ISSN 1330-3651 (Print), ISSN 1848-6339 (Online)

https://doi.org/10.17559/TV-20230618000745
Original scientific paper

Enhanced Secure Storage of Big Data at Rest with Improved ECC and Paillier Homomorphic Encryption Algorithms

Rong HU*, Ping HUANG

Abstract: With the rapid growth of Big Data, securing its storage has become crucial. This study proposes to enhance the secure storage of big data at rest in Hadoop by improving encryption algorithms. The Elliptic Curve Cryptography Algorithm (ECC) is upgraded by a parallel two-threaded approach for unstructured data. For structured data, enhance Paillier Homomorphic Encryption to support operations on ciphertexts. Experiments on datasets up to 4 G show that the modified ECC method reduces encryption time to 60 - 80 seconds, compared to 100 - 160 seconds for standard ECC, AES, and DES. It can also use shorter key lengths than RSA with comparable levels of security. Enhanced Paillier encryption uses large prime numbers to ensure the validity of the ciphertext. By combining these improved encryption techniques within a secure Hadoop framework, this research demonstrates an effective way to address vulnerabilities in Big Data storage.

Keywords: hadoop; Paillier Homomorphic Encryption; static Big Data; secure storage;

1 INTRODUCTION

The advent of big data has brought forth significant challenges in terms of storing and managing extremely large volumes of data, characterized by their variety and timeliness [1-2]. The current focus of research in this field revolves around valuable knowledge extraction and value mining from these massive datasets. Data storage emerges as a critical issue in big data analysis, as effective storage is a prerequisite for subsequent analysis and mining activities. The challenges associated with big data storage stem from its inherent nature as an untrustworthy third party, vulnerable to problems such as server failures that can potentially compromise data integrity [3-4]. Moreover, unauthorized access to the platform leads to frequent incidents of data theft and tampering. To tackle these concerns, Hadoop, an open-source big data framework, has emerged as the most popular platform employed by internet businesses. Built on a distributed computing framework and a distributed file system (HDFS), Hadoop provides a transparent infrastructure that addresses these challenges [5-6]. Its applications span various domains, including healthcare, banking, and other data-intensive services. Selective data encryption has gained prominence as a crucial technique to protect sensitive data and reduce processing costs. It enables the utilization of multiple encryption algorithms, accommodating various forms of data encryption. However, applying encryption algorithms such as AES, ECC, and RSA directly in Hadoop introduces a significant amount of overhead in the process of encrypting and decrypting large-scale data. Existing data encryption and decryption techniques within the Hadoop design overlook the challenges posed by standard encryption methods when directly applied using the MapReduce parallel computing framework [7]. As a result, data leakage becomes a potential concern following decryption. To address the need for improved secure storage of static big data, this study proposes the encryption of structured data using the Paillier encryption algorithm and the upgrading of Elliptic Curve Cryptography (ECC) within the Hadoop architecture. By leveraging Paillier encryption for structured data and enhancing ECC for unstructured data, the study aims to mitigate the limitations of traditional encryption methods and ensure secure storage.

To sum up, the rapid growth of big data necessitates effective storage mechanisms to accommodate the challenges associated with its variety and timeliness. This study will shed light on the significance of data storage in big data analysis, highlighting the vulnerabilities posed by untrustworthy third parties and unauthorized access. Additionally, the popularity of Hadoop and the importance of selective data encryption have been emphasized. The study's focus on improving secure storage through the encryption of structured and unstructured data using Paillier and ECC within the Hadoop framework demonstrates a promising approach to address these challenges. This paper is organized as follows: in section 2 we presented a comprehensive review of the related works in big data research, especially those related to data storage. In section 3, we proposed an Improved ECC Parallel Encryption Algorithm as our research method. In section 4, we used experiment to validate the algorithm we proposed. Then we discussed the practical and theoretical implications of the research results in section 5. Finally, we came up with a conclusion of our study.

2 RELATED WORKS

The Internet is an important part of human working life today and the large amount of data stored on the web has made experts aware of the importance of secure storage of static big data. A number of experts have conducted research related to secure data storage. A blockchain-based storage structure was suggested by Liang W. and other researchers as a solution to the issue of simple data manipulation in the network environment. During the procedure, the faulty node data was rectified using blockchain distributed encoding and the neighbouring regeneration code. The outcomes demonstrate that the suggested approach may successfully repair node data [8]. For the issue of audit deduplication, Dong Q. et al. provide a pseudo-random function-based encryption approach. The procedure gets rid of the aggregated evidence structure and uses surrogate signatures to give data blocks audit labels. The findings demonstrate the effectiveness and applicability of the suggested strategy [9]. In an effort to address the data sharing security concerns raised by drones, Feng C. presents a blockchain-based data sharing approach. The procedure integrates ABEM to increase overall computing speed, uses ABE to improve command security, public key encryption for accounts, and both. The results of the experiments demonstrate that the proposed technique offers faster encryption and improved data security [10]. A sine-cosine optimization technique for data security in IoMT was proposed by Joel D. and colleagues. The procedure employs the SCO method for node clustering that is energy-efficient, and intuitionistic fuzzy is used to choose the most efficient paths. The findings demonstrate that the suggested strategy offers superior data security and customer service quality [11]. Data security in 5G network environments has been proposed by researchers like Abd El-Latif A. A. using the S-box encryption approach. The encryption policy is set using the quantum walk's unpredictability. The experimental findings demonstrate the effectiveness of the suggested strategy to produce safe file passwords [12].

