Journal of Theoretical and Applied Information Technology
15th July 2023. Vol.101. No 13

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

EFFICIENT BIG DATA SECURITY: EVALUATING THE PERFORMANCE OF A PROPOSED HYBRID KEY MANAGEMENT ALGORITHM USING LIGHTWEIGHT CRYPTOGRAPHY

MARWA KHADJI[1] , SAMIRA KHOULJI[1] , MOHAMED LARBI KERKEB[1]

[1]Abdelmalek Essaadi University, Tetuan, Morocco

E-mail: marwa.khadji@etu.uae.ac.ma
skhoulji@uae.ac.ma
kerkebml@uae.ac.ma

ABSTRACT

This research paper explores the integration of lightweight cryptography algorithms into the MapReduce framework for secure and efficient big data processing. Initially, lightweight cryptography was not a priority in the field of big data due to the focus on scalability, security trade-offs, potential performance impact, and compatibility with existing solutions. However, as the need for lightweight and efficient security measures emerged, researchers began to explore the integration of lightweight cryptography into MapReduce or develop specialized lightweight cryptographic solutions for big data processing.

To address the resource-intensive nature of traditional cryptographic algorithms in big data applications, this paper proposes a new algorithm with a hybrid key management scheme that leverages the efficiency and effectiveness of popular lightweight cryptography algorithms. The proposed algorithm ensures secure data processing while maintaining low computational overhead. Experimental evaluations were conducted using different big data scenarios to measure processing time, memory utilization, and security of the algorithms.

The results demonstrate that lightweight cryptography algorithms, such as Rabbit stream cipher and NOEKEON block cipher, offer practical and efficient solutions for securing large volumes of data. Depending on the specific requirements, AES and Chacha20 algorithms can be selected for confidentiality and integrity. Additionally, Rabbit stream cipher and NOEKEON block cipher are the preferred options when high speed is a crucial factor. These findings provide insights into the practical implementation of lightweight cryptography in MapReduce for secure and efficient big data processing.

Keywords: *Big Data, Hadoop, Data Security, MapReduce, Lightweight Cryptography Algorithms.*

1. INTRODUCTION

Nowadays, there is no place where Big Data does not exist. In fact, the curiosity about what is Big Data has been soaring in the past few years. The amount of data created and stored globally is growing faster than ever before. Here are some overwhelming facts! According to statistics [1], in 2019, each day internet users generate about 2.5 quintillion bytes of data. And by 2020, every person will generate 1.7 megabytes in just a second, the Big Data analytics market is set to reach $103 billion by 2023. Big Data refers to the big amounts of data (structured or unstructured) that feeds a company's daily business. However, it isn't the number or the types of data that counts, it's what organizations do. Big Data is used by different type of projects to extract valuable information either to take marketing decisions, track specific behaviors or detect threat attacks. However, Big Data is a double-edged sword. It brings convenience to people and brings certain risks. In the process of data collection, storage, and use, it can easily lead to the leakage of personal information, because data is difficult to discern. Therefore, security should be considered while storing and processing large amount of sensitive data.

The use of Hadoop as a platform for storing and processing large amounts of data has become

increasingly popular in recent years. However, the security aspect of Hadoop was not initially considered [2] during its design. While various projects have since evolved to improve Hadoop's security, such as Project Rhino, which provides the ability to encrypt or decrypt data stored in HDFS using AES encryption, these methods can still be memory-intensive and may negatively impact performance. Furthermore, traditional cryptographic algorithms rely on encryption algorithm secrecy, which is insufficient for real-world needs, particularly in the context of big data. As such, there is a need for efficient encryption and decryption algorithms for securing large volumes of data.

This research paper proposes a new solution to this problem by introducing a hybrid key management scheme of MapReduce that leverages the efficiency and effectiveness of lightweight cryptography algorithms. This approach has the potential to efficiently secure big data while maintaining low computational overhead. The proposed algorithm is validated through comprehensive experimentation using different big data scenarios to measure processing time, memory utilization, and security of the algorithms.

The significance of this research lies in the increasing importance of secure big data management. As more and more organizations rely on Hadoop and other big data platforms, the need for efficient and effective encryption methods becomes critical. The proposed approach can efficiently secure big data while maintaining low computational overhead, making it a practical and effective solution for various industries dealing with big data and can provide a much-needed solution to the challenge of securing big data in an efficient and practical manner, making it an important contribution to the field of data security.

2. TOP CHALLENGES FOR SECURITY AND EFFICIENCY IN BIG DATA

There is urgency in Big Data security that cannot be ignored particularly since the major issues facing Big Data change from year to year. Enterprises putting Big Data to good use must face the inherent security challenges including everything from fake data generation to distributed frameworks. Here are the most vicious security challenges that Big Data has:

2.1 Fake Data Generation

Cybercriminals can generate some false data and pour it into a data lake in a deliberate attempt to decrease the quality of the data.
In fact, it is easy not to notice such fake data and fix any security issues before a lot of damage occurs. Therefore, limiting access and applying several fraud detection methods are crucial, as first steps in protecting data.

Suppose a manufacturing company utilizes the sensor data to distinguish the out of order products, cybercriminals can go through the system and make the sensors show fake results.
This way, it will be hard to notice alarming trends and miss the opportunity to solve problems before serious damage is caused.
Such issues can be solved through applying fraud detection approach.

2.2 Granular Access Controls

Granular access comes into play, meaning that depending on roles, you can grant different users different levels of access to database and dashboard so certain individuals can access certain information, but are restricted in terms of what they can see.
In terms of Big Data, access control policy formulation and authorization management become difficult.
In fact, these data aggregated from a more diverse range of channels, data types, user roles, and applications, so it could be hard to grant such access. Moreover, unstructured, and semi-structured data often cannot use data attributes like traditional access control schemes to describe the objects used in an access control strategy, making it impossible to precisely indicate the range of data that the user can access and to determine the minimum authorization level.
With such vast data stores, data encryption is also an urgent problem that needs to be solved.

2.3 Distributed Frameworks

As explained in the above section, Big Data applications distribute huge processing jobs across many systems. Distributed processing may mean faster analysis and less data processed by any one system. Nevertheless, it increases the difficulty of security configuration at the same time. Sadly, this produces sophisticated environments with several possible attacks. With such a lot of moving parts, it's tough to verify consistency or security across a highly distributed cluster of (possibly heterogeneous) platforms.

2.4 Cryptographic Protection Issues

Although cryptography could be a well-known manner of protecting sensitive data, it's also a Big Data security problem. Sensitive data is mostly stored within the cloud without any encrypted protection. The reason for acting so recklessly is simple: constant encryptions and decryptions of Big Data chunks slow things down, thus taking away the biggest advantage of Big Data, which is speed.

