Entropy **2014**, *16*, 4185-4198; doi:10.3390/e16084185

OPEN ACCESS

entropy

ISSN 1099-4300

www.mdpi.com/journal/entropy

Article

Block Access Token Renewal Scheme Based on Secret Sharing in Apache Hadoop

Su-Hyun Kim and Im-Yeong Lee *

Department of Computer Sciences and Software Engineering, Soonchunhyang University, 646, Eumnae-ri, Sinchang-myeon, Asan-si, Chungcheongnam-do, Korea; E-Mail: kimsh@sch.ac.kr

* Author to whom correspondence should be addressed; E-Mail: imylee@sch.ac.kr; Tel.: +82-041-530-1323.

Received: 24 June 2014; in revised form: 15 July 2014 / Accepted: 17 July 2014 / Published: 24 July 2014

Abstract: In a cloud computing environment, user data is encrypted and stored using a large number of distributed servers. Global Internet service companies such as Google and Yahoo have recognized the importance of Internet service platforms and conducted their own research and development to utilize large cluster-based cloud computing platform technologies based on low-cost commercial off-the-shelf nodes. Accordingly, as various data services are now allowed over a distributed computing environment, distributed management of big data has become a major issue. On the other hand, security vulnerability and privacy infringement due to malicious attackers or internal users can occur by means of various usage types of big data. In particular, various security vulnerabilities can occur in the block access token, which is used for the permission control of data blocks in Hadoop. To solve this problem, we have proposed a weight-applied XOR-based efficient distribution storage and recovery scheme in this paper. In particular, various security vulnerabilities can occur in the block access token, which is used for the permission control of data blocks in Hadoop. In this paper, a secret sharing-based block access token management scheme is proposed to overcome such security vulnerabilities.

Keywords: Hadoop; block access token; authentication; secret sharing; entropy

---

1. Introduction

Recently, considerable attention has been paid to cloud computing in Korea as well as overseas, thereby increasing the number of studies on the subject. Many companies have become interested in cloud computing, which is extendable to various areas based on the advances in information technology (IT) and facilitates an efficient use of computing power. Global Internet service companies such as Google and Yahoo have recognized the importance of Internet service platforms and conducted their own research and development projects to utilize large cluster-based distributed computing platform technologies based on low-cost commercial off-the-shelf nodes. As a representative application requiring big data processing and storage management, not only Internet service areas but also other applications such as business intelligence can be targets for cloud services so that many business models are presented. Thus, as various data services are now allowed over a distributed computing environment, the distributed management of big data has become a major issue. However, if the data stored in such distributed servers are managed without data encryption, data losses may occur by tracking the location of the distributed file stored in the master server. To prevent such a problem, distributed data should be encrypted using a secret key. However, big data can be divided into tens or hundreds of segments over a cloud computing environment, which causes difficulty in data management because of the different secret keys used in the distributed servers and leads to a substantial overhead due to the numerous processes required for authentication and encryption/decryption. To solve this problem, we have proposed in this paper a weight-applied XOR-based efficient distribution storage and recovery scheme. In particular, various security vulnerabilities can occur in the block access token, which is used for the permission control of data blocks in Hadoop. In this paper, a secret sharing-based block access token management scheme is proposed to overcome such security vulnerabilities.

The rest of this paper is organized as follows: in Section 2, the related technologies are introduced to aid in the understanding the proposed scheme. In Section 3, the basic security requirements of a cloud computing environment are discussed. In Section 4, the proposed scheme is explained. In Section 5, the safety of the proposed scheme is analyzed, and finally, the conclusions and the future research directions are presented in Section 6.

2. Related Study

In this section, related technologies and existing schemes are introduced to aid in the understanding of the scheme proposed in this paper.