To address the issue of record protection in electronic medical records, Anand A. and Singh A. K. have suggested an ECC-based record-keeping solution. The procedure creates an electronic medical record watermark and inserts it into the Paillier cryptosystem. The proposed approach has good resilience and imperceptibility, according to experimental data [13]. For the issue of data decryption, Kumar P. et al. provide an ECC-based decryption technique. The method employs DNA coding for a multi-stage performance improvement before the computational efficiency of the procedure is assessed. Experimental findings demonstrate that the suggested technique decrypts data quickly and efficiently [14]. Aparna P. and Kishore P. propose an ECC-based recording method for the problem of image watermarking in e-healthcare. The process segments the image, compresses the bit stream with arithmetic coding and embeds the cover. Experimental results show that the proposed method has good peak signal-to-noise ratio and can generate image watermarks effectively [15]. Dwivedi R. K. et al. proposed an ECC-based key length control method for the access control problem of medical monitoring sensors. The process is implemented in the in.NET framework and the key length is shortened using ECC. The experimental results show that the proposed method has better length control speed and can effectively shorten the key length [16]. For the issue of video compression and encryption, Rajagopal S. and Shenbagavalli A. suggest an ECC-based encryption technique. The motion vector is encrypted during the procedure, and IABC is then optimized using the generated private key. The suggested approach has a good video compression rate and quick video encryption, according to experimental data [17].

In summary, the shortcomings of these studies are that they only address specific problems in specific domains, lack comprehensiveness and generality, and may be validated only in small-scale experimental environments without large-scale testing in real environments. In contrast, the novelty of the improved ECC encryption methods lies in the introduction of blockchain technology to solve the data manipulation problem, providing distributed and decentralised features that increase data security and reliability. Overall, the proposed method is based on ECC encryption algorithm with high security and computational efficiency.

3 RESEARCH METHOD

3.1 Static Big Data Secure Storage Framework Design

Hadoop uses a master-slave architecture to carry out distributed data storage. The NameNode node acts as a master node in this process, with the primary responsibility of storing metadata. The DataNode, in contrast, performs its own job as a slave node by storing data blocks [18, 19]. As a result, in order to assure the secure storage of static big data in Hadoop, the security of data blocks must come first on the list of improvements to be made. Based on this tenet and concept, the study develops a secure storage architecture for huge static data. The Hadoop Distributed File System Control module, the Data Encryption module, and the Metadata Protection module make up the framework. These modules' primary functions are to connect the client to the NameNode servers and to initialize any encryption algorithms that may be present on those servers. The distribution pass operation, or the transmission of the NameNode service to the four servers, is initially carried out by the metadata protection module. The four servers are split into two groups and operate the cluster as a whole while also providing external services. The NameNode metadata in the same group is essentially identical during this operation. It should be noted that at the same moment, one and only one is Active, when the role is to provide services to the outside world. The other one assumes the function of a backup node, i.e. a hot backup of the entire process of metadata, in case the node goes down and the backup node is able to directly replace the original node for cluster management. The final module in the secure storage framework involved is the data encryption module, which performs homomorphic encryption of textual data to be computed and light encryption operations on ordinary data through an improved encryption algorithm. At the same time, the module implements data encryption and storage processing using a multi-threaded approach to ensure secure storage while providing efficient computing capabilities. The designed secure storage framework is shown in Fig. 1.

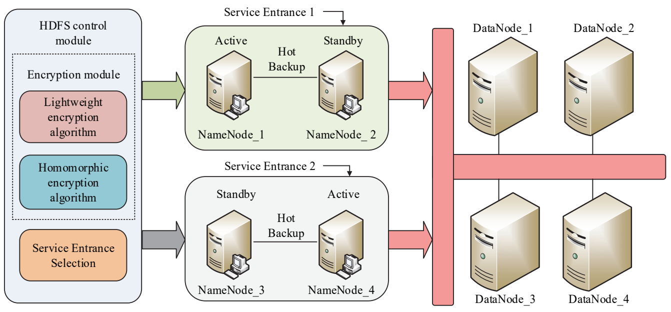


Figure 1 The overall framework of the static big data secure storage solution designed by the research institute

A prerequisite for metadata big data security is to ensure the reliability of the metadata. Since metadata is mainly stored in NameNode servers, metadata can be greatly affected and appear uncontrollable in the event of NameNode downtime [20]. The study combines the

Hadoop distributed file system high availability model and federation mechanism for the design of a multi-NameNode Hadoop cluster, and by designing a standby NameNode, a full process of hot backup of metadata is performed, thus reducing the occurrence of metadata loss caused by a single point of failure. When trying to implement the highly available Hadoop distributed file system concept, ZooKeeper's active NameNode selection and troubleshooting features are mostly used. In the event of a server loss, failure detection is a feature that enables ZooKeeper to promptly terminate a connection and start a failover service. If the active NameNode is in a crash state, a particular exclusion lock prevents another node from taking over as the active NameNode. The corresponding ZKFC process, which provides health monitoring, ZooKeeper session management, and node selection, is maintained by all servers hosting a NameNode. ZKFC periodically sends health check commands to the NameNode during health monitoring. Feedback from this node indicates that it is functioning properly to the ZKFC. The NameNode on this node is found to be flawed or in a subpar operational state in the absence of input, and this node is identified. ZKFC keeps a session active in ZooKeeper when the neighbourhood NameNode is deemed to be in good health. ZKFC employs distributed locking to stop the other nodes from submitting requests to enter the Active state when the NameNode's state is Active. A lock release procedure is carried out to make sure that the remaining nodes may get the distributed lock and continue processing when the session is terminated, which means that the current Active node has failed. ZKFC will decide in the node selection section if the current remaining node has acquired the status of the appropriate ZNode lock, which is now split into two scenarios. This node will be granted the permission of the related ZNode lock if the required object is not obtained and the NameNode is healthy. And this node is enrolled after the locking is successful. Fig. 2 depicts one of them, the operation flow for a distributed lock by ZooKeeper.

