National College of Ireland

Data Encryption in HDFS

MSc Research Project
Cloud Computing

Rushikesh Manoj Nikumbh
Student ID: x22136851

School of Computing
National College of Ireland

Supervisor: Mr. Vikas Sahni

National College of Ireland
Project Submission Sheet
School of Computing

National College of Ireland

Student Name:	Rushikesh Manoj Nikumbh
Student ID:	x22136851
Programme:	Cloud Computing
Year:	2023
Module:	MSc Research Project
Supervisor:	Mr. Vikas Sahni
Submission Due Date:	14/12/2023
Project Title:	Data Encryption in HDFS
Word Count:	4768
Page Count:	15

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	
Date:	14th December 2023

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Data Encryption in HDFS

Rushikesh Manoj Nikumbh
x22136851

Abstract

Hadoop's Hadoop Distributed File System (HDFS) faces security challenges such as internode communication vulnerabilities, replication storage mode vulnerabilities, and user access monitoring. Traditional encryption techniques can increase data size and slow down performance. Therefore, this research will focus on modifying specific variants of the Blowfish block-cipher encryption algorithm to improve efficiency and security in HDFS. In this research, a successful studies related to challenges in HDFS were conducted. Also the implementation of blowfish algorithm for storage and retrieval of file in HDFS was experimented. The method implemented showed improved performance with compared with other algorithms - DES and 3DES.

1 Introduction

Researchers had been investigating methods for safeguarding and protecting data within the Hadoop Distributed File System (HDFS). As Hadoop gained widespread adoption across various industries, the need to safeguard data stored in HDFS from cybersecurity threats and potential breaches had become increasingly critical. Privacy regulations had also emphasized the significance of effective encryption techniques to safeguard sensitive information, including personal data, intellectual property and financial records stored in HDFS. Researchers had explored hybrid encryption methods as a promising solution to maintain a balance between performance and security, addressing challenges posed by traditional encryption methods. By including the evolving threat landscape and experimenting with various encryption techniques, researchers had contributed to enhancing HDFS's resilience against cyber threats. This included reducing reliance on external cloud service providers and ensuring data integrity and protection.

Researchers had identified different security vulnerabilities in HDFS, for example communication issues between nodes, weaknesses in data storage mechanisms, and challenges in monitoring access privileges. They had focused on improving existing data encryption methods, particularly the Blowfish method, to enhance encryption speed and efficiency. The overall objective was to achieve a balance between security and performance by optimizing encryption processes and streamlining data protection without compromising safety.

This research aims to study Hadoop and the security challenges of the Hadoop Distributed File System (HDFS), various methodologies that are been used to deal with the security challenges and implementing one of the method to check the efficiency. For this research purpose, we have considered the encryption and decryption time [1] as major

[1] `https://networksimulationtools.com/encryption-simulator/`

parameters and have implemented using Blowfish encryption algorithm which performs well for certain applications.

While the rest of the paper has been divided in following sections

- Related Work - In this section, the major discussion is related to the security challenges of Hadoop and different mechanism that has been used to tackle these issues.
- Methodology - This section discusses the different methodologies, different test case scenarios to evaluate the results and the experimental setup.
- Design Specifications - This section emphasis of the different underlying such has HDFS, Blowfish Algorithm and proposed Encryption and Decryption process in HDFS.
- Implementation - This section gives us an overview of implementation environment that is setup to conduct our experiments.

  At the end, evaluation parameters, results and discussion are included followed by conclusion and future works.

2 Related Work

Hadoop's distributed file system (HDFS) is a popular storage system for big data, but it faces several security challenges. Researchers have proposed various solutions to address these challenges, including data encryptionFunde and Swain (2022). Traditional encryption techniques can increase data size and slow down performance, so researchers have explored hybrid encryption techniques that combine traditional encryption with other methods, such as attribute-based encryption and honey encryption.

2.1 The security challenges of the HDFS

Since its initial development, the Hadoop Distributed File System (HDFS) has been vulnerable to security breaches that could be exploited by unauthorized users. The primary focus during HDFS's early development was on providing basic functionality without prioritizing comprehensive security measures. This oversight left the system susceptible to unauthorized access and malicious attacks. If unauthorized users gained access to HDFS, they could damage or steal data, disrupt system operations, or even gain access to other sensitive data within the organization.

HDFS security challenges have included weak authentication and authorization mechanisms, allowing unauthorized users to access the system or data. Insufficient monitoring and auditing of HDFS logs further compounded the vulnerability, making it more difficult to detect and respond to possible security breaches. Unauthorized access to HDFS has resulted in serious consequences that includes data breaches, system downtime, and reputational damage. Therefore, it is crucial to implement strong security measures and consistently monitor and audit HDFS activity to mitigate these risks. Sharma and Navdeti (2014)

The Key challenges are discussed below

- Authentication and Authorization

  HDFS provides basic authentication and authorization mechanisms, but they are not sufficient for protecting sensitive data in production environments. It is important to implement stronger authentication mechanisms, such as Kerberos or LDAP, to ensure that only authorized users can access the HDFS cluster. Additionally, fine-grained authorization policies should be implemented to restrict users' access to specific data sets and operations.Bhathal and Singh (2019)

- Data Encryption

  Data stored in HDFS is not encrypted by default, making it vulnerable to unauthorized access. To protect sensitive data, encrypt both at rest and in transit. Encryption at rest can be achieved using file-level or whole-disk encryption. Encryption in transit can be done using SSL/TLS to secure network communication between HDFS nodes.

