J Comput Virol Hack Tech (2015) 11:137–142
DOI 10.1007/s11416-014-0236-5

ORIGINAL PAPER

A token-based authentication security scheme for Hadoop distributed file system using elliptic curve cryptography

Yoon-Su Jeong · Yong-Tae Kim

Received: 2 September 2014 / Accepted: 29 December 2014 / Published online: 27 February 2015

Abstract In recent years, a number of platforms for building Big Data applications, both open-source and proprietary, have been proposed. One of the most popular platforms is Apache Hadoop, an open-source software framework for Big Data processing used by leading companies like Yahoo and Facebook. Historically, earlier versions of Hadoop did not prioritize security, so Hadoop has continued to make security modifications. In particular, the Hadoop Distributed File System (HDFS) upon which Hadoop modules are built did not provide robust security for user authentication. This paper proposes a token-based authentication scheme that protects sensitive data stored in HDFS against replay and impersonation attacks. The proposed scheme allows HDFS clients to be authenticated by the datanode via the block access token. Unlike most HDFS authentication protocols adopting public key exchange approaches, the proposed scheme uses the hash chain of keys. The proposed scheme has the performance (communication power, computing power and area efficiency) as good as that of existing HDFS systems.

Special Issue: Convergence Security Systems.

This work was supported by the Security Engineering Research Center, granted by the Korea Ministry of Knowledge Economy.

Y.-S. Jeong
Division of Information and Communication Convergence Engineering, Mokwon University, 88 Doanbuk-ro, Seo-gu, Daejeon 302-318, Korea
e-mail: bukmunro@mokwon.ac.kr

Y.-T. Kim (✉)
Division of Multimedia Engineering, Hannam University, 133 Ojeong-dong, Daedeok-Gu, Daejeon 306-791, Korea
e-mail: ky7762@hnu.kr

1 Introduction

With the growth of social networks and smart devices, the use of Big Data has increased dramatically over the past few years. Big Data requires new technologies that efficiently collect and analyze enormous volumes of real-world data over the network [1–3]. In particular, open-source platforms for scalable and distributed processing of data are being actively studied in Cloud Computing in which dynamically scalable and often virtualized IT resources are provided as a service over the Internet [4].

Apache's Hadoop is an open-source software framework for Big Data processing used by many of the world's largest online media companies including Yahoo, Facebook and Twitter. Hadoop allows for the distributed processing of large data sets across clusters of computers [5,6]. HDFS, Hadoop's distributed file system, is designed to scale up from single servers to thousands of machines, each offering local computation and storage. MapReduce, another Hadoop component, is a scalable, parallel processing framework that works in tandem with HDFS. Hadoop enables users to store and process huge amounts of data at very low costs, but it lacks security measures to perform sufficient authentication and authorization of both users and services [7,8].

Traditional data processing and management systems such as Data Warehouse and Relational Database Management Systems are designed to process structured data, so they are not effective in handling large-scale, unstructured data included in Big Data [9,10]. Initially, secure deployment and use of Hadoop were not a concern [11–13]. The data in Hadoop was not sensitive and access to the cluster could be sufficiently limited. As Hadoop matured, more data and more variety of data including sensitive enterprise and personal information are moved to HDFS. However, HDFS's

lax authentication allows any user to impersonate any other user or cluster services.

For additional authentication security, Hadoop provides Kerberos that uses symmetric key operations [4,6,14]. If all of the components of the Hadoop system (i.e., namenodes, datanodes, and Hadoop clients) are authenticated using Kerberos, the Kerberos key distribution center (KDC) might have a bottleneck. To reduce Kerberos traffic (and thereby putting less load on the Kerberos infrastructure), Hadoop relies on delegation tokens. Delegation token approaches use symmetric encryption and the shared keys may be distributed to hundreds or even thousands of hosts depending upon the token type [15,16]. This leaves Hadoop communication vulnerable to eavesdropping and modification, thus making replay and impersonation attacks more likely. For example, an attacker can gain access to the data block by eavesdropping the block access token [17]. Hadoop security controls require the namenode and the datanode to share a private key to use the block access token. If the shared key is disclosed to an attacker, the data on all datanodes is vulnerable [11,18].