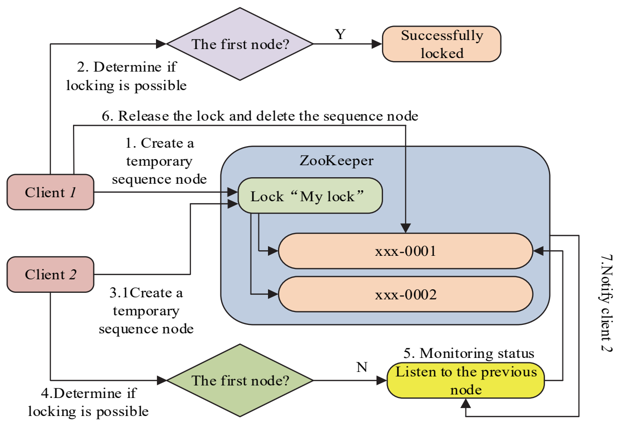

The first node?
Y
Successfully locked
2. Determine if locking is possible
6. Release the lock and delete the sequence node
1. Create a temporary sequence node
ZooKeeper
Client 1
Lock "My lock"
xxx-0001
xxx-0002
Client 2
3.1Create a temporary sequence node
7.Notify client 2
5. Monitoring status
N
The first node?
Listen to the previous node
4.Determine if locking is possible

Figure 2 ZooKeeper Distributed lock execution process

Through ZooKeeper, services are shifted to the standby node in the event of a NameNode failure. The primary function of ZooKeeper on the NameNode node of the Hadoop cluster, where it is installed, is to continuously monitor the node state. In order to maintain uninterrupted cluster service when a NameNode fails, ZooKeeper quickly launches the hot backup node. The service must be started in order to begin keeping track of the server's condition. The HealthMonitor thread broadcasts packets to the NameNode at a set time and checks the NameNode status in the second step of monitoring and detection. The third phase is the administration of ZooKeeper sessions, which involves feeding back the new state of the NameNode whenever its status changes. Implementing node selection based on ZooKeeper in the fourth stage entails managing the node's status there in real time after receiving feedback. Finally, the service is isolated, and when the backup node's state is Active, the service is isolated as well. As demonstrated in Fig. 3, in the case that an Active node fails, the backup node will start up right away and eventually offer the necessary service to the cluster, concluding the automated failover.

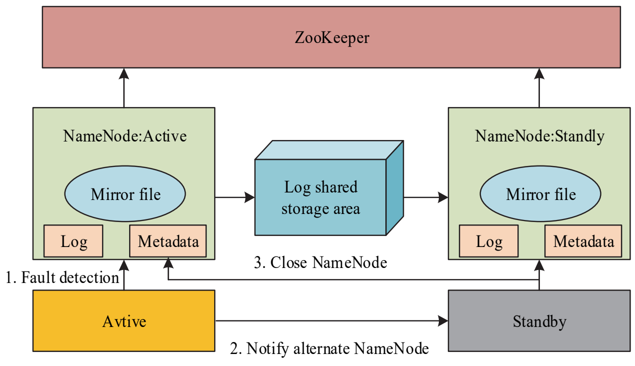


Figure 3 Fault service switching process

The failover between Active and Standby nodes is possible with ZooKeeper. This feature allows the Standby node to switch to the Active state in order to maintain the cluster's normal operation in the event that the NameNode fails. The study has set up a log sharing management module to record the information of metadata changes to assure the reliability of the metadata in order to synchronize the metadata of the two NameNodes. As demonstrated in Fig. 3, as soon as an issue is discovered by the Active fault controller, the Standby fault controller is notified right away, and the remaining NameNode processes force a shutdown procedure, putting the node into the Standby state. The node corresponding to the Standby fault controller goes to the Active state, balancing the dual objectives of metadata integrity and failed service recovery, with every process associated with the failed NameNode in a stalled state.

3.2 Big Data Storage Based on Improved ECC Parallel Encryption Algorithm

Elliptic Curve Cryptography (ECC), an asymmetric cryptography algorithm built on the elliptic curves mathematical theory, is simply discrete logarithm encryption [21]. In elliptic curve cryptography, the supposedly encrypted data is really created by performing a geometric operation on two points on the curve to create the ciphertext. The point-addition and point-doubling operations of elliptic curves are used to encrypt and decrypt data, respectively. Fig. 4 depicts the encryption and decryption of elliptic curves.

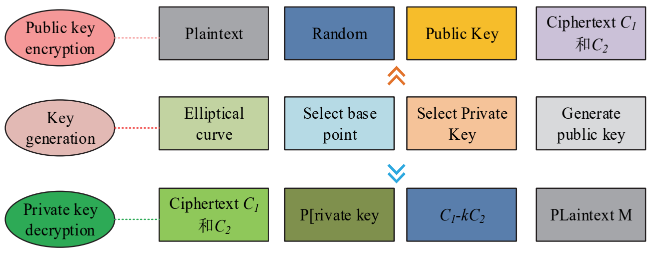


Figure 4 Elliptic curve encryption and decryption process

The definition of an elliptic curve E is shown in Eq. (1)

$$E: y_2 + a_1 xy + a_3 y = x^3 + a_2 x^2 + a_4 x + a_6 \tag{1}$$

In Eq. (1), denotes the domain of rational numbers as defined. The simplified representation of an elliptic curve is shown in Eq. (2).

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

The asymmetric encryption characteristic of the ECC encryption technique is employed in distributed storage plus number because it allows for improved security with shorter keys. Nevertheless, it has drawbacks, such as poor encryption performance when handling bigger data quantities. To accomplish its encryption efficiency, the study therefore enhances the ECC algorithm in terms of both storage technique and encryption mode. To achieve encryption and storage under mutual disruption, two-threaded encryption is employed. Data slicing, where the sliced data blocks are independently encrypted and saved, is first used to increase the efficiency of encryption and decryption of a single file. In one procedure, encryption is examined through byte streams because not all encrypted files are strings. When processing a lot of tiny files, Hadoop's speed suffers dramatically [22]. Before encryption, the research will split the file into two sections, each of which shows a one-to-one relationship with the corresponding two service portals. This will help to better integrate the two-channel storage model with the encryption technique. Prior to data slicing, the study also performed a document size analysis, and only files that fulfilled specific thresholds were permitted to carry out this stage. The study only began the data cut process for files larger than 256 MB in order to achieve the objective of lowering the number of little files, making sure that both finished cuts exceeded 128 MB. Fig. 5 depicts the two-threaded encryption model.