  In file-level encryption, each file is encrypted independently. This method offers granular control over which files are encrypted and provides flexibility in managing encryption keys. Whereas, With whole-disk encryption, the entire disk containing HDFS data is encrypted. This approach provides a more comprehensive level of protection, as it ensures that all data on the disk is safeguarded from unauthorized access. SSL/TLS (Secure Sockets Layer/Transport Layer Security) is a cryptographic protocol that encrypts data in transit between HDFS nodes. This helps to prevent eavesdropping and data tampering during data transfers.Parmar et al. (2017)

- Access Control Lists (ACLs)

  ACLs provide a mechanism for controlling access to individual files and directories in HDFS. However, ACLs are not granular enough to protect sensitive data, as they only allow or deny access to entire files and directories. Fine-grained access control policies are needed to restrict users' access to specific data elements within files and directories.[2]

- Network Security

  HDFS communicates between nodes using TCP/IP, which is a well-known protocol but is not inherently secure. To protect HDFS from network-based attacks, it is important to implement firewalls and intrusion detection systems (IDS) to filter traffic and detect suspicious activity. Additionally, encryption should be used to protect data in transit between HDFS nodes.Sinha et al. (2019)

- Access Control to HDFS NameNode

  The HDFS NameNode is a critical component of the HDFS architecture, as it manages the file system metadata. Unauthorized access to the NameNode can enable attackers to corrupt or delete data, or even take control of the entire HDFS cluster. Access to the NameNode should be restricted to authorized users, and strong authentication and authorization mechanisms should be implemented.Rajeh (2022)

[2]https://securosis.com/blog/securing-hadoop-architectural-security-issues

- HDFS Daemon Vulnerabilities

  HDFS daemons, such as DataNodes and NameNodes, are susceptible to vulnerabilities. These vulnerabilities are prone to be exploited by attackers to gain unauthorized access to the HDFS cluster or to disrupt its operation. It is essential to consistently apply security updates and patches to HDFS daemons to reduce these risks.

- Data Theft and Modification

  HDFS is a distributed file system, making it more vulnerable to data theft and modification attacks. Attackers could exploit vulnerabilities in HDFS or its underlying infrastructure to steal data or modify it without leaving traces.

- Insider Threats

  Insider threats pose a significant challenge to HDFS security. Malicious insiders, such as authorized users with access to sensitive data, could intentionally or accidentally compromise the HDFS cluster or steal confidential information. conducting regular security awareness training, Implementing strong access control policies, auditing mechanisms and robust logging can help to mitigate insider threats.

2.2 Mechanisms to tackle the security issues in HDFS

Zhonghan et al. (2013) had proposed a hybrid encryption method for HDFS, employing symmetric encryption for data blocks and asymmetric encryption for key protection. This method involved integrating encryption and decryption modules into datanodes and utilizing AES for data security. Each data block was encrypted and decrypted with a key stored on the corresponding datanode. The client generated an RSA key pair, where the private key was retained locally and the public key was stored in the namenode's metadata.Naisuty et al. (2020) The private key was also encrypted with a key that is derived from the user's password which stored as a file.

The proposed method preserved the original protocol between datanodes and the namenode, while introducing authentication and key exchange protocols at the client and datanode levels, aiming to protect data from theft even if datanodes were compromised. However, the evaluation had focused solely on performance degradation compared to standard HDFS, lacking a comprehensive analysis of security effectiveness, scalability, and impacts on other HDFS aspects.

The hybrid method between RSA and Rabin cryptosystems to encrypt the files saved in HDFS, proposed a promising approach to enhancing information security in Hadoop and addressing the critical need for robust data protection in cloud computing environments. Yousif et al. (2020) This method involved encrypting the file's content before saving it on HDFS, that used the hybrid cipher asymmetric key algorithm that incorporated the most important cloud computer system service models and effectively addressed data administration and protection issues for key management and security in data transfer. However, a limitation was the amount of time taken for decryption method to identify the proper messages in plaintext format produced through Rabin method decryption.

This limitation potentially impacted the overall efficiency and performance of the encryption and decryption processes, particularly when dealing with large volumes of data. Therefore, while the proposed approach offered significant advancements in data

security, the potential impact of increased computational complexity and decryption time should have been carefully considered in future developments.

The individual implementations of RSA or ElGamal methods had shown limitations in terms of computational complexity and efficiency for the security of sensitive data in Hadoop, particularly in the Hadoop Distributed File System (HDFS). To enhance data confidentiality in Hadoop, an optimized hybrid encipherment algorithm had been developed and compared with classical public-key cryptosystems before applying it on Hadoop to secure big data. Shehzad et al. (2016)

The hybrid encryption system, which combines two popular asymmetric key cryptosystems, RSA and ElGamal, had provided a more powerful computational complexity and enhanced data confidentiality in HDFS, resulting in shorter decryption times. Kareem et al. (2020) However, the research had not provided a comprehensive evaluation of the method's performance in terms of security and efficiency, and further research had been needed.

