFromLocal**.

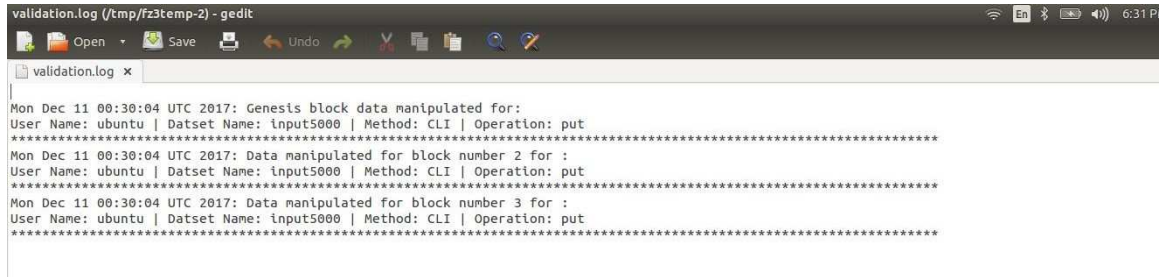
The blockchain is now modified. Next we execute the blockchain validation.



```
ubuntu@ip-172-31-24-71: ~  
ubuntu@ip-172-31-24-71:~$ java -cp /usr/local/hadoop/CBC.jar blockchain/ValidationScheduler  
[main] INFO org.quartz.impl.StdSchedulerFactory - Using default implementation for ThreadExecutor  
[main] INFO org.quartz.simpl.SimpleThreadPool - Job execution threads will use class loader of thread: main  
[main] INFO org.quartz.core.SchedulerSignalerImpl - Initialized Scheduler Signaller of type: class org.quartz.core.SchedulerSignalerImpl  
[main] INFO org.quartz.core.QuartzScheduler - Quartz Scheduler v.2.2.1 created.  
[main] INFO org.quartz.simpl.RAMJobStore - RAMJobStore initialized.  
[main] INFO org.quartz.core.QuartzScheduler - Scheduler meta-data: Quartz Scheduler (v2.2.1) 'DefaultQuartzScheduler' with instanceId 'NON_CLUSTERED'  
Scheduler class: 'org.quartz.core.QuartzScheduler' - running locally.  
NOT STARTED.  
Currently in standby mode.  
Number of jobs executed: 0  
Using thread pool 'org.quartz.simpl.SimpleThreadPool' - with 10 threads.  
Using job-store 'org.quartz.simpl.RAMJobStore' - which does not support persistence. and is not clustered.  
[main] INFO org.quartz.impl.StdSchedulerFactory - Quartz scheduler 'DefaultQuartzScheduler' initialized from default resource file in Quartz package: 'quartz.properties'  
[main] INFO org.quartz.impl.StdSchedulerFactory - Quartz scheduler version: 2.2.1  
[main] INFO org.quartz.core.QuartzScheduler - Scheduler DefaultQuartzScheduler_$_NON_CLUSTERED started.
```

Figure 33: Validating blockchain

In the screenshot, the validation scheduler is executed. This scheduler executes the validation logic after every fixed interval. In this case, the validation logic is executed every 5 minutes.



```
validation.log (/tmp/fz3temp-2) - gedit
validation.log x
Mon Dec 11 00:30:04 UTC 2017: Genesis block data manipulated for:
User Name: ubuntu | Dataset Name: input5000 | Method: CLI | Operation: put
*****
Mon Dec 11 00:30:04 UTC 2017: Data manipulated for block number 2 for :
User Name: ubuntu | Dataset Name: input5000 | Method: CLI | Operation: put
*****
Mon Dec 11 00:30:04 UTC 2017: Data manipulated for block number 3 for :
User Name: ubuntu | Dataset Name: input5000 | Method: CLI | Operation: put
*****
```

Figure 34: Validation report

In the previous steps we modified the blockchain input5000\_blockc and we modified the genesis block. The screenshot of the report explains the same. Since we modified the genesis block, the entire blockchain was affected. This report is in decrypted format, so the admin can see which dataset was modified at what time.