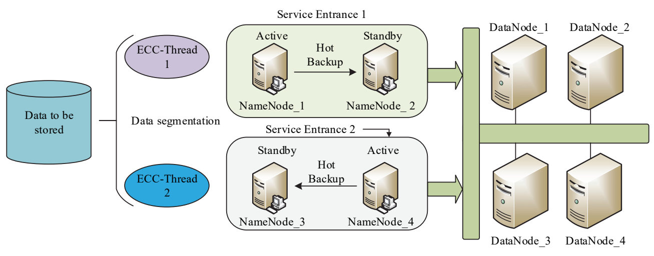


Figure 5 Research on an Improved ECC parallel encryption model

The study uses a homomorphic encryption scheme to ensure that data is not compromised during the processing of enormous volumes of data. The data to be computed is normally transmitted to the Hadoop distributed file system via text files. Homomorphic encryption enables particular operations to be carried out on the ciphertext as well as the direct computation of encrypted data. In order to achieve a result equivalent to one under plaintext, the resulting ciphertext is simultaneously decrypted. As a result, the privacy of the data is better secured and there is a significant reduction in the expensive consumption of frequent encryption and decryption. Fig. 6 depicts the homomorphic encryption algorithm's operational procedure.

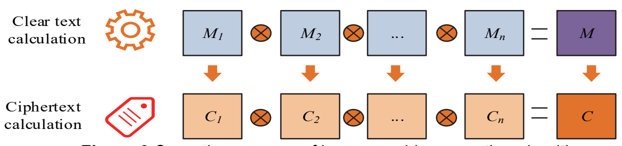


Figure 6 Operation process of homomorphic encryption algorithm

Not only is the Paillier homomorphic encryption algorithm capable of public key encryption, but it also performs better in a variety of cloud computing applications and is essential for resolving issues with data privacy. To better fulfill the storage needs, the study adds the Paillier homomorphic encryption method to the structured data storage phase and enhances the read and write data processes. Eq. (3) demonstrates how the Paillier homomorphic encryption algorithm first creates two huge prime integers, and at random by selecting the least common multiple of each minus 1.

$$\lambda = \mathrm{lcm}(q-1)(p-1) \tag{3}$$

In Eq. (3), represents the least common multiple, while represents the least common multiple sought, and represent two randomly generated large primes. A random integer is then obtained, which must conform to an order that can divide, as shown in Eq. (4).

$$gcd\,(g^{\lambda} \mathrm{mod} n^2, n) = 1 \tag{4}$$

In Eq. (4), is the maximum convention number. In the encryption stage, for the plaintext, the conditions and are satisfied, while a random number BB is determined for the secondary encryption, and the conditions and are satisfied. The encrypted ciphertext is obtained as shown in Eq. (5).

$$c = E(m, r) = g^m \cdot r^n \mathrm{mod} n^2 \tag{5}$$

In Eq. (5), is the encryption process for and denotes the random number obtained from the auxiliary encryption process. In the decryption stage, the plain text obtained by decrypting the ciphertext is shown in Eq. (6).

$$m = D(c, sk) = \frac{L(c^{\lambda} \mathrm{mod} n^2)}{L(g^{\lambda} \mathrm{mod} n^2)} \mathrm{mod} n \tag{6}$$

The study reads in the plain text and writes out the cipher text at the same time through a buffered character stream since the data to be encrypted are strings and text, which helps to enhance the speed of reading and writing files during encryption and decryption. I/O (Input/Output) reduction is mostly used by buffered character streams to increase data processing effectiveness. A buffered stream may read and write 8 k bytes in comparison to a standard physical stream's 1 byte, significantly lowering the number of I/O operations required to read and write the same amount of data. Additionally, text data often has one row for each piece of data in projects where data is generated using the MapReduce distributed computing framework and follows a particular alignment constraint. To ensure that MapReduce can efficiently recognize the alignment criteria once the encryption is stored, the Paillier method needs to be further enhanced. Since the cipher text is printed out in the same arrangement format as the plain text, the study employs the division of strings to encrypt all the strings corresponding to the fields one by one throughout the encryption process. The data is split during the read phase of the plaintext data by reading it line by line in conjunction with the plaintext separator, after which the Paillier homomorphic encryption method is initialized. The divided data is then encrypted and output in a regular format in accordance with how the plain texts are organized.

Regarding parameter configuration, I usually need to choose the right parameters according to specific application scenarios and requirements. For example, for machine learning tasks, hyperparameters such as learning rate, batch size, and number of iterations need to be adjusted to optimise the performance of the model. In practice, techniques such as grid search are used to find the optimal parameter configuration. During implementation, various open source software and tools such as TensorFlow, PyTorch, Hadoop, etc. are used. These software and tools help to implement the algorithms more efficiently and make the best use of the available computational resources. For example, TensorFlow provides a series of APIs and tools to easily build and train deep learning models. For the setup of a Hadoop cluster, we need to consider various factors, such as hardware configuration, network topology, storage system, and so on. In practice, Hadoop's command line tools are used to manage and configure the cluster, such as hadoop-config.sh and yarn-config.sh. Regarding the steps for key encryption, a simple pseudo-code is provided here:

