Journal Pre-proof

Hadoop-based secure storage solution for big data in cloud computing environment

Shaopeng Guan, Conghui Zhang, Yilin Wang, Wenqing Liu

PII: S2352-8648(23)00027-5

DOI: https://doi.org/10.1016/j.dcan.2023.01.014

Reference: DCAN 611

To appear in: *Digital Communications and Networks*

Received Date: 17 March 2021

Revised Date: 19 October 2022

Accepted Date: 17 January 2023

Please cite this article as: S. Guan, C. Zhang, Y. Wang, W. Liu, Hadoop-based secure storage solution for big data in cloud computing environment, *Digital Communications and Networks* (2023), doi: https://doi.org/10.1016/j.dcan.2023.01.014.

Digital Communications and Networks(DCN)

HOSTED BY

ELSEVIER

Available online at www.sciencedirect.com

ScienceDirect

journal homepage: www.elsevier.com/locate/dcan

Hadoop-based secure storage solution for big data in cloud computing environment

Shaopeng Guan*, Conghui Zhang, Yilin Wang, Wenqing Liu

School of Information and Electronic Engineering, Shandong Technology and Business University, Yantai 264003, China

Abstract

In order to address the problems of the single encryption algorithm, such as low encryption efficiency and unreliable metadata for static data storage of big data platforms in the cloud computing environment, we propose a Hadoop based big data secure storage scheme. Firstly, in order to disperse the NameNode service from a single server to multiple servers, we combine HDFS federation and HDFS high-availability mechanisms, and use the Zookeeper distributed coordination mechanism to coordinate each node to achieve dual-channel storage. Then, we improve the ECC encryption algorithm for the encryption of ordinary data, and adopt a homomorphic encryption algorithm to encrypt data that needs to be calculated. To accelerate the encryption, we adopt the dual-thread encryption mode. Finally, the HDFS control module is designed to combine the encryption algorithm with the storage model. Experimental results show that the proposed solution solves the problem of a single point of failure of metadata, performs well in terms of metadata reliability, and can realize the fault tolerance of the server. The improved encryption algorithm integrates the dual-channel storage mode, and the encryption storage efficiency improves by 27.6% on average.

KEYWORDS:
Big data security, Data encryption, Hadoop, Parallel encrypted storage, Zookeeper

1. Introduction

The rapid development of cloud computing provides technical support for big data storage and processing. Hadoop is an open-source big data framework that implements Google's cloud computing system. With a distributed file system HDFS and a distributed computing framework MapReduce as the core, Hadoop provides a distributed infrastructure with transparent underlying details [1], and it can combine multiple ordinary computers or servers to form a big data cluster. Thanks to its advantages in the form of impressive compatibility, great computing power and low cost [2–5], Hadoop is enjoying a vast market in finance, medical treatment, e-commerce and other fields of structured and unstructured data services, thus becoming the mainstream big data platform used by Internet companies.

Currently, more and more data is stored on the cloud platform based on the Hadoop architecture. In order to use the cloud platform resources to process big data more efficiently, Hadoop adapts to various big data processing businesses by adding components and modules [6]. Its flexibility renders Hadoop more versatile. However, it also brings complexity and security issues to the framework [7]. According to the Common Vulnerabilities and Exposures (CVE) vulnerability list, there are currently nearly 6 information disclosure vulnerabilities in Hadoop, which can bring serious security risks [8, 9].

As shown in Fig. 1, in order to store massive amounts of data and perform large-scale parallel computing on them, Hadoop adopts a distributed model to form a big data cluster with multiple servers [10]. The distributed architecture allows the hardware resources

*Corresponding author.
[1]Shaopeng Guan: an associate professor in the School of Information and Electronic Engineering at Shandong Technology and Business University (email:konexgsp@gmail.com).

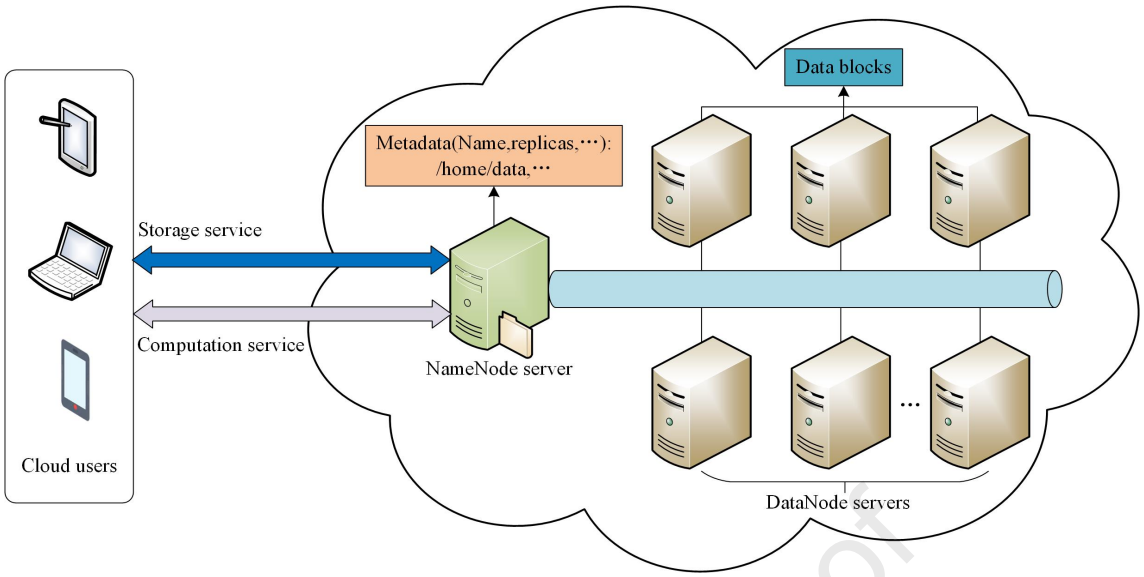

Fig. 1: Hadoop architecture in a cloud computing environment

of the big data platform to be scalable, but it also exposes Hadoop to the following attacks. (1) Security threats caused by NameNode performance and metadata reliability issues. As the master server of the entire cluster, the NameNode must be accessed by the client before reading or writing to HDFS. And any file operation can be recorded by the NameNode in the form of metadata. Therefore, the NameNode is more like the 'steward' of the Hadoop distributed file system. If it fails, it will have an impact on the entire big data system's functionality and may even result in data leakage. With the continuous growth of data, the NameNode encounters a performance bottleneck. Once the NameNode fails, metadata will be lost, which will seriously threaten the data security of the entire big data environment. Therefore, the reliability of metadata is a prerequisite for the secure storage of big data. (2) The security of the data block itself. As a slave node in the Hadoop cluster, the DataNode is also called a data node, which is mainly used to store and for parallel calculations of massive data. A big data cluster usually has hundreds of DataNode servers, and it is difficult for people to have good control over the data access to each node. If a DataNode in the cluster is disconnected from the master node, malicious users can directly obtain stored data from the node, leading to information leakage.

To solve the performance bottlenecks and security problems caused by a single NameNode, Hadoop introduced the HDFS Federation mechanism. This mechanism supports the cluster configuration of multiple NameNodes, but the metadata namespace of each NameNode is relatively independent, and it does not coordinate multiple NameNodes to manage the cluster together. Although this mechanism enhances the scalability of the cluster, there are still problems with the single point of failure of the NameNode and unreliable metadata [11]. The HDFS High Availability (HA) is introduced in Hadoop 2.x [12]. The advantage of this mechanism lies in the fact that when a single NameNode fails, the standby node replaces the original NameNode node management cluster, realizing the great availability of the NameNode server. This mechanism is equivalent to performing a hot backup of metadata on a spare NameNode, but it does not store metadata in multiple nodes, so it cannot solve the metadata storage performance bottleneck.

