International Journal of Communication Networks and Information Security
2024, 16(4)
ISSN: 2073-607X,2076-0930
https://ijcnis.org/
Research Article

Efficient Dual-Level Encryption for Securing Data in HDFS Using Hybrid User-Defined Function (HUDF)

Shivani Awasthi*
Department of Computer Science, HBTU, Kanpur
**Corresponding authorE-mail:200304002@hbtu.ac.in*

Narendra Kohli
Department of Computer Science, HBTU, Kanpur

ARTICLE INFO
Received: 17 Aug 2024
Accepted: 27 Sep 2024

ABSTRACT
Big Data is a new class of technology that gives businesses more insight into their massive data sets, allowing them to make better business decisions and satisfy customers. Big data systems are also a desirable target for hackers due to the aggregation of their data.The Hadoop Distributed File System (HDFS) stores massive data in the Hadoop framework. Since HDFS does not safeguard data privacy, encrypting the file is the right way to protect the stored data in HDFS but takes a long time. In this paper, regarding privacy concerns, we use different compression-type data storage file formats with the proposed hybrid user-defined function (HUDF)based on an XOR-Onetime pad with AES to securedata in HDFS. In this way, we provide a dual level of encryptionby maskingselective data and whole data in the file. Our experiment demonstrates that this scheme offersoverall data security and also faster processing time than the conventional methods. The proposed HUDF with ORC, Zlib (Z) file format (HUDF-ORC-Z) gives 9-10% better performanceresults than 2DES and other method. Finally, we efficiently utilized the space, improved query processing time,and decreased data load timewith the Hive engine.
Keywords- *AES, Avro, Deflate, ORC, Parquet, Snappy, Gzip, HDFS, cloud environments*

1. Introduction

Today, data is one of the most valuable resources for business in all industries. The increasing volume and significance of data have given rise to a new issue that conventional analytic methods are unable to address. The solution to this issue arose from developing a novel paradigm known as big data(Morenoet al., 2016).The need for large organizations, like Google, Facebook, and Yahoo, to analyze enormous amounts of data led to the term "big data"(Yaqoobet al., 2016; Garlasu et al., 2013). Numerous methods have characterized big data, including 4V Volume, Velocity, Variety, and Veracity and 3V Volume, Variety, and Velocity (Gandomiet al., 2015; Chen & Zhang,2014;Rodríguez-Mazahua et al., 2015; Hashem et al., 2015).
Doug Laney, who works at Gartner, characterized big data using the 3 Vs: volume, velocity, and variety. The terms volume, diversity, and velocity refer to the quantity, variety, and pace at which data comes from and goes out of a given source (Chen & Zhang, 2014). IBM and Microsoft included veracity, sometimes known as variability, as the fourth V of their concept of big data. The unreliability and complexity of data are referred to as veracity. As the fourth V in the definition of big data, McKinsey & Co. added value (Chen, et al., 2014). Value is the worth of unexpected findings within massive data collections. Vast data, as defined generally, is a collection of huge, complicated data sets that are too

vast for current data processing technology to handle effectively (Chen & Zhang, 2014).However, in addition to the amount and variety of data it contains, big data has also raised new issues with data privacy and security (Moreno et al., 2016). As a result, establishing security in big data has become one of the main challenges that might prevent technological growth; big data cannot get the necessary degree of confidence without sufficient security assurances (Thuraisingham, 2015;Van Rijmenam, 2014).

Since massive data sets come with great responsibility, we offer storage-level security in this research. When using the Hadoop HDFS data storage service, clients are not strongly motivated to save data locally, so they transmit it continuously. As is customary, the data is stored on the Hadoop HDFS storage server so users may access individual data at any time and from any location. The Hadoop HDFS storage server provides information security assurance and hardware allocation. The client may obtain their information as long as they have an internet connection. Hadoop is one recent innovation that offers a way to store vast amounts of information. The structure depends on Java and is implemented in an open-source manner. Hadoop is used in big clusters or cloud environments(Polato et al. 2014).Although Hadoop is widely utilized, indicating its versatility, it does not have robust security features to safeguard the data stored in HDFS. Researchers use various methods for storing the data in HDFS. An encryption scheme is one of the methods for securing data in HDFS.

HDFS may be deployed on low-cost hardware. HDFS supports file sizes up to terabytes or even petabytes. Programs that use it typically do so as a result. Hadoop uses a master/slave design, in which a master computer is one of the cluster's machines and all other machines are its slaves(Balusamy, 2021). HDFS stores the file's metadata information on a single standalone server called the Namenode and data inanotherdistributed node called the Datanode. The HDFS architecture isshown in **Fig. 1.**

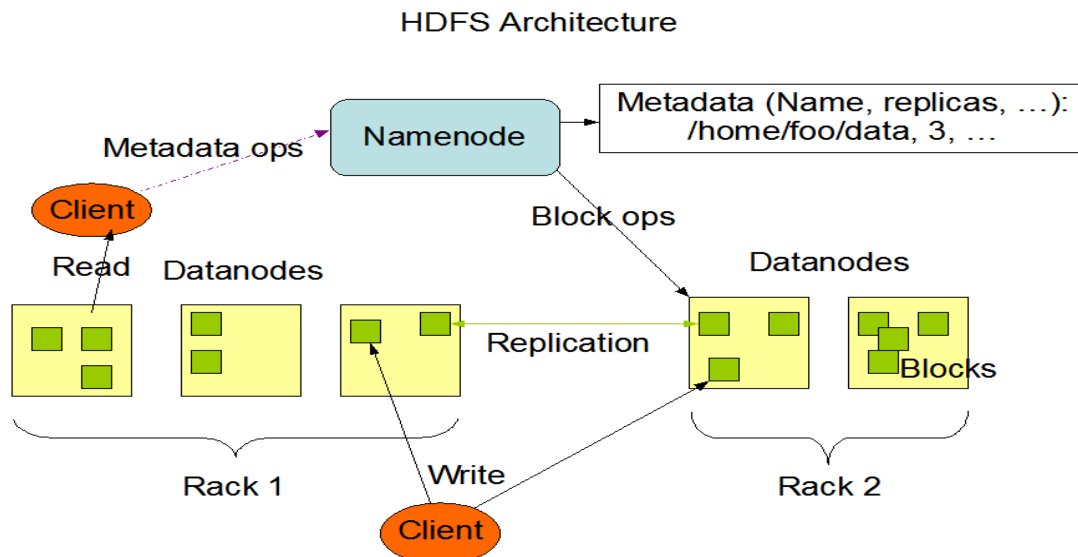


Fig. 1. HDFS architecture (https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs)

The Namenode and its corresponding job tracker daemon are present on the master node. The Namenode oversees the whole file system's namespace, regulates end-user access to the files, and uses the Heartbeat signal to monitor the Datanode's health. While the actual data is not included, the Namenode is the directory for the Datanode that holds the details about which blocks together make up the file and where those blocks are located. Additionally, keeping a lot of little files is not a good use for HDFS. This is because Namenode memory capacity restricts the total number of files that can be stored in HDFS and stores the file system information. More metadata must be retained when many little files need to be kept, and more metadata takes up memory. The Datanode is made up of a set of all slave nodes that have the Task Tracker daemon attached to them. The real data are spread throughout the cluster and are located at the Datanode. The user data file is distributed by first breaking it into blocks, which are stored in the Datanodes and have a default size of 64 MB. We can manage the size of the data

block if it is not fixed. The Namenode determines which file block has to be mapped to which Datanode. The same file has many Namenode(Metadata) components. By default, every block is mapped to three Datanodesto replicate data and offer fault tolerance and dependability. Every block's position in the Datanode is known to the Namenode. In addition, Namenode does several other tasks, including renaming files and directories and opening and shutting files. Additionally, inside a given rack, the Namenodedetermines which file block has to be written to which Datanode. The rack is a section of storage where several Datanodes are arranged. Based on the proximity of the nodes to each other, the Namenode decides where these blocks belong. The Datanodes communicate more quickly the closer they are to one another.