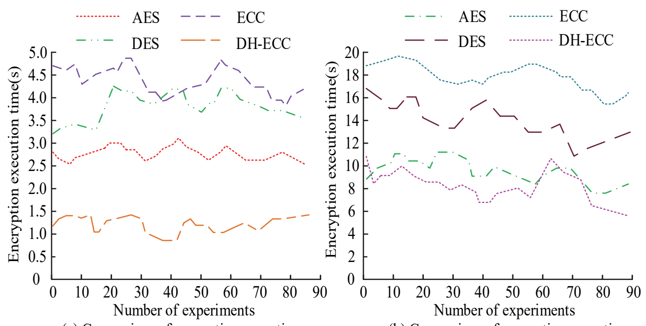
```
[# Generate public and private keys
generate_keypair().
  # Generate a random private key
private_key = generate_random_private_key()
    # Generate the corresponding public key based on the private key
public_key = generate_public_key(private_key)
    # Return the public and private keys
  return public_key, private_key
# Encrypt data with the public key
encrypt_data(data, public_key).
  # Encrypt the data using the public key
encrypted_data = encrypt_with_public_key(data, public_key)
    # Return the encrypted data
  return encrypted_data
# Transfer the encrypted data
# Decrypt data with private key
decrypt_data(encrypted_data, private_key).
  # Decrypt the encrypted data using the private key
decrypted_data = decrypt_with_private_key(encrypted_data, private_key)
    # Return the decrypted data
  return decrypted_data

# Main program
public_key, private_key = generate_keypair()
data = "Data to be encrypted"
encrypted_data = encrypt_data(data, public_key)
transmit_data(encrypted_data)
decrypted_data = decrypt_data(encrypted_data, private_key)]
```

This process uses the Paillier encryption algorithm, which is an RSA-based encryption scheme that enables additive encryption and decryption. By this method, the privacy and security of the data can be ensured.

4 RESULTS AND DISCUSSION

The study uses encryption speed and Hadoop distributed file system storage speed as evaluation metrics to contrast the suggested two-threaded ECC encryption with the pre-modified ECC, AES (Advanced Encryption Standard), and DES (Data Encryption Standard) under conventional Hadoop. Fig. 7 compares the encryption execution times of the four algorithms for data sizes of 55 M and 500 M. As can be seen in Fig. 7a, the encryption execution times obtained by all four methods fluctuate to varying degrees over the 90 experiments. The AES algorithm fluctuates between 2.5 s and 3.0 s, the ECC algorithm fluctuates relatively more, with a maximum of nearly 5 s and a minimum of 4 s. The DES algorithm has a minimum of 3.0 s and a maximum of less than 4.5 s. The DH-ECC algorithm only fluctuates between 1.0 s and 1.5 s, the shortest time among the four methods. As can be seen in Fig. 7b, both AES and DES methods have encryption times above 10 s when processing 500 M data volume, and there are relatively large fluctuations. The difference between the two methods, ECC and DH-ECC, is smaller, and the proposed method still has a small advantage.



(a) Comparison of encryption execution time under a data size of 55M
(b) Comparison of encryption execution time under a data size of 500M

Figure 7 Comparison of encryption execution time between four methods at 55 M and 500 M data scales

The main highlights of this research include the use of encryption speed and Hadoop distributed file system storage speed as evaluation metrics, which integrates algorithm execution efficiency and storage performance. Comparing the proposed two-threaded ECC encryption with premodified ECC, AES and DES under traditional

Hadoop helps to understand the performance of different algorithms in big data environments.

Fig. 8 displays the encryption execution times for the four algorithms using 1G and 2G data sizes. Fig. 8a shows that the time needed for AES to encrypt and execute 1 G of data varies between 12 and 16 seconds for each of the four algorithms, making it the quickest of the four. Although the proposed DH-ECC technique takes a little longer than the AES algorithm, it differs from it less. The ECC algorithm performs the worst, taking up to 36 s, whereas both DES and other approaches are over 24 s. When processing data of the 2 G scale, the four approaches, as shown in Fig. 8b, exhibit a comparatively wide disparity. Between them, there is just a 5 second gap between DH-ECC and AES. In comparison, the time needed for the DES approach varied between 50 and 55 seconds, exceeding 40 seconds for both ECC and DES. The highlight of this research is the use of larger data sizes (1 G and 2 G) for the experiments, enabling a more comprehensive assessment of the performance of different algorithms when dealing with large-scale data. The performance of the four algorithms is analysed in detail in a large-scale data scenario and clear differences are demonstrated in terms of experimental results.

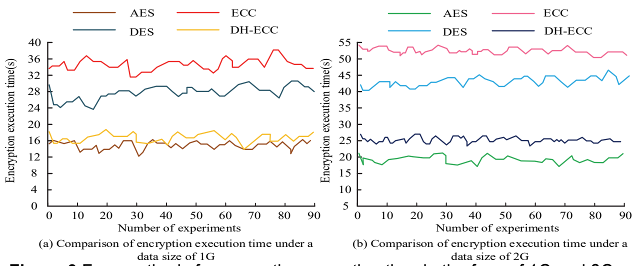


Figure 8 Four methods for encryption execution time in the face of 1G and 2G data scales

The suggested DH-ECC algorithm was shown to be less dissimilar from the AES technique after verification of the encryption execution time at data sizes of 55 M, 500 M, 1 G, and 2 G. The study also continued to raise the data size and validate the encryption execution time for the four approaches at 3.5 G and 4 G because the initial four data sizes were quite small. Fig. 9 displays the outcomes after grouping the 90 studies into an average of nine groups and eliminating the lowest and highest values. In Fig. 9, the group is represented by the horizontal axis, and the average encryption time is shown by the vertical axis. Fig. 9a shows that for all nine sets of trials at the 3.5 G data scale, the average encryption time of the proposed DH-ECC algorithm is between 40 s and 60 s, demonstrating good performance. The encryption times of all three techniques are longer, with DES reaching 110 s and AES and ECC varying between 80 and 100. Fig. 9b shows that the ECC algorithm's encryption time varies between 150 and 160 seconds for 4.0 gigabytes of data size. The AES and DES techniques differ by less, each taking between 120 and 140 seconds. On the other side, the suggested DH-ECC method is below 80 s, with a minimum of close to 60 s, which offers higher processing power. The highlights of this research include is the validation of encryption execution time for different data sizes (55 M, 500 M, 1 G, 2 G, 3.5 G, and 4 G), demonstrating the performance of different algorithms for different data sizes. The performance of the algorithms is verified under different data sizes, especially the DH-ECC algorithm shows better performance when dealing with large-scale data.