In terms of data security, Hadoop mainly uses the Kerberos identity authentication mechanism to maintain the security of big data clusters. Only users with successful identity authentication can access cluster resources. However, after passing the identity verification, the user is no longer under the supervision of the authentication system, which makes it impossible to control the illegal operations of legitimate users. Once a malicious user is authenticated, the user can arbitrarily read the data in the cluster by disguising the identity, thus resulting in the leakage of data privacy. Data security also starts with the data block itself. The encryption algorithm stores and transmits data in ciphertext, which can better protect the data from leakage [13, 14]. Therefore, Hadoop introduced a transparent encryption mechanism [15]. In spite of the increased data security, this encryption method is too inefficient in the process of mass data storage. And in a Hadoop cluster, the parallel computing framework cannot be used directly to process ciphertext data. If you want to calculate the data, you must decrypt it first. This process is complicated and risky in that we may face data leakage during the calculation process.

For the above problems, we propose a Hadoop-based secure storage solution for big data in a cloud environment. The solution can not only ensure the safe storage of data but also improve storage efficiency.

Our main contributions are summarized as follows:

(1) By eliminating the single point of failure of the Hadoop master node as an entry point, we propose a dual-channel distributed storage model. Metadata is distributed on multiple nodes, and the Zookeeper distributed coordination mechanism is used to achieve metadata consistency among the nodes.
(2) We improve the ECC encryption algorithm, analyze the reading characteristics of data in Hadoop, and introduce a homomorphic encryption algorithm to realize the secure storage of data in Hadoop.
(3) We design a parallel encrypted storage scheme for big data, and verify the data storage performance through experiments. With the optimization of the dual-channel storage model, the data parallel storage rate is significantly improved.

The rest of this paper is organized as follows. Section 2 introduces the research related to big data secure storage under the current Hadoop architecture. Section 3 describes the proposed model and method, including the metadata reliability model and the encryption scheme. In section 4, the experimental environment configuration is shown, and the experimental results are analyzed. Section 5 is a conclusion.

2. Related works

There are two main security threats after the data is uploaded to the cloud big data platform. First, as an untrusted third party, the cloud platform may cause data leakage in the wake of the server's failure. Second, after the cloud platform is illegally accessed, there are risks of data theft and tampering [16, 17]. At present, the research on Hadoop data security mainly focuses on the reliability of the Hadoop cluster and the security of the data itself.

In the Hadoop architecture, the reliability of NameNode and metadata is the prerequisite for data security in the Hadoop big data cluster. In terms of metadata reliability and storage performance, Jena et al. [18] proposed that by using an aggregator in Hadoop to aggregate data, it can reduce the burden of NameNode, and improve the processing performance of NameNode. Shaha et al. [19] migrated the rarely-used metadata to a separate memory by segmenting metadata, and established a secondary storage mechanism for metadata to ensure the great availability of metadata. Choi et al. [20] put forward a HDFS metadata namespace with non-volatile memory. This scheme accesses metadata through byte addresses to ensure data durability and high performance, alleviating the bottleneck problem of the NameNode performance. Won et al. [21] transferred the metadata in the NameNode to a separate database table to achieve the robustness and high availability of the Hadoop cluster. The methods above all begin with a single NameNode, but without taking into account the NameNode's failure to provide metadata. For this problem, Solissa et al. [22] proposed the use of Distributed Replicated Block Device to quickly transfer the metadata in the failed NameNode. This method implements data-level fault tolerance, which means it can ensure that metadata is transferred in time after the NameNode fails. However, the service will still be interrupted when the NameNode fails. To solve the problem of the failure of a single point and the metadata storage space bottleneck caused by a single NameNode in traditional Hadoop clusters, MOSES et al. [23] proposed a rack-aware model. By deploying Rack_Unit NameNode in each rack, the load of a single NameNode is reduced and the reliability of metadata is increased. This scheme requires the deployment of local NameNode in each rack, and the NameNode of the master node is responsible for overall scheduling, which achieves metadata reliability and brings additional communication consumption. Awaysheh et al. [24] proposed a federated access control model, which constructed Federation HDFS through multiple independent NameNodes. This solution can store different types of data onto different NameNodes, and realize the isolation of different types of metadata. However, there is no hot backup of the metadata in the NameNode, so there is still a single point of failure problem. Wu et al. [25] proposed a metadata dynamic load balancing mechanism based on reinforcement learning, which dynamically adjusts the load according to the performance of the metadata server, breaking the system performance bottleneck caused by the unbalanced load of the metadata server. Although this scheme is suitable for Federation HDFS clusters with multiple NameNodes, it lacks a reliable server fault recovery mechanism, leading to the unavailability of metadata when the NameNode is down.

In terms of data block security protection, the application of encryption algorithms is the most direct and effective way [26–29]. As the encryption algorithm provided by Hadoop is relatively single, it is not suitable for the encryption of many different types of data, and cannot guarantee data security while ensuring data storage efficiency. Many scholars use other encryption algorithms to replace and supplement the encryption algorithm that comes with Hadoop. In regard of the problem of secure storage of IoT data in cloud computing, Mo [30] proposed a data storage scheme based on an improved ant colony algorithm, and discussed the adoption of encryption algorithms to enhance Hadoop data storage. The improved ant colony algorithm is used to rationally schedule Hadoop storage tasks, which realizes the secure storage of IoT data in a shorter time. Regarding the storage, analysis and privacy protection of medical big data, Rallapalli et al. [31] discussed the storage and analysis process of the Hadoop architecture. They proved that

the Hadoop architecture combined with encryption algorithms can process medical big data safely and efficiently. Kareem et al. [32] combined two asymmetric encryption algorithms: RSA and ElGamal, and proposed a method to enhance the confidentiality of Hadoop data. This scheme complements the advantages of the two encryption algorithms, and improves the efficiency of data encryption and decryption. However, the RSA encryption algorithm usually requires a longer key to ensure data security. Kapil et al. [33] combined attribute encryption with a honey encryption algorithm to encrypt data stored in Hadoop. Jain et al. [34] directly introduced a security protection module in HDFS and MapReduce to encrypt the data in transmission to ensure the security of the data transmission process. Song et al. [35] proposed a data encryption scheme that supports both ARIA and AES algorithms on Hadoop, which solved the single problem of the Hadoop encryption algorithm. Heping et al. [36] proposed a hyperchaotic data encryption algorithm based on Hadoop, and they used the MapReduce model to design a chaotic cryptographic system to ensure the security of data in Hadoop. The above research mainly focuses on the encryption and decryption in the process of data storage and transmission under the Hadoop architecture. However, they did not use common encryption algorithms to encrypt and store data. The MapReduce parallel computing framework cannot be used to directly process the ciphertext, and the data after decryption easily leads to leakage. For this reason, some scholars began to study homomorphic encryption algorithms for data protection in cloud computing environments [37]. Literature [38–40] uses a homomorphic encryption algorithm to encrypt the data stored in the cloud platform, which better protects the security of data during transmission and operation. Soo et al. [41] used a homomorphic encryption algorithm to encrypt data in Hadoop, which can make full use of parallel computing resources of big data clusters while ensuring data security. However, the data stored in Hadoop is not all data that needs to be calculated, and the homomorphic encryption algorithm is inferior to ordinary encryption algorithms in performance. Therefore, if the homomorphic encryption algorithm is used to encrypt all the data in Hadoop, it will seriously affect the storage efficiency.

Selective data encryption is considered as a way to reduce computing costs while protecting data security in the cloud. Homomorphic encryption can better support the secure calculation of data in distributed systems, but the efficiency of encryption and decryption is low. Symmetric encryption algorithms such as AES, DES, and 3DES have high encryption efficiency, but they have higher risks of data leakage. When sharing data in a cloud computing environment, encryption and decryption use the same key, and the key needs to be told to the other party, so it easily leads to the leakage of the key. Asymmetric encryption algorithms, such as RSA and ECC [42], use public-key encryption and private key decryption, so it provides security in the cloud computing environment. Studies have shown that 160-bit ECC encryption is equivalent to 1024-bit RSA encryption, and 210-bit ECC encryption is equivalent to 2048-bit RSA encryption [43]. Therefore, for the security issues of data blocks in Hadoop, we propose to use homomorphic encryption and ECC lightweight encryption for data that needs to be calculated and ordinary data, respectively. This solution can ensure data security while considering encryption efficiency.