Hence, a change in the genesis block from “+w7” to “777” is detected because all the values were recalculated during the validation process. The process of creating the block header i.e. the concatenation of merkle root, the version number, the timestamp and the number of bits provided a block header value which was different from the present block header value. Since the block header value from the genesis block is being used to create the subsequent block, the values in the subsequent blocks changed as well. The concatenation of merkle root, the version number, the timestamp, number of bits and the previous block header was different than the block header value present.

```

ubuntu@ip-172-31-24-71: ~
ubuntu@ip-172-31-24-71:~$ java -cp /usr/local/hadoop/CBC.jar blockchain/ViewBlockchain
[main] INFO org.apache.hive.jdbc.Utils - Supplied authorities: ec2-18-218-1-144.us-east-2.compute.amazonaws.com:10000
[main] INFO org.apache.hive.jdbc.Utils - Resolved authority: ec2-18-218-1-144.us-east-2.compute.amazonaws.com:10000
Enter UserName:
pshbhatt
Enter Password:
piyush69
2017-12-11 00:33:00,033 WARN NativeCodeLoader - Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Please enter the dataset name to view it's blockchain or enter ALL to view all blockchains
all
Block number: 1

Merkle Root: SnUdhitygVoAXZW+emUdwywg0wk8HR3ckDEXWL11jS0C2qy+pxvas1f8uWl2WT2QVgQn/FPoHHeNWDhRLWQF0+kcS/SPJWt62L/4yF83qpE= | Block Header: K/GK8ps3inBU3
qgfjwMRl6rT/9iLHCRAZGeThwoSfyhczWlbrFGlcQoNGUqYppHPPrk82JoSr3bndtYmJTSFukcS/SPJWt62L/4yF83qpE= | Username: ubuntu | Dataset Name: input1000 | Method:
CLI | Operation: put

Block number: 1

Merkle Root: 462CzeuHvvsMNdKgyOfId/oiJzTqxsAr4sTtnalJ6+rhu+F56FonbL5Xfz90vZs80xEEy4A7l+XrFamtodkZZ+kcS/SPJWt62L/4yF83qpE= | Block Header: kWflu6lOGFAtn
/D56AUHh8/Ag/pHfZwBlrenXxa61fZtFCCsqGYdkPDnmIrJL9IdQW08j6F8BYkg7z7Mn0+kcS/SPJWt62L/4yF83qpE= | Username: ubuntu | Dataset Name: input10000 | Method:
CLI | Operation: put

Block number: 2

Merkle Root: 462CzeuHvvsMNdKgyOfId/oiJzTqxsAr4sTtnalJ6+rhu+F56FonbL5Xfz90vZs80xEEy4A7l+XrFamtodkZZ+kcS/SPJWt62L/4yF83qpE= | Block Header: n+GWTpaVgVPnW
L6tOxpZJuhj9HHVMMITEP1u79P11aGr3MOhejHMZOGa0tD0n3WCFa49IemwK0rJ8IazBhhI00kcS/SPJWt62L/4yF83qpE= | Username: ubuntu | Dataset Name: input10000 | Method:
CLI | Operation: cat

Block number: 1

Merkle Root: ItPq5qll+r/Ub+wg9cBycrljDF3wQo+ItkDXzM4TA3vfoGfE0k69sscSfKEuL+A6ucnqghyVQhj+w5L9tpYx0kcS/SPJWt62L/4yF83qpE= | Block Header: 8sp777i4yP4VU
4+jB0j0y7VI2ghMfVhpUbyPcNJFFLfbqank6YTLfNGexPn9tsZYGxc/UFAwIdLJnERIAxwRekcS/SPJWt62L/4yF83qpE= | Username: ubuntu | Dataset Name: input5000 | Method:
CLI | Operation: put

Block number: 2

Merkle Root: 98wxsW0Cuzthv/cZQBHmB72jROKduke1QdwQh3NRA8B0Cj0Ic2Cud7nedTjqGukOlch5nh0B6PIzW4xdAH550kcS/SPJWt62L/4yF83qpE= | Block Header: YnzAgLSq5FBGZ
lZS+LuoA1U+laFhr1kmpuug5SIBusXjGu606MavAGBwbyFqk1qou03tN0uwMLeWwGvtgFLOkcS/SPJWt62L/4yF83qpE= | Username: ubuntu | Dataset Name: input5000 | Method:
MR | Operation: mraccess

Block number: 3