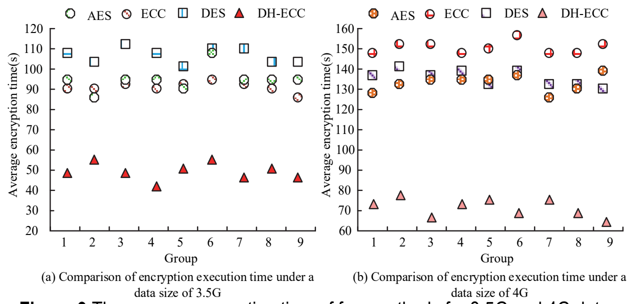


Figure 9 The average encryption time of four methods for 3.5G and 4G data scales

Combining the encryption time verification results of the four methods at different data sizes shows that when the data volume is below 2 G, the encryption speed of the AES algorithm differs less from that of the proposed DH-ECC algorithm, and there are cases where the former is superior. In comparison with the ECC algorithm, it can be seen that the DH-ECC algorithm improves the improved encryption efficiency when it is below 2 G. The reason for this is that the asymmetric encryption algorithm does not simply perform a byte round conversion operation, so the encryption speed is on the lower side compared to the symmetric encryption algorithm. At the same time, the Institute's DH-ECC algorithm has trouble taking advantage of dual-threading's benefits when the volume of data is modest. Because AES and DES algorithms primarily require several round-robin operations such row shifting and column obfuscation to encrypt data, their encryption performance substantially declines when data volumes exceed 2 G. Too many round-robin procedures when there is a lot of data will significantly slow down encryption. The suggested DH-ECC technique, in contrast, offers dual-threaded encryption and buffered byte stream modes that may both fully utilize system resources and speed up reading and writing of huge volumes of data.

As the research combines the designed two-channel storage model with it to boost the security and effectiveness of data storage, it is not simply an upgrade of the encryption method. Therefore, it is necessary to assess the Hadoop distributed file system's upload efficiency. A thorough experiment on the data encryption and upload process was then carried out, and the results are shown in Fig. 10. The study's proposed dual-channel storage scheme was compared to the Hadoop-based highly available storage model (RU_NN), the HDFS-based Federated Access Control Model (FACRM), and the original Hadoop distributed file system.

Fig. 10b displays the cumulative results of the encrypted uploads for the four techniques, whereas Fig. 10a displays the upload speed at various data sizes. Fig. 10a shows that for various data sizes, the research-improved HDFS data upload speed is superior than the other three models, reaching up to 60 MB/s. The research-improved HDFS system's cluster configuration includes two Active NameNodes, and two storage portals allow for simultaneous data uploads. Fig. 10b shows that even for 4 G scale data, the suggested DH-ECC encryption algorithm only needs a total upload time of about 170 s, which is faster than the other three approaches. This shows that when used in conjunction with the dual-channel storage option, the total upload efficiency of this encryption technique is greatly increased.

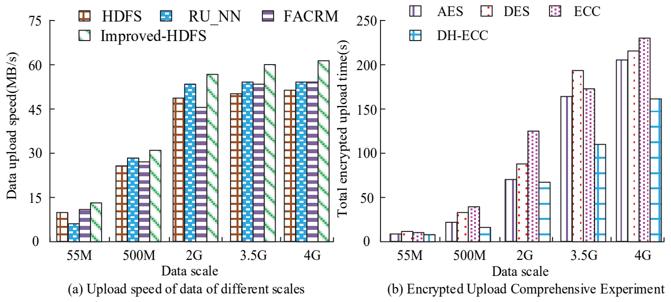


Figure 10 Comprehensive experimental results on upload speed and encrypted upload of data of different scales

The suggested DH-ECC encryption technique's security was then further examined by comparison with the RSA (Rivest-Shamir-Adleman) algorithm, an asymmetric encryption algorithm. The resulting comparison of key length and security level is displayed in Tab. 1.

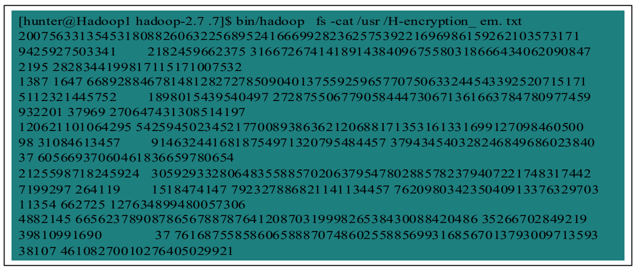
Table 1 Key length and security level of DH-ECC and RSA

Confidentiality level	Key length of DH-ECC / bit	Key length of RSA / bit	Confidentiality period
80	160	1024	2010
112	224	2048	2030
128	256	3072	2040
192	384	7680	2080
256	512	15360	2120

As can be observed from Tab. 1, the proposed DH-ECC algorithm requires smaller keys than the RSA algorithm for the same secrecy level, requiring keys of 160 bits, 224 bits, 256 bits, 384 bits, and 512 bits, respectively. It demonstrates how the suggested DH-ECC encryption technique can be more effectively used to store static huge data securely, make greater use of distributed resources, and protect private keys.