3. Models and methods

As shown in Fig. 2, we mainly start from the two aspects of metadata reliability and data block security. The proposed secure storage model can be divided into three modules: HDFS control module, metadata reliability protection module, and data encryption storage module. Among them, the HDFS control module is mainly responsible for establishing a connection between the client and the NameNode server in the Hadoop cluster. It is also responsible for initializing the encryption algorithm. The metadata reliability protection module distributes the services in the NameNode to multiple servers and divides them into two groups. These servers jointly manage the entire cluster and provide external services. Furthermore, the metadata of the two NameNodes in each group is consistent, and only one NameNode in the 'Active' state provides external services at the same time, and the other NameNode acts as a backup node to perform the hot backup of the metadata to ensure that the node is in standby mode. When the NameNode fails, the standby node can directly take over and manage the cluster. The data encryption storage module mainly adopts encryption algorithms and improved ones. We use improved lightweight encryption and homomorphic encryption for ordinary data and text data that need to be calculated. And we use dual-threaded data encryption and storage to increase the security of data storage and computing.

Proper nouns and abbreviations involved in the article are explained in Tab. 1.

3.1. Metadata reliability protection model

Reliable metadata is a prerequisite for big data security, and metadata is mainly stored in the NameNode server. Metadata will lose control when the NameNode fails. We have built several backup NameNodes for Hadoop to implement hot backup of metadata and to reduce the probability of metadata loss due to a single point of failure. Furthermore, we use the Zookeeper distributed coordination service [44] to automatically switch the service to the standby node when the NameNode fails. We deploy Zookeeper on

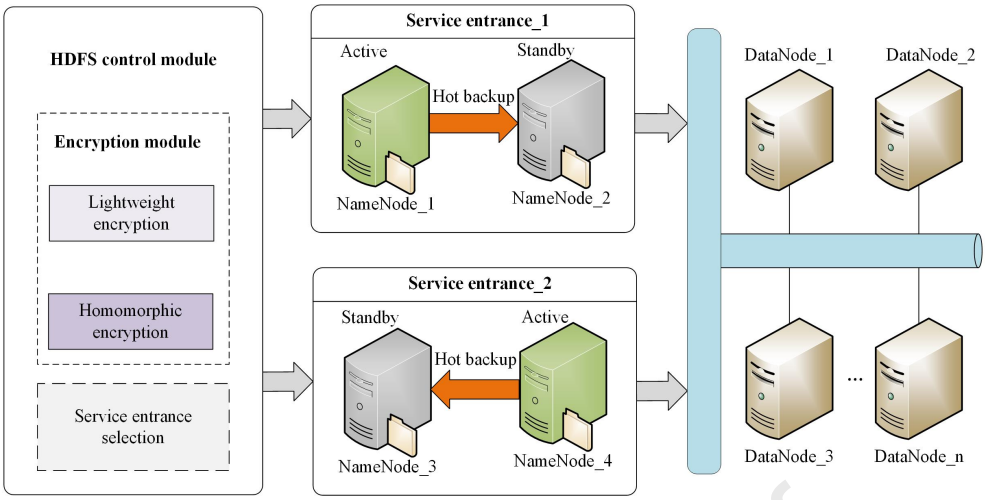

Fig. 2: Secure storage model

Term	Abbreviation	Description
NameNode	NN	The Name node. It manages namespace and data block location by metadata.
DataNode	DN	The data node is responsible for data block storage and task calculation.
ResourceManager	RM	Responsible for the process of resource coordination and management.
NodeManager	NM	The agent process on a single node reports information to the RM and monitors the resources of the node.
Zookeeper FailoverController	ZKFC	Mainly used to achieve fault recovery between two NNs.
QuorumPeerMain	QPM	Entry process of Zookeeper.

Tab. 1: Explanation of professional terms and abbreviations

the NameNode nodes of the Hadoop cluster to monitor the node status in real time. When one of the NameNodes fails, Zookeeper immediately starts the hot backup node to continue providing services to the cluster. The steps of Zookeeper to realize the automatic switchover of failed nodes are as follows:

Step 1 Start listening service. After starting all services, the main thread starts to wait in a loop.

Step 2 Monitoring and detection. The HealthMonitor thread periodically sends data packets to NN to check the status of NN. ZKFC uses a health-check command to pin the NameNode on the same host periodically. As long as the NameNode responds to its health status in time, ZKFC considers the node to be healthy; otherwise, the health monitor identifies the node as unhealthy.

Step 3 Zookeeper session management. When the local NameNode is healthy, ZKFC maintains a session opened in Zookeeper. When the state of NN changes, the HealthMonitor thread calls back and feeds back the changing state of NN to the failover controller.

Step 4 Node selection in Zookeeper. The failover controller receives the feedback information and invokes the active node selector to manage the node status on Zookeeper.

Step 5 Service isolation. When a node is in the "Active" state, isolate its backup node, keep the backup node in standby state, and synchronize the metadata information of the NameNode. Once the NameNode fails, the standby node will start immediately, and then provide services for the cluster.

The failover process is shown in Fig. 3.

Zookeeper has a failover controller for Active and Standby nodes. We use Zookeeper's failover function to ensure that the Standby node can switch to the Active state, restore the metadata access service in the cluster, and ensure the cluster Normal operation when the Active NameNode fails. To ensure the synchronization of the metadata of the two NameNodes, we provide a log sharing management module. The logs generated by the Active NameNode record metadata changes in the namespace, and they are stored in the log sharing module. Then, the Standby node reads these shared logs and applies the metadata information in the log to its namespace, thus realizing the real-time synchronization of the metadata in the two NameNodes, breaking the metadata information island, and

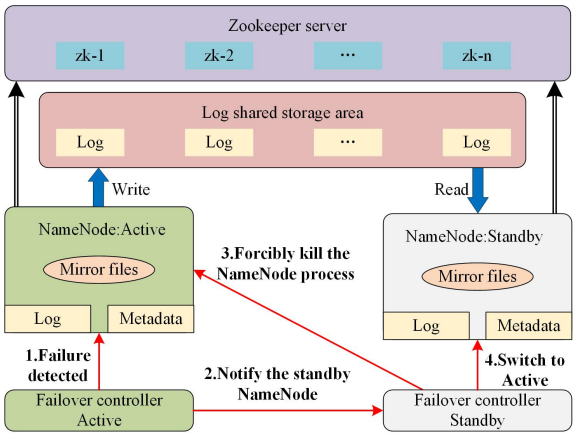

Fig. 3: The process of failover

ensuring the metadata reliability.

In Fig. 3, when the Active faulty controller detects that the NameNode is failed (Step 1), it immediately informs the Standby faulty controller (Step 2). After receiving the failure notification, the Standby fault controller forcibly shuts down the remaining NameNode related processes, and then switches the node to the Standby state to prevent conflicts when starting a new NameNode service process (Step 3). When all the processes related to the failed NameNode are stopped, the Standby failure controller switches its node to the Active state, ensuring that the metadata is not lost and completing the recovery of the failed service.