The paper does not compare the proposed BlowFish encryption scheme with other existing encryption algorithms, limiting the understanding of its effectiveness and suitability in the context of Hadoop and HDFS.

An encryption scheme proposed by Kaushik and Srivastava (2021) applies the BlowFish algorithm to sensitive attributes in files stored in HDFS. This approach offers several advantages over encrypting the entire file using the AES algorithm, including reduced storage space requirements and improved performance. Experiments with varying file sizes and demonstrate that the proposed scheme outperforms AES in terms of both storage efficiency and speed. They also discuss the integration of the scheme with the MapReduce programming model, which facilitates distributed data processing in Hadoop. While comparison of the BlowFish scheme with other encryption algorithms would presented a promising approach for securing sensitive attributes in HDFS files.

While hybrid encryption algorithms offer high security, they also consider performance factors. By combining different algorithms, it's possible to achieve a balance between security and performance, ensuring that encryption processes doesn't significantly impact system performance.

Johri et al. (2018) have created Privacy Preserve Hadoop (PPH), an implementation that enhances the security of Hadoop by integrating data encryption and decryption processes into the MapReduce framework. PPH proves to be superior to DES in terms of encryption and decryption runtime in both single-node and multi-node Hadoop environments. It is an effective solution for organizations handling sensitive and voluminous data, enabling secure data management and accessibility within Hadoop without the expense of specialized hardware. They also propose potential enhancements for future implementations, such as incorporating a dedicated key management server and utilizing multiple cryptography algorithms in multi-node Hadoop setups with larger data files. They acknowledge the challenges posed by big data and the need for solutions like PPH to address issues like integrating, storing, managing, and processing vast amounts of data efficiently. The MapReduce programming model, a cornerstone of Hadoop, facilitates parallel processing of large datasets on commodity hardware, minimizing the complexities of managing execution details by leveraging the Hadoop runtime.

The research papers reviewed in this section helped us to identify and study the different security challenges posed by HDFS which helped to conclude that data encryption is the most effective way to address the security challenges. However, it been observed that conventional encryption techniques can increase data size and slow down perform-

ance. Hybrid encryption techniques can improve the efficiency and security of HDFS by combining traditional encryption with other methods, such as attribute-based encryption and honey encryption.

3 Methodology

Hybrid encryption algorithms combine multiple encryption algorithms providing a higher level of security and can provide a more comprehensive approach to data protection by addressing the limitations of individual encryption algorithms. Also, that allows flexibility in choosing the most suitable encryption algorithms for different aspects of data protection.

While the research revolves around discussion of the security challenges of the Hadoop Distributed File System (HDFS), various methodologies that are been used to deal with the security challenges and implementing one of the method to check the efficiency, we will check these details one by one in this section.

Encryption and decryption are both computationally intensive tasks. This means that it takes a lot of processing power to encrypt and decrypt data. However, the speed of encryption and decryption has improved significantly over time as computers have become more powerful and the encryption mechanism used

Sekar and Padmavathamma (2016) introduces an integrated approach to encrypt and decrypt data before sending it to the system and compares different encryption algorithms, including DES, AES, and RSA, for their encryption and decryption times and buffer sizes.

To enhance security, a modified Blowfish algorithm, proposed by Alaojan and Alwattar (2022) reducing the number of substitution boxes from four to three and incorporating a dynamic key. The performance of this modified algorithm was compared to the original and previously modified Blowfish algorithms based on evaluation criteria such as throughput, encoding time, and decoding time. The comparative performance of the original, pre-modified, and modified Blowfish algorithms was assessed for various file types and sizes, with the processing speed measured in terms of encoding, decoding, and throughput.

Algorithm's performance can been assessed by considering communication complexity and various computational parameters, such as energy usage, processing time, memory, and communication overhead. Selecting the most suitable performance parameters helps to enhance the efficiency of the method. The various considerable parameters for cryptographic encryption techniques includes cost, key length, susceptibility versus known attacks, encryption time, cipher randomness, correlation data, application flexibility versus various platforms, and computation time versus varied data format. Although, through the discussions and review in the previous sections, there are few of common parameter that has been assessed are taken into considerations in this research. They are as follows:

- **Encryption time** - the amount of time it takes to convert plain text into ciphertext. This is the process of scrambling the data so that data becomes unreadable to anyone who does not have the specific decryption key. Encryption time can vary depending on the type of encryption algorithm used, the length of the key, and the amount of data being encrypted.

- **Decryption time** - the amount of time it takes to convert ciphertext back into plain text. This is the process of unscrambling the data using the decryption key.

Decryption time is typically the same as encryption time, but it can be slightly faster or slower depending on the specific algorithm and the hardware being used.