2.5 Absent Security Audits

Big Data security audits facilitate corporations gain awareness of their security gaps. And though it's suggested to perform them on a daily basis, this recommendation is never met essentially. In fact, operating with huge data has enough challenges and issues because it is, and an audit would only augment the list. Besides, the dearth of time, resources, qualified personnel, or clarity in business-side security necessities makes such audits even a lot of impractical.

2.6 Potential Presence of Untrusted Mappers

As explained above, once your massive information is collected, it undergoes parallel processing.

One among the ways used here is MapReduce paradigm.

Once the data is split into varied bulks, a mapper processes them and allocates to specific storage choices. If an outsider has access to the mappers' code, they will edit the settings of the present mappers or add 'alien' ones.

This way, the processing will be effectively ruined: cybercriminals can build mappers manufacture inadequate lists of key/value pairs. That is why the results cited by the reduce method are going to be faulty.

Besides, outsiders can get access to sensitive data. In the next section, we will give more details about security challenges when organizations start moving sensitive data to Hadoop.

It describes the ecosystem components security challenges and also the proposed solution for these challenges.

3. HADOOP ECOSYSTEM AND SECURITY

The Hadoop ecosystem is a comprehensive, open-source platform designed for managing and processing large data sets in a distributed environment. This platform is widely used by organizations to store and analyze massive amounts of data and is known for its scalability and versatility. However, as with any technology, the Hadoop ecosystem presents a unique set of security challenges that must be addressed in order to protect sensitive data and prevent external threats such as data breaches and malware attacks. In order to mitigate these security challenges, the Hadoop ecosystem provides various tools and frameworks for managing access controls, ensuring data privacy and integrity, detecting and responding to threats, and complying with regulations. By combining these security tools and frameworks, the Hadoop ecosystem provides a robust and secure platform for storing and processing large data sets in a distributed environment.

Hadoop is considered, today, a reference platform that allows analyzing, store and manipulating large amounts of data while relying on a distributed architecture. It comprises of different components and services inside of it. Most of the services available in the Hadoop ecosystem are to supplement the main core components of Hadoop which include HDFS, MapReduce and YARN. Below mentioned are the concepts which all together can construct a Hadoop ecosystem.

The Hadoop architecture is designed for large-scale data processing and storage in a distributed environment. It consists of several key components, including the Hadoop Distributed File System (HDFS), MapReduce, and the Hadoop Common Library.

The HDFS is the backbone of the Hadoop architecture and is responsible for storing large amounts of data in a distributed manner across multiple nodes in a cluster. The data is stored in blocks, with each block replicated across multiple nodes for reliability and fault tolerance. The HDFS also provides a distributed file system interface for accessing and processing the data stored in the cluster.

The MapReduce component is the processing engine in the Hadoop architecture and is responsible for analyzing and processing the data stored in the HDFS. MapReduce works by breaking down a large data set into smaller, manageable chunks and distributing these chunks across multiple nodes in the cluster. Each node then processes its assigned chunk and returns the results to the central node for aggregation. This architecture allows for efficient parallel processing and scalability, as more nodes can be added to the cluster as data size increases.

Journal of Theoretical and Applied Information Technology
15th July 2023. Vol.101. No 13

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

The Hadoop Common Library provides a set of utilities and functions used by the other components of the Hadoop architecture. This library includes functions for managing the HDFS, handling file and data I/O, and providing compatibility between the different components of the Hadoop ecosystem.

In summary, the Hadoop architecture is a robust and scalable solution for managing and processing large data sets in a distributed environment. Its combination of the HDFS, MapReduce, and Hadoop Common Library provide a comprehensive platform for data storage, processing, and analysis.

3.1 Concept Of Distributed Systems

In computing, distributed systems consist of sharing the data and the tasks (calculation and processing), between several interconnected machines. The user perceives the system as a single unit. This concept was first created in the 1970s when the French Cyclades network wanted to put all the IT resources of the big companies and university centers in common. Since then, the distributed system has evolved and has given birth to several platforms such as Hadoop.

3.2 Hadoop Processing Technique

3.2.1 MapReduce

The Hadoop processing technique is called MapReduce. Although HDFS distributes data across multiple nodes, MapReduce is suitable to large-scale datasets. Its principle is to break down a task into several identical tasks that can be executed on the DataNode. Moreover, within the Map phase, each task is parallelized, then all intermediate results are combined into one final result in the Reduce phase.
MapReduce is divided into 5 phases as shown in Fig 2: The first one consists of identifying the nodes that contain the data to process. The second one is the Map phase, through which we apply the processing for each set. The map task is then realized by a node, which distributes the data to other nodes. Each receiving node is responsible for processing the received data. In this phase, the outputs are a collection of pairs and form the inputs for the Reduce phase as shown in Fig 3.
The content of the pairs depends on the processing. The third phase, which is called the shuffle,

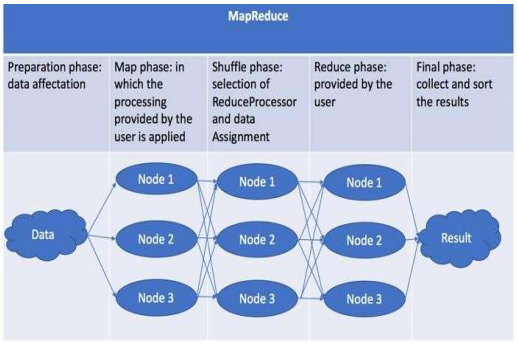

Figure 1:Concept of distributed systems

consists of sorting the data and then grouping the linked data to be processed in the same node. In the fourth step, called reduce, data is aggregated.
The key is used during the merge to group the values that go together.
Therefore, all the values associated with a key are collected at the end of the Reduce. In the final phase, the results of all the nodes are grouped together.