This paper proposes a token-based authentication scheme that protects sensitive HDFS data against replay and impersonation attacks. The proposed scheme utilizes the hash chain of authentication keys, rather than public key-based authentication key exchanges commonly found in existing HDFS systems. The proposed scheme allows clients to be authenticated to the datanode via the block access token. In terms of performance, the proposed scheme achieves communication power, computing power and area efficiency similar to those of existing HDFS systems.

The remainder of this paper is organized as follows. Section 2 describes Hadoop's two primary components: HDFS and MapReduce. Section 3 presents the proposed authentication scheme for HDFS. In Sect. 4, the proposed scheme is evaluated in terms of security and performance. Finally conclusions are given in Sect. 5.

2 Related work

Hadoop is an open-source platform that supports the processing of Big Data in a distributed computing environment [1,2]. Hadoop is made up of stand-alone modules such as a distributed file system called HDFS, a library for parallel processing of large distributed datasets called MapReduce, and so on. HDFS and MapReduce were inspired by technologies created inside Google—Google File System (GFS) and Google MapReduce, respectively [5,7].

2.1 HDFS

HDFS is a distributed, scalable file system that stores large files across multiple machines. Files are stored on the HDFS in blocks which are typically 64 MB in size. That is, an HDFS file is chopped up into 64 MB chunks and data that is smaller than the block size is stored as it is, not occupying the full block space. Figure 1 shows a typical architecture of an HDFS cluster, consisting of a namenode, a secondary namenode and multiple datanodes [6,14].

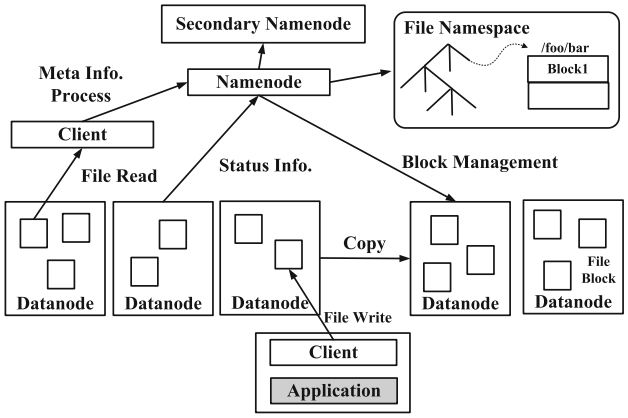

Fig. 1 The architecture of HDFS

HDFS stores its metadata (i.e., the information about file names, directories, permissions, etc.) on the namenode. HDFS is based on a master–slave architecture so a single master node (namenode) maintains all the files in the file system. The namenode manages the HDFS's namespace and regulates access to the files by clients. It distributes data blocks to datanodes and stores this mapping information. In addition, the namenode keeps track of which blocks need to be replicated and initiates replication whenever necessary. Multiple copies of each file provide data protection and computational performance.

The datanodes are responsible for storing application data and serve read/write requests from clients. The datanodes also perform block creation, deletion and replication upon instruction from the namenode.

Datanodes periodically send Heartbeats and Blockreports to the namenode. Receipt of a Heartbeat implies that the datanode is functioning properly. The namenode marks datanodes without recent Heartbeats as dead and does not forward any new I/O requests to them. A Blockreport contains a list of all blocks on a datanode. Blockreports provide the namenode with an up-to-date view of where blocks are located on the cluster. The namenode constructs and maintains latest metadata from Blockreports.

Figure 2 depicts the overall operations of HDFS. As shown in the figure, all servers are fully connected and communicate with each other using TCP-based protocols.

2.2 MapReduce

MapReduce is a framework for processing large data sets in parallel across a Hadoop cluster. Like HDFS, it is based