In order to perform analysis based on the chosen parameters, we had followed the steps as below:

Step 1 : Connect with HDFS Client - HDFS Client initiates communication with the Master Node and sends a request to create a new file with desired attributes and specifies file name, permissions, and other relevant metadata.

Step 2 : Encryption Process - The file with the specified data is then encrypted using Blowfish algorithm and HDFS Client transfers encrypted file data from the client to the Data Node.

Step 3: Acknowledgement and output - Upon successful replication, the file creation process is finalized, and the file is ready for access by authorized users. HDFS client receives confirmation from the Data Node which can be seen in local host.

To extract the stored file, following steps were followed:

Step 1 : Connect with HDFS Client - HDFS Client initiates communication with the Master Node and sends a request to retrieve a new file with desired attributes and specifies file name, permissions, and other relevant metadata.

Step 2 : Decryption Process - The file with the specified data is then decrypted using Blowfish algorithm and HDFS Client transfers decrypted file data from the Data Node to client.

Step 3: Acknowledgement and output - Upon successful file retrieval process, the file is ready for access by authorized users.

4 Design Specification

The techniques and architecture that underlies the implementation and the associated requirements are as follows.

4.1 Hadoop Distributed File System

The Hadoop Distributed File System (HDFS) has been designed based on the master-slave principle, comprising a single NameNode and a group of DataNodes. The HDFS Architecture is as shown Figure 1. The NameNode is responsible for managing the HDFS directory tree and maintaining all the metadata related to the file system. Client directly communicates with the NameNode to execute standard file operations.

The NameNode is responsible for managing a comprehensive mapping of files stored at DataNodes and their corresponding file names. It had also been assigned the crucial task of proactively monitoring DataNodes for any potential failures. Upon detecting such instances, the NameNode would swiftly initiate the creation of additional block replicas to safeguard the data and maintain system resilience.

The NameNode can also act as a CheckpointNode and a Backup-Node. The CheckpointNode protects system metadata through periodic checkpoints, while the BackupNode maintains a synchronized file image with the NameNode state and handles potential failures. In enterprise versions of Hadoop, a Secondary NameNode is introduced as a backup

in case the original NameNode crashes. It uses the saved HDFS checkpoint to restart the crashed NameNode.

DataNodes are responsible for physically storing file blocks and fulfilling instructions assigned by the NameNode. Each file block is subdivided into smaller chunks, each marked with a unique identification timestamp. As a standard practice, each data block is triplicated, with two copies residing on distinct DataNodes within the same rack and the third copy hosted on a DataNode in a different rack. This replication approach promotes data redundancy and fault tolerance, ensuring the robustness of the distributed file system. In the upcoming section, we will discuss regarding the Blowfish Encryption Algorithm that will be used.

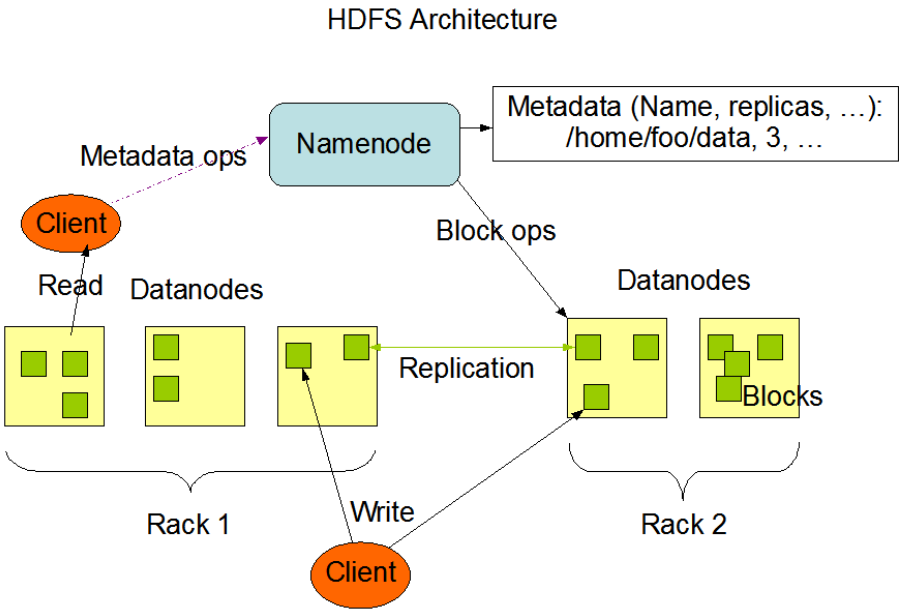

Figure 1: HDFS Architecture [3]

4.2 Blowfish Encryption

Blowfish encryption algorithm is a secret-key block cipher, that had been introduced for data encryption purposes. Its is based on the Feistel network structure, which repeatedly applies a simple encryption function 16 times, a well-established and widely used cryptographic design approach is shown in Figure 2. The block size of Blowfish had been set at 64 bits, and the key length could vary from 32 bits up to 448 bits. This flexibility in key length enabled stronger encryption and enhanced security. However, there had been potential attacks identified on mini versions of Blowfish with smaller block sizes, such as 32-bit or even 16-bit. Schneier (1994) The encryption of data using Blowfish had proven to be very efficient, making it suitable for applications demanding fast encryption and decryption and also supports secure user authentication for remote access.