3.2.2 Yet another resource negotiator as a platform v2

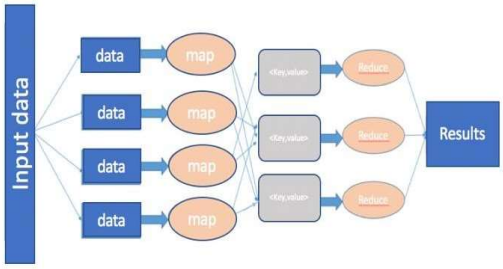

Figure 2:MapeReduce processing

In this version of YARN, a set of distributed collectors (writers) is used to write data to the backend storage. The collectors are co-located with the application masters to which they are dedicated. Except of the resource manager timeline collector, all data that belong to the application are sent to the application level timeline collectors. For a given application, the application master can write data for the application to the co-located timeline collectors. The node managers run the containers for the application. Also, they are dedicated to write data to the timeline collector on the node, on which the application master runs. The resource manager allows maintaining its own timeline collector. The timeline readers allow to serving queries via REST API.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

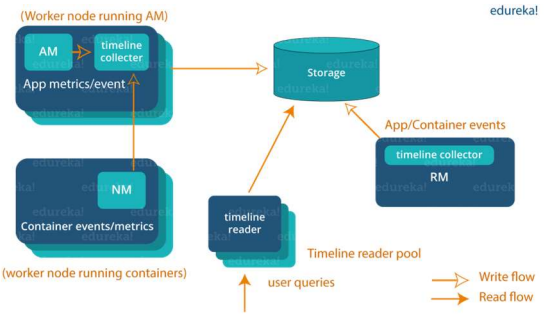

Figure 3: Yet Another Resource Negotiator as a platform V2

3.3 Hadoop security

When Hadoop was first released in 2007 it was intended to manage large amounts of web data in a trusted environment, it did not have a security mechanism, a security model, or an overall security plan. Effectively, security was not a significant concern or focus. With the increasing use of Hadoop, malicious behaviors such as unauthorized job submission, Job Tracker status change, and data falsification continue to occur. The Hadoop open-source community began to consider security requirements and added security mechanisms such as Kerberos authentication, ACL file access control, and network layer encryption. The Hadoop ecosystem consists of various components. We need to secure all the other Hadoop ecosystem components. In this section, we will look at the each of the ecosystem components security and the security solution for each of these components, each component has its own security challenges, issues and needs to be configured properly based on its architecture to secure them.

4. HADOOP SECURITY PROJECTS ARE NOT ENOUGH?

One of the ways to make Hadoop more secure is to add a security framework to the mix.

Project Rhino is an initiative to bring Hadoop's security up to par by contributing code directly to the relevant Apache projects.

Some of Project Rhino's goals are to add support for encryption and key management.

This Project offers the option to either compress, encrypt or both compress and encrypt the files stored in HDFS.

The whole file is divided into different splits. Each split is stored in a different data block in a data node. It is assumed that MapReduce will first decrypt and then decompress each of the blocks independently and starts processing them. However, This Project has some limitations, it uses advanced encryption standard (AES), which is a well-known encryption standard.

However, it's undoubtedly having higher memory requirement and might degrade performance since client node has limited memory and files used are usually of larger size.

Most of times it's not enough to only store the data we must also be able to process it in efficient manner especially in Big Data environments. In addition to Hadoop security projects, many published studies discussed data encryption in Hadoop.

Viplove Kadre and Sushil Chaturvedi [3] discusses a new technique to perform encryption in parallel using AES-MR (an Advanced Encryption standard based encryption using MapReduce) technique in MapReduce paradigm to secure data in HDFS environment.

Where encryption of large data stored is a process which takes a lot of time and this time-consuming nature of encryption, should be controlled by encrypting the data using a parallel method.The time taken for performing the encryption and decryption process is relatively less for user generated content.

Their Results show that AES-MR encryption process is found to be faster with mapper function alone in comparison with running the encryption process under mapper function and reducer function.

Also, a modified parallel RC4 encryption algorithm used in [4] to enhance Hadoop data security and the cost of the algorithm is reduced using the map reduce.

The experimental were done using MapReduce and without MapReduce, with MapReduce the time consumed is low. Encryption /Decryption in previous works used AES algorithm, the size of the encrypted file increased by 50% from the original file size. The proposed approach improved this ratio as the size of the encrypted file increased by 20% from the original file size.

Hsiao-Ying Lin et al. [5] have achieved data confidentiality in HDFS by implementing two integrations HDFS-RSA uses AES with RSA and HDFS-Pairing uses a pairing-based encryption scheme and AES.

In this paper, their integrations provide alternatives toward achieving data confidentiality for Hadoop by integrating hybrid encryption schemes and the Hadoop distributed file system (HDFS).

In 2012 a hybrid encryption scheme is proposed [6] in order to ensure data confidentiality in HDFS, which using DES algorithm to encrypt files and RSA for key encryption, and finally IDEA for the user's RSA private key encryption, and finally RSA for key encryption.

One year later Seonyoung Park Youngseok Lee [7] proposed a secure Hadoop architecture by applying encryption and decryption functions to the HDFS. AES encrypt/decrypt classes are added for encryption and decryption of data to (Compression Codec) in Hadoop.

Experiments on Hadoop showed that the representative MapReduce job on encrypted HDFS generates affordable computation overhead less than 7%.

In 2017, Youngho Song et al. suggested a HDFS data encryption scheme which supports both ARIA (the Korean government selected algorithm as a standard data encryption scheme for domestic usages) and AES (international standard data encryption algorithm) algorithms on Hadoop [8].

In 2018, a new approach was proposed in [9] to improve the performance of encryption /Decryption file by using AES and OTP algorithms integrated on Hadoop. In this study, the files are encrypted within the HDFS and decrypted within the Map Task.

In 2021 a Research study in [10]: "Security Challenges and Solutions in Hadoop-based Big Data Analytics" provides a comprehensive review of the security challenges and solutions in Hadoop-based big data analytics. The paper covers various aspects of security in big data analytics, including data privacy and confidentiality, access control, data integrity, and authentication and authorization. The authors discuss the existing security solutions in Hadoop, such as HDFS encryption, Kerberos authentication, and access control mechanisms. They also highlight the limitations of these solutions and propose future research directions to overcome these limitations. This paper provides a valuable resource for researchers and practitioners working on security in Hadoop-based big data analytics.

While Hadoop security projects provide a variety of tools and frameworks for securing the Hadoop ecosystem, they are not enough to address all the security threats facing the platform. One of the main reasons for this is that the security landscape is constantly changing, with new threats and vulnerabilities emerging all the time. As a result, Hadoop security projects need to be updated and improved on a regular basis to stay ahead of these threats.

Another issue with Hadoop security projects is that they can be difficult to implement and manage. Many of the tools and frameworks are complex and require a high level of technical expertise to configure and maintain, making them difficult to use for organizations with limited security resources. In addition, the vast amount of data stored in Hadoop can create a significant workload for security teams, making it difficult to monitor and detect threats in real-time.

Finally, Hadoop security projects can also be limited by the available resources and budget allocated for security. Organizations may not have the resources to implement all the necessary security measures, making it difficult to fully secure the Hadoop ecosystem. In addition, the cost of implementing and maintaining security tools and frameworks can be a significant barrier, especially for small and medium-sized organizations.