3.2. Data encryption and storage

3.2.1. Improved ECC lightweight encryption algorithm

ECC is a cryptosystem of the discrete logarithm problem of the elliptic curve [45]. The elliptic curve E can be defined as shown in formula 1:

$$E: \quad y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \tag{1}$$

where $a_i(i = 1, 2, 3, ..., 6) \in K$, and $\Delta \neq 0$, K is the defined rational number field, and Δ is the discriminant of the elliptic curve equation. The specific definition is shown in formula 2:

$$\begin{cases} \Delta = -d_2^2d_8 - 8d_4^3 - 27d_6^2 + 9d_2d_6 \\ d_2 = a_1^2 + 4a_2 \\ d_4 = 2a_4 + a_1a_3 \\ d_6 = a_3^2 + 4a_6 \\ d_8 = a_1^2a_6 + 4a_2a_6 - a_1a_3a_4 + a_2a_3^2 - a_4^2 \end{cases} \tag{2}$$

When the elliptic curve E satisfies the formula 2, the formula 1 is called the Weierstrass equation [46]. Formula 1 is simplified, that is, the general expression of the elliptic curve is formula 3:

$$y^2 = x^3 + ax + b \tag{3}$$

where $(a, b, x, y) \in Fp$, Fp is a finite field, and p is a large prime number.

ECC encryption needs to select an elliptic curve $Ep(a, b)$ and takes a point on the elliptic curve as the base point G. Then it needs to select a private key k, and generates a public key $K = kG$, where K and G are points on the elliptic curve $Ep(a, b)$. Finally, the plaintext is embedded in a point of the elliptic curve, and the public key K is used to complete the encryption of the plaintext.

The ECC encryption algorithm can use a shorter key to achieve higher security. The asymmetric encryption method is more suitable for distributed storage, but its encryption efficiency is lower than that of the symmetric encryption algorithm when the amount of data is large. For this reason, we have improved the ECC algorithm in terms of the encryption mode and storage method to improve its encryption efficiency. We use a dual-threaded encryption method in the encryption process because of the benefits of the dual-channel storage method. This can ensure that encryption and storage are performed simultaneously without affecting each other. To improve the efficiency of encryption and decryption of a single file, we first segment the data, and then encrypt and store the segmented data blocks. Since the encrypted file is not only a string, but also various types of data, we use byte stream encryption. Moreover, processing a large number of small files will significantly reduce the performance of Hadoop, since the more blocks the original file is divided into, more small files there are. In order to better integrate the encryption algorithm with the dual-channel storage mode, we only split the file into two before encrypting the file, and the two pieces of data correspond to two service entries. This method not only ensures that the two pieces of data are encrypted and stored at the same time, but also avoids generating more small files. Besides, we first judge the file size before splitting the data, and the file that reaches the threshold will be split. A small file in Hadoop refers to a file whose size is smaller than the storage block size of HDFS (128MB in Hadoop 2.x). Therefore, in order to ensure that the two pieces of data after splitting are both larger than 128MB, we only implement it for files larger than 256MB Data segmentation operation to avoid too many small files. The dual-thread encryption model is shown in Fig. 4.

The improved ECC encryption algorithm is shown in Algorithm 1.

The improvement of the encryption algorithm also closely follows the characteristics of the dual-channel storage mode. Each thread corresponds to a service entry, which improves the efficiency of encrypted storage and breaks the single point of metadata.

Dual-threaded encrypted storage is shown in Algorithm 2.

3.2.2. The Paillier homomorphic encryption

The data that needs to be calculated is usually stored in HDFS in the form of text files (.txt). In order to en-

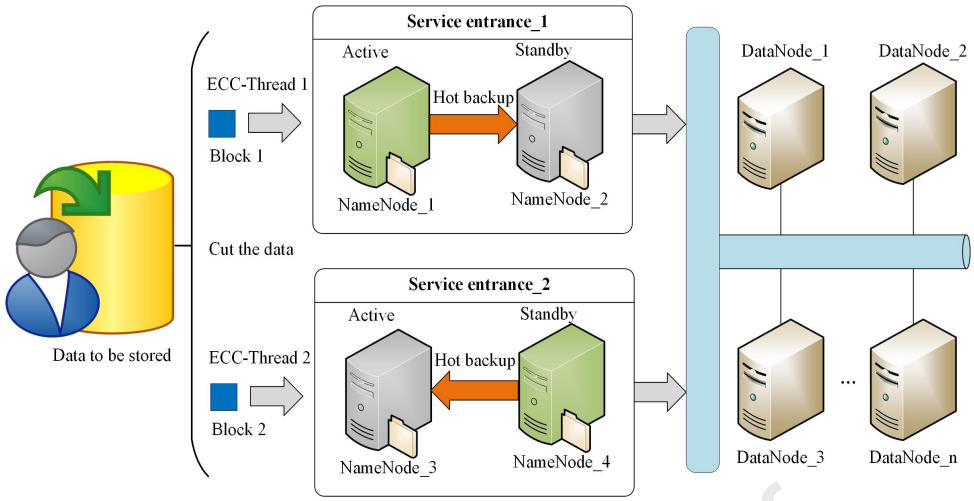

Fig. 4: Improved ECC parallel encryption

Algorithm 1 Improved dual-thread ECC encryption algorithm (DH-ECC).

Input: Initialize ECC parameters and files to be encrypted.

Output: Encrypted files.

1: The client gets the file information;
2: **if** File size > 256MB, **then**
3: Cut into two pieces;
4: Elliptic curve generation $Ep(a, b)$;
5: Obtain a group of elliptic curves (x, y);
6: **while** $x \leq p - 1$, **do**
7: Obtain y from elliptic curve $y^2 = x^3 + ax + b$;
8: Get all the points that satisfy $Ep(a, b)$, and get the base point $G(x_0, y_0)$;
9: Randomly generate an integer n;
10: Read data in a buffered byte stream, and XOR the read bytes with n;
11: Map byte information to elliptic curve to realize encryption;
12: Output encrypted files;
13: **end while**
14: **end if**

Algorithm 2 Encrypted storage thread.

1: Call algorithm 1, start the encryption thread: Thread1.start(), Thread2.start();
2: Obtain Hadoop configuration information and service entry information;
3: **if** The service entrance is normally connected, **then**
4: The two threads correspond to the two service entries and save the encrypted file, which is recorded as metadata;
5: **else**
6: **if** The standby node IP is normal, **then**
7: Get the connection, and save the encrypted file;
8: **else**
9: Get connected with another service portal;
10: The encrypted file is stored in HDFS;
11: **end if**
12: **end if**

sure that the information is not leaked when the parallel computing framework is used to perform massive data computing tasks, we use the Paillier homomorphic encryption algorithm to encrypt it. The steps are as follows:

Step 1 Algorithm initialization:

(1) Randomly generate two large prime numbers p and q, and let $n = p \times q$;

(2) Take the least common multiple λ of $p - 1$ and $q - 1$, which is specifically expressed as formula 4:

$$\lambda = lcm(p - 1)(q - 1) \tag{4}$$

where lcm represents the least common multiple.

(3) Choosing a random integer g needs to satisfy the order that n can divide g, which is specifically expressed as formula 5:

$$gcd(g^{\lambda} \bmod n^2, n) = 1 \tag{5}$$

where gcd represents the greatest common divisor, the public key is $pk = (n, g)$, and the private key is $sk = (p, q)$;

Step 2 Encryption and decryption process:

(1) Encryption stage. For plaintext m, $m \in Z_n$, and $m < n$. Choose a random number r to assist encryption, where $r \in Z_n^*$, and $r < n$. Then the encrypted ciphertext is c, which can be expressed as formula 6:

$$c = E(m, r) = g^m \times r^n \bmod n^2 \tag{6}$$

where $E(m, r)$ represents the m encryption process, and r is a random number generated by auxiliary encryption.

(2) Decryption stage. For the ciphertext c, the plaintext obtained by decryption is formula 7:

$$m = D(c, sk) = \frac{L(c^{\lambda} \bmod n^2)}{L(g^{\lambda} \bmod n^2)} \bmod n \qquad (7)$$

where $L(u) = \frac{u-1}{n}$.