4.2.1 Encryption and Decryption mechanism of Blowfish algorithm

Data encryption with Blowfish:

[3] https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html

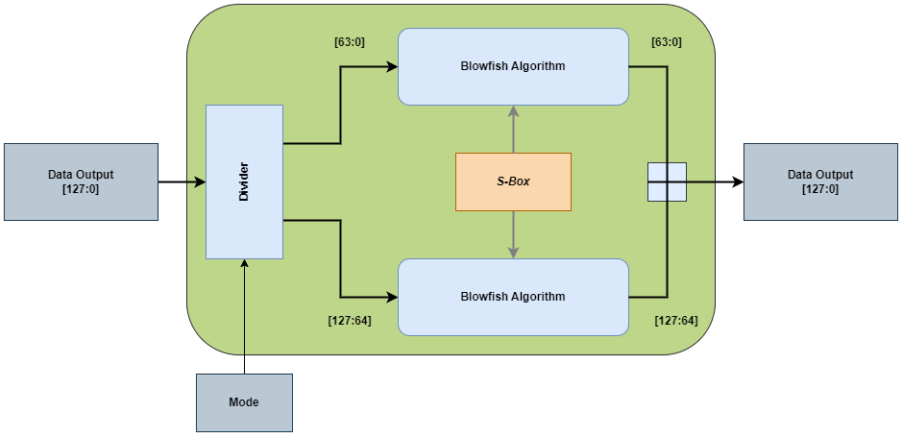

Figure 2: Feistel network structure in Blowfish

1. First, you need to create a key. This key can be any length, from 32 to 448 bits, which is like a very long number.
2. You give the key to Blowfish. Blowfish then uses this key to create a special set of instructions, called the P-array, S-boxes, and L-box.
3. You split the data you want to encrypt into two pieces, called L and R.
4. Blowfish performs a series of steps on L and R, using the P-array, S-boxes, and L-box. These steps are like mixing up the data in a special way.
5. After 16 rounds of these steps, Blowfish puts the L and R pieces back together, and the encrypted data is ready.

Data decryption with Blowfish:

1. You give the key to Blowfish again. Blowfish uses this key to recreate the P-array, S-boxes, and L-box.
2. You split the encrypted data into two pieces, L and R.
3. Blowfish performs the same series of steps on L and R, but in reverse order.
4. After 16 rounds of these steps, Blowfish puts the L and R pieces back together, and the decrypted data is ready.

4.3 Encryption and Decryption process in HDFS

In the previous sections, we studied the associated frameworks that will help for underlying the research. In this section, the Encryption and Decryption process in HDFS will be discussed for the proposed methodology as shown in Figure 3.

First beginning with the steps for the encryption, the steps are as follows:

1. Communication and Request Submission: The HDFS client initiates the process by communicating with the master node through the distributed file system. It sends a request to create a new file, specifying the desired file name and other relevant attributes.

2. Space Availability Check and Data Node Selection: The master node, acting as the central coordinator, receives the request and checks the available storage space on the data nodes within the cluster. It identifies a data node with sufficient space to accommodate the new file and assigns it the responsibility of storing the file's data blocks.

3. Data Node Details Transfer: The master node communicates the details of the selected data node to the distributed file system, which forwards the information to the HDFS client. This allows the client to establish a connection with the designated data node for file creation and data transfer.

4. Attribute Transfer and File Encryption: The HDFS client transfers the file attributes to the selected data node, including file name, permissions, and any other relevant metadata. Once the attributes are received, the data node initiates the file encryption process, applying an encryption algorithm to protect the file's contents. The encryption process is used to encrypt the file, ensuring its confidentiality and protecting it from unauthorized access.

5. Writing Process Initiation: The HDFS client establishes a connection to the selected data node using the distributed file system output data streams. This connection serves as the channel for transferring the encrypted file data from the client to the data node.

6. Data Replication and Master Node Awareness: Once the file writing process is complete, the data node replicates the encrypted file data onto another data node within the cluster. This replication ensures data redundancy and fault tolerance, safeguarding the file against potential data loss due to node failures. The master node is kept informed about the current data node and the replication data node, maintaining a comprehensive map of file distribution throughout the cluster.

7. File Creation Completion: Upon successful replication, the file creation process is finalized, and the file is ready for access by authorized users. The HDFS client receives confirmation from the data node, signifying the successful creation and replication of the encrypted file.

5 Implementation

The encryption of data using Blowfish is very efficient, making it suitable for use in applications that require fast encryption and decryption. To support this claim, various experiments will performed and output against different parameters will be captured. Blowfish algorithm will be used to improve the security of the existing Hadoop setup. For capturing the output, we will execute the methodology for multiple iterations for a varied file size of few MB upto a GB.