The ciphertext after homomorphic encryption can be examined using the terminal in the study because Paillier homomorphic encryption is compatible with MapReduce distributed computing. Fig. 11 displays the outcome of the enhanced Paillier homomorphic encryption after it has been finished. Fig. 11 shows how the modified Paillier encryption technique primarily uses big prime numbers to encrypt data and then uses the same large prime numbers to decrypt the data, essentially assuring the operation of the ciphertext. While the data after Paillier encryption can be better utilised by distributed computing frameworks like MapReduce, it is more applicable because the study reads data per line through buffered character streams, making it impossible to determine its alignment criteria.

Compared with the RSA algorithm, the DH-ECC algorithm has a small difference in encryption speed compared with the AES algorithm at less than 2 G data volume, and sometimes it is better. Whereas, the DH-ECC algorithm improves the encryption efficiency compared to the ECC algorithm on data volumes below 2 G. This is because asymmetric encryption algorithms (including DH-ECC and RSA) have lower encryption speeds compared to symmetric encryption algorithms such as AES and DES.


```
[hunter@Hadoop1 hadoop-2.7 .7]$ bin/hadoop   fs -cat /usr /H-encryption_ em. txt
200756331354531808826063225689524166699282362575392216969861592621035731 71
9425927503341       2182459662375 3166726741418914384096755803186664340620908 47
2195 2828344199817115171007532
1387 1647 668928846781481282727850904013755925965770750633244543392520715171
5112321445752       1898015439540497 2728755067790584447306713616637847809774 59
932201 37969 270647431308514197
1206211010642 95 5425945023452177008938636212068817135316133169912709846050 0
98 31084613457       9146324416818754971320795484457 37943454032824684968602384 0
37 605669370604618366597806 54
2125598718245924   3059293328064835588570206379547802885782379407221748317442
7199297 264119       1518474147 7923278868211411344 57 762098034235040913376329703
11354 662725 127634899480057306
4882145 665623789087865678878764120870319998265384300884204 86 35266702849219
39810991690          37 7616875585860658887074860255885699316856701379300971359 3
38107 461082700102764050299 21
```
Figure 11 Improved Paillier homomorphic encryption rendering

Asymmetric encryption algorithms need to perform complex operations, whereas symmetric encryption algorithms can use simple byte operations and therefore encrypt faster. In addition, it is difficult for DH-ECC algorithm to take advantage of dual-threading when the amount of data is small, whereas AES and DES algorithms mainly need to perform multiple cyclic operations such as row shifting and column obfuscation to encrypt the data, and their encryption performance decreases significantly when the amount of data is more than 2 G, because too many cyclic operations will reduce the encryption speed. The DH-ECC algorithm, on the other hand, can make full use of system resources and accelerate the reading and writing speed of massive data by providing two-threaded encryption and buffered byte stream mode. In summary, the DH-ECC algorithm has a certain advantage over the RSA algorithm in terms of encryption efficiency, and it is able to better exploit the performance advantage when dealing with large data.

4 CONCLUSION

One of the most important components of data security is the secure storage of static massive data. The main research problem of this study is the secure storage of massive data. In order to solve this problem, the study proposes a storage framework consisting of a Hadoop distributed file system control module, a data encryption module, and a metadata protection module, and proposes improved ECC and Paillier homomorphic encryption algorithms. Through experiments, it is found that the encryption time of the DH-ECC algorithm ranges from 1.0 to 1.5 seconds, while the encryption time of the AES algorithm ranges from 2.5 to 3.0 seconds. The difference in encryption time of the ECC algorithm is even greater, with a maximum value of close to 5 seconds and a minimum value of 4 seconds. In the nine sets of experiments with 3.5 G of data, the average encryption time for the DH-ECC method ranged from 40 to 60 seconds, while both the AES and ECC algorithms fluctuated between 80 and 100 seconds. All three have longer encryption times. The key lengths of the DH-ECC algorithm are 160, 224, 256, 384 and 512 bits at the secrecy level of 80256, which are smaller than the key lengths of the RSA method at the same secrecy level. Enhanced Paillier encryption for structured data uses large prime numbers for data encryption and large prime numbers after encryption is complete, which greatly extends the amount of massive data that can be stored securely. However, the study does not take into account the complexity of the Paillier algorithm for application in Hadoop, and thus further improvements need to be introduced with fully homomorphic encryption algorithms. The research results show that the DH-ECC algorithm has better performance and security in data encryption, and has potential application value for encrypted storage of big data. However, in practical applications, the complexity, scalability and difficulty of implementation of the algorithm need to be considered comprehensively to choose a suitable encryption method. Meanwhile, future research can further improve the complexity of Paillier's algorithm to enhance its application in Hadoop.

5 REFERENCES

[1] Gurav, Y. B. & Patil, B. M.(2023). De-centralized information flow control for cloud virtual machines with hybrid AES-ECC and improved meta-heuristic optimization based optimal key generation. *International journal of intelligent robotics and applications*, *7*(2), 406-425. https://doi.org/10.1007/s41315-022-00268-6

[2] Suzhen, W., Chunfeng, D., Weidong, Z., Jindong, Z., Hong J., Bo, M., & Lingfang, Z. (2023). EaD: ECC-Assisted Deduplication With High Performance and Low Memory Overhead for Ultra-Low Latency Flash Storage.*IEEE Transactions on Computers*, *72*(1), 208-221. https://doi.org/10.1109/TC.2022.3152665

[3] Yin, B., Yin, H., Wu, Y., & Jiang, Z. (2020). FDC: A secure federated deep learning mechanism for data collaborations in the Internet of Things. *IEEE Internet of Things Journal*, *7*(7), 6348-6359. https://doi.org/10.1109/JIOT.2020.2966778

[4] Smarandache, F. (2022). Plithogeny, plithogenic set, logic, probability and statistics: a short review. *Journal of Computational and Cognitive Engineering*, *1*(2), 47-50. https://doi.org/10.47852/bonviewJCCE2202191

[5] John, Y. M., Sanusi, A., Yusuf, I., & Modibbo, U. M. (2020). Reliability Analysis of Multi-Hardware–Software System with Failure Interaction. *Journal of Computational and Cognitive Engineering*, *2*(1), 38-46. https://doi.org/10.47852/bonviewJCCE2202216

[6] John, N. & Sam, S. (2021). Provably Secure Data Sharing Approach for Personal Health Records in Cloud Storage Using Session Password, Data Access Key, and Circular Interpolation. *International journal on Semantic Web and information systems*, *17*(4), 76-98.