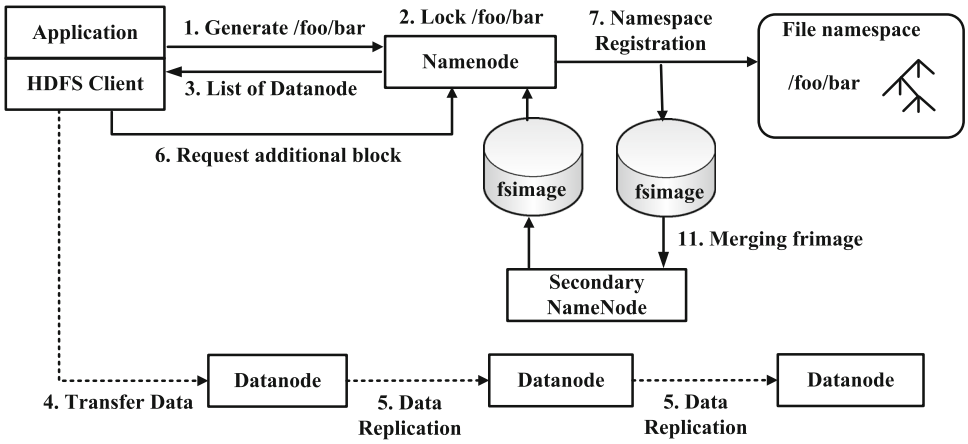

Fig. 2 The operational view of HDFS

on a master–slave model. The master is a special node which coordinates activity between several worker nodes. The master receives the input data that is to be processed. The input data is split into smaller chunks and all these chunks are distributed to and processed in parallel on multiple worker nodes. This is called the Map phase. The workers send their results back to the master node which aggregates these results to produce the sum total. This is called the Reduce phase.

MapReduce operates on key-value pairs. That is, all input and output in MapReduce is in key-value pairs [14]. Conceptually, a MapReduce job takes a set of input key-value pairs and produces a set of output key-value pairs by passing the data through Map and Reduce functions. The Map tasks produce an intermediate set of key-value pairs that the Reduce tasks uses as input [15,17].

In the MapReduce model designed for batch-oriented computations over large data sets, each Map job runs to completion before it can be consumed by a Reduce job. MapReduce Online, a modified MapReduce architecture, allows data to be pipelined between Map and Reduce jobs. This enables Reduce jobs to have access to "early returns" from Map jobs as they are being produced. That is, the Reduce job can start earlier, which reduces MapReduce completion times and improves the utilization of worker nodes. This architecture is effective in continuous service environments where MapReduce jobs can run continuously accepting new data as it arrives and analyzing it immediately [16,18].

3 Elliptic curve-based authentication token generation

This section presents the proposed HDFS authentication scheme that creates delegation tokens using elliptic curve cryptography (ECC).

3.1 Overview

The proposed authentication scheme strengthens the protection of authentication information exchanged between

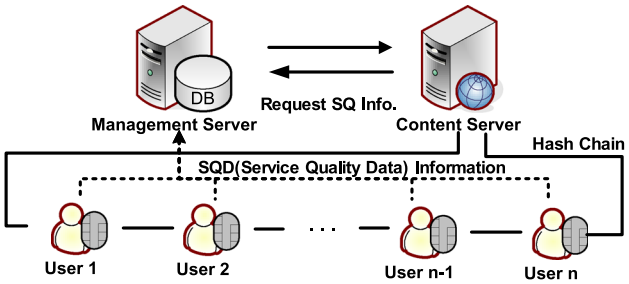

Fig. 3 The overview of the proposed authentication scheme

namenode and datanode. The proposed scheme is designed to operate in a master–slave architecture presented in Fig. 3. That is, it requires an operation environment where a single master node (namenode) coordinates activity between several slave nodes (datanodes). The proposed scheme allows HDFS clients to be authenticated by the datanode via the block access token. Unlike most HDFS authentication protocols adopting public key exchange approaches, the proposed scheme uses the hash chain of keys. The proposed scheme has the performance (communication power, computing power and area efficiency) as good as that of existing HDFS systems. Figure 3 shows an overview of the proposed authentication scheme. The proposed scheme exploits delegation tokens to add another layer of protection to the existing HDFS authentication.

3.2 Assumptions

The proposed authentication scheme has the following assumptions:

- Clock synchronization is provided. That is, all HDFS nodes (client, namenode, and datanode) refer to a common notion of time.
- The symmetric key encryption functions that encrypt plaintext M with private key K share hash function $H()$.

- For secure communication between namenode and datanode, Secure Shell (SSH) is used.

3.3 Terms and definitions

Table 1 presents the terms used in the proposed scheme.

3.4 Token generation

In the proposed scheme, tokens are generated based on ECC and used to enhance the security of authentication messages communicated between namenode and datanode. This offline token generation process consists of the following four steps.