Every bit of information stored in RAM on the Namenode is regularly backed up by HDFS's secondary Namenode. The secondary Namenode functions as a recovery mechanism in the event of a failure instead of taking over as the Namenode. The secondary Namenode needs memory space equal to the Namenode's to back up the data kept there while running on a different computer. The solution does not guarantee high availability even with the backup Namenode present since the Namenode is still a single point of failure. The filesystem cannot be read or written until a fresh Namenode is activated.

Apache Hive is a distributed, fault-tolerant data warehousing solution on top of the Hadoop that enables large-scale analytics. A key component of many data lake systems, the Hive Meta Store (HMS) provides a central repository for quickly assessed information to enable data-driven, well-informed decision-making (Due, 2018). Hive is not a relational database, despite storing the schemas in a database and the processed data in HDFS. Furthermore, Hive features a faster query language similar to SQL called Hive Query Language (HiveQL or HQL)(Wadhera et al., 2021). A database technology called Hive aids in the meaning of databases and schema to analyze structured data.All of the HQL statements are internally translated by a compiler into Map Reduce jobs that are sent to Hadoop for job execution.HDFS is a data warehouse and storage system, while the hive UI is an interface for managing data warehouses. HQL can design and execute Map Reduce queries as statements, and the Hive execution engine is produced by integrating the Map Reduce and HiveQL process engines. (Bansal et al. 2014; Wadhera, et al., 2021;Priya, 2019; Due, 2018).

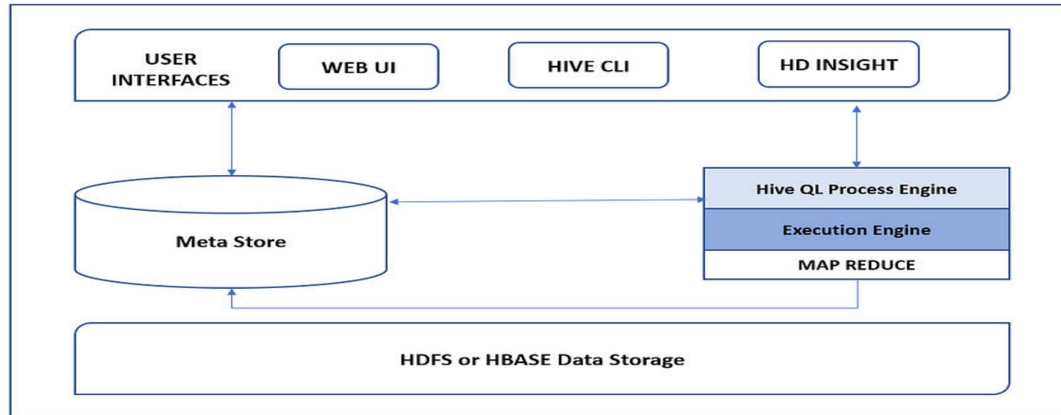


Fig. 2. Hive Architecture (Bansal et al., 2018)

TheHive architecture has been shown in **Fig.2**.,which works on batch processing. Moreover,the key/value pair is supported by the Hive. It supports all of the database's ACID features. The compiler converts HQL commands into a MapReduce DAG graph (Bansal et al., 2018).

Privacy, managing, and comprehending vast amounts of data are difficult tasks.Although HDFS can handle a wide variety of file formats, the use case must be taken into account when selecting a format(Naidu, 2022). The most common text formats for storing raw data are CSV, text files, JSON, and XML. The drawback of raw data formats is that they require a significant amount of storage capacity. Another more important problem is storing raw data in HDFS, where each piece of data is duplicated three times. As a result, the quantity of data that HDFS copies would be tripled, making the file size three times larger than it was initially. As a result, compressing and storing the files in proper format into HDFS becomes essential.

HDFS allows for more effective use of storage space based on job definition, and it contains some binary storage data types file formatssuch as ORC, Avro, and Parquet(Moránet al., 2015;Menon, 2014). These formats are designed for systems that use MapReduce types of frameworks. They are a methodical combination of several components, including data storage format, data compression, data encoding, and optimization algorithms for data reading(Morán et al., 2015).These file formats provide store data in the form of columns or row-wise in binary form format, so reading and writing from HDFS is easily done from HDFS storage space.Various file formatsforthe compression of data have been shown in **Table 1**.

Table 1 Various file formatsforthe compression of data

File Format	Description	Advantages	Disadvantages
ORC	The file format combines the benefits of both row-wise and column-wise storage for efficient query processingand compression	Faster read and write large datasets. Efficient compression.	Limited supportfor schema evaluation
Parquet	A columnar storage format that is optimized for processing large data sets in parallel.	Columnar storage allows for efficient querying and processing of largedatasets. Efficient compression for reduced storage requirements.	Slower writing speed than a row-basedstorage format and limited schema evaluation
AVRO	A binary data format that is compact and efficient for storing and exchanging data in big data applications.	Full support for schema evaluation. compact binary format for efficient data exchange.	Slower than another file format due to schema parsing.

On the other hand, the outcomes of various encryption techniques have shown that the original file document size can be increased. Additionally, it is possible to extend both the uploading and downloading times for a particular file. Therefore, to address these issues, we proposed HUDFwith a different compression type storage file format as a novel encryption-decryption technique.This approach is a highly scalable and efficient solution for data protection in HDFS. According to simulation data, this method has demonstrated significant gains based on performance parameters such as encryption-decryption time, throughput, and total map-reduce time for encryption-decryption.
Our contributionsare:

- Examine Hadoop, weigh its benefits and drawbacks concerning privacy and security, and learn about the various file formats.
- To propose a hybrid user-defined function(HUDF) based on an XOR-Onetime pad with AES forencryption and decryption.
- In the Hive shell, HUDF is applied withvarious compression techniquesindifferent file formats,such as ORC, Parquet, and Avro.
- Carry out tests using test data to demonstrate the effectiveness of our suggested strategy based on proposed HUDF with differentcompression types storage file formats.

2. Related Works

(Song et al., 2017) proposean HDFS data encryption method compatible with both AES and ARIA algorithms on Hadoop. The Korean government took ARIA as the standard encryption system, even though the available research only supports AES encryption. It includes a block-splitting component and a variable-length data processing component. In this paper, parallel compression and decompression operations on HDFS dataSplittableCompressionCodecare used.

(Mahmoud et al., 2018)suggested a method for enhancing Hadoop's encryption and decryption performance based on inbuilt AES and OTP algorithms.Using this method, the encrypted file's size rose from its original size by 20%.