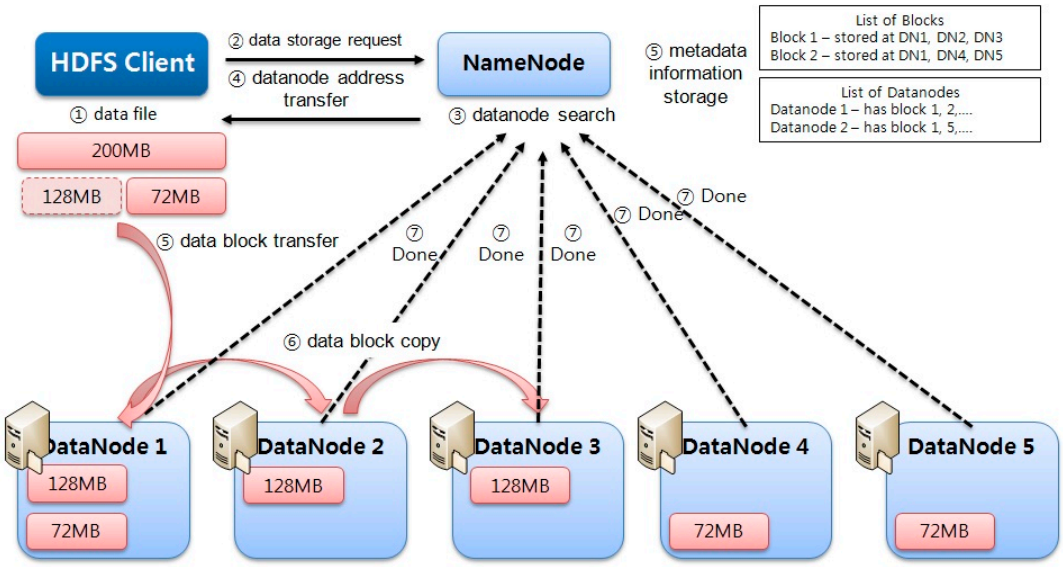
2.1. Hadoop Distributed File System (HDFS)

2.1.1. Structure of HDFS

The Apache Hadoop Distributed File System (HDFS) is a file system produced to be an executable in off-the-shelf hardware which has many similarities with the existing distributed file systems. However, many differences exist, which are designed for better failover features and applicability to low-cost hardware. The HDFS has most widely been used as a distributed file system over a cloud

computing platform by global IT companies such as Amazon, IBM, and Yahoo [1].

Figure 1. Distributed data storage process.



One of the distinctive characteristics of the HDFS design and implementation is that it is implemented using Java in order to guarantee easy portability between many platforms, and the similarity of the GFS. The use of the Java language with high portability has an advantage that the HDFS can be run over many different servers.

An HDFS client divides a user's data into 128-MB block units, thereby sending storage requests to NameNode. Then, NameNode identifies the address of DataNode to which a data block is stored and sends back the address to a client. A client who receives three addresses of DataNode sends the data to the first DataNode directly, and then, the first DataNode sends a copy of the data as soon as it receives the data. In such a manner, a single block is backed up and stored in three DataNodes in preparation of failure and error in DataNode as well as data loss (Figure 1).

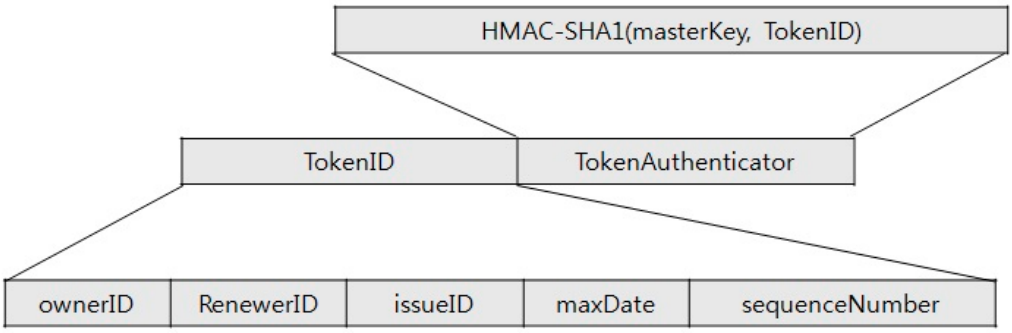
2.1.2. Block Access Token

In the typical Hadoop system, the access to a data block stored in a DataNode is not controlled. In this case, clients who do not have authentication can access the data block to read or write. In Hadoop version 1.0.0, the concept of block access token was introduced to confirm the access authorization to the data block.

A secret key is shared by NameNode and DataNode. The secret key is regularly renewed by NameNode and is shared with DataNode through a heartbeat. Using the secret key, the block access token requested by the clients is encrypted. When the secret key transferred between NameNode and DataNode is exposed to attackers, the data block stored in DataNode runs the risk of exposure to those attackers. Although the use of a public key has been considered for token creation, it incurs a considerably high computing cost, and hence, a symmetric key scheme is used. The structure of the block access token is shown in Figure 2. The block access token is composed of a token ID and a TokenAuthenticator. Its detailed structure is as follows:

- expirationDate: the date of expiry;
- keyID: secret key identification used for making a TokenAuthenticator;
- ownerID: token owner identification;
- blockID: block identification authenticated by a token;
- accessModes: a combination of the rights to block reading, writing, copying, *etc.*

Figure 2. Structure of a block access token.