```

Figure 35: View all blockchains by admin

```

ubuntu@ip-172-31-24-71: ~
ubuntu@ip-172-31-24-71:~$ java -cp /usr/local/hadoop/CBC.jar blockchain/ViewBlockchain
[main] INFO org.apache.hive.jdbc.Utils - Supplied authorities: ec2-18-218-1-144.us-east-2.compute.amazonaws.com:10000
[main] INFO org.apache.hive.jdbc.Utils - Resolved authority: ec2-18-218-1-144.us-east-2.compute.amazonaws.com:10000
Enter UserName:
pshbhatt
Enter Password:
piyush69
2017-12-11 00:33:58,974 WARN NativeCodeLoader - Unable to load native-hadoop library for your platform... using builtin-java classes where applic
Please enter the dataset name to view it's blockchain or enter ALL to view all blockchains
input1000
Block number: 1

Merkle Root: SnUdhitygVoAXZW+emUdwywg0wk8HR3ckDEXWL11jS0C2qy+pxvas1f8uWl2WT2QVgQn/FPoHHeNWDhRLWQF0+kcS/SPJWt62L/4yF83qpE= | Block Header: K/GK8ps3
qgfjwMRl6rT/9iLHCRAZGeThwoSfyhczWlbrFGlcQoNGUqYppHPPrk82JoSr3bndtYmJTSFukcS/SPJWt62L/4yF83qpE= | Username: ubuntu | Dataset Name: input1000 | Met
CLI | Operation: put

```

Figure 36: View single blockchain by admin

```

ubuntu@ip-172-31-24-71: ~
ubuntu@ip-172-31-24-71:~$ java -cp /usr/local/hadoop/CBC.jar blockchain/ViewBlockchain
[main] INFO org.apache.hive.jdbc.Utils - Supplied authorities: ec2-18-218-1-144.us-east-2.compute.amazonaws.com:10000
[main] INFO org.apache.hive.jdbc.Utils - Resolved authority: ec2-18-218-1-144.us-east-2.compute.amazonaws.com:10000
Enter UserName:
pshbhattM
Enter Password:
piyush69
2017-12-11 00:36:00,900 WARN NativeCodeLoader - Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Block number: 1

Block Header: K/GK8ps3inBU3qgfjwRl6rT/9iIHCRAGeThwoSfyhcZwIbwrfGICqoNGUqYppHPrrk82Jo5r3bmdtYMJTSFukcS/SPJWt62L/4yF83qpE=
Block number: 1

Block Header: kmFLu6l0GFAtn/DS0AUHhB/Ag/pWf2wBlremXxa61fZtFCCsQGVDkKPDnmIrJL9IdQM08j6F88YqZkg7z7Nm0+kcS/SPJWt62L/4yF83qpE=
Block number: 2

Block Header: m+GWTpaVgVPnWL6tOxpZJuhj9HHvMWITEP1u79P11agr3MOhejHWZOGa0tD0n3WcFA49IemwK0rJ8IazBhhI00kcS/SPJWt62L/4yF83qpE=
Block number: 1

Block Header: 8sp777i4yP4VU4+jB0j0y7VI2ghMfvhpUbyPcnJFFLF0bqank6YTLfNGexPn91szYGxc/UFawIdLJnERIAxwRekcS/SPJWt62L/4yF83qpE=
Block number: 2

Block Header: YnzAgISq5FBGZLZS+lUoA1U+LaFHR1kmpuug5SIBusXjGu606MavAGBwbyFqk1qou03tN0buwtMleWGVtgFLOkcS/SPJWt62L/4yF83qpE=
Block number: 3

Block Header: jLExCPqMs7TL3uityNdPE/7eJ0ub7jvQJREJekUOLPjndjXhRN08Q+UwDb+k8QEhKx18ZdDilhAvFnZf4nME+0kcS/SPJWt62L/4yF83qpE=
ubuntu@ip-172-31-24-71:~$

```

Figure 37: View blockchain non-admin

A user can view the blockchains. All they have to do is provide their credentials (username and password). If their role is admin, they can view the decrypted blockchain. They can either view all the blockchains (figure 35) or a specific blockchain (figure 36). If the user is not an admin, he/she can view the blockheaders of all the blockchain. No other information is visible to them.

```

ubuntu@ip-172-31-24-71: ~
ubuntu@ip-172-31-24-71:~$ java -cp /usr/local/hadoop/CBC.jar blockchain/Role_Registration
[main] INFO org.apache.hive.jdbc.Utils - Supplied authorities: ec2-18-218-1-144.us-east-2.compute.amazonaws.com:10000
[main] INFO org.apache.hive.jdbc.Utils - Resolved authority: ec2-18-218-1-144.us-east-2.compute.amazonaws.com:10000
First Name:
temporary
Last Name:
user
User Name:
tempuser
Password:
temp000
Role:
employee