(Kamaruzaman et al., 2018)demonstrate that data stored in HDFS can be somewhat secured with the use of the 256-bit AES encryption technique. It provides some degree of security for data saved in HDFS.

(Teng et al., 2020)suggestedmodified data encryption algorithmsin cloud computing to secure data. Modified AES used random disturbance information, boost key choreography and column mix operation. It ensures attribute privacy and improves decryption efficiency for outsourced data storage.

(Jain et al., 2019)suggested a blowfish encryption methodfor access control, authentication, and confidentiality on the Hadoop serverinstead of the AES algorithm(Kamaruzaman et al., 2018).A privacy layer is created between the user and the real owner of data, whichaddresses vulnerabilities in the Hadoop framework and improves information security.

(Khafagy et al., 2020) propose a Hybrid-Key Stream Cipher Mechanism (HKSCM) for encryption.HKSCM integrates AES and OTP algorithms to enhance data security. In (Khafagy et al., 2020), HKSCM performance is better than AES, RC4, and hybrid algorithms.

(Jayapandian et al., 2020) discuss the merits and demerits of big data. In this paper, MHT and AES algorithms are used to solve security problems.APEX20KCFPGA is used by AESto improve performance. The drawback of the above schemeis cost and pattern analysis.

Relational database's incapacity to manage unstructured data gave rise to big data. Hadoop and MapReduce are used for processing and handling large amounts of data. HDFS used to store the data. AES encryption is used for the security of large data. It addressessecurity issuessuch asthe lack of a security model due to the fragmented data in Big Data clusters,and vulnerable environments due to distributed computing and parallel processing(Gattojuet al., 2021).

(Kaushik et al., 2020)evaluate the performance of the popular data encryption methods AES and DESede on the private data kept in HDFS based on data size, encryption time, and decryption time.

(Algaradi et al., 2021) studied methods and technologies for securing big data in cryptography. Existing encryption algorithms have limitations in performance and execution time. The proposed scheme uses a map-reduce programming model for parallel processing of large data setsand overcomes the limitation.

(Mohanraj et al., 2022)use a hybrid encryption algorithm using CP-ABE and AES for big data security. This scheme

gives 96.5% maximum efficiency, 7.12-minute encryption, and 6.51-minute decryption time.

(Khadji et al., 2023)investigate how to use lightweight cryptography with the MapReduce architecture to process large amounts of data. It suggests a hybrid key management system to process vast amounts of data securely and effectively. In a lightweight cryptography approach, the author recommended AES and Chacha20 for confidentiality and integrity,although Rabbit stream cipher and NOEKEON block cipher for speed.

(Ramya et al.,2020)focused on HDFS performance, data node selection, security, and deduplication in a Big data environment. Elliptic curve cryptography with key generation and a PSO-based MapReduce model were employed by the author in this paper. Security, a lack of space, conventional PSO, and FCM issues were the limitations of this paper.

(Gupta et al., 2023)covered the essential issue of security and privacy in large data processing. To strengthen the security and privacy safeguards that are already in place inside the MapReduce paradigm, the authors suggest a solution that they name the Fortified MapReduce Layer. Their proposed method entails incorporating additional security layers into the MapReduce framework to solve security and privacy challenges.The drawback of this paper was installing and incorporating such a security layer into existing MapReduce systems could be complex and require major modifications.

Muhammad et al. [42] proposed a customized genetic algorithm to improve cloud data encryption that offers potential benefits in optimization and performance, it also introduces several drawbacks related

to computational complexity, optimization overhead, implementation challenges, security concerns, performance variability, and usability. These factors must be carefully considered when designing and deploying such a system to ensure that the benefits outweigh the costs and risks.
It is seen from the above results that hybrid encryptionmethods and compression-type storage file formats may be a more suitable option for enhancing the security of large data setswithout compromising on performance. Our research, taking into account these observationssuggeststhe dual level of security through HUDF,with the different compression typesof storage file formats. This schemeefficientlyutilizes space to store large datain a distributed file system context without writing map-reduce jobs.It reduces load time and improvesquery processing time with the least computing cost.

3. Proposed Methodology

To protect large amounts of HDFS data, we suggested proposed HUDFbased on XOR-Onetime pad and AES as a hybrid encryption and decryption technique. When comparing the twofold encryption process to other encryption methods, data security is improved. XOR-Onetime Pad is more secure in comparison to the OnetimePad. Due to its decentralized access control and secure communication capabilities in dynamic environments, the XOR-Onetime pad with AES has recently drawn increased attentioncompared to the Onetime pad. The data is cryptographically secured during transmission between the Datanodesas well as storage level by implementing encryption policies with various file formats like ORC, Parquet, and Avro. In this way, we focus on thedual-level encryption of confidential data and overall encryptingthe file dataviathe proposed HUDFwith different compression-typestorage file formats such as ORC, Parquet, and Avro. The overview of the proposed HUDF for encryption and decryption to/from HDFS has been shown in **Fig3**.

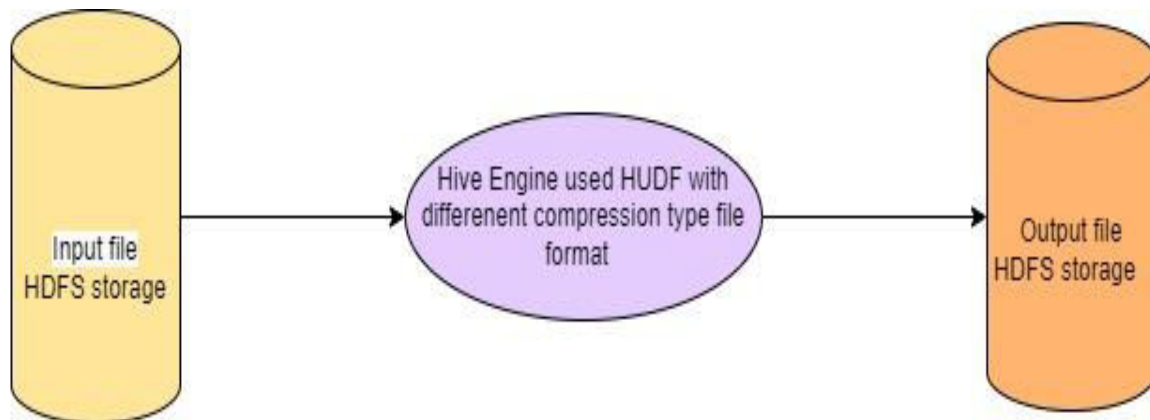


Fig.3. Overview of the proposed scheme for encryption and decryption

HDFS stores the file in CSV format then HQL is used for reading and writing the data from HDFS. Throughthe Hive engine, this HQL easily converts into the map-reduce job and gives the output in the HDFS warehouse (Hanandeh et al.,2022) based on the above **Fig. 3.**

3.1 XOR-Onetime pad

A symmetric encryption scheme called the Vernam Cipher was created in 1917 by Gilbert Vernam, an employee of AT & T. One system that demonstrates complete cryptographic stability and perfect secrecy is the Vernam cipher. OTP is a simple Verman cipher cryptosystem. OTP used the "Exclusive OR" (xor) Boolean function(Iavich et al.,2018).
The message is XOR^{ed} with the key to obtain the cipher in the one-time pad.
c = m XOR k
The cipher is XOR^{ed} with the message for decryption.
m = x XOR k
When using the XOR-Onetime pad, data is encrypted using a technique that is difficult to decrypt using a brute-force approach, which involves creating random encryption keys and matching them with the right one.