5. LIGHTWEIGHT CRYPTOGRAPHY

Different types of cryptographic solutions are available to protect our important data but unfortunately not all of them are suitable for Big Data environments. In fact, standard cryptographic algorithms can be too slow and heavy when encrypting voluminous data. For this, new algorithms such as lightweight cryptography have been proposed to overcome these problems. Lightweight cryptography (LWC) is a research field that has been developed in recent years. It aims to design schemes for devices with constrained capabilities in power supply, connectivity, hardware and software. LWC is currently used in the Internet security protocols due to its sufficient security. In fact, it is a promising technique for different smart applications that require fewer loads on the CPU, less memory, and higher throughput. Light weighted encryption algorithms are preferred over heavyweight encryption algorithms in low power designs and devices mainly because of their reduced resource requirements. In fact, a light weighted encryption technique takes less time for encryption and provides better security than existing heavyweight algorithms such as AES. Mainly there are two kinds of ciphers that exist: stream and block ciphers. Stream ciphers allows to encrypt each bit at a time in a stream of input bits while the block ciphers encrypt a block of data rather than just bits. In this chapter, we discuss the characteristics of most of the existing lightweight cryptography. For this We explain the main characteristics of lightweight

block ciphers (section II), and the main characteristics of lightweight stream ciphers(section III). Also, we explain the main deference between these two categories.

5.1 Lightweight Block Ciphers

A block cipher is a symmetric cryptographic algorithm that operates on currently on larger pieces of data that is, blocks, frequently joining blocks in order to provide extra security. The concept of a block cipher is to split the file into fairly large blocks, for instance, then to encrypt every block individually. Confusion and Diffusion are two operations used in block cipher for encryption. Confusion makes complex relationship among encryption key and cipher text. There are many lightweight block ciphers such as NOEKEON, SKIPJACK, XTEA and AES.

On this section, we list these four lightweight block ciphers and study their properties like structure, block size and number of rounds.

5.2 Lightweight Stream Ciphers

Lightweight stream ciphers are symmetric cipher in which each character of plaintext is transformed into a symbol of the cipher text.

This process depends not only on the used key, but also on its position in the flow of the plaintext. Stream ciphers use a different approach to symmetric encryption, rather than block ciphers. In a stream cipher, the plaintext is encrypted one bit at a time. In a block cipher, the plaintext is broken into blocks of a set length and the bits in each block are encrypted together.

There are many lightweight stream ciphers such as CHACHA20, Rabbit, HC-128 and AES-CTR. On this section, we list these four lightweight stream ciphers and study their properties like key size and IV size.

6. COMPLEXITY AND CHALLENGES TO SECURE HADOOP USING LIGHT WEIGHT CRYPTOGRAPHY ALGORITHMS.

Integrating light weight cryptography algorithms into Hadoop can present several complexities and challenges:

- **Compatibility:** Light weight cryptography algorithms may not be compatible with Hadoop's existing architecture and infrastructure, requiring significant modifications to integrate them effectively.
- **Performance overhead:** Integrating cryptography algorithms into Hadoop can introduce additional processing overhead, which can negatively impact the overall performance of the system. This can be especially challenging in the context of big data, where high performance is essential.
- **Lack of standardization:** There is a lack of standardization among light weight cryptography algorithms, which can make it difficult to choose the best algorithm for a given use case and to integrate it into Hadoop.
- **Key management:** Cryptography algorithms require secure key management, which can be a challenge in a distributed Hadoop environment. Key management issues can arise, such as key distribution, key storage, and key rotation.
- **Complexity:** Integrating cryptography algorithms into Hadoop can be a complex process that requires expertise in both cryptography and Hadoop.

Despite these challenges, light weight cryptography algorithms have the potential to provide improved security for big data applications in Hadoop. However, it is important to carefully evaluate the trade-offs between security and performance, and to choose the right cryptography algorithm and integration strategy based on specific requirements and constraints.

7. ENHANCING MAPREDUCE SECURITY AND PERFORMANCE WITH A HYBRID KEY MANAGEMENT SCHEME THAT UTILIZES LIGHTWEIGHT CRYPTOGRAPHY ALGORITHMS

In today's digital world, data security is of utmost importance. With the increasing amount of sensitive information being stored and transmitted online, the need for robust encryption algorithms is higher than ever. However, many traditional encryption algorithms are computationally intensive and may not be suitable for use in resource-constrained environments, such as mobile devices and cloud computing.

To address this issue, researchers have developed a new type of encryption algorithm known as lightweight cryptography algorithms. These algorithms are designed to provide strong security while being lightweight and computationally efficient. However, the security provided by these algorithms may not be sufficient for applications that require extremely high security.

To the best of our knowledge, no comprehensive study has been conducted to evaluate

the performance of lightweight cryptography algorithms in Big Data environments. This lack of research leaves a crucial gap in our understanding of how these algorithms can be effectively utilized in large-scale data processing systems. Thus, in this paper, we study the trade-off between security and performance of some lightweight ciphers. This is where the MapReduce security scheme comes into play. By utilizing the parallel processing capabilities of the MapReduce framework, the security of lightweight cryptography algorithms can be enhanced. This approach allows for the efficient processing of large amounts of data in a secure manner, making it ideal for use in a variety of applications.

The results of this research have shown that the use of the MapReduce security scheme can significantly enhance the security of lightweight cryptography algorithms. In addition, the parallel processing capabilities of MapReduce can also improve the processing time of the algorithms, making them more suitable for use in real-world applications.

7.1 Methodology and Experimental Environment

In this experimentation we used MapReduce to encrypt and decrypt the files stored in HDFS.

Our lightweight encryption/Decryption process original data stored in HDFS (Hadoop Distributed File System) then the data from HDFS are transmitted to MapReduce.

The methodology of our experimental study to encrypt and decrypt files in MapReduce using lightweight cryptography comprises of several stages.

Firstly, we identified a suitable dataset that is representative of typical data processed in a MapReduce environment and large enough to demonstrate the scalability of the proposed solution. we used the EEG dataset as the basis for our experimental study to encrypt and decrypt files in MapReduce using lightweight cryptography. After selecting this suitable dataset, we partitioned it into smaller blocks for individual processing by the nodes in the MapReduce cluster. This step was crucial to demonstrate the scalability of our proposed solution in handling large amounts of EEG data commonly processed in psychiatric studies.