In general, a block access token is used as follows: a client requests a block ID that constitutes a file and the DataNode location that stores the block from NameNode to read a file. NameNode then sends the requested block ID and the DataNode location information that stores the block using the meta-information maintained in the memory. A single block is replicated in many DataNodes; therefore, the locations of all DataNodes that have the replicated data are returned. A single block is sent with a single block access token to represent the authentication information of the block. A client selects the closest DataNode based on the location information of the many DataNodes received from NameNode followed by requesting a block with the block access token from the closest DataNode. The DataNode that receives the request verifies the authentication information of the block requested. Thus, a token authenticator is created using the block access token ID received and the secret key shared by NameNode, and the created and received token authenticators are compared. If the two token IDs are the same, authentication is completed. If correct authentication has been performed, the requested block is sent.

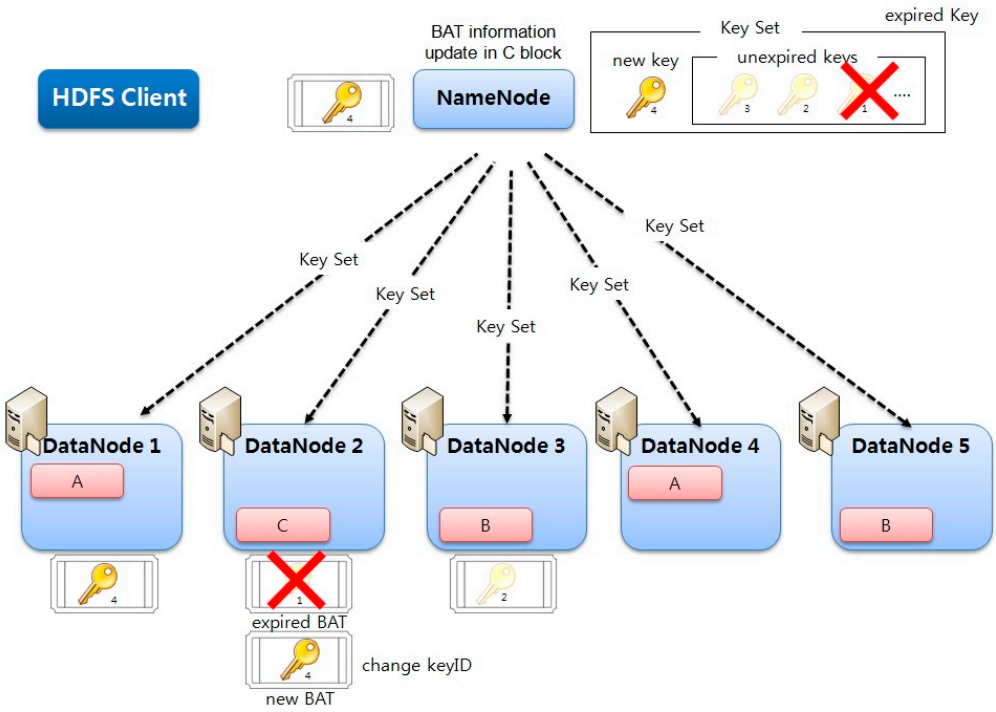
2.1.3. Key Rolling Mechanism

A secret key, which is used to calculate the token authenticator that constitutes a block access token, is randomly chosen by NameNode and sent to DataNode when first register with the NameNode. Then, the secret key is updated by NameNode periodically and is shared via a heartbeat, which is communicated with DataNode frequently [2]. The key rolling mechanism is executed as follows (Figure 3):

(1) A random key is chosen in NameNode. The key that is currently used is called, and NameNode selects a new key regularly and discards the previous key. The previous key is stored for a certain period (until the expiry date). NameNode maintains a set of keys that are not expired. The current key in this set is used only for token creation (validation).

(2) When DataNode starts, a key set in NameNode, which is not expired, is fetched. DataNode is not required to test token validation.
(3) NameNode removes the expired keys when the current key is updated. A new key is used for token creation from the time of creation. DataNode obtains a new key set through heartbeats.
(4) DataNode removes the expired keys when a key is received.

Figure 3. Key rolling mechanism of Hadoop.



2.2. Online Secret Sharing

The existing secret sharing schemes have advantages in that a partial information size that each participant has to store is almost the same as that of the original secret K and the highly secure secret sharing scheme in which unauthorized participants cannot obtain any information regarding the secret K. However, after the secret is restored once, it has a drawback that partial information of the participants cannot be reused. In addition, if the access structure of a set of participants, who can restore a secret, is modified, the distributor has to create a new polynomial, and new partial information needs to be shared with the existing participants as well.