3.2AESAlgorithm

The block length of the AES block cipher is 128 bits. Three distinct key lengths are supported by AES: 128 bits, 192 bits, or 256 bits(Zodpe et al., 2018). 10 processing cycles make up encryption for 128-bit keys, 192-bit key rounds are 12, and 256-bit key rounds are 14-bit keys. One single-byte is included in each processing round-based substitution stage, a permutation step that is row-wise, and amixing step in columns, as well as the round key. The sequence in which these four actions are carried out varies dependingon cryptography and decryption. In contrast to DES, the decryptionalgorithm and the encryption algorithm are very different.The general architecture of the AES (Advanced Encryption Standard) encryption mechanism has been shown in **Fig.4**.(Zodpe et al.,2018; Rihan et al., 2017).

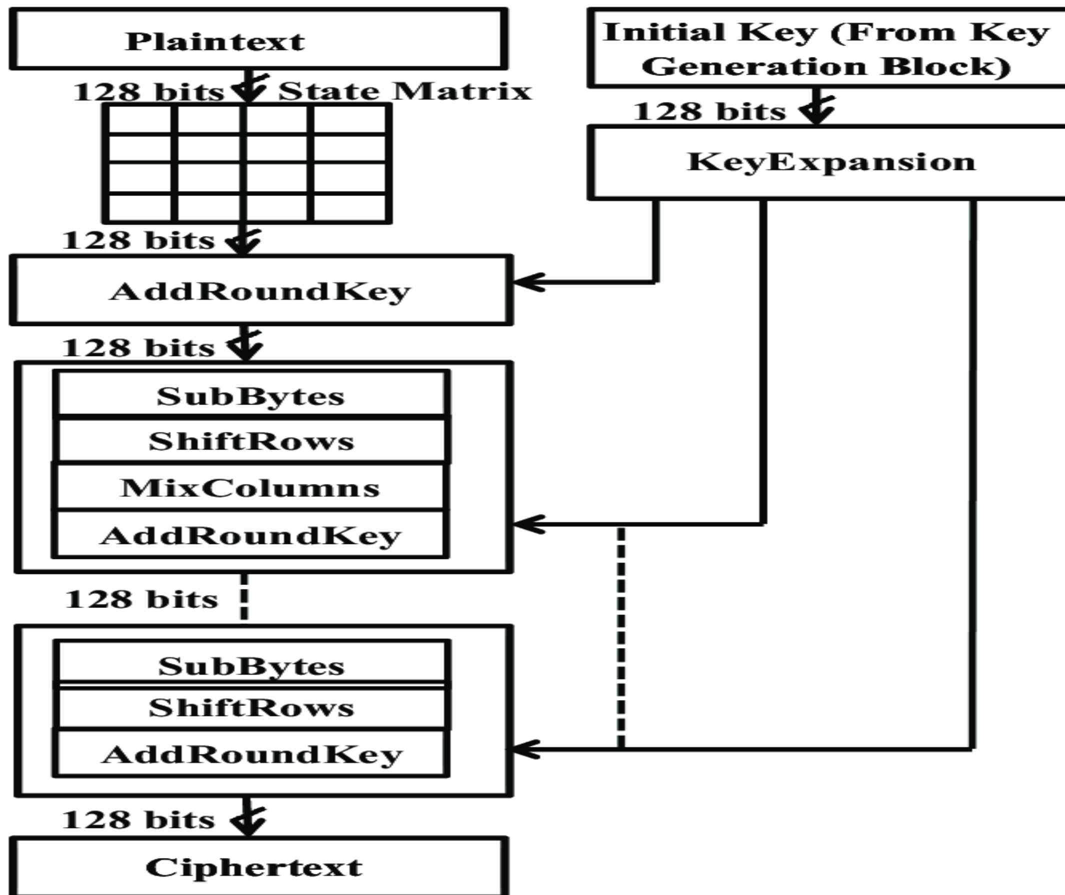


Fig. 4.AES Encryption Mechanism (Zodpe et al.,2018)

Differentcompression-type storage file formatsare used with theproposed HUDF in the hive shellto providedouble-layer security. The first layer usesencryption and decryption functions based on aproposed HUDFto protect the sensitive data in the table after thatuses different file formatssuch as ORC, Parquet, and Avro with their compression method to provide dual-level security of important data and overall data that are arranged in the table. Using the above approach,we provide security forimportant data that is stored in HDFS and easily move data tothe cloudenvironment. The compression method is used by the different storage file formats, by which we compress more data in less space as well as reduce load timeand query processing time over the distributed environment. Thedifferent file formats and their types of compressionhave been shown in **Table 2**(Due, 2018).

Table 2 File format and its Compression types

ORC and its Compression types	Parquet and its Compression types	Avro and its Compression types
Snappy (S)	Snappy (S)	Snappy (S)
Zlib (Z)	Gzip (G)	Deflate (D)
None (N)	Uncompressed (U)	Uncompressed (U)

3.3*Pseudocodeof proposed HUDF*

```
1    Generate XOR-Onetime pad pseudocode
function generate_one_time_pad12(len):
pad = random_bytes(len) // Generate random bytes of specified length return pad
2    XOR encryption using Onetime pad Pseudocode
function xor_encrypt1(plaintext1, one_time_pad12):
ciphertext1 = ""
for i from 0 to length(plaintext1) - 1:
ciphertext1 += char_to_byte(plaintext1[i]) XOR one_time_pad12[i]
return ciphertext1
3    XOR decryption using Onetime pad Pseudocode
function xor_decrypt1(ciphertext1, one_time_pad12):
plaintext1 = ""
for i from 0 to length(ciphertext1) - 1:
plaintext1 += byte_to_char(ciphertext1[i] XOR one_time_pad12[i])
return plaintext1
4    AES Encryption Pseudocode
function aes_encrypt1(plaintext1, key):
cipher = AES.new(key, AES.MODE_ECB) // Initialize AES cipher in ECB  mode
ciphertext1 = cipher.encrypt(pad(plaintext1)) // Pad plaintext if necessary and then encrypt
return ciphertext1
5    AES Decryption Pseudocode
function aes_decrypt1(ciphertext1, key):
cipher = AES.new(key, AES.MODE_ECB) // Initialize AES cipher in (ECB) mode
plaintext1 = unpad(cipher. decrypt(ciphertext1)) // Decrypt ciphertext and then unpad plaintext
return plaintext1
6    Combine XOR-Onetime pad with AES Pseudocode
function xor_aes_encrypt1(plaintext1, aes_key_length):
one_time_pad12 = generate_one_time_pad12(aes_key_length)
ciphertext1 = xor_encrypt1(plaintext1, one_time_pad12)
aes_key = generate_one_time_pad12(aes_key_length) // Generate a separate AES key
encrypted_aes_key = aes_encrypt1(aes_key, one_time_pad12) // Encrypt AES key using XOR-one-
time pad
return (ciphertext1, encrypted_aes_key)
function xor_aes_decrypt1(ciphertext1, encrypted_aes_key):
(one_time_pad12, _) = xor_aes_decrypt1(encrypted_aes_key, encrypted_aes_key) // Decrypt AES key
using XOR-one-time pad
plaintext1 = xor_decrypt1(ciphertext1, one_time_pad12)
return plaintext1
```

3.4 Proposed HUDFbased on XOR-Onetime pad with AES

The proposed HUDF is shown in **Fig.5.** This HUDF converts into a jar file and loadsinto HDFS to work as a permanentfunction.