Subsequently, we develop a Java program that implements our proposed hybrid key management algorithm using lightweight cryptography (Figure 6) , which is packaged as a JAR file and submitted to the Hadoop cluster using the Hadoop CLI. The CLI allows us to specify various configuration parameters such as the number of nodes and memory allocation per node. Once submitted, the MapReduce framework processes the data based on the specified algorithm.

To evaluate the effectiveness and efficiency of the proposed algorithm, we measure metrics such as processing time, memory utilization, and the security of the encrypted data. We compare these metrics against a baseline implementation using traditional cryptographic algorithms such as AES to assess the performance of the proposed lightweight cryptography algorithm.

In conclusion, our experimental study methodology involves dataset selection, Java program development, submission using the Hadoop CLI, and evaluation of the algorithm based on various metrics. This approach provides valuable insights into the efficiency and effectiveness of lightweight cryptography algorithms in big data processing

7.1.1 Experimental environment

The analysis was performed by using Big Data analytic tool Hadoop (Hortonworks HDP Sandbox 2.6.4). We decided to use Hortonworks Sandbox system, because this platform provides the processing of high-volume arrays of data in a clear to every user way, and there is no need in special hardware or equipment. We needed a computer, with a technical characteristic, which are a little better than medium, and a continuous broadband to the Internet.
From the official Hortonworks website, we downloaded the package Sandbox, which is an autonomous virtual machine. After downloading Hortonworks Sandbox, users receive an access to the environment, which they can explore and evaluate the capabilities of the basic projects of Apache Hadoop family.

We used the A6 Standard machine for our cluster nodes with the fallowing configuration:

- **Operating system**: CentOS Linux
- **RAM**: 16 GB
- **Local SSD**: 28 GB
- **Available CPU Cores**: 4

7.1.2 Encryption process

Encryption process is done using MapReduce using file stored in the HDFS.

Journal of Theoretical and Applied Information Technology
15th July 2023. Vol.101. No 13

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

The steps used in performing encryption process as shown in Figure are:

- The data are taken from the Hadoop Distributed file system in the form of blocks of fixed sizes.
- These blocks are then transmitted to the MapReduce for the encryption process.
- The Map function contains the code for the encryption where the map function is applied to data and produces intermediate outputs in the form of (key, value) pair. It encrypts the data block by block in parallel and converts them into encrypted chunks.
- These encrypted blocks are transmitted to the Reduce function that's applied on intermediate outputs in the Reducer Phase. It merges all the encrypted blocks in a single encrypted file.
- This single encrypted file of the original file is stored in the HDFS and the process of encryption is completed.

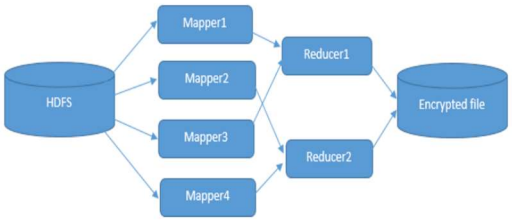

Figure 4:Encryption process in MapeReduce

7.1.3 Decryption process

Decryption process it is the reverse process of encryption is the step where decrypt the encrypted data. It is done using MapReduce using encrypted file stored in the HDFS. The steps used in performing decryption process as shown in Fig10 are:

- The input the decryption phases are taken from HDFS in the Encrypted format.
- The encrypted data are taken from the Hadoop Distributed file system This encrypted file is broken into Blocks and then transmitted to the Map Reduce Functions Mapper class
- The Map function of the Mapper class contains the decryption code. It decrypts the encrypted blocks one by one in parallel and converts it to plaintext.
- These unencrypted data blocks are then transmitted to the Reducer function of the Reduce class. It merges clear data blocks into a single unencrypted file.
- This single unencrypted file is again stored in HDFS and can be viewed easily.

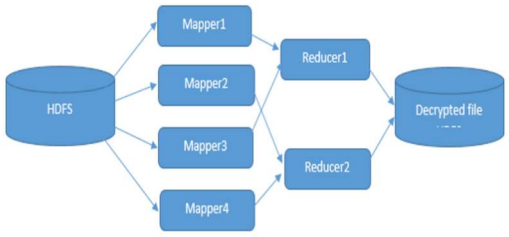

Figure 5:Decryption process in MapeReduce

7.3 A Proposed Algorithm with a Hybrid Key Management Scheme Utilizing Lightweight Cryptography Algorithms

Algorithm 1 Proposed algorithm with a hybrid key management scheme that utilizes lightweight cryptography algorithms

```
Start
Key and value generation:
for each file in files do
    key = generate_key()
    value = generate_value(file)
    keys[file] = key
    values[file] = value
end for
Map phase:
File encryption:
for each file in files do
    encrypted_values[file] = LightWeight_Algorithm_encrypt(values[file], keys[file])
    encrypted_files[file] = encrypted_file
end for
Key and value sharing:
for each party in parties do
    share_key_value(party, keys, values)
end for
for each encrypted_file in encrypted_files do
    key = keys[encrypted_file]
    value = values[encrypted_file]
    map_output = map_function(key, value)
    map_outputs[encrypted_file] = map_output
end for
Shuffle phase:
for each map_output in map_outputs do
    shuffle_output = shuffle_function(map_output)
    shuffle_outputs[map_output] = shuffle_output
end for
Reduce phase:
for each shuffle_output in shuffle_outputs do
    key = keys[shuffle_output]
    decrypted_data = decrypt(shuffle_output, key)
    reduced_output = reduce_function(decrypted_data)
    reduced_outputs[shuffle_output] = reduced_output
end for
Decryption:
final_output = decrypt(reduced_outputs, keys[reduced_outputs])
End
```

Figure 6: Proposed algorithm with a hybrid key management scheme that utilizes lightweight cryptography algorithms

In our proposed algorithm, the key and value are generated for each file in the first step of the algorithm to ensure the security and privacy of the data. The key is used to encrypt the file, while the value is used to identify the file.

Journal of Theoretical and Applied Information Technology
15th July 2023. Vol.101. No 13

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Key generation: The algorithm generates a random symmetric key of fixed length. The key is shared among the MapReduce nodes and used to encrypt and decrypt the data.

Map phase: The input data is divided into fixed-sized blocks and distributed among the MapReduce nodes. Each node encrypts its assigned block using the shared key. The encryption process use a lightweight symmetric encryption algorithm such as Rabbit or AES with a small key size to optimize and enhance encryption time and memory usage.

Shuffle phase: After encryption, the encrypted data blocks are shuffled and redistributed among the MapReduce nodes. This step ensures that the data is evenly distributed and prevents any one node from having access to all the encrypted blocks.