5.1 Implementation Environment

The experiments were conducted using a single-node Hadoop cluster running on a laptop with an Intel Core i5-7200U processor and 4GB of RAM. The cluster consisted of one NameNode and one DataNode. The NameNode manages the cluster and stores information

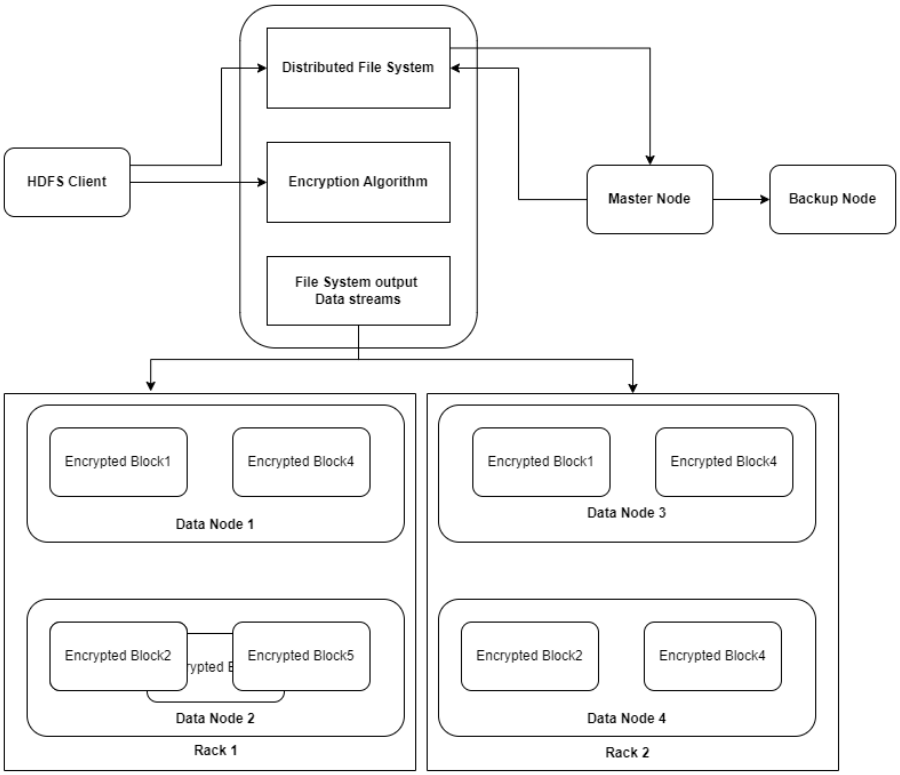

Figure 3: Encryption and Decryption process in HDFS

about the data, while the DataNode provides the physical storage space. The operating system was Linux (Ubuntu 22.04.3 LTS) with Hadoop version 3.3.6 installed. The implementation and standalone applications were written in Java, and the Java Development Kit (JDK) was installed using the *apt-get* command. The running components of the Hadoop cluster could be checked using the *jps* command. Additionally, Eclipse IDE for Java Developer was also used.

6 Evaluation

To evaluate out results, output for different parameters has be captured. by executing the methodology for multiple iterations for a varied file size of few MB upto a GB. They are as discussed below.

6.1 Encryption time

The graph in figure Figure 4 shows the encryption times for three different algorithms: DES, Blowfish, and 3DES. In terms of encryption time, DES takes 19.5 minutes, 3DES takes 18.6 minutes, and Blowfish takes 17.7 minutes. While decryption time of DES, 3DES and Blowfish algorithms is 24.9, 23.1 and 19.2 minutes. These metrics provide insights into the comparative efficiency of these algorithms, with Blowfish generally demonstrating quicker performance in both encryption.

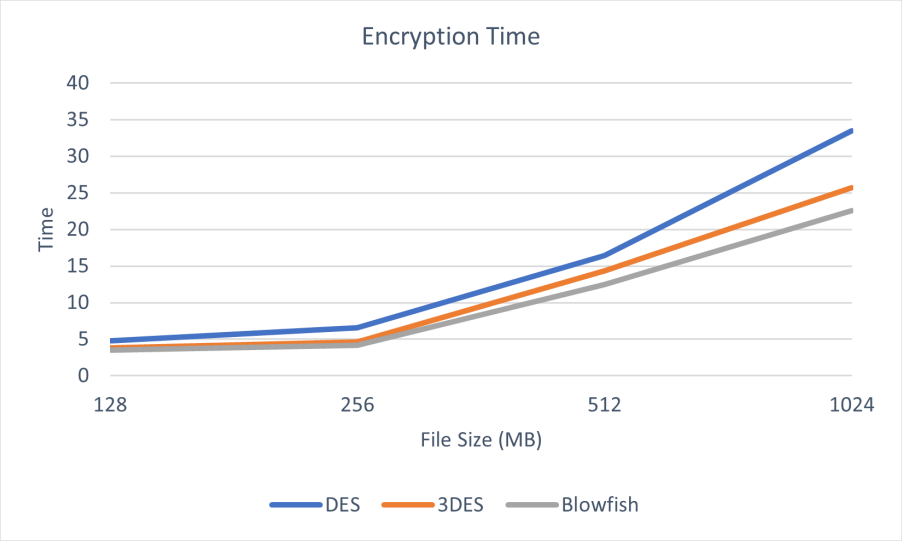

Figure 4: Encryption times for different file size

6.2 Decryption Time

The graph in figure Figure 5 the decryption time for different file sizes and encryption algorithms. DES is the slowest algorithm, followed by 3DES and Blowfish. Blowfish is the fastest algorithm for all file sizes.