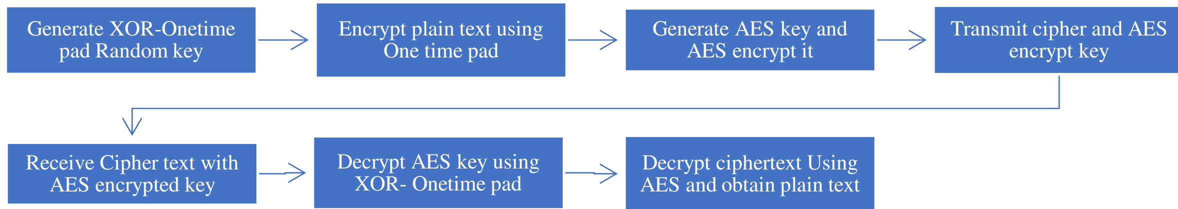


Fig.5. Flowchart of HUDF

Thesteps of the Proposed methodology in the Hive shell are shown in **Fig.6.**

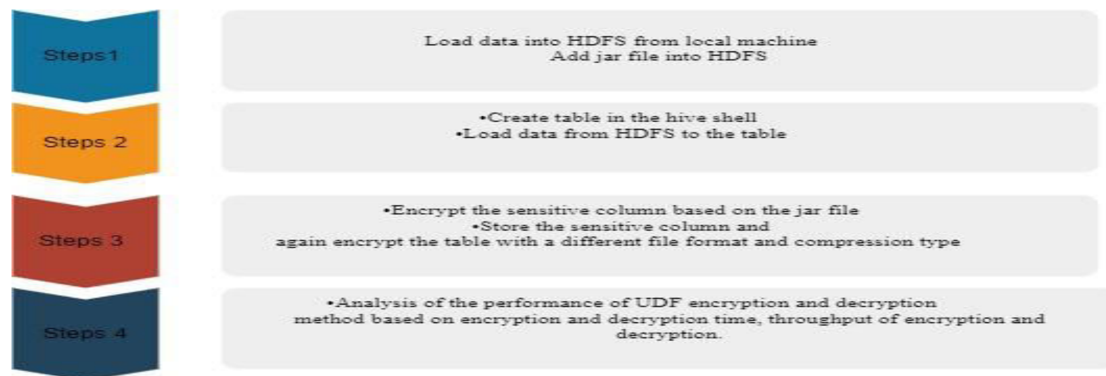


Fig.6.Steps forintegrating proposed HUDF with different compression-types file formatsin Hive shell

A way to increase security that combines proposed HUDF with different compression types of storage data file formats is shown in **Fig.5.** and **Fig.6.**After processing, HDFS-encrypted files stay encrypted in the HDFS storage warehouse in the Apache Hadoop environment. The results, including the intermediate findings, are always kept in an encrypted format on a non-HDFS file system within the cluster. Using a parallel computing technique, data has been divided into smaller chunks at the client level, and encrypted storage has been made at HDFS. Additionally, encrypted data can be directly used for Map Reduce jobs, which can be decrypted before processing by importing the relevant decryption library. A MapReduce job divides its input into fixed-size chunks. These are the input splitportions of the input used by one map alone. The input data is processed in decrypted form at the Map function, while the output data is saved in encrypted form in the HDFS. After the intermediate results of the Map function are decrypted, the Reduce function is applied, and the final encrypted data is placed in HDFSwhich is only accessible by authorized clients. The encryption and decryption processes are identical, and they are both easy to use, affordable, and scalable without sacrificing functionality, performance, or scalability. As a result, they are simple to suggest and successfully solve huge data cluster security flaws.In this proposed methodology, we used theHQL with proposed HUDFand different compression-types storage fileformatsto provide dual-levelsecurity at REST and Motion. Serialization-deserialization and compression-decompression are easily done in different file formats, so the researchershouldn't be worried aboutserialization-deserialization and compression-decompression.As a result, this approach is a scalable and efficient solution for data protection in HDFS.

4. Result and Discussion

Experiments on a Hadoop cluster running on an i7 CPU running at 2.80 GHz with 16 GB of RAM, 64-bit OS, and installedvia Cloudera VM have validated the proposed encryption approach for large data

security in the HDFS environment. Other nodes are chosen as Datanodes, and one node is designated as the master node. Different-size files[41] such as 8KB, 11KB, and 16KBare utilized to assess the performance of the encryption and decryption processes. Encryption and decryption times,the throughput of encryption, and decryption, and total map-reduce time for encryption and decryption are among the parameters that are compared with the proposed methodology.

4.1Encryption Time of Proposed Methodology

The encryption time is obtained bycalculating the time taken to generate the cipher text from the plain text. It is observed in **Fig. 7., Fig.8**, and **Fig. 9.** that the encryption time gradually increases when the file size is too long.**Fig. 7.**shows the result of encryption with ORC file format and itsdifferent compression techniques such as N,S,& Z withproposed HUDF(HUDF-ORC-N/S/Z). The result of encryption is based on the parquet file format and its different compression techniques such as U, S, & G with proposed HUDF (HUDF-PARQUET-U/S/G) shown in **Fig. 8**.The encryption time result based on the Avro file format and its various compression methods, including U, S, & D with proposedHUDF (HUDF-AVRO-U/S/D), is displayed in **Fig.9.**

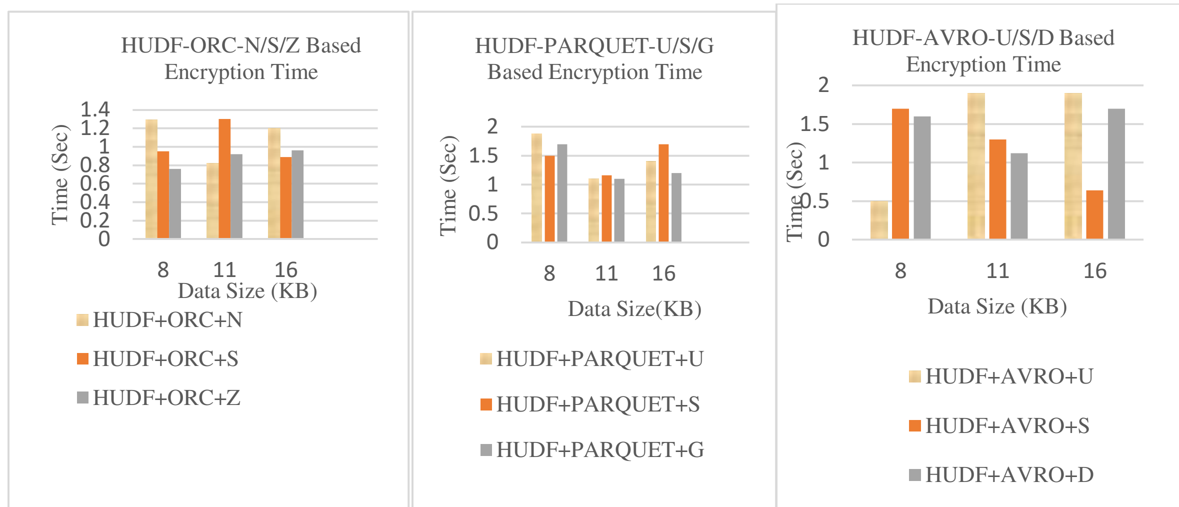
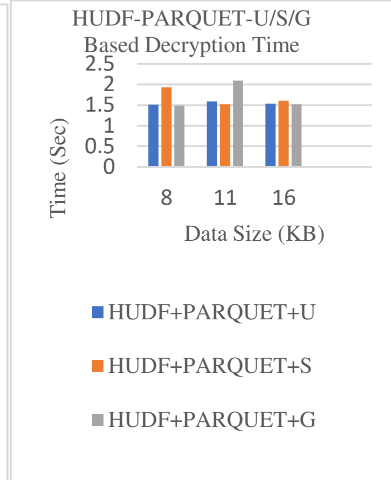