Since the data to be encrypted is text and character strings, in order to increase the speed of reading and writing files during encryption and decryption, we use a buffered character stream to read in plaintext and write out ciphertext. The buffered character stream improves the data processing speed by reducing I/O (Input/Output) operations in the process of reading/writing data. In one I/O operation, the buffered character stream can read/write 8k (the default value) bytes of data, while the regular physical stream can only read/write 1 byte of data. That is, using buffered character streams can significantly reduce the number of I/O operations for the same scale of data. Additionally, when using the MapReduce distributed computing framework to calculate data, the text data is usually a row of data, and there are certain arrangement rules. To ensure that MapReduce can normally recognize the arrangement rules after encrypted storage, we have improved the Paillier algorithm. We implement string segmentation when reading data, and encrypt the string corresponding to each field separately. When writing the ciphertext, we can arrange it in the format of the plaintext.

The Paillier homomorphic encryption algorithm used is shown in Algorithm 3.

Algorithm 3 Paillier homomorphic encryption.

Input: Initialize Paillier parameters and files to be encrypted.

Output: Encrypted file.

1: Randomly generate a large prime number p and q, and get λ according to formula 4;
2: Randomly generate integer g, and $g \in (0, 1, 2, ..., 100)$;
3: **if** $gcd(g^{\lambda} \bmod n^2, n) \neq 1$, **then**
4: Regenerate g;
5: **else**
6: Read plaintext data line by line in character stream mode;
7: **while** Read row data is not empty, **do**
8: Cut a line of data that has been read according to "tab";
9: Establish a buffered output character stream, and encrypt the cut line of content and write it out in plain text;
10: **end while**
11: Close the buffered input and output character stream;
12: Output encrypted files encapsulated in the format;
13: **end if**

In summary, the big data security storage model we propose considers both the reliability of metadata and the security of data blocks, and stores metadata in multiple NameNodes. A real-time synchronization and failure recovery scheme leveraging the Zookeeper for metadata is designed to realize the high availability of big data clusters. Additionally, the model provides two data storage channels. We design an encryption algorithm based on the storage model, and use dual threads to store data from the two data storage channels in parallel, which not only achieves secure data storage but also improves the data storage efficiency.

4. Experiment and analysis

4.1. Experimental environments

We use VMware virtual machines to build a Hadoop distributed cluster, and the experimental environment software configuration information is provided in Tab. 2. The Hadoop cluster deployment information is provided in Tab. 3.

Virtual machine version	VMware15
Hadoop version	Hadoop2.7.7
Java version	JDK 1.8.0-131
Operating system	Centos7.0
Cluster size	4 nodes
Computer CPU	Intel Core i7-10750H

Tab. 2: Software configuration of the experimental environment

Items	The name of the node	Description of the role
Namespace-1	Hadoop-1	NN1,DN,RM, NM,ZKFC,QPM
	Hadoop-2	NN2,DN,NM, ZKFC,QPM
Namespace-2	Hadoop-3	NN3,DN,RM, NM,ZKFC,QPM
	Hadoop-4	NN4,DN,NM, ZKFC,QPM

Tab. 3: Information of cluster deployment

4.2. Metadata reliability testing and analysis

In order to test the reliability of the metadata of the proposed scheme, we forcefully kill one of the NameNode processes in the Hadoop cluster that normally started, simulate its failure, and check the status of other nodes in the cluster and whether HDFS can work through the Hadoop browser client. Fig. 5 shows the status of each node when the Hadoop starts normally.

According to Fig. 6, we can see that when the Active NameNode server fails, Zookeeper will switch the Standby node to Active after detecting the failed node.

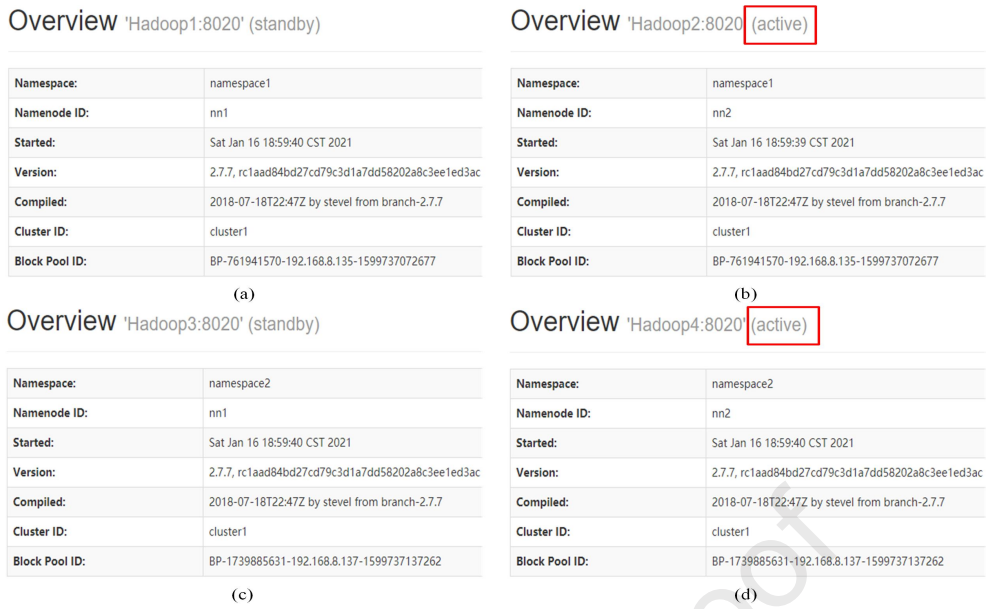

Fig. 5: The normal state of Hadoop

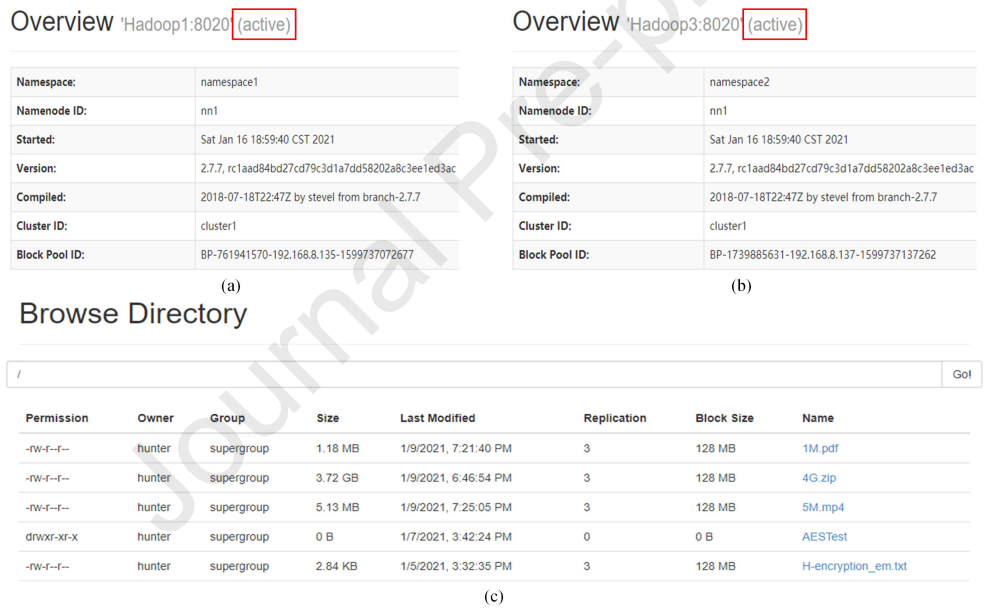

Fig. 6: Node status after one of the NameNodes fails

It ensures the normal operation of HDFS, and all services of the cluster are managed by the new NameNode server, ensuring that the Hadoop nodes are still under server control. We can now ensure that the failed node is repaired as part of the cluster's normal operation, and the repaired node only needs to restart the service to perform a hot backup of the current metadata as a standby node. After the failed node is repaired, the roles of the original active and standby nodes are reversed. That is, the original standby node becomes the current active node, and the original active node runs as the standby node. Most importantly, metadata is allocated to multiple nodes in a distributed manner, effectively alleviating the performance bottleneck of a single NameNode.