[7] Li, P. & Lo, K. T. (2020). Survey on JPEG compatible joint image compression and encryption algorithms. *IET Signal Processing*, *14*(8), 475-488. https://doi.org/10.1049/iet-spr.2019.0276

[8] Liang, W., Fan, Y., Li, K. C., Zhang, D., & Gaudiot, J. L. (2020). Secure data storage and recovery in industrial blockchain network environments. *IEEE Transactions on Industrial Informatics*, *16*(10), 6543-6552. https://doi.org/10.1109/tii.2020.2966069

[9] Dang, Q., Ma, H., Liu, Z., & Xie. Y.(2020). Secure and Efficient Client-Side Data Deduplication with Public Auditing in Cloud Storage. *International Journal of Network Security*, *22*(3), 462-475.

[10] Feng, C., Yu, K., Bashir, A. K., Al-Otaibi, Y. D., Lu, Y., Chen, S., & Zhang, D. (2021). Efficient and secure data sharing for 5G flying drones: a blockchain-enabled approach. *IEEE Network*, *35*(1),130-137. https://doi.org/10.1109/MNET.011.2000223

[11] Joel, D. & Juliet, S. E. (2021). A Secure Data Storage Architecture for Internet of Medical Things (IoMT) Using an Adaptive Gaussian Mutation Based Sine Cosine Optimization Algorithm and Fuzzy-Based Secure Clustering. *Journal of Medical Imaging and Health Informatics*, *11*(12), 2883-2890. https://doi.org/10.1166/jmihi.2021.3838

[12] Abd El-Latif, A. A., Abd-El-Atty, B., Mazurczyk, W., Fung, C., & Venegas-Andraca, S. E. (2020). Secure data encryption based on quantum walks for 5G Internet of Things scenario. *IEEE Transactions on Network and Service Management*, *17*(1),118-131. https://doi.org/10.1109/TNSM.2020.2969863

[13] Anand, A. & Singh, A. K. (2020). Joint Watermarking-Encryption-ECC for Patient Record Security in Wavelet Domain. *IEEE Transactions on Multimedia*, *27*(3), 66-75. https://doi.org/10.1109/MMUL.2020.2985973

[14] Kumar, P. & Kumar, B. A. (2020). Enhancing multi-tenancy security in the cloud computing using hybrid ECC-based data encryption approach. *IET Communications*, *14*(18), 3212-3222. https://doi.org/10.1049/iet-com.2020.0255

[15] Aparna, P. & Kishore, P. (2020). An iris biometric-based dual encryption technique for medical image in e-healthcare application. *International Journal of Computational Vision and Robotics*, *10*(1), 1-20. https://doi.org/10.1504/IJCVR.2020.104353

[16] Dwivedi, R. K., Kumar, R., & Buyya, R. (2021). Secure Healthcare Monitoring Sensor Cloud With Attribute-Based Elliptical Curve Cryptography. *International Journal of Cloud Applications and Computing (IJCAC)*, *11*(3), 1-18. https://doi.org/10.4018/IJCAC.2021070101

[17] Rajagopal, S. & Shenbagavalli, A. (2020). OptimalECC-based signcryption algorithm for secured video compression process in H.264 encoder. *International Journal of Biomedical Engineering and Technology*, *32*(1), 36-65. https://doi.org/10.1504/IJBET.2020.104676

[18] Li, L. (2020). Secure encryption algorithms for wireless sensor networks based on node trust value. *International Journal of Internet Protocol Technology*, *3*(13), 117-123. https://doi.org/10.1504/IJIPT.2020.107967

[19] Ding, L., Wang, Z., Wang, X., Wang, X., & Wu, D. (2020). Security information transmission algorithms for IoT based on cloud computing. *Computer Communications*, *155*(4), 32-39. https://doi.org/10.1016/j.comcom.2020.03.010

[20] Abroshan, H. (2021). A hybrid encryption solution to improve cloud computing security using symmetric and asymmetric cryptography algorithms. *International Journal of Advanced Computer Science and Applications*, *12*(6), 31-37. https://doi.org/10.14569/IJACSA.2021.0120604

[21] Yin, S., Liu, J., & Teng, L. (2020). Improved Elliptic Curve Cryptography with Homomorphic Encryption for Medical Image Encryption. *International Journal of Network Security*, *22*(3),421-426.

[22] Viswanathan, S., Bhuvaneswaran, R. S., Ganapathy, S., & Kannan, A. (2022). Euler phi function and gamma function based elliptic curve encryption for secured group communication. *Wireless Personal Communications*, *125*(1), 421-451. https://doi.org/10.1007/S11277-022-09557-6

Contact information:

Rong HU, Associate Professor
(Corresponding author)
School of Intelligence Technology, Geely University of China, Chengdu Sichuan, 641423, P. R. China,
No. 123, SEC. 2, Chengjian Avenue, Eastern New District, Chengdu City
Sichuan Province
E-mail: 727749104@qq.com

Ping HUANG, Master lecturer
School of Mathematics and Computer Science, Panzhihua University,
Panzhihua, Sichuan, 617000, P. R. China
No. 10, North Section of Sanxian Avenue, East District, Panzhihua City, Sichuan Province
E-mail:1269228282@qq.com