Fig.7. **Fig.8.** **Fig. 9.**

4.2 DecryptionTime of Proposed Methodology

The decryption time is obtained by calculating the time taken to generate the plain text from the cipher text. It is observed in **Fig.10., Fig. 11.**, and **Fig. 12.**The outcome of decrypting an ORC file format using several compression methods, including N, S, & Z is displayed in **Fig.10.** with proposed HUDF(HUDF-ORC-N/S/Z). **Fig.11.**shows the result of decryption based on the parquet file format and its different compression techniques such as U, S, & G withproposed HUDF (HUDF-PARQUET-U/S/G).

Fig. 12. displays the result of decryption time depending on Avro file format and its various compression strategies, such as U, S, & D withproposed HUDF (HUDF-AVRO-U/S/D).



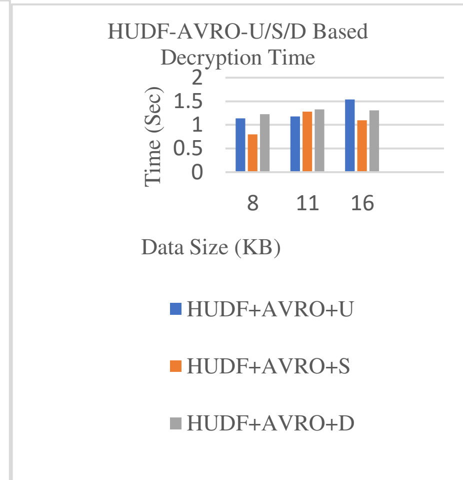
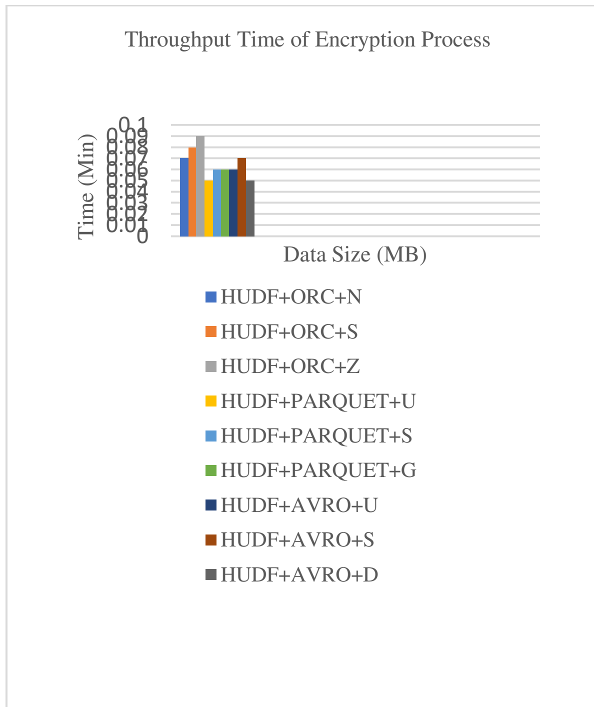


Fig.10. **Fig.11.** **Fig. 12.**

4.3 Throughput

Throughput is obtained by calculating thefile size divided by the time for encryption and the file sizedivided by the decryption time. **Fig.13**.shows the throughput based on encryption time and the throughput based on decryption time is shown in **Fig. 14.**



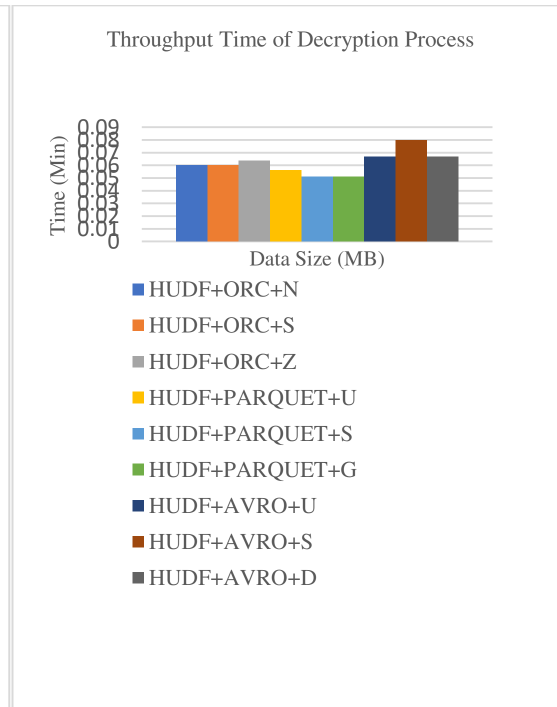


Fig. 13. **Fig.14.**

Table3, and Table 4.compare the results based on encryption and decryption time. It proves that the proposed HUDF-ORC-Z format isbetter than the other methods. Tables 5, andTable6 present the overall map-reduce time of the encryption and decryption process.

Table 3 File Encryption Time Performance Comparison BetweenHUDF-ORC-N/S/Z, HUDF-PARQUET-U/S/G, HUDF-AVRO-U/S/D, and 2DES

File Size	HUDF-ORC-N/S/Z			HUDF-PARQUET-U/S/G			HUDF-AVRO-U/S/D			2DES
KB	N (Sec)	S (Sec)	Z (Sec)	U (Sec)	S (Sec)	G (Sec)	U (Sec)	S (Sec)	D (Sec)	(Sec)
8	1.29	0.95	0.76	1.88	1.50	1.70	0.50	1.70	1.60	6.4
11	0.82	1.30	0.92	1.1	1.16	1.10	1.90	1.30	1.12	7.2
16	1.20	0.89	0.96	1.40	1.70	1.20	1.90	0.64	1.70	10.2
Throughput (MB/Min)	0.07	0.08	0.09	0.05	0.06	0.06	0.06	0.07	0.05	0.01

Table 4 File Decryption Time Performance Comparison Between HUDF-ORC-N/S/Z, HUDF-PARQUET-U/S/G, HUDF-AVRO-U/S/D and 2DES