4.3. Encryption algorithm testing and analysis

4.3.1. Encryption efficiency experiment and analysis

We compare the improved dual-thread ECC encryption algorithm (DH-ECC) with AES, DES, and other encryption algorithms used in traditional Hadoop, and adopt encryption speed and HDFS storage speed as evaluation indicators. For each group of experiments, we experimented 20 times separately. After removing the highest and the lowest values, we took the average value as the final experimental result. The result of the encryption experiment is shown in Fig. 7.

As shown in Fig. 7, when the data scale is less than 2G, the encryption speed of DH-ECC is slightly lower

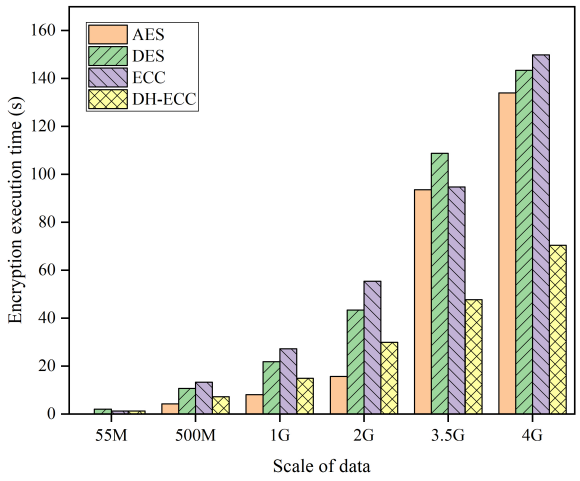

Fig. 7: Data encryption time with different scales of data

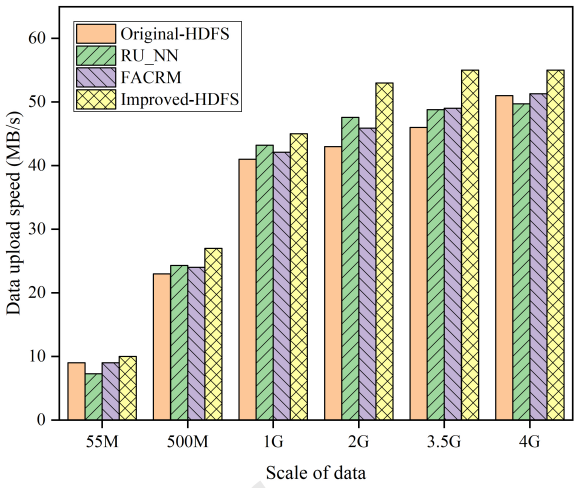

Fig. 8: Data upload rate with different scales of data

than that of the AES encryption algorithm. However, compared to the ECC encryption algorithm before improvement, the encryption efficiency is improved. As the asymmetric encryption algorithm has high security, the encryption process is not a simple byte round conversion operation, so the encryption speed is slightly lower than that of the symmetric encryption algorithm. In addition, when the data scale is small, the advantages of dual-threading can't be well reflected. When the data is larger than 2G, it can be seen that the encryption efficiency of the AES and DES encryption algorithms is significantly reduced. The main reason is that AES and DES encrypt data through multiple rounds of row shifting and column confusion. When the data is large, multiple rounds of bytes will seriously affect its encryption speed. Due to the dual-thread encryption mode, the file reading method of buffered byte streams, and the addition of an exclusive OR operation on bytes, the DH-ECC algorithm performs best. The dual-thread encryption mode can make better use of system resources when the data is large. The way of buffering the byte stream to read files can improve the speed of reading and writing data.

4.3.2. Comprehensive analysis of encrypted storage

We not only improve the encryption algorithm, but also fuse the encryption algorithm with the proposed dual-channel storage mode to achieve safe and efficient data storage. Therefore, the upload efficiency of HDFS is also an important indicator of experimental evaluation. We first conduct a separate data upload experiment to test the data upload performance of the proposed dual-channel storage scheme (Improved-HDFS). And then we conduct a comprehensive experiment on the process of data encryption and upload to analyze the overall efficiency. In the data upload experiment, we compare and analyze the proposed scheme with the schemes in literature [23] and literature [24]. Literature [23] proposed a rack-aware model where each rack is provided with a Rack_Unit NameNode (RU_NN), which uses a Standby NameNode to hot backup metadata to ensure the great availability of the Hadoop cluster, and uses multiple unit NameNodes to reduce the workload of the master node. Literature [24] also configures multiple NameNodes for Hadoop. The difference is that this scheme proposes a Federated Access Control Reference Model (FACRM). Each NameNode is independent of each other, which provides multiple independent namespaces for Hadoop clusters. These two schemes are similar to the dual-channel storage solution technology and framework we proposed, and they are optimized for Hadoop through multiple NameNodes. In the comprehensive experiment of encrypted uploading, AES, DES and ECC encryption algorithms are used for experimental comparison. The results of the data upload experiment are shown in Fig. 8. The results of the encrypted upload comprehensive experiment are shown in Fig. 9.

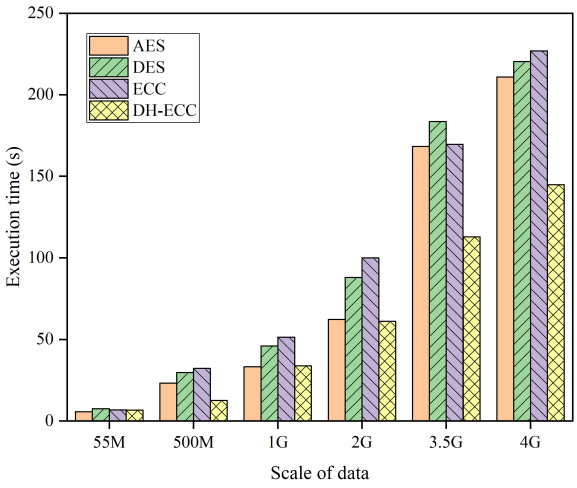

Fig. 9: Encryption combined with uploading comprehensive test results

As shown in Fig. 8, our improved dual-channel stor-

age scheme (Improved-HDFS) has a faster data upload speed compared with other schemes. As we have configured two Active NameNodes for the cluster, the two storage portals can upload data at the same time. Theoretically, the data upload speed of the improved dual channel mode should be twice that upload speed of a single NameNode. However, due to the limitations of the network bandwidth and disk read and write speed, the upload speed does not reach the theoretical value, but compared with the traditional HDFS, the upload speed is promoted. RU_NN uses a Standby NameNode to hot backup metadata, and each rack sets the unit NameNode. This solution increases the reliability of metadata and reduces the workload of the master node. But at the same time, only one Activate NameNode can provide data upload services. FACRM uses Federation HDFS. Although there are multiple NameNodes, these NameNodes are independent of each other, which is equivalent to a combination of multiple single-node NameNodes. It is suitable for the partition storage of different types of data. The data upload speed is almost the same as RU_NN. However, compared with the traditional HDFS of a single NameNode, the data upload speed of these two schemes has been improved.

As shown in Fig. 9, the total time of DH-ECC encrypted upload is less than other encrypted storage methods under different scales of data. The comparison between Fig. 7 and Fig. 9 shows that although the DH-ECC encryption rate is slightly lower than other encryption algorithms when the amount of data is small, combined with the dual-channel storage model we proposed, the overall encryption upload efficiency is better than other algorithms. Moreover, when the amount of data is large, the overall efficiency of the DH-ECC algorithm is relatively stable, and it can better reflect the advantages of the improved dual-thread encrypted storage solution.