```

Figure 38: User registration



```

hive> use blockchain;
K
time taken: 0.755 seconds
hive> select * from registration;
K
Piyush Bhatt pshbhatt 1d9c893b7075da81c6da18f9bf57494 admin 10807868
Jennifer Bhatt jenni3og 88f20e3885e9e31ac60a234319bdce9 admin 10807868
Piyush Bhatt pshbhattM 1d9c893b7075da81c6da18f9bf57494 manager NA
temporary user tempuser dc128acb45f752e3f1c2a724c849e5b employee NA
time taken: 1.054 seconds, Fetched: 4 row(s)
hive>

```

Figure 39: User data

The employees of the organization can register themselves if they wish to see the blockchain. They will have to provide their information and their information will be inserted into the database. If their role is admin, an extra field will be added to the registration process. The password would be inserted into the database as a hash value.

```

ubuntu@ip-172-31-24-71: ~
ubuntu@ip-172-31-24-71:~$ hdfs dfs -copyfromlocal /home/ubuntu/inputs/input20000 /user/ubuntu
-copyfromlocal: Unknown command
ubuntu@ip-172-31-24-71:~$ hdfs dfs -copyFromLocal /home/ubuntu/inputs/input20000 /user/ubuntu
-copyFromLocal: Unknown command
ubuntu@ip-172-31-24-71:~$ hdfs dfs -copyFromLocal /home/ubuntu/inputs/input20000 /user/ubuntu
ubuntu@ip-172-31-24-71:~$ hdfs dfs -ls /blockchain
Found 5 items
drwxr-xr-x - ubuntu supergroup 0 2017-12-10 23:39 /blockchain/input10000_blockc
drwxr-xr-x - ubuntu supergroup 0 2017-12-10 23:37 /blockchain/input1000_blockc
drwxr-xr-x - ubuntu supergroup 0 2017-12-11 21:58 /blockchain/input20000_blockc
drwxr-xr-x - ubuntu supergroup 0 2017-12-11 00:05 /blockchain/input5000_blockc
-rw-r--r-- 1 ubuntu supergroup 181 2017-12-11 21:58 /blockchain/list
ubuntu@ip-172-31-24-71:~$ hdfs dfs -copytolocal /user/ubuntu/input1000 /home/ubuntu
-copytolocal: Unknown command
ubuntu@ip-172-31-24-71:~$

```

Figure 40: Incorrect statements

If incorrect statements are entered, the Hadoop file system throws an error and does not perform any action. In the screenshot, **copyFromLocal** command was misspelled twice and it was copied when the command was executed correctly. The next command **copyToLocal** was misspelled and Hadoop file syetem caught the error.

```

ubuntu@ip-172-31-24-71:~
ubuntu@ip-172-31-24-71:~$ hdfs dfs -ls /blockchain
Found 5 items
drwxr-xr-x - ubuntu supergroup      0 2017-12-10 23:39 /blockchain/input1000_blockc
drwxr-xr-x - ubuntu supergroup      0 2017-12-10 23:37 /blockchain/input1000_blockc
drwxr-xr-x - ubuntu supergroup      0 2017-12-11 21:58 /blockchain/input20000_blockc
drwxr-xr-x - ubuntu supergroup      0 2017-12-11 00:05 /blockchain/input5000_blockc
-rw-r--r--  1 ubuntu supergroup    229 2017-12-11 22:05 /blockchain/list
ubuntu@ip-172-31-24-71:~$ hdfs dfs -copyToLocal /blockchain/input20000_blockc/input20000_blockc
ubuntu@ip-172-31-24-71:~$ hdfs dfs -ls /blockchain
Found 6 items
drwxr-xr-x - ubuntu supergroup      0 2017-12-10 23:39 /blockchain/input1000_blockc
drwxr-xr-x - ubuntu supergroup      0 2017-12-10 23:37 /blockchain/input1000_blockc
drwxr-xr-x - ubuntu supergroup      0 2017-12-11 21:58 /blockchain/input20000_blockc
drwxr-xr-x - ubuntu supergroup      0 2017-12-11 22:06 /blockchain/input20000_blockc_blockc
drwxr-xr-x - ubuntu supergroup      0 2017-12-11 00:05 /blockchain/input5000_blockc
-rw-r--r--  1 ubuntu supergroup    254 2017-12-11 22:06 /blockchain/list
ubuntu@ip-172-31-24-71:~$