File Size	HUDF-ORC-N/S/Z			HUDF-PARQUET-U/S/G			HUDF-AVRO-U/S/D			2DES
KB	N (Sec)	S (Sec)	Z (Sec)	U (Sec)	S (Sec)	G (Sec)	U (Sec)	S (Sec)	D (Sec)	(Sec)
8	1.43	1.36	1.24	1.51	1.93	1.49	1.14	0.80	1.23	6.3
11	1.30	1.39	1.37	1.59	1.52	2.10	1.18	1.28	1.33	7.5
16	1.10	1.38	1.44	1.54	1.61	1.52	1.54	1.1	1.31	10.4
Throughput (MB/Min)	0.06	0.06	0.06	0.05	0.05	0.05	0.06	0.08	0.067	0.01

Table 5 Total Map-Reduce Timeof HUDF-ORC-N/S/Z, HUDF-PARQUET-U/S/G, and HUDF-AVRO-U/S/D Encryption process

File Size	HUDF-ORC-N/S/Z			HUDF-PARQUET-U/S/G			HUDF-AVRO-U/S/D		
KB	N (msec)	S (msec)	Z (msec)	U (msec)	S (msec)	G (msec)	U (msec)	S (msec)	D (msec)
8	1290	950	760	1880	1500	1700	500	1700	1600
11	820	1300	920	1100	1160	1100	1900	1300	1120
16	1200	890	960	1400	1700	1200	1900	640	1700

Table 6 Total Map-Reduce Timeof HUDF-ORC-N/S/Z, HUDF-PARQUET-U/S/G, and HUDF-AVRO-U/S/D Decryption process

File Size	HUDF-ORC-N/S/Z			HUDF-PARQUET-U/S/G			HUDF-AVRO-U/S/D		
KB	N (msec)	S (msec)	Z (msec)	U (msec)	S (msec)	G (msec)	U (msec)	S (msec)	D (msec)
8	1430	1360	1240	1510	1930	1490	1140	800	1230
11	1300	1390	1370	1590	1520	2100	1180	1280	1330
16	1100	1380	1440	1540	1610	1520	1540	1100	1310

5. Trade-offsand Strategies for Balancing between Performance and Security

The proposed method for safeguarding data in HDFS involves combining XOR-One-time pad and AES encryption techniques with compression-type storage file formats (ORC, Parquet, and Avro). This approach aims to strike a balance between security and performance. However, it requires a balance between security and performance. The use of XOR-One-time pad and AES encryption creates a strong

dual layer of encryption, making it harder for malicious actors to access data. However, this method also increases computing overhead, potentially affecting system performance during data input and retrieval. Key management is crucial for ensuring the security of both encryption methods. Compression techniques like ORC, Parquet, and Avro optimize storage and query efficiency used to prevent pattern recognition attacks. However, compression can cause latency and impair query performance. High data volumes require strong encryption to prevent large-scale breaches, and scalability concerns arise. Algorithm selection is crucial for balancing security and performance. Strategies to optimize algorithms include optimizing versions of the XOR-One-time pad and AES algorithms, developing efficient key management, applying selective encryption, leveraging parallel processing, exploring hybrid encryption approaches, and using adaptive compression techniques. In conclusion, the proposed dual encryption method offers a solid solution for safeguarding data in HDFS, but trade-offs must be considered.

6. Case Study

A healthcare company manages sensitive patient data sets, including medical records, treatment history, and personal data, to ensure data security and privacy. To meet regulatory requirements like HIPAA, the company adopted a proposed encryption solution that combines compression-type storage file formats (ORC, Parquet, and Avro) with AES and XOR-One-time pad encryption algorithms in their Hadoop Distributed File System (HDFS). The company classified data based on sensitivity, using dual encryption for sensitive data like medical records and patient identities. The encryption method used XOR-One-time Pad and AES encryption for maximum secrecy. Data was compressed using ORC, Parquet, or Avro formats, reducing storage footprint and improving query performance. An advanced key management system was implemented to manage key production, distribution, and rotation. Hardware acceleration was used to optimize encryption and decryption procedures. The dual encryption method enhanced data security, demonstrated compliance with HIPAA, and improved performance. The combination of encryption and compression ensured efficient storage usage, reducing costs and expediting data recovery.

7. Limitations and Future Work

The proposed approach of integrating an XOR-One-time pad with AES encryption algorithms and compression-type storage file formats for data protection in HDFS has shown promising results. However, it faces several constraints, including performance complexity, key management complexity, scalability issues, compatibility concerns, security flaws, and maintenance and upkeep. The dual encryption process adds computational complexity, which can be problematic in resource-limited contexts or dealing with large datasets. The proposed encryption technologies may require major adjustments to current systems, and any flaws in implementation or integration can pose risks. Regular updates and maintenance are necessary to ensure the security and efficiency of the method. Future work should focus on optimizing encryption algorithms, enhancing key management solutions, conducting scalability testing, integrating encryption algorithms with developing technologies, conducting security audits and penetration testing, developing user-friendly tools, and exploring hybrid approaches that combine encryption and compression techniques. By addressing these constraints, the proposed approach can be further enhanced to provide a more robust, efficient, and secure solution for preserving sensitive data in HDFS.

8. Conclusion

This paper providesa proposed HUDFbased on an XOR-Onetimepad with AES encryption and decryptiontechniqueswith different compression-type storage file formats providingtwo levels of data securityin a Hadoop environment. In this paper, ourfocus wason encryptingsensitive data and whole file data within Hadoop. Hive has properties to read and write data from/into HDFS via a user-defined function (UDF) so the researcher does not face the problem of building complex map-reduce jobs. Additionally, hive uses different storage file formats and compressiontechniquesto store and move data easily so that the decreasesload time and improvesquery processing time of large data sets. The above approach used the ETL (Extract-Transform-Load) pipeline method based on HDFS to Hive.Compared to the 2DES and other methods, the above novel method results prove that the proposed methodology based onthe HUDF-ORC-Z file format is more beneficial andincreases the performance by 9-10%. It alsoprovides dual-level securitywithout compromising its performance. In a real-world scenario, our

main aim is to providethe security of sensitive data and the entire file data storage level and transit levelin the distributed environment.The suggested model's effectiveness was confirmed in terms of encryption time, decryption time, throughput time for the encryption and decryption process, and total map-reduce time for encryption and decryption. The outcomes confirm the effectiveness and dependability of the strategy in safeguarding private information in HDFS.

Statements and Declarations

- ☑ The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.
- ☑ The authors declare that no funds, grants, or other support were received during the preparation of this manuscript.
- ☑ The authors of this paper have directly participated in the planning, execution, or analysis of this study.
- ☑ The authors of this paper have read and approved the final manuscript.
- ☐ The authors declare the following financial interests/personal relationships which may be considered as potential competing interests:

References

Apache Hadoop 3.3.6: HDFS Architecture [Internet]. Available from: https://hadoop.apache.org/docs/stable/hadoop- project-dist/hadoop-hdfs/HdfsDesign.html

Algaradi, T., & Rama. (2021) A new encryption scheme for performance improvement in big data environment using Map Reduce. Journal of Engineering Science and Technology.