- Step 1: The namenode chooses a random number, as represented in Eq. (1). The namenode then generates a private/public key pair using Eq. (2).

$$d_S \in [2, \mathrm{n}-2] \tag{1}$$
$$Q_S \in d_S \times P \tag{2}$$

- Step 2: The namenode sends public key Q_S generated using Eq. (2) to the datanode. Upon receiving Q_S, the datanode performs Eq. (3) through (6) in order to create a digital signature. This digital signature is sent to the namenode.

$$Choose \quad K_S \in [2, n-2], \quad l_S \tag{3}$$
$$Choose \quad R_S = K_S \times P, K^{-1} \mod P \tag{4}$$
$$t_S = R_{S,X} \tag{5}$$
$$TK_S = K_S^{-1}(H(Q_{S.X}, I_S, t_S) + d_{CH} \times r_S \tag{6}$$

Table 1 Notation

Symbol	Description
E	GF(p) on the elliptic curve
N	The largest number
P	Large prime
Q_X	The public key of x
d_S	The selected private key of [2, n-2]
t_z	Expiration time of the authentication of x
I_x	Temporary identifier of x
$Q_{XY.Z}$	Mutually agreed Key between x and y
e_Z	Selected value at the $h(Q_{xy.Z}, x, t_x, I_x)$
TK_x	Token Information of x
R_x	Points of the elliptic curve x
r_x	Coordinate x values of elliptic curve
(r,S)	Certificate pair for the message

In Eq. (6), temporary datanode identifier I_S and credentials expiry time t_S are included to facilitate managing datanodes. $R_{S,X}$, a pre-shared key between namenode and datanode, is replaced with r_S. A pair of r_S and token TK_S serves as credentials denoted as $(r_S,\ TK_S)$.

- Step 3: In Eq. (7), the datanode sends its public key Q_{CH} to the namenode along with I_S, $(r_S,\ TK_S)$ and t_S.

$$Q_{CH},\ I_S,\ (r_S,\ TK_S),\ t_S \tag{7}$$

- Step 4: Among the credentials sent from the datanode to the namenode, Q_{CH}, I_S and t_S are processed by the hash function, as denoted in Eq. (8). The resulting value e_S is stored along with other authentication information sent by the datanode, as represented in Eq. (9).

$$e_S \in H(Q_S,\ Q_{S.x},\ I_S,\ t_S) \tag{8}$$
$$Store \quad S = Q_S,\ Q_{Ch},\ I_S,\ t_S,\ e_S,\ (r_S,\ TK_S) \tag{9}$$

3.5 Authentication in client and datanode communication

The namenode stores private key K_i that is used in the hash chain and sets seed value $c_{j,0}$ for generating and verifying a delegation token. Seed value $c_{j,0}$ is generated by applying the hash function to random number $c_{i,j}$ n times. $c_{i,j}$ denotes a random number of client i and j. The clients are divided into several groups, each of which has at least n clients. The authentication between client and datanode is performed in the following six steps.

- Step 1: The client sends block access token $(r_S,\ TK_S)$, timestamp t_S and seed value $c_{j,0}$ to the datanode, as represented in Eq. (10).

$$Transfer \quad E_{K_{i,j}}\left((r_S,\ TK_S)\, t_S,\ c_{j,0}\right),\ K_i,\ E_{c_{j,0}}(K_i||K_j) \tag{10}$$

- Step 2: The datanode decrypts the message from the client using private key $K_{i,j}$ that is shared by client and datanode. This is represented in Eq. (11).

$$D_{K_{i,j}}\left((r_S,\ TK_S)\, t_S,\ c_{j,0}\right) \tag{11}$$

- Step 3: The datanode obtains seed value $c_{j,0}$ after the decryption represented in Eq. (11). The datanode decrypts $E_{c_{j,0}}(K_i||K_j)$ using seed value $c_{j,0}$ and key K_i, thereby obtaining key K_j. This is denoted in Eq. (12).

$$D_{K_{i,j}}\left(K_i||K_j\right) \tag{12}$$

- Step 4: The datanode verifies timestamp t_S and authenticates block access token $(r_S,\ TK_S)$. If the client is a

legitimate user, the datanode encrypts the requested data using private key $K_{i,j}$ and sends it to the client.

$$Check \quad (r_S, TK_S), t_S \tag{13}$$

$$Transfer \quad E_{K_{i,j}}(Data) \tag{14}$$

- Step 5: The datanode decrypts the ciphertext presented in Eq. (15) using private key $K_{i,j}$. If the decrypted data is valid, the datanode regenerates $E_{K_{i,j}}((r_S, TK_S)\, t_S, c_{j,0})$, K_i, $E_{c_{j,0}}(K_i||K_j)$ and sends it to the client.

$$\text{Re}\, generate \quad E_{K_{i,j}}\left((r_S, TK_S)\, t_S, c_{j,0}\right), \quad K_i, E_{c_{j,0}}\left(K_i||K_j\right) \tag{15}$$

- Step 6: The client decrypts the data received from the datanode using private key $K_{i,j}$.