```

Figure 41: Illegal command

Copying blockchain files to local file system is an illegal operation. The Blockchain data usage tracker detects that and creates a new blockchain which has 2 “**\_blockc**” in the name. This shows that someone tried to tamper with the blockchain. The user’s username will be captured in this new blockchain and the validation process will detect it as an anomaly.

```

validation - Notepad
File Edit Format View Help

Mon Dec 11 22:40:19 UTC 2017: Operation was done on blockchain for input20000_blockc_blockc
*****

```

Figure 42: Validation report

The report shows that an illegal operation was performed on a blockchain.



```

ubuntu@ip-172-31-24-71: ~
ubuntu@ip-172-31-24-71:~$ hdfs dfs -ls /blockchain
Found 6 items
drwxr-xr-x - ubuntu supergroup 0 2017-12-10 23:39 /blockchain/input1000_blockc
drwxr-xr-x - ubuntu supergroup 0 2017-12-10 23:37 /blockchain/input1000_blockc
drwxr-xr-x - ubuntu supergroup 0 2017-12-11 22:42 /blockchain/input20000_blockc
drwxr-xr-x - ubuntu supergroup 0 2017-12-11 22:42 /blockchain/input20000_blockc_blockc
drwxr-xr-x - ubuntu supergroup 0 2017-12-11 00:05 /blockchain/input5000_blockc
-rw-r--r-- 1 ubuntu supergroup 365 2017-12-11 22:42 /blockchain/list
ubuntu@ip-172-31-24-71:~$ hdfs dfs -rm -r /blockchain/input20000_blockc_blockc
17/12/11 22:44:04 INFO fs.TrashPolicyDefault: Namenode trash configuration: Deletion interval = 0 minutes, Emptier interval = 0 minutes.
Deleted /blockchain/input20000_blockc_blockc
ubuntu@ip-172-31-24-71:~$ hdfs dfs -rm /user/ubuntu/input20000
17/12/11 22:44:15 INFO fs.TrashPolicyDefault: Namenode trash configuration: Deletion interval = 0 minutes, Emptier interval = 0 minutes.
Deleted /user/ubuntu/input20000
ubuntu@ip-172-31-24-71:~$ hdfs dfs -ls /blockchain
Found 4 items
drwxr-xr-x - ubuntu supergroup 0 2017-12-10 23:39 /blockchain/input1000_blockc
drwxr-xr-x - ubuntu supergroup 0 2017-12-10 23:37 /blockchain/input1000_blockc
drwxr-xr-x - ubuntu supergroup 0 2017-12-11 00:05 /blockchain/input5000_blockc
-rw-r--r-- 1 ubuntu supergroup 365 2017-12-11 22:42 /blockchain/list
ubuntu@ip-172-31-24-71:~$

```

Figure 43: Delete hdfs files

```

delete.log (/tmp/fz3temp-2) - gedit
delete.log x
Mon Dec 11 22:44:05 UTC 2017: File input20000_blockc_blockc deleted.
*****
Mon Dec 11 22:44:16 UTC 2017: File input20000 deleted.
*****
Mon Dec 11 22:44:20 UTC 2017: File input20000_blockc deleted.
*****

```

Figure 44: Log of files deleted

If a file is deleted (blockchain or dataset), it will be recorded in delete.log. Like in the screenshot, first the blockchain “**input20000\_blockc\_blockc**” was deleted (figure 43).

This was recorded in delete.log (figure 44).

Next the dataset **input20000** was deleted. As explained previously, if a dataset is deleted, the corresponding blockchain will be deleted as well. The delete.log records the deletion of **input20000** and its corresponding blockchain **input20000\_blockc**.

## CHAPTER V

### **Conclusion**

Data is captured by organizations and converted into datasets. Two or more datasets may be combined to fetch critical or sensitive data which then can be misused for a variety of purposes. The thesis proposes a novel way to track the usage of the datasets using a blockchain. Blockchain is used because the data inside a blockchain is immutable. The data inside the blockchain are validated at regular intervals and if an individual tries to modify the data using unauthorized manner, the changes would be known.