To solve such problems, an online secret sharing scheme is proposed [7]. If the access structure of a set of participants who can restore the original secret K is dynamically modified, *i.e.*, if a participant is randomly removed or a new participant is added, this scheme does not require a re-distribution of the new partial information but uses the existing partial information as is. Authenticated information is disclosed on the public board to which all participants have access, and if the access structure is modified, only the disclosed information values are modified so that the partial information of the existing participants can remain as is.

2.3. Analysis of Existing Research

2.3.1. Park *et al.*

Park *et al.* [8] proposed a block access token management scheme in 2012 that can prevent replay attacks using hash chains in order to overcome the vulnerability of the existing block access token. When a client sends a block access token along with a block request to DataNode, a security vulnerability is incurred. To overcome such vulnerability, each DataNode manages independent hash chains, thereby sending a single block access token including a hash chain-based one-off token, preventing attackers from performing replay attacks even if a token is exposed to attackers.

This scheme uses a one-off token so that each DataNode maintains a different hash value, minimizing the previous vulnerability. However, when a block access token is issued to a client, the token needs to be re-created every time there is a replicated block, causing a certain computational overhead. Thus, NameNode needs to maintain a number of values per DataNode ID; hence, the amount of data to be maintained increases with an increase in the number of DataNodes

2.3.2. Jin *et al.*

Jin *et al.* [9] analyzed a data security issue under the Hadoop platform and proposed a reliable file system design for Hadoop in 2013. This scheme was designed by using a fully homomorphic and authentication agent technology. The homomorphic encryption technology uses encrypted data in order to protect the efficiency of the application program and data security. The authentication agent technology provides various access control rules by a combination of the access control mechanisms. Based on these two technologies, permission right separation and security audit mechanism are provided to ensure safe storage of data in the Hadoop file system. However, only the access control rules cannot prevent Impersonate attacks.

2.3.3. Zhang *et al.*

Zhang *et al.* [10] proposed a flexible and extensible scheme for data storage using a Linux cluster technology, distributed file systems, and a cloud computing framework based on Hadoop in 2011. The existing HDFS file system structure has a high throughput for large file storage systems, but it has a limitation that it is not suitable for small file processing and storage. To overcome this limitation, they proposed a flexible and extensible scheme for data storage that uses the object metadata creation query and is managed in the entity metadata storage hierarchy based on HBase.

3. Security Requirement

One of the core mechanisms used by cloud computing is data encryption via key management. A failure in the key storage server can make data access impossible; hence, research into key management methods is essential. The following security requirements are needed to prevent various failures [5]:

- *Safe Key Storage*: as with other types of sensitive data, the key must be protected during storage, transmission, and back-up processes. Inadequate key storage could destroy all of the encrypted data.
- *Key Storage Access*: the access to key storage should be limited only to those entities who require private keys. The key storage management policy should also maintain separate access control. The entity used to provide the keys should be different from the entity used to store the keys.
- *Key Back-up and Recovery*: a lost key inevitably leads to a loss of data, which is an effective way of destroying data. The accidental loss of a key can also lead to major losses for a business. Therefore, safe back-up and recovery solutions must be implemented.
- *Confidentiality*: data communications between the storage server and a client terminal should only be verified using legitimate objects.
- *Authentication*: the legitimacy of a storage server where the data is distributed and stored should only be verifiable by a legitimate object recognition client.
- *Availability*: certification and secrecy should be performed at a fast rate to ensure availability for transmitting large amounts of data.
- *Computational Efficiency*: only the minimum number of computations should be performed to reduce frequent overheads on the client terminals and cloud servers.

4. Proposed Scheme

4.1. System Model and Assumption

The overall cloud computing environment was designed based on HDFS of Apache. The cloud computing environment, will be distributed saving is divided into blocks of 128 MB for the data storage capacity. It has been stored in the form of text data stored in the distributed storage server is not encrypted, when the distributed storage server is taken by an attacker, the problem of some data is exposed as it occurs.

In the proposed scheme, a secret key value, which can create a block access token required for authentication, is divided into many segments and stored. That is, a block access token is not encrypted with only a single secret key, but a secret key is created by collecting secret segments via communication between DataNodes that store data blocks, thereby preventing the man-in-the-middle attacks in advance. In the proposed scheme, Pinch's online secret sharing scheme was applied [7].