The decryption time increases with file size for all three algorithms. However, the rate of increase is different for each algorithm. DES has the steepest rate of increase, followed by 3DES and Blowfish.

For small files (128 MB), Blowfish is about 2 times faster than 3DES and 3 times faster than DES. For large files (1024 MB), Blowfish is about 5 times faster than 3DES and 7 times faster than DES.

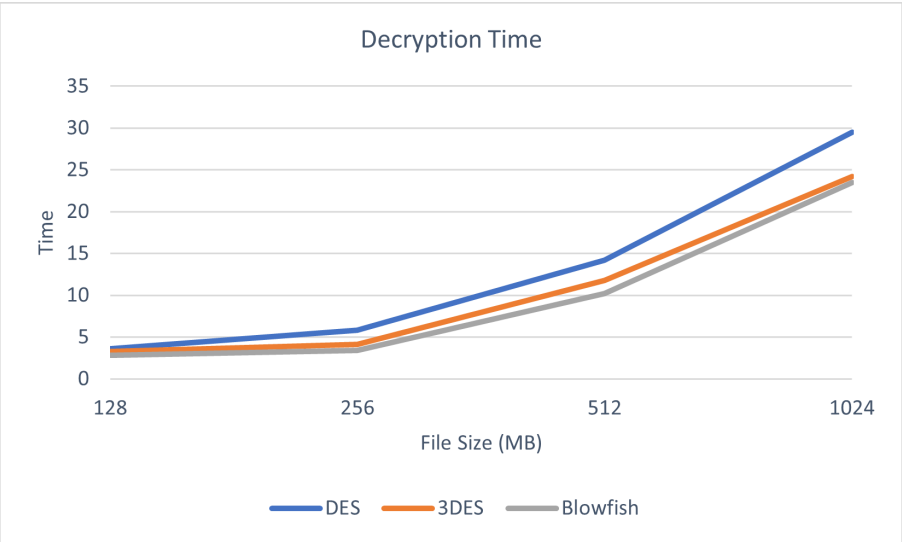

Figure 5: Decryption times for different file size

File Size	DES	3DES	Blowfish
128	4.8	3.8	3.5
256	6.6	4.6	4.2
512	16.4	14.3	12.5
1024	33.5	25.7	22.6

Table 1: Encryption time taken by different algorithms

6.3 Average encryption and decryption time

The graph in figure Figure 6 shows the average encryption and decryption times for three different algorithms: DES, Blowfish, and 3DES. In terms of average encryption time, DES takes 15.4 minutes, 3DES takes 12.2 minutes, and Blowfish takes 10.7 minutes. While average decryption time of DES, 3DES and Blowfish algorithms is 13.3, 10.9 and 10 minutes. These metrics provide insights into the comparative efficiency of these algorithms, with Blowfish generally demonstrating quicker performance in both encryption and decryption.

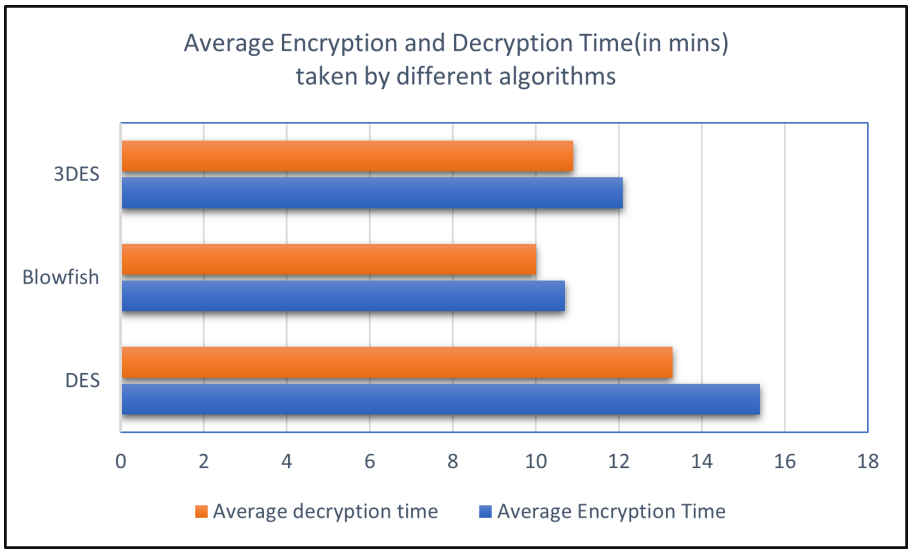

Figure 6: Average encryption and decryption times for different algorithms

6.4 Discussion

The proposed research conducted various experiment to evaluate the parameters that included Encryption and decryption time taken by the Blowfish algorithm on a varied size of data file.

The tables 1 and 2 shows the size of a file (in bytes), and the encryption time, decryption time (in minutes) respectively for three encryption algorithms: DES, 3DES, and Blowfish. The file sizes are 128, 256, 512, and 1024 bytes.