When an user accesses/creates a dataset in Hadoop File System (HDFS), his username, the name of the dataset accessed/created, the method using which he accessed the dataset (Command Line Interface or Map Reduce) and the Command Line Operation performed (cat, copy and move) are captured. These values are then hashed to create a single hash value called Merkle root. If a new dataset is created or an existing dataset is accessed for the first time, a genesis block is created. A version number is calculated, the timestamp for creation of the block is captured and the number of bits of the data (data used for creation of Merkle root) is calculated. These three values are modified and are concatenated with the Merkle root. This creates the block header which is used in creation of the subsequent blocks. The blockheader of the following blocks is calculated by concatenating the Merkle root, the modified version number, the modified timestamp, modified

number of bits and the block header of the previous block. Since all the data inside each block depends on the data from the previous blocks, all the data is connected. At every regular intervals a validation process is executed. This process recomputes all the values and if any modification was done to any of the blocks, the values in the following blocks would not match. This would mean that an unauthorized modification was performed. This is the reason blockchain is secure and reliable.

This implementation was tested on a real Hadoop cluster with 1 namenode and 3 datanodes. One more backup cluster was created. Each of these nodes was configured with 4GB RAM and 8GB hard drive. When a dataset in HDFS was accessed, the details were stored in blocks in the blockchain. The blocks consisted of a multiple hash values. These hash values were the Merkle root, the blockheader, hash values of Version number, the timestamp and number of bits. One of these values were modified and the validation process was executed. The modification caused an anomaly and was caught during the validation process. This anomaly was reported to a log file. The testing was successful with the expected results.

In this implementation dataset name, username, method of access (Command Line Interface or Map Reduce) and command line operation (copy, move, cat, put) are being used to create a blockchain. In the future many more relevant entities like data items, the access rights each user has and user role could be captured to make the use of Blockchain Data Usage Tracker more detailed on a larger scale.

This was done on a small scale testing environment. In future, this could be made to be executed on a large scale scope like for public/government use. That would require high performance servers which would make it much better.

## REFERENCES

- [1] Every day big data statistics  
(<http://www.vcloudnews.com/every-day-big-data-statistics-2-5-quintillion-bytes-of-data-created-daily/>)[02/27/2017], indentation - check for all
- [2] How much data is out there?  
([http://www.webopedia.com/quick\\_ref/just-how-much-data-is-out-there.html](http://www.webopedia.com/quick_ref/just-how-much-data-is-out-there.html)), [02/27/2017]
- [3] Hadoop Mapreduce([https://hadoop.apache.org/docs/r1.2.1/mapred\\_tutorial.html](https://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html)), [02/27/2017]
- [4] Big data analytics ([http://www.sas.com/en\\_us/insights/analytics/big-data-analytics.html](http://www.sas.com/en_us/insights/analytics/big-data-analytics.html)), [02/27/2017]
- [5] Bitcoin (<https://bitcoin.org/en/>), [02/27/2017]
- [6] Cryptographic Hash Functions  
([https://en.wikipedia.org/wiki/Cryptographic\\_hash\\_function](https://en.wikipedia.org/wiki/Cryptographic_hash_function)), [02/27/2017]
- [7] Access control of sensitive data in HDFS  
([https://www.thinkmind.org/download.php?articleid=infocomp\\_2013\\_4\\_10\\_10050](https://www.thinkmind.org/download.php?articleid=infocomp_2013_4_10_10050)) [02/27/2017]
- [8] Vigiles: Fine-grained Access Control for MapReduce Systems  
(<http://www.utdallas.edu/~muratk/publications/vigilespaper.pdf>) [02/27/2017]
- [9] Zeng, W., Yang, Y., Luo, B. (2013). Access control for Big Data using data content. In IEEE International Conference on Big Data, 2013, pp. 45-47.

- [10] Hu, V. C., Grance, T., Ferraiolo, D. F., Kuhn, D. R. (2014). An Access Control scheme for Big Data processing. In IEEE International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 2014, pp. 1-7.
- [11] Storm (<http://storm.apache.org/>) [03/20/2017]
- [12] Hortonworks Data Platform(<https://hortonworks.com/products/data-center/hdp/>)[03/20/2017]
- [13] Apache Pig(<https://pig.apache.org/>)[03/20/2017]

VITA

PIYUSH BHATT

Candidate for the Degree of

Master of Science in Computer Science

Thesis: ACCESS CONTROL AND SECURITY OF DATASETS BY USAGE  
TRACKING USING BLOCK CHAIN TECHNOLOGY

Major Field: BIG DATA

Biographical:

Education:

Completed the requirements for the Master of Science in Computer Science at Oklahoma State University, Stillwater, Oklahoma in May, 2017.

Completed the requirements for the Bachelor of Science in Information Technology at SRM University, Chennai, India in 2011.

Experience: 3.5 years of software development in Java and J2EE.