4.2. System Parameters

K secret key value
T_x public value
Γ access structure
Γ^* minimal authorized set
f One-way function

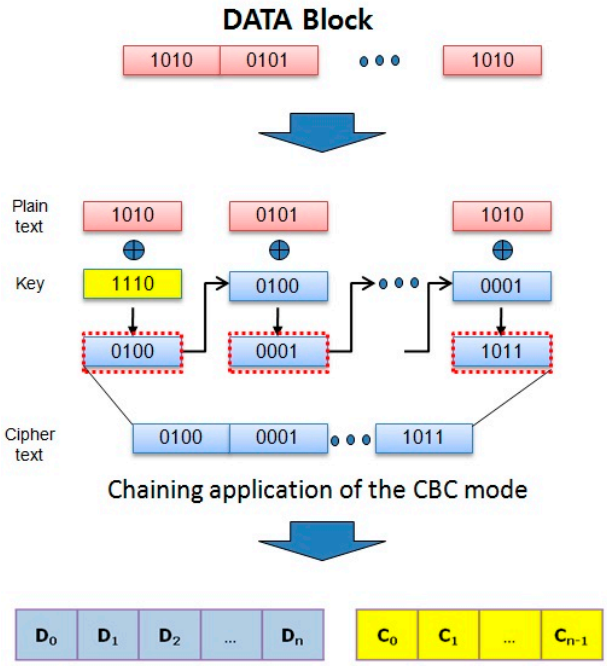
4.3. Data Storage and Error Recovery

HDFS proposes having the user encrypted to prevent an attack, but protection for the system itself is not provided. We use Chaning of CBC mode as one of the block encryption modes in the method proposed in this paper. The original data block is converted into a random data block. The parity block is generated as shown in Equation (1):

$$\begin{aligned} C_i &= D_{0-n} \oplus D_i \\ &\cdots \\ C_{n-1} &= D_{0-n} \oplus D_{n-1} \end{aligned} \tag{1}$$

If an error occurs in the original data blocks, a generated parity block is able to recover the original data block. The steps to generate and recover the data block are as follows (Figure 4).

Figure 4. Distributed data storage phase.



Process of generating the C_1

$$C_0 = D_0 \oplus D_1 \oplus D_2 \oplus D_3 \oplus D_0 = D_1 \oplus D_2 \oplus D_3$$

$$C_1 = D_0 \oplus D_2 \oplus D_3$$

$$C_2 = D_0 \oplus D_1 \oplus D_3$$

Process of recovering the D_1

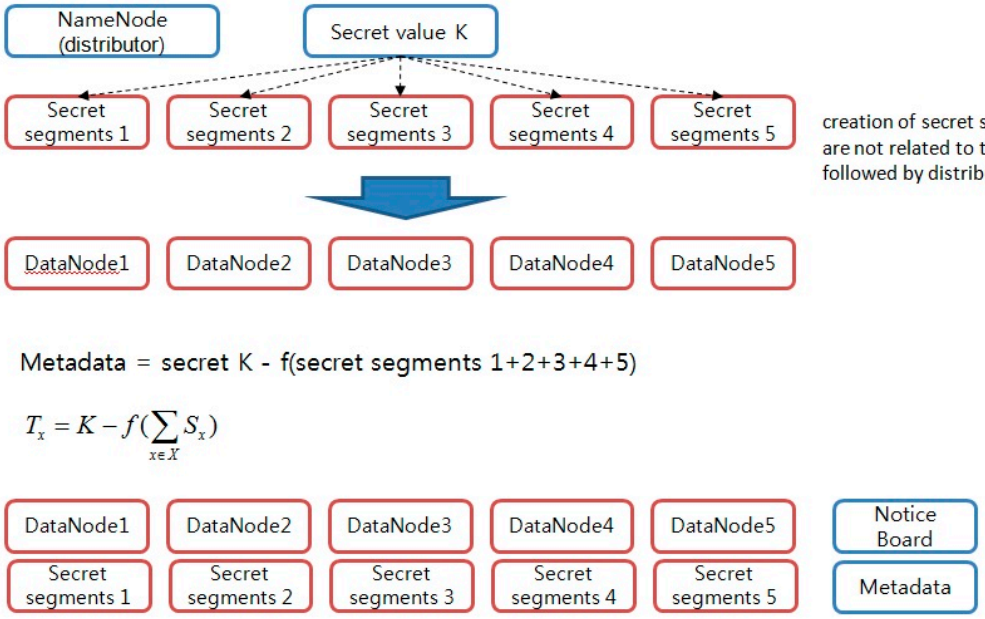
$$D_1 = C_1 \oplus D_1 \oplus D_4$$

4.4. Secret Sharing Protocol

In this phase, we then distribute the value of the secret to the distributed storage server. The value of this secret serves to generate a block access token. When a client requests data storage from NameNode, NameNode secures DataNode to store the data and creates a block access token, which is

required for user authentication with respect to data segments. Here, a secret key value for the creation of a block access token is stored in a number of DataNodes as a form of a secret segment (Figure 5). The secret segment's value has no association with the secret key value. Therefore, even if it is exposed to attackers, there is no adverse effect on the system.