4.3.3. DH-ECC safety analysis

Take $y^2 = x^3 + 6x + 12(\text{mod}23)$ as an example. After processing according to the ECC multiplication and addition operation rules, the figure is no longer an elliptic curve in the geometric sense, but a series of discrete points, as shown in Fig. 10. The encryption process is to map the data to the elliptic curve, and the irregularly scattered points make it more secure. In order to display the experimental data clearly, here we only set p to 23, and in the actual encryption process, p uses 131 (p is usually greater than 120 to achieve higher security). Therefore, the actual data encryption is more secure.

Research shows that ECC can use a shorter key to achieve higher security. The key length and confidentiality period corresponding to the same security level are shown in Tab. 4. Compared with symmetric encryption algorithms, ECC adopts public key encryption and private key decryption methods, which can

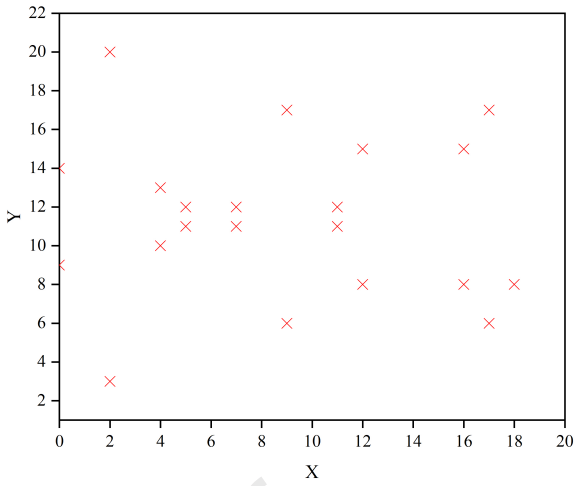

Fig. 10: Scatter plot of DH-ECC

be better suited for distributed data storage in a cloud computing environment. It can use public keys to encrypt data without revealing the private key, making better use of distributed resources.

4.4. Paillier homomorphic encryption algorithm analysis

The purpose of using the homomorphic encryption algorithm is to better adapt to MapReduce distributed computing. The ciphertext after homomorphic encryption can be viewed through the terminal. Fig. 11 shows the effect of homomorphic encryption.

As the Paillier encryption algorithm encrypts data through large prime numbers, the ciphertext can be calculated. The data encrypted by Paillier is stored in HDFS in the form of text files (.txt), which can be better used by distributed computing frameworks such as MapReduce and Spark. Also, when we encrypt data, we read data line by line in a buffered character stream and arrange the encrypted ciphertext according to the arrangement of the plaintext. The encrypted ciphertext cannot see its arrangement rules, but distributed computing frameworks such as MapReduce can read and calculate the ciphertext in rows, which is better suited to the Hadoop distributed computing framework.

```
[hunter@Hadoop1 hadoop-2.7.7]$ bin/hadoop fs -cat /usr/H-encryption_em.txt
200756331354531808826063225689524166699282362575392216969861592621035731710
94259275033411  218245966237531667267414189143840967558031866643406209084721
219528283441998171151710075321
138716476689288467814812827278509040137559259657707506332445433925207151710
5112321445752  189801543954049727287550677905844473067136166378478097745910
932201379692706474313085141971
120621101064295542594502345217700893863621206881713531613316991270984605000
9831084613457  914632441681875497132079548445737943454032824684968602384037
376056693706046183665978065421
212559871824592430592933280648355885702063795478028857823794072217483174421
7199297264119  151847414779232788682114113445776209803423504091337632970311
1135466272512763489948005730648
488214566562378908786567887876412087031999826538430088420486352667028492193
9810991690  377616875585860658887074860255885699316856701379300971359338
1081074610827001027640502992151
518269492863454755573819553798729496864587225566532690425428745039595041291
002551760375  152861658074416029464374390213018746414662147801750251997683
1191771682426448338286916511
166621072966754142518892422293315849926930074517743064660514057804136544000
7037769400694006  946843554107459000019053093042994334286882586336547536359838
7388111874049931934481286455
```

Fig. 11: Ciphertext after homomorphic encryption

Confidentiality level	The key length of RSA (bit)	The key length of ECC (bit)	Confidentiality period (years)
256	15360	512	2120
192	7680	384	2080
128	3072	256	2040
112	2048	224	2030
80	1024	160	2010

Tab. 4: Security level and key length analysis [43]

5. Conclusion

Hadoop is a typical big data framework in the cloud computing environment. It uses distributed storage to expose data to various security threats. We analyzed the impact of the reliability of metadata and the security of data blocks on the secure storage of big data under the Hadoop architecture and we proposed a safe and efficient big data storage solution. In terms of metadata reliability, we combine HDFS federation and HDFS high-availability mechanisms to realize the fault tolerance of the NameNode server, which breaks the single point of failure of metadata in traditional Hadoop. To protect the security of data blocks, we improved the dual-threaded ECC encryption algorithm to achieve the efficient encrypted storage of data in Hadoop. Furthermore, we add a buffered character stream reading method to the Paillier homomorphic encryption algorithm, so that it can better support MapReduce ciphertext calculations. The selective encryption method ensures data security while considering encryption efficiency. Experiments in the Hadoop distributed system show that the proposed scheme can effectively improve the efficiency of encrypted storage of Hadoop, solve the performance bottleneck problem caused by a single NameNode server, and store data safely and efficiently. In the cloud computing environment, the use of homomorphic encryption algorithms enhances the security of data in the operation process, but the encrypted ciphertext will be significantly expanded, which will increase the amount of calculation. In future work, we will invest in research on the ciphertext calculation of homomorphic encryption algorithms, and combine the Spark computing framework to achieve the efficient calculation of ciphertext.

Declaration of competing interest

The authors declare that they have no competing interests.

References

[1] G. Li, J. Tan, S. S. Chaudhry, Industry 4.0 and big data innovations, Enterprise Information Systems. 13 (2) (2019) 145–147.

[2] C. Wen, J. Yang, L. Gan, Y. Pan, Big data driven internet of things for credit evaluation and early warning in finance, Future Generation Computer Systems. 124 (6) (2021) 295–307.

[3] X. Zhang, Y. Wang, Research on intelligent medical big data system based on hadoop and blockchain, EURASIP Journal on Wireless Communications and Networking. 2021 (1) (2021) 1–21.

[4] S. Bhavsar, P. Shah, T. Trambadiya, A survey on e-commerce log analysis using hadoop, International Journal of Computerences & Engineering. 7 (3) (2019) 486–489.

[5] X. Huang, W. Yi, J. Wang, Z. Xu, Hadoop-based medical image storage and access method for examination series, Mathematical Problems in Engineering. 2021 (3) (2021) 1–10.

[6] S. Mazumdar, D. Seybold, K. Kritikos, Y. Verginadis, A survey on data storage and placement methodologies for cloud-big data ecosystem, Journal of Big Data. 6 (1) (2019) 1–37.

[7] W. Rajeh, Hadoop distributed file system security challenges and examination of unauthorized access issue, Journal of Information Security. 13 (2) (2022) 23–42.

[8] G. S. Bhathal, A. Singh, Big data: Hadoop framework vulnerabilities, security issues and attacks, Array. 1-2 (4) (2019) 1–8.

[9] G. Kapil, A. Agrawal, R. A. Khan, Big data security challenges: Hadoop perspective, International Journal of Pure and Applied Mathematics. 120 (6) (2020) 11767–11784.

[10] B. H. Husain, S. R. Zeebaree, et al., Improvised distributions framework of hadoop: A review, International Journal of Science and Business. 5 (2) (2021) 31–41.

[11] M. Naisuty, A. N. Hidayanto, N. C. Harahap, A. Rosyiq, G. M. S. Hartono, Data protection on hadoop distributed file system by using encryption algorithms: a systematic literature review, Journal of Physics Conference Series. 1444 (4) (2020) 1–8.