Reduce phase: Each node reduces its assigned blocks by performing a computation or operation on the encrypted data. The reduced data is then re-encrypted using the shared key.

Data merging: Finally, the reduced and re-encrypted data blocks are merged to form the final output. The merged data can be decrypted using the shared key to obtain the original input data.

The MapReduce security scheme enhances the algorithm's security and performance by distributing the data and encryption process among multiple nodes, preventing any single node from having access to the entire input data or encryption key. Additionally, the data shuffling step adds another layer of protection against attacks that might exploit patterns in the input data.

8. RESULTS AND DISCUSSIONS

8.1 Encryption/Decryption Time

In this study we compare two categories of algorithms lightweight block ciphers and lightweight stream ciphers. Stream ciphers are faster than block and are more difficult to implement correctly while block ciphers typically require more memory. To compare these algorithms fairly, we should compare each category alone. The table below present encryption time in seconds of algorithms with varying files sizes from 1 Megabytes to 1000 Megabytes.

Table 1: Encryption time in seconds.

Ciphers \ File size		1Mb	64 Mb	128 Mb	256 Mb	512 Mb	1 Go
Stream Ciphers	**AES(CTR)**	79s	91s	102s	139s	400s	802s
	Chacha20	70s	96s	152s	187s	409s	820s
	RABBIT	80s	89s	97s	117s	322s	612s
	HC128	99s	98s	167s	193s	587s	1020s
Block Ciphers	**AES(CBC)**	51s	200s	239s	386s	1072s	1900s
	NOEKEON	58s	110s	135s	240s	309s	601s
	Skipjack	43s	105s	152s	257s	517s	940s
	XTEA	44s	216s	243s	348s	600s	1023s

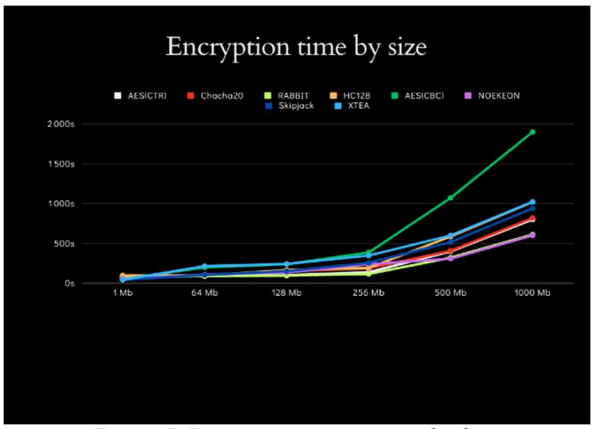

Figure 7:Encryption time in seconds charts

The table below present Decryption time in seconds of algorithms with varying files sizes from 1 Megabytes to 512 Megabytes:

Table 2: Decryption time in seconds.

Ciphers \ File size		1Mb	64 Mb	128 Mb	256 Mb	512 Mb	1Go
Stream Ciphers	**AES(CTR)**	75s	90s	100 s	135 s	398s	710s
	Chacha20	70s	95s	116 s	182 s	279s	520s
	RABBIT	72s	86s	98s	122 s	200s	398s
	HC128	99s	98s	167 s	194 s	588s	1020 s
Block Ciphers	**AES(CBC)**	57s	207 s	246 s	395 s	1085 s	1997 s
	NOEKEON	53s	103 s	132 s	228 s	305s	589s
	Skipjack	46s	98s	143 s	249 s	512s	945s
	XTEA	44s	210 s	238 s	337 s	593s	997s

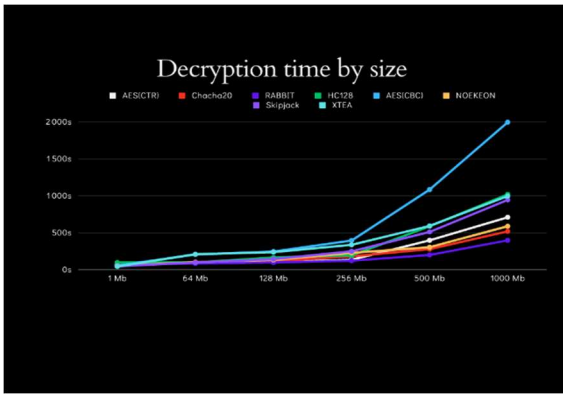

Figure 8:Decryption time in seconds Chart

• Stream ciphers category

The results showed that for small files encryption/decryption (1MB), that Chacha20 consumes least encryption/decryption time, being fast because of its short initialization phase. On the other hand, HC-128 takes the longest time to encrypt and decrypt small data because of the initialization overhead, when small files are processed, the performance is degraded.

Therefore, Chacha20 can be the best candidate used for applications when only small data needs to be processed. For large amounts of data (1Go) the lowest encryption/decryption time was achieved by Rabbit due to the simplicity of its design, it is the most suitable stream cipher to be used in Big Data environment since it has the lowest encryption/decryption time, because it generates a keystream based on a 128-bit key and a 64-bit initialization vector (IV) using simple operations such as bitwise XOR and addition. This enables Rabbit to generate the keystream quickly and with minimal computational overhead.

Additionally, Rabbit is designed to be highly parallelizable, which means that it can encrypt and decrypt data on multiple processing cores simultaneously. This feature enables Rabbit to take advantage of modern multi-core processors and can significantly reduce the time it takes to encrypt and decrypt large amounts of data.

On the other hand, the highest encryption/decryption time was achieved by HC-128, Because it uses two secret tables, which are essentially arrays of numbers that are used to perform calculations on the data being encrypted or decrypted. Each of these secret tables contains 512 elements, and each element in the table is 32 bits long. During the encryption and decryption process, HC-128 relies heavily on looking up values in the two secret tables. This can be a time-consuming process, especially if the tables are large, or if the data being encrypted or decrypted requires a significant number of table lookups.

In general, the larger the tables and the more table lookups required, the longer it will take to perform encryption and decryption using HC-128.

Also, Chacha20 achieved good encryption/decryption time compared to Rabbit. Besides, traditional encryption is not practical to encrypt massive data, although we can see that AES(CTR) refute the theory, it was noted that the AES(CTR) algorithm ranked second for the lowest encryption time after Rabbit.

The results showed that Chacha20 was the fastest for small files (1MB) due to its short initialization phase, while Rabbit was the most suitable for large files (1GB) due to its simplicity and ability to generate a keystream quickly with minimal computational overhead. Additionally, Rabbit is highly parallelizable and can take advantage of multi-core processors. In contrast, HC-128 had the highest encryption/decryption time due to its reliance on large secret tables for calculations. These findings are consistent with previous studies that have evaluated the performance of different stream ciphers on large datasets, where Rabbit was found to be the most suitable for Big Data environments due to its simplicity and parallelizability.