Figure 5. Distributed data storage phase.



Step 1. The distributor NameNode selects the partial information S_i ($1 \le i \le n$) to be distributed to each participant randomly.

Step 2. The distributor NameNode sends the value of S_i ($1 \le i \le n$) secretly to each participant P_i.

Step 3. The distributor NameNode computes the T_x following value with respect to X, which is: $X \in \Gamma^*$.

$$T_x = K - f(\sum_{x \in X} S_x)$$

Step 4. The distributor NameNode discloses the T_x value to the public board with respect to each element X that belongs to Γ^*.

4.5. Secret Restoration Protocol

Upon receiving the data read request sent by a user, DataNode restores a secret key using the secret segment value that each node has and then creates a block access token (Figure 6).

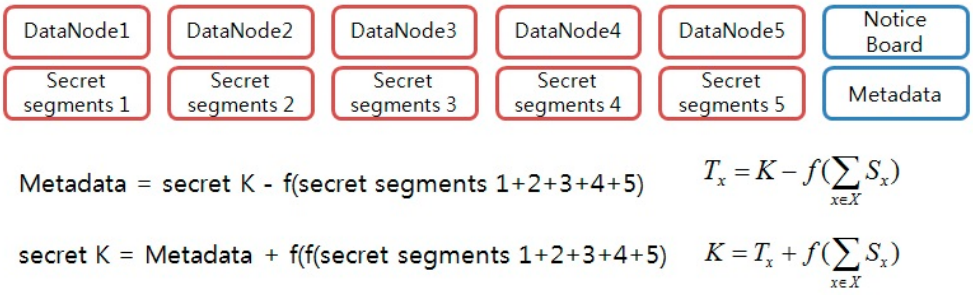
Step 1. A participant set for the restoration of secret K: The following V_x is calculated using the partial information of the participants P_i ($1 \le i \le n$) that belong to set X from $\{P_1, P_2, \ldots, P_t\}$:

$$V_x = \sum_{x \in X} S_x$$

Step 2. A one-way function f(x) is applied to the calculated V_x value, and the T_x value is read from the public board to calculate K, as shown in the equation below:

$$K = T_x + f(V_x)$$

Figure 6. Secret restoration process.

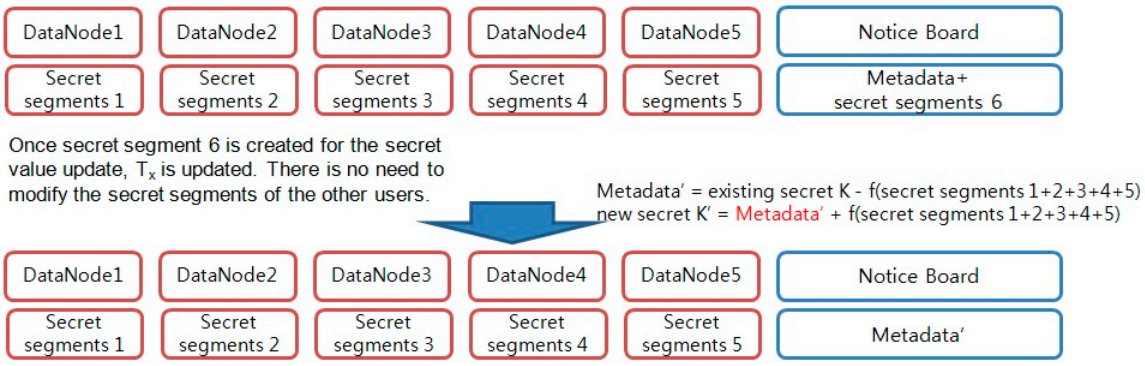


4.6. Secret Key Update

In Hadoop, a secret key that creates a block access token is regularly updated by means of the key rolling mechanism. During this update of the secret key, the key set is vulnerable to the risk of exposure to attackers. As a method of hiding a key set, a secret key value may be divided via secret sharing, but this method has the drawback that the key segments have to be re-distributed during the key update. However, the existing distributed secret segments can be maintained without re-distributing new segments by using an online secret sharing scheme during the key update.

The distributor NameNode creates a new single secret segment without modifying the existing secret segments that were distributed beforehand, which is followed by the updating of the disclosed metadata information. Based on the updated metadata, a new secret key value can be obtained by using only the existing secret segments that were distributed earlier. Here, the secret segments that were newly created by NameNode are discarded after being used for the metadata information update (Figure7).