4 Evaluation

4.1 Security evaluation

Table 2 show security evaluation of attack type (replay attack, disguise attack, data encryption) used in HDFS, HDFS with public key system, [11] and the proposed scheme. Replay and impersonation attack scenarios that can occur in the authentication of HDFS clients and data were devised and the resilience of the proposed scheme against those attacks was assessed.

An attacker posing as a legitimate client sends the intercepted message $E_{g_{j,n}}(\mathrm{D}||K_i||E_{c_{j,0}}(K_i||K_j))$ and $E_{k_i}(c_{j,n})$ to the namenode. The attacker then receives $E_{g_{j,n}}(\mathrm{B}||K_i||E_{c_{j,0}}(K_i||K_j))$ and $E_{k_i}(c_{j,n})$ from the namenode. However, the attacker does not know K_i and K_j so the attacker cannot decrypt the message sent by the namenode. Therefore, the replay attack fails.

An attacker posing as a legitimate client sends the intercepted message $E_{K_{i,j}}((r_S, TK_S)\, t_S, c_{j,0})$, K_i and $E_{c_{j,0}}(K_i||K_j)$ to the datanode. The datanode verifies timestamp t_S and realizes that it is not a valid request. Note that the attacker is unaware of private key $K_{i,j}$ so the attacker cannot modify the timestamp.

Given that there is a malicious node impersonating a datanode, this node cannot know the request message sent by the client because it is unaware of hash chain value $c_{j,n}$ and $g_{j,n}$, the information kept private by each client. Even if this node is able to produce the block access token, it cannot create $E_{K_{i,j}}((r_S, TK_S)\, t_S, c_{j,0})$, K_i and $E_{c_{j,0}}(K_i||K_j)$ to be sent to the client. Therefore, the propose scheme eliminates the threat of datanode impersonation attacks.

Table 2 Security evaluation

Security analysis	*HDFS*	HDFS with public key system	[11]	Proposed scheme
Replay attack	×	○	○	○
Disguise attack	×	○	○	○
Data encryption	×	○	○	○

4.2 Performance evaluation

In conventional HDFS, the datanode requires only one key vk to issue a delegation token, and it keeps the session key until the delegation token is issued. Compared to conventional HDFS, HDFS with public key-based authentication requires m additional keys at the namenode and m additional keys at the datanode. At the client side, a public key that is shared by the client and the namenode and n additional keys are required.

In the authentication protocol suggested in [11], the namenode does not need to store any key for the client because it receives $E_{mk}(K_{NC})$ sent by the client. Similarly, the datanode does not store any key for the client as it receives $E_{vk}(K_{CD})$ from the client.

In the proposed scheme, the client and datanode store K_i and $E_{c_{j,0}}(K_i||K_j)$ in tables in order to load the block access token in memory. The stored information is revoked when the token expires. The memory required by the proposed scheme is similar to that required by conventional HDFS and the protocol in [11]. Compared to HDFS with public key-based authentication, the proposed scheme requires much less memory.

HDFS with public key-based authentication requires m additional key exchanges between namenode and client and $n \times m$ additional key exchanges between datanode and client, in addition to the 6-round key exchange of conventional HDFS systems. In the proposed scheme and the protocol in [11], all key exchanges are performed during the conventional 6-round authentication process so additional key exchanges are not needed.

Figure 4 is shows the bandwidth and storage time according to the ECC key length. As a result of the Fig. 2, ECC key length is increasing bandwidth and storage time. The time of the total time needed for the key establishment is also consumed more which was added to the decode time of the material only as a message time.