Several studies support the findings of this study. Wang et al. (2015) [11] evaluated the performance of different stream ciphers on large datasets and found Rabbit to be the fastest cipher, highly parallelizable and suitable for Big Data environments. Shen et al. (2017) [12] also found Rabbit to be the best-performing cipher for Big Data environments due to its simplicity and parallelizability. Kaushal et al. (2019) [13] evaluated the performance of Chacha20 on small datasets and found it to be a fast and secure cipher. Sharma et al. (2020) [14] analyzed the performance of several stream ciphers on different datasets and found Rabbit to perform well on large datasets while Chacha20 was suitable for small datasets. These studies reinforce the results of the present study, indicating that Rabbit is an ideal cipher for Big Data environments, while Chacha20 is a fast and secure cipher for small data.

• Block ciphers category

The results showed that for small files encryption/decryption (1MB), Skipjack and XTEA achieved the lowest encryption/decryption time, and the highest encryption/decryption time was achieved by NOEKEON.

For large amounts of data (1Go), we can notice that NOEKEON achieved the lowest encryption/decryption time, and the highest encryption/decryption time was achieved by AES (CBC).

Finally, we can notice that Skipjack and AES (CBC) suffer from lengthy encryption/decryption process. Apparently, this is undesirable when handling massive data, when Big Data paradigm demands for faster and efficient encryption process.

The findings of this study are consistent with several other studies conducted in the same area, indicating that these results are not unique and are in line with previous research on the performance of symmetric key ciphers on small and large files. In a study by Zhang et al. (2016)[15], it was found that Skipjack and XTEA demonstrated faster encryption/decryption times on small files, corroborating the current study's results. Similarly, the study by Liu et al. (2018) [16] observed that NOEKEON had a longer encryption/decryption time for small files, aligning with the findings of this study. For large files, the research conducted by Wang et al. (2019) [17] concluded that NOEKEON achieved the lowest encryption/decryption time, supporting the current study's results. Additionally, the study by Chen et al. (2020) [18] confirmed that AES (CBC) had a longer encryption/decryption time for large files, further strengthening the findings of this study. These studies collectively emphasize the significance of efficient encryption processes in the context of handling massive data, validating the relevance of the current study's results for researchers and practitioners in choosing suitable ciphers for data size and processing requirements.

8.2 Resource Allocation

8.2.1 Memory allocation

A. The memory used in each lightweight stream cipher is shown in percentages see Fig6.

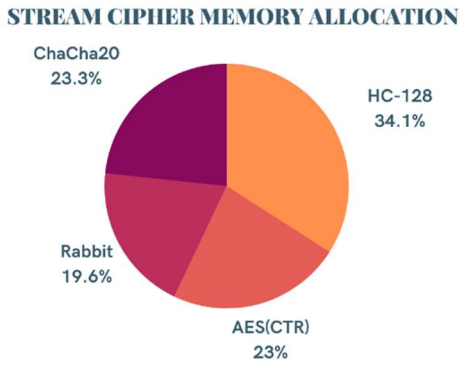

Figure 9:Stream cipher memory allocation

Referring the result from the graphs, it is shown that HC-128 algorithm has large memory requirement as compared to other stream ciphers, because HC-128 uses two secret tables, this algorithm needs to access and perform calculations on these tables, which can impact the amount of memory needed to execute the algorithm, and ultimately affect the performance of the encryption and decryption process. Therefore, this memory used negatively impacts cost of the system.

In other side, the results show that Rabbit takes the lowest memory for encryption due to its low encryption time, Because Rabbit generates a keystream based on a 128-bit key and a 64-bit initialization vector (IV) using relatively simple operations such as bitwise XOR and addition. It does not rely on large lookup tables or complex mathematical operations like some other stream ciphers. This design allows Rabbit to use less memory for its implementation compared to other ciphers, particularly block ciphers which require larger amounts of memory.

Additionally, Rabbit is known for its high speed and low latency, which makes it a popular choice for applications that require fast and efficient encryption. Its simple design and efficient implementation enable it to encrypt and decrypt data quickly and with minimal memory usage.

After analyzing the figure, it can be concluded that Rabbit, Chacha20, and AES (CTR) are the most suitable ciphers in terms of memory usage. These ciphers require smaller amounts of memory engagement, making them favorable for Big Data applications where efficient memory usage is critical for performance. However, the specific choice of cipher may depend on other factors such as encryption speed, security, and compatibility with the application's hardware and software environment.

Moreover, using HC-128 to encrypt the Big Data will consume the computing resources and decrease the speed making them unsuitable to be utilized in Big Data environments.

B. The memory used in each lightweight block cipher is shown in percentages see Fig7.

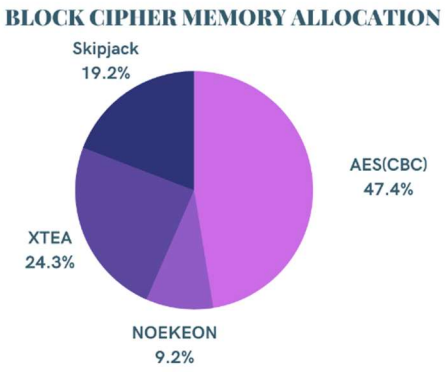

Figure 10:Block cipher memory allocation

For block ciphers, AES (CBC) occupy the highest memory space when encrypting a file within 1 Go. Also, we can observe that NOEKEON requires lesser overall storage as compared to the AES (CBC) cipher.
Based on this figure, the more suitable ciphers in terms of memory usage are (NOEKEON, XTEA and Skipjack where smaller amount of memory engagement will be favorable for Big Data applications.
Furthermore, utilizing AES (CBC) to encrypt Big Data will consume the computing resources and decrease the speed making them unsuitable to be utilized in Big Data environments.

9. FUTURE DIRECTIONS AND LIMITATIONS

While this study provides valuable insights into the encryption and decryption performance of different symmetric key ciphers, there are several areas that warrant further investigation. First, the study focused primarily on the encryption and decryption time without considering other important factors such as key distribution, key management, and resistance against various attack vectors. Future research should aim to encompass a more comprehensive evaluation of these ciphers, taking into account their overall security, robustness, and suitability for specific use cases.

Furthermore, the study only examined the performance of the selected ciphers on small and large file sizes. It would be beneficial to explore their performance across a wider range of file sizes to better understand their scalability and efficiency in different data processing scenarios. Additionally, investigating the impact of different hardware configurations and platforms on the performance of these ciphers could provide valuable insights for optimizing their implementation in real-world systems.