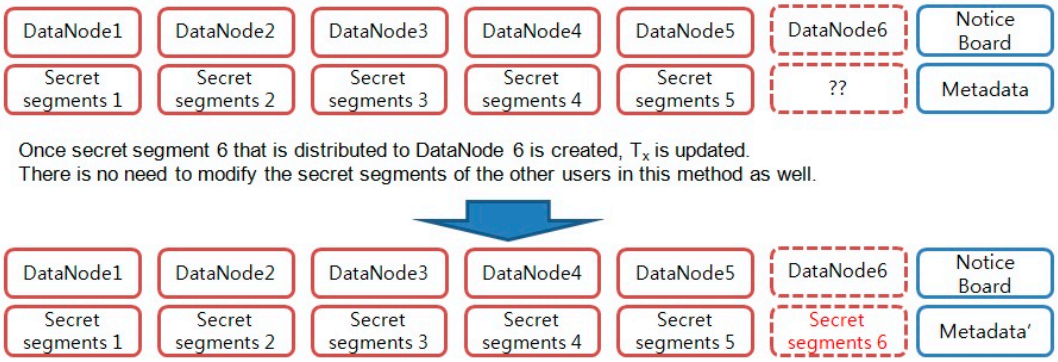
Figure 7. Secret key update.



4.7. Addition of New Data Block

The scheme proposed in this paper can be employed for new data addition as well. When a new data block is stored in DataNode, the previously stored secret segments do not need to be distributed or modified again. NameNode creates a new secret segment in DataNode where new data blocks are stored and this updates the metadata information. The secret key value to be updated after this is the same as that in the key update process (Figure 8).

Figure 8. Addition of new data block.



5. Analysis of the Proposed Scheme

In this section, five schemes, namely an existing block access token-added scheme supplemented by Hadoop without authentication or permission, three other existing schemes (from [8–10]), and the proposed scheme are compared (Table 1).

Table 1. Comparison with the conventional analysis method.

	Hadoop	**[8]**	**[9]**	**[10]**	**Proposed Scheme**
Exposure of Block Access Token Information	○	○	○	○	×
Vulnerable to the Man-in-the-Middle Attacks	○	×	○	○	×
Computational Complexity	Symmetric Key Operations	Hash (n)	-	-	Hash (1)
User Access Control	Δ	○	○	○	○
Against Impersonate Attack	×	Δ	×	×	○
Integrity Check of Data	Δ	Δ	Δ	○	○
Availability of Data	○	○	○	○	○

5.1. Exposure of Block Access Token Information

In the existing Hadoop, a secret key is updated between NameNode and DataNode frequently, and block access tokens are sent after they are encrypted using the appropriate secret key upon a user's request. A hash chains-used scheme is one that adds a one-off token to a block access token, which exposes the block ID, user ID, and key ID to attackers due to a lack of an encryption process. The other existing schemes also have the same problem as that in Hadoop because they use the same existing Hadoop environment. In the proposed scheme, a block access token created by sending or receiving secret segments is transmitted using a hash function, and only an authorized user or DataNode can create a block access token, which is followed by taking a hash function to prove that a user is authentic.

5.2. Man-in-the-Middle Attacks

In the existing schemes, a temporary block access token can be created by the attacker's interception during the secret key update between NameNode and DataNode. The hash chain and the proposed schemes are proposed to prevent the man-in-the-middle attacks. Thus, man-in-the-middle attacks cannot be successful under these schemes. Other existing schemes also experience the abovementioned man-in-the-middle attacks because they use the same existing Hadoop environment.

5.3. Computational Complexity

When comparing the computational complexity, the existing Hadoop systems update secret keys frequently and perform a symmetric key encryption-based operation for block access token creation. Here, various types of symmetric key encryption algorithms such as AES and SEED can be used. The hash chain scheme iterates n hash operations by using random numbers in each data block. However, the proposed scheme is relatively efficient because a block access token is applied to a hash function only once and sent along with a secret segment after the initial secret segment creation and distribution process.

5.4. Against Impersonate Attack

The existing scheme [9] performs user access control by using the access control list, but it cannot prevent the impersonated user's data access with only an access control module. However, in the proposed scheme, only authorized creators can generate a block access token required for data access so that it can prevent the impersonate attack.

5.5. Integrity Check of Data

The existing Hadoop systems and scheme [8] check data block integrity and error occurrence in the data storage servers frequently via heartbeat messages. Heartbeat messages are transferred by DataNode to NameNode frequently. The scheme proposed in [9] also frequently checks data block integrity through checksums, but it has the drawback of an increase in the computational complexity. The scheme proposed in [10] can check validation by using the index information created in the object metadata table. In the proposed scheme, validation can be performed via secret segments required for block access token creation, which is quite efficient because of the absence of frequent validations through heartbeat messages.