5 Conclusion

As the popularity of cloud-based Hadoop grows, efforts to improve the state of Hadoop security increase. This paper proposes a token-based authentication scheme that aims to provide more secure protection of sensitive HDFS data with-

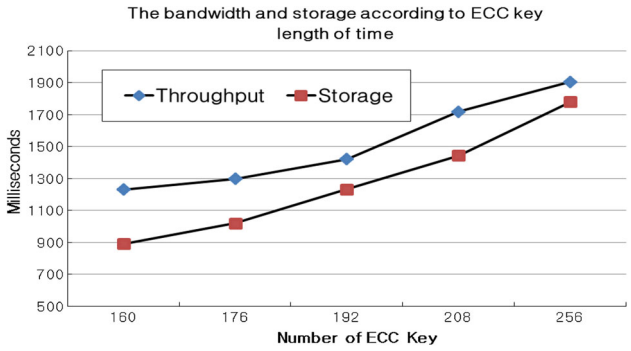

Fig. 4 The bandwidth and storage according to ECC key length of time

out overburdening authentication operations. The proposed scheme adds another layer of protection to the existing symmetric key HDFS authentication. ECC technology employed in the proposed scheme makes generated authentication keys anonymous, thereby protecting them against breaches or accidental exposures. The proposed scheme allows the Hadoop clients to be authenticated by the datanode via the block access token while thwarting replay and impersonation attacks. In addition, the proposed scheme makes use of the hash chain of authentication keys, gaining performance advantage over public key-based HDFS authentication protocols. In the future, ways to protect the privacy of enterprise and personal data stored in HDFS will be studied.

References

1. Apache Hadoop: http://hadoop.apache.org/
2. Barakat, O.L., Hashim, S.J., Raja Abdullah, R.S.A.B., Ramli, A.R., Hashim, F., Samsudin, K., Rahman, M.A.: Malware analysis performance enhancement using cloud computing. J. Comput. Virol. Hacking Tech. **10**(1), 1–10 (2014)
3. Cao, Y., Miao, Q., Liu, J., Gao, L.: Abstracting minimal security-relevant behaviors for malware analysis. J. Comput. Virol. Hacking Tech. **9**(4), 171–178 (2013)
4. Hong, J.K.: The security policy for Big Data of US government. J. Digit. Converg. **11**(10), 403–409 (2013)
5. Ghemawat, S., Gobioff, H., Leung, S.: The google file system. In: Proceedings of the Nineteenth CM Symposium on Operating Systems Principles, vol. 37, issue no. 5, pp. 29–43 (2003)
6. Apache Hadoop MapReduce Tutorial: http://hadoop.apache.org/docs/r1.0.4/mapred_tutorial.html
7. Dean, J., Ghemawat, S.: MapReduce: simplified data processing on large clusters. In: Proceedings of Sixth Symposium on Operating System Design and Implementation (OSDI04), pp. 137–150 (2004)
8. Shvachko, K., Kuang, H., Radia, S., Chansler, R.: The Hadoop distributed file system. In: Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST), pp. 1–10 (2010)
9. Vatamanu, C., Gavilut, D., Benchea, R.M.: Building a practical and reliable classifier for malware detection. J. Comput. Virol. Hacking Tech. **9**(4), 205–214 (2013)
10. Cimpoesu, M., Gavilut, D., Popescu, A.: The proactivity of perceptron derived algorithms in malwawre detection. J. Comput. Virol. Hacking Tech. **8**(4), 133–140 (2012)
11. Hong, J.K.: Kerberos Authentication Deployment Policy of US in Big Data environment. J. Digit. Converg. **11**(11), 435–441 (2013)
12. Jeong, Y.S., Han, K.H.: Service Management Scheme using security identification information adopt to Big Data environment. J. Digit. Converg. **11**(12), 393–399 (2013)
13. Lee, S.H., Lee, D.W.: Current status of Big Data utilization. J. Digit. Converg. **11**(2), 229–233 (2013)
14. White, T.: Hadoop The Definitive Guide, 2nd edn, pp. 41–47. O'Reilly Media, Sebastopol (2009)
15. Condie, T., Conway, N., Alvaro, P., Hellerstein, J.M., Elmeleegy, K., Sears, R.: MapReduce Online. In: Proceedings of NSDI'10 (2010)
16. Lee, M.Y.: Technology of distributed stream computing. Electron. Telecommun. Trends **26**(1) (2011)
17. Dean, J., Ghemawat, S.: MapReduce: simplified data processing on large clusters. Commun. ACM **51**(1), 107–113 (2008)
18. Jeong, S.W., Kim, K.S., Jeong, I.R.: Secure authentication protocol in Hadoop distributed file system based on Hash chain. JKIISC **23**(5), 831–847 (2013)