It is observed that, as the file size increases, the encryption and decryption time increases for all three algorithms. However, the time taken for Blowfish is consistently faster than the time taken for DES or 3DES. For example, it takes 3.5 seconds to encrypt a 128-byte file with Blowfish, but it takes 4.8 seconds to encrypt the same file with DES. Similarly, it takes 3.5 seconds to decrypt a 128-byte file with Blowfish, but it takes 4.8

File Size	DES	3DES	Blowfish
128	3.6	3.3	2.8
256	5.8	4.1	3.4
512	14.2	11.8	10.2
1024	29.5	24.2	23.5

Table 2: Decryption time taken by different algorithms

seconds to decrypt the same file with DES. The table shows that Blowfish is a faster encryption algorithm than DES or 3DES. The encryption time for all three algorithms is proportional to the file size. This means that the larger the file, the longer it will take to encrypt. For small files (128 MB), Blowfish is about 2 times faster than 3DES and 3 times faster than DES. For large files (1024 MB), Blowfish is about 5 times faster than 3DES and 7 times faster than DES. This is because Blowfish uses a larger key size and more complex encryption algorithm.

7 Conclusion and Future Work

The objective of the research was to study different possible security threats and issues of Hadoop HDFS. This research has significantly helped to enhance the understanding of Hadoop and its components. The findings have revealed Blowfish algorithm can be effectively used for encryption and decryption of files in HDFS, and can be further improvised to optimize other evaluation parameters. These findings have the potential to be used for inter-cloud data transfer with greater security and efficiently. Although, The current research conducted is limited to single node and the file size is comparably smaller than what it is in real production environment, future research should focus on methodology involving different evaluation parameters such as energy usage, processing time, memory, communication overhead and computation time versus varied data format will be studied to achieve the objectives.

References

Alaojan, S. E. and Alwattar, A. H. (2022). A modified blowfish algorithm to secure data in cloud, *2022 International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT)*, IEEE, pp. 218–222.

Bhathal, G. S. and Singh, A. (2019). Big data: Hadoop framework vulnerabilities, security issues and attacks, *Array* **1-2**: 100002.
URL: *https://www.sciencedirect.com/science/article/pii/S2590005619300025*

Funde, S. and Swain, G. (2022). Big data privacy and security using abundant data recovery techniques and data obliviousness methodologies, *IEEE Access* **10**: 105458–105484.

Johri, P., Arora, S. and Kumar, M. (2018). Privacy preserve hadoop (pph)—an implementation of big data security by hadoop with encrypted hdfs, *Information and Communication Technology for Sustainable Development: Proceedings of ICT4SD 2016, Volume 2*, Springer, pp. 339–346.

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

Kaushik, A. and Srivastava, V. K. (2021). An approach to secure sensitive attributes stored on hdfs using blowfish, *2021 3rd International Conference on Advances in Computing, Communication Control and Networking (ICAC3N)*, IEEE, pp. 1260–1264.

Naisuty, M., Hidayanto, A. N., Harahap, N. C., Rosyiq, A., Suhanto, A. and Hartono, G. M. S. (2020). Data protection on hadoop distributed file system by using encryption algorithms: a systematic literature review, *Journal of Physics: Conference Series*, Vol. 1444, IOP Publishing, p. 012012.

Parmar, R. R., Roy, S., Bhattacharyya, D., Bandyopadhyay, S. K. and Kim, T.-H. (2017). Large-scale encryption in the hadoop environment: Challenges and solutions, *IEEE Access* **5**: 7156–7163.

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

Schneier, B. (1994). Description of a new variable-length key, 64-bit block cipher (blowfish), *in* R. Anderson (ed.), *Fast Software Encryption*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 191–204.

Sekar, K. and Padmavathamma, M. (2016). Comparative study of encryption algorithm over big data in cloud systems, *2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)*, IEEE, pp. 1571–1574.

Sharma, P. P. and Navdeti, C. P. (2014). Securing big data hadoop: a review of security issues, threats and solution, *Int. J. Comput. Sci. Inf. Technol* **5**(2): 2126–2131.

Shehzad, D., Khan, Z., Dag, H. and Bozkus, Z. (2016). A novel hybrid encryption scheme to ensure hadoop based cloud data security, *International Journal of Computer Science and Information Security (IJCSIS)* **14**(4).

Sinha, S., Gupta, S. and Kumar, A. (2019). Emerging data security solutions in hadoop based systems: Vulnerabilities and their countermeasures, *2019 International Conference on Computing, Communication, and Intelligent Systems (ICCCIS)*, pp. 235–240.

Yousif, R. Z., Kareem, S. W. and Abdalwahid, S. M. (2020). Enhancing approach for information security in hadoop, *Polytechnic Journal* **10**(1): 81–87.

Zhonghan, C., Diming, Z., Hao, H. and Zhenjiang, Q. (2013). Design and implementation of data encryption in cloud based on hdfs, *1st International Workshop on Cloud Computing and Information Security*, Atlantis Press, pp. 274–277.