Bansal, Krati, et al. “Analyzing Performance of Apache Pig and Apache Hive with Hadoop.” Lecture notes in electrical engineering, 2018, pp. 41–51, doi:10.1007/978-981-13-1642-5_4.

Balusamy, B., R, N. a., Kadry, S., &Gandomi, a. H. (2021) Big data. John Wiley & Sons. Wiley.

Chen, C. L. Philip, and Chun-Yang Zhang. “Data-intensive applications, challenges, techniques and technologies: A survey on Big Data.” Information Sciences, vol. 275, Aug. 2014, pp. 314–47, doi:10.1016/j.ins.2014.01.015.

Chen, Min, et al. “Big Data: A Survey.” Journal on Special Topics in Mobile Networks and Applications/Mobile Networks and Applications, vol. 19, no. 2, Jan. 2014, pp. 171–209, doi:10.1007/s11036-013-0489-0.

Due. D (2018) Apache Hive Essentials.

Gandomi, A., and Murtaza H.. “Beyond the hype: Big data concepts, methods, and analytics.” International Journal of Information Management, vol. 35, no. 2, Apr. 2015, pp. 137–44, doi:10.1016/j.ijinfomgt.2014.10.007.

Garlasu, D., et al. A big data implementation based on Grid computing. Jan. 2013, doi:10.1109/roedunet.2013.6511732.

Gattoju, Saritha, and Nagalakshmi. “An Efficient Approach ForBig Data Security Based On HadoopSystem Using Cryptographic Techniques.” Indian Journal of Computer Science and Engineering, vol. 12, no. 4, Aug. 2021, pp. 1027–37, doi:10.21817/indjcse/2021/v12i4/211204132.

Gupta, M, and Rajendra K. D. “Fortified MapReduce Layer: Elevating Security and Privacy in Big Data.” ICST Transactions on Scalable Information Systems, Oct. 2023, doi:10.4108/eetsis.3859.

Hanand, A & H, R & R, F & A, F& Al-D, Ali & E, Ala'a & Ghaith, Yahya & Rateb, Saddam. (2022). The effects of big data, artificial intelligence, and business intelligence on e-learning and business performance: Evidence from Jordanian telecommunication firms. International Journal of Data and Network Science. 7. 35-40. 10.5267/j.ijdns.2022.12.009.

Hashem, I. A. T., et al. “The rise of ‘big data’ on cloud computing: Review and open research issues.” Information Systems, vol. 47, Jan. 2015, pp. 98–115, doi:10.1016/j.is.2014.07.006.

Iavich, M., &Kevanishvili, Z.(2018) Modified one time pad. ResearchGate. Scientific and Practical Cyber Security Journal (SPCSJ).

Jayapandian, N. “SECURING CLOUD DATA AGAINST CYBER-ATTACKS USING HYBRID AES WITH MHT ALGORITHM.” Computing, Dec. 2020, pp. 561–68, doi:10.47839/ijc.19.4.1989.

Kamaruzaman, Siti Hanisah, et al. “Design and Implementation of Data-at-Rest Encryption for Hadoop.” International Journal of Engineering & Technology, vol. 7, no. 2.15, Apr. 2018, p. 54, doi:10.14419/ijet.v7i2.15.11212.

Kaushik, Anju, and Vinod Kumar Srivastava. “Performance Analysis Of AES And DESede On The Sensitive Data Stored
In HDFS.” 2020 2nd International Conference on Advances in Computing, Communication Control and Networking (ICACCCN), Dec. 2020, doi:10.1109/icacccn51052.2020.9362755.
Khadji, Khoulji, &Kerkeb, M. (2023) Efficient Big Data security: Evaluating the performance of a proposed hybrid key
management algorithm using lightweight cryptography. Journal of Theoretical and Applied Information Technology.
Khafagy, Omar Helmy, et al. Hybrid-Key Stream Cipher Mechanism for Hadoop Distributed File System Security. Feb. 2020, doi:10.1109/itce48509.2020.9047775.
Mahmoud, Hadeer, et al. An approach for big data security based on Hadoop distributed file system. Feb. 2018,
doi:10.1109/itce.2018.8316608.
Menon R. (2014) Cloudera administration handbook.
Mohanraj, T., & R. Santhosh. (2022) Hybrid encryption algorithm for big data security in the Hadoop distributed file system. Computer-Assisted Methods in Engineering and Science.
Morán, Jesús, et al. Testing data transformations in MapReduce programs. Aug. 2015, doi:10.1145/2804322.2804326.
Moreno, Julio, et al. “Main Issues in Big Data Security.” Future Internet, vol. 8, no. 3, Sept. 2016, p. 44, doi:10.3390/fi8030044.
Naidu V. (2022) Performance enhancement using appropriate file formats in big data Hadoop ecosystem. International Research Journal of Engineering and Technology (IRJET).
Polato, Ivanilton, et al. “A comprehensive view of Hadoop research—A systematic literature review.” Journal of Network and Computer Applications, vol. 46, Nov. 2014, pp. 1–25, doi:10.1016/j.jnca.2014.07.022.
Priya S. (2019) Big data: analytics, technologies, and applications.
Ramya, P., and C. Sundar. “SecDedoop: Secure Deduplication with Access Control of Big Data in the HDFS/Hadoop Environment.” Big Data, vol. 8, no. 2, Apr. 2020, pp. 147–63, doi:10.1089/big.2019.0120.
Rihan, S., Khalid, & F. Oshman, S.(2015) A Performance Comparison of Encryption Algorithms AES and DES.
International Journal of Engineering Research & Technology, vol 4(12). International Journal of Engineering Research & Technology.
Rodríguez-Mazahua, Lisbeth, et al. “A general perspective of Big Data: applications, tools, challenges and trends.” the
Journal of Supercomputing/Journal of Supercomputing, vol. 72, no. 8, Aug. 2015, pp. 3073–113, doi:10.1007/s11227-015-1501-1.
Sonal Jain, Mohit Jain. (2019) Privacy-Preserving mining using data encryption scheme for Hadoop Ecosystem.
International Journal for Rapid Research in Engineering Technology & Applied Science.
Song, N. Youngho, et al. Design and implementation of HDFS data encryption scheme using ARIA algorithm on Hadoop. Feb. 2017, doi:10.1109/bigcomp.2017.7881720.
Teng, L., Li, H., Yin, S., & Sun, Y. (2020) A modified advanced encryption standard for data security. International Journal of Network Security.
Thuraisingham, Bhavani. Big Data Security and Privacy. Mar. 2015, doi:10.1145/2699026.2699136.
Van Rijmenam, M. Think Bigger.(2014) AMACOM
Wadhera, Shweta, et al. “A Systematic Review of Big data tools and Applications for developments.” 2021 2nd International Conference on Intelligent Engineering and Management (ICIEM),2021, doi:10.1109/iciem51511.2021.9445326.
Yaqoob, Ibrar, et al. “Big data: From beginning to future.” International Journal of Information Management, vol. 36,
no. 6,2016, pp. 1231–47, doi:10.1016/j.ijinfomgt.2016.07.009.
Zodpe, Harshali &Sapkal, Ashok. (2018). An Efficient AES Implementation Using FPGA with Enhanced Security Features. Journal of King Saud University - Engineering Sciences. doi:/0.1016/j.jksues.2018.07.002.