[12] A. Oriani, I. C. Garcia, From backup to hot standby: High availability for hdfs, in: 2012 IEEE 31st Symposium on Reliable Distributed Systems, IEEE, 2012, pp. 131–140.

[13] I. Hababeh, A. Gharaibeh, S. Nofal, I. Khalil, An integrated methodology for big data classification and security for improving cloud systems data mobility, IEEE Access. 7 (12) (2019) 9153–9163.

[14] Y. Kim, J. Son, R. M. Parizi, G. Srivastava, H. Oh, 3-multi ranked encryption with enhanced security in cloud computing, Digital Communications and Networks. 3 (7) (2022) 1–18.

[15] F. Jiang, Z. Pan, Q. Li, L. Huang, D. Zhang, Research on the application of transparent encryption in distributed file system hdfs, in: 2020 19th International Symposium on Distributed Computing and Applications for Business Engineering and Science (DCABES), IEEE, 2020, pp. 1–4.

[16] A. J. Ouda, A. N. Yousif, A. S. Hasan, H. M. Ibrahim, M. A. Shyaa, The impact of cloud computing on network security and the risk for organization behaviors, Webology. 19 (1) (2022) 195–206.

[17] A. Razaque, M. B. H. Frej, B. Alotaibi, M. Alotaibi, Privacy preservation models for third-party auditor over cloud computing: A survey, Electronics. 10 (21) (2021) 1–22.

[18] B. Jena, M. K. Gourisaria, S. S. Rautaray, M. Pandey, Name node performance enlarging by aggregator based hadoop framework. 6 (2) (2017) 112–116.

[19] T. R. Shaha, M. N. Akhtar, F. T. Johora, M. Z. Hossain, M. Rahman, R. B. Ahmad, A noble approach to develop dynamically scalable namenode in hadoop distributed file system using secondary storage, Indonesian Journal of Electrical Engineering and Computer Science. 13 (2) (2019) 729–736.

[20] W. G. Choi, S. Park, A write-friendly approach to manage

namespace of hadoop distributed file system by utilizing non-volatile memory, The Journal of Supercomputing. 75 (10) (2019) 6632–6662.

[21] H. Won, M. C. Nguyen, M.-S. Gil, Y.-S. Moon, K.-Y. Whang, Moving metadata from ad hoc files to database tables for robust, highly available, and scalable hdfs, The Journal of Supercomputing. 73 (6) (2017) 2657–2681.

[22] D. F. Solissa, M. Abdurohman, Hadoop high availability with linux ha, in: 2018 6th International Conference on Information and Communication Technology (ICoICT), IEEE, 2018, pp. 66–69.

[23] T. MOSES, A proposed rack-aware model for high-availability of hadoop distributed file system (hdfs) architecture, University of Pitesti Scientific Bulletin Series: Electronics and Computer Science. 20 (1) (2020) 25–34.

[24] F. M. Awaysheh, M. Alazab, M. Gupta, T. F. Pena, J. C. Cabaleiro, Next-generation big data federation access control: A reference model, Future Generation Computer Systems. 108 (7) (2020) 726–741.

[25] Z.-q. Wu, J. Wei, F. Zhang, W. Guo, G.-w. Xie, Mdlb: a metadata dynamic load balancing mechanism based on reinforcement learning, Frontiers of Information Technology & Electronic Engineering. 21 (7) (2020) 1034–1046.

[26] S. Aljawarneh, M. B. Yassein, W. A. Talafha, A multithreaded programming approach for multimedia big data: encryption system, Multimedia Tools & Applications. 77 (6) (2017) 1–20.

[27] K. Gai, M. Qiu, H. Zhao, Privacy-preserving data encryption strategy for big data in mobile cloud computing, IEEE Transactions on Big Data. 7 (4) (2017) 1–12.

[28] L. Guo, H. Xie, Y. Li, Data encryption based blockchain and privacy preserving mechanisms towards big data, Journal of Visual Communication and Image Representation. 70 (7) (2019) 1–11.

[29] K. Wang, J. Yu, X. Liu, S. Guo, A pre-authentication approach to proxy re-encryption in big data context, IEEE Transactions on Big Data. 7 (4) (2017) 657–667.

[30] Y. Mo, A data security storage method for iot under hadoop cloud computing platform, International Journal of Wireless Information Networks. 26 (3) (2019) 152–157.

[31] S. Rallapalli, R. Gondkar, U. P. K. Ketavarapu, Impact of processing and analyzing healthcare big data on cloud computing environment by implementing hadoop cluster, Procedia Computer Science. 85 (8) (2016) 16–22.

[32] W. Kareem, R. Z. Yousif, S. M. J. Abdalwahid, An approach for enhancing data confidentiality in hadoop, Indonesian Journal of Electrical Engineering and Computer Science. 20 (3) (2020) 1547–1555.

[33] G. Kapil, A. Agrawal, A. Attaallah, A. Algarni, R. Kumar, R. A. Khan, Attribute based honey encryption algorithm for securing big data: Hadoop distributed file system perspective, PeerJ Computer Science. 6 (99) (2020) 1–11.

[34] P. Jain, M. Gyanchandani, N. Khare, Enhanced secured map reduce layer for big data privacy and security, Journal of Big Data. 6 (1) (2019) 1–17.

[35] Y. Song, Y.-S. Shin, M. Jang, J.-W. Chang, Design and implementation of hdfs data encryption scheme using aria algorithm on hadoop, in: 2017 IEEE International Conference on Big Data and Smart Computing (BigComp), IEEE, 2017, pp. 84–90.

[36] W. Heping, Research on hyperchaotic encryption algorithm based on mapreduce, in: 2017 International Conference on Computer Systems, Electronics and Control (ICCSEC), IEEE, 2017, pp. 1358–1361.

[37] M. Alloghani, M. M. Alani, D. Al-Jumeily, T. Baker, A. J. Aljaaf, A systematic review on the status and progress of homomorphic encryption technologies, Journal of Information Security and Applications. 48 (6) (2019) 1–10.

[38] M. M. Potey, C. A. Dhote, D. H. Sharma, Homomorphic encryption for security of cloud data, Procedia Computer Science. 79 (4) (2016) 175–181.

[39] M. Farooqui, H. Gull, M. Ilyas, S. Z. Iqbal, M. A. A. Khan, G. Krishna, M. S. Ahmed, Improving mental healthcare using a human centered internet of things model and embedding homomorphic encryption scheme for cloud security, Journal of Computational and Theoretical Nanoscience. 16 (5-6) (2019) 1806–1812.

[40] A. Alabdulatif, I. Khalil, X. Yi, Towards secure big data analytic for cloud-enabled applications with fully homomorphic encryption, Journal of Parallel and Distributed Computing. 137 (3) (2019) 192–204.

[41] T. Soo, A. Samsudin, Securing big data processing with homomorphic encryption, Systems Engineering. 82 (10) (2020) 11980–11991.

[42] A. R. Omondi, Elliptic-curve cryptosystems, in: Cryptography Arithmetic, Springer, 2020, pp. 243–252.

[43] D. Mahto, D. K. Yadav, Rsa and ecc: A comparative analysis, International Journal of Applied Engineering Research. 12 (19) (2017) 9053–9061.

[44] L. B. Goel, R. Majumdar, Handling mutual exclusion in a distributed application through zookeeper, in: Computer Engineering & Applications, IEEE, 2015, pp. 457–460.

[45] D. He, H. Wang, M. K. Khan, L. Wang, Lightweight anonymous key distribution scheme for smart grid using elliptic curve cryptography, Iet Communications. 10 (14) (2016) 1795–1802.

[46] M. I. Falcao, F. Miranda, R. Severino, M. J. Soares, Weierstrass method for quaternionic polynomial root-finding, Mathematical Methods in the Applied Sciences. 41 (1) (2018) 423–437.

Conflict of interest

We declare that we do not have any commercial or associative interest that represents a conflict of interest in connection with the work submitted.