5.6. Availability of Data

All schemes facilitate the availability of data stored in distributed servers. The distributed Hadoop storage system replicates a data block into at least three locations in order to respond to a single location error first. The availability of data maximally increases by frequently conducting an error and loss check of data blocks stored in DataNode.

5.7. Authentication and Confidentiality

The user data are divided into many pieces and stored on storage servers by the master server. If a user request for data is received, a collection process is required for the distributed data. The storage servers are physically and logically separated from the master server, so the certifications for all of the individual distributed servers should be provided secretly in the communication networks used during the transmission of the data pieces. The use of an existing public key encryption system is very inefficient in terms of its costs and computational burden.

6. Conclusions and Future Research

In this paper, a new block access token management scheme was proposed by modifying vulnerable parts of user data management over the Hadoop system, which is one of the cloud computing structures. Security vulnerability and privacy infringement due to malicious attackers or internal users by means of various usage types of big data could be prevented in advance, and a secret sharing-based block access token management scheme was proposed to overcome various security vulnerabilities.

The problem of secret key management shared between DataNode and NameNode and the problem of NameNode that posed the risk of man-in-the-middle attacks were solved, and the operation overhead, which was concentrated on NameNode, could be reduced more efficiently than by the existing schemes. In the future, performance measurement and analysis comparison with the existing schemes will be conducted via a practical implementation based on the proposed scheme.

Acknowledgments

This research was supported by the MSIP (Ministry of Science, ICT&Future Planning), Korea, under the ITRC (Information Technology Research Center)) support program (NIPA-2014-H0301-14-1010) supervised by the NIPA (National IT Industry Promotion Agency). This work was supported by the Soonchunhyang University Research Fund.

Author Contributions

Su-Hyun Kim and Im-Yeong Lee both designed research and wrote the paper. Both authors have read and approved the final manuscript.

Conflicts of Interest

The authors declare that there is no conflict of interests regarding the publication of this paper.

References and Notes

1. Apache Hadoop. Available online: http://hadoop.apache.org (accessed on 18 July 2014).
2. O'Malley, O.; Zhang, K.; Radia, S.; Marti, R.; Harrell, C. *Hadoop Security Design*; Yahoo Inc.: Santa Clara, CA, USA, 2009.
3. Becherer, A. *Hadoop Security Design: Just Add Kerberos? Really?* iSEC Partners, Inc.: San Francisco, CA, USA, 2010.

4. Ghemawat, S.; Gobioff, H.; Leung, S.T. The Google file system. In Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles, New York, NY, USA, 19–22 October 2003; pp. 29–43.
5. Hubbard, D.; Sutton, M. *Top Threats to Cloud Computing V1.0*; Cloud Security Alliance: Las Vegas, NV, USA, 2010.
6. Shamir, A. How to share a secret. *Commun. ACM* **1979**, *22*, 612–613.
7. Pinch, R.G.E. On-line multiple secret sharing. *Electron. Lett.* **1996**, *32*, 1087–1088.
8. Park, S.-J.; Kim, H. Improving Hadoop Security Through Hash-chain. *J. Korean Inst. Inf. Tech.* **2012**, *6*, 65–73. (In Korean)
9. Jin, S.; Yang, S.; Zhu, X.; Yin, H. Design of a Trusted File System Based on Hadoop. In *Trustworthy Computing and Services*; Springer: Berlin/Heidelberg, Germany, 2013; pp. 673–680.
10. Zhang, D.W.; Sun, F.Q.; Cheng, X.; Liu, C. Research on Hadoop-based enterprise file cloud storage system. In Proceedings of 2011 IEEE 3rd International Conference on Awareness Science and Technology (iCAST), Dalian, China, 27–30 September 2011; pp. 434–437.
11. Zheng, Y. Digital signcryption or how to achieve cost (signature & encryption) << cost (signature) + cost (encryption). In *Advances in Cryptology—CRYPTO'97*, Proceedings of 17th Annual International Cryptology Conference, Santa Barbara, CA, USA, 17–21 August 1997; Lecture Notes in Computer Science, Volume 1294; Kaliski, B.S., Jr., Ed.; Springer: Berlin/Heidelberg, Germany, 1997; pp. 165–179.
12. Kwak, D.; Moon, S. Efficient distributed signcryption scheme as group signcryption. In *Applied Cryptography and Network Security*, Proceedings of the 1st International Conference on "Applied Cryptography and Network Security" (ACNS 2003), Kunming, China, 16–19 October 2003; Lecture Notes in Computer Science, Volume 2846; Springer: Berlin/Heidelberg, Germany, 2003; pp. 403–417.