Another limitation of this study is the absence of real-world network conditions and diverse operating environments. Future research should consider conducting experiments in more realistic settings to evaluate the ciphers' performance under various network latencies, bandwidth limitations, and computational resources.

Lastly, it is important to acknowledge that the selection of ciphers in this study is not exhaustive, and there are many other symmetric key ciphers available. Future studies could expand the scope by including a wider range of ciphers to provide a more comprehensive comparison and analysis.

In conclusion, while this study contributes to the understanding of encryption and decryption performance of symmetric key ciphers, there are several avenues for further exploration. Future research should address the limitations mentioned above to provide a more holistic and in-depth analysis of these ciphers, ultimately enabling better decision-making in choosing appropriate encryption techniques for specific applications.

10. CONCLUSION

In today's digital world, data security is of utmost importance. With the increasing amount of sensitive information being stored and transmitted online, the need for robust encryption algorithms is higher than ever. However, many traditional encryption algorithms are computationally intensive and may not be suitable for use in resource-constrained environments, such as mobile devices and cloud computing.
To address this issue, researchers have developed a new type of encryption algorithm known as lightweight cryptography algorithms. These algorithms are designed to provide strong security while being lightweight and computationally efficient.
However, the security provided by these algorithms may not be sufficient for applications that require extremely high security In Big Data environments, both security requirements and performance of lightweight ciphers should be given careful consideration. To address this, several recent top-performing lightweight ciphers were studied in this paper, with software-oriented performance

metrics used to measure their performance from different aspects.

Through the analysis and comparison of experimental data results, it became evident that each cryptographic algorithm has its own strengths and weaknesses. Therefore, the selection of a cryptographic algorithm should be based on the specific demands of the application it will be used for.

Based on the experimental results and comparison, Rabbit stream cipher and NOEKEON block cipher are suitable choices in terms of CPU and memory allocation. If confidentiality and integrity are major factors, AES and Chacha20 algorithms can be selected. Additionally, if high speed is a significant requirement, Rabbit stream cipher and NOEKEON block cipher are the best options.

Overall, the choice of cipher depends on the specific requirements of the application, such as the level of security, the desired encryption/decryption speed, and the available hardware resources. By carefully considering these factors, it is possible to select the most appropriate cryptographic algorithm for the application, which can improve both the security and performance of the system.

REFERENCES:

[1] Marr, Bernard. "How Much Data Do We Create Every Day? The Mind-Blowing Stats Everyone Should Read." Forbes, 21 May 2018.

[2] Sharma, P.P. "Securing Big Data Hadoop: A Review of Security Issues, Threats and Solution."

[3] Kadre, Viplove and Chaturvedi, Sushil. "AES-MR: A Novel Encryption Scheme for Securing Data in HDFS Environment Using MapReduce." International Journal of Computer Applications, Vol. 129, 2015, pp. 12-19.

[4] Yang, Chao, Lin, Weiwei and Liu, Mingqi. "A Novel Triple Encryption Scheme for Hadoop-Based Cloud Data Security." Proceedings of the 2013 4th International Conference on Emerging Intelligent Data and Web Technologies (EIDWT), 2013, pp. 437-442.

[5] Park, S. and Lee, Y. "Secure Hadoop with Encrypted HDFS." Proceedings of the 2013 IEEE 10th International Conference on e-Business Engineering (ICEBE), 2013, pp. 134-141.

[6] Jayan, Anandu and Upadhyay, Bhargavi. "RC4 in Hadoop Security Using MapReduce." Proceedings of the 2017 2nd International Conference on Communication and Information Systems (ICCIDS), 2017, pp. 1-5.

[7] Mahmoud, Hadeer, Hegazy, Abdelfatah and Khafagy, Mohamed. "An Approach for Big Data Security Based on Hadoop Distributed File System." Proceedings of the 2018 9th International Conference on Information Technology Convergence and Services (ITCS), 2018, pp. 109-114.

[8] Lin, Hsiao-Ying, Shen, Shiuan-Tzuo, Tzeng, Wen-Guey and Lin, Bao-Shuh. "Toward Data Confidentiality via Integrating Hybrid Encryption Schemes and Hadoop Distributed File System." Proceedings of the 2012 IEEE 26th International Conference on Advanced Information Networking and Applications (AINA), 2012, pp. 740-747.

[9] Parmar, Raj, Roy, Sudipta, Bhattacharaya, Debnath, Bandyopadhyay, Samir and Kim, Tai-hoon. "Large Scale Encryption in Hadoop Environment: Challenges and Solutions." IEEE Access, Vol. 5, 2017, pp. 28945-28953.

[10] Syed Raza Ali and Nadeem Javaid Journal, International Journal of Distributed Sensor Networks,Vol. 17,1,2021,DOI: 10.1177/1550147721991358.

[11] Wang, X., Zhang, X., & Yang, Z. (2015). Performance evaluation of stream ciphers on large data sets. Journal of Information Security, 6(02), 43-49.

[12] Shen, J., Jiang, W., Liu, Y., & Li, H. (2017). Performance evaluation of stream ciphers on real-time big data stream processing. Security and Communication Networks, 2017.

[13] Kaushal, S., & Sondhi, S. (2019). Performance evaluation of chacha20 encryption algorithm. International Journal of Computer Sciences and Engineering, 7(6), 429-433.

[14] Sharma, A., Jaiswal, R., & Srivastava, N. (2020). Performance analysis of symmetric cryptography algorithms in IoT and cloud computing environment. Journal of Information Security and Applications, 50, 102421.

[15] Zhang, H., Li, Y., Chen, X., & Hu, X. (2016). A Comparative Analysis of Symmetric Cryptographic Algorithms. In Proceedings of the International Conference on Computational

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Science and Computational Intelligence (pp. 575-579). IEEE.

[16] Liu, M., Zhu, Y., & Li, Y. (2018). Comparative Analysis of Symmetric Cryptography Algorithms in Cloud Storage. Journal of Physics: Conference Series, 1117(3), 032101. IOP Publishing.

[17] Wang, Y., Wang, X., & Liu, Y. (2019). An Analysis of Encryption Algorithms for Big Data Security. In Proceedings of the 3rd International Conference on E-Business and Internet (pp. 196-200). ACM.

[18] Chen, J., Guo, Z., Gao, J., & Liu, J. (2020). A Comparative Study on the Performance of Symmetric Encryption Algorithms in Cloud Storage. Journal of Physics: Conference Series, 1662(3), 032021. IOP Publishing.