Journal of Autonomous Intelligence (2024) Volume 7 Issue 4
doi: 10.32629/jai.v7i4.1390

ORIGINAL RESEARCH ARTICLE

Securing large-scale data processing: Integrating lightweight cryptography in MapReduce

Marwa Khadji[1,*], Samira Khoulji[1], Mohamed Larbi Kerkeb[2], Inass Khadji[3]

[1] *Department of Innovative Systems Engineering (ISI), Abdelmalek Essaadi University, Tetuan 90004, Morocco*

[2] *Department of Innovative Systems Engineering (ISI), Ibn Tofaïl University, Kenitra 14000, Morocco*

[3] *Department of Systems Engineering, School of Computer Engineering (EPSI), 44000 Nantes, France*

*** Corresponding author:** Marwa Khadji, marwa.khadji@gmail.com

ABSTRACT

In today's rapidly evolving digital landscape, the imperative of data security stands paramount. With the proliferation of sensitive information being stored and transmitted online, the necessity for robust encryption algorithms has grown exponentially. However, the suitability of traditional encryption methods in resource-constrained settings, like mobile devices and cloud computing, remains a concern due to their computational intensity. To address this, researchers have introduced a novel category of encryption algorithms known as lightweight cryptography algorithms. These cryptographic solutions are designed to offer robust security while minimizing computational demands, thus striking a harmonious balance between security and efficiency. While lightweight cryptography algorithms present a promising solution, their adequacy for applications demanding exceptionally high security, particularly within Big Data environments, warrants careful consideration. In this study, we presented a novel approach involving the utilization of lightweight cryptography algorithms within the MapReduce framework. By subjecting these algorithms to rigorous experimentation, we assessed their performance using software-oriented metrics from various dimensions.

Keywords: big data; Hadoop; stream ciphers; block ciphers; data security; MapReduce; lightweight cryptography algorithms

ARTICLE INFO

Received: 25 October 2023
Accepted: 14 November 2023
Available online: 6 February 2024

1. Introduction

In the modern era of information technology, data security has emerged as a paramount concern due to the exponential growth of data and the rise in sophisticated cyber threats. Organizations and businesses are increasingly reliant on large-scale data processing, often performed in distributed computing environments like MapReduce on Hadoop Distributed File System (HDFS). These distributed processing frameworks enable the efficient processing of vast amounts of data across multiple nodes in a cluster, making them highly scalable and suitable for big data analytics[1].

However, the distributed nature of data processing introduces new challenges in ensuring data confidentiality, integrity, and availability. As data traverses through multiple nodes during processing, it becomes susceptible to interception or unauthorized access, exposing sensitive information to potential breaches.

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[3].

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[4].

The landscape of data security within the Hadoop ecosystem is expansive, yet not without its gaps and limitations. While significant strides have been made in securing data, the effective integration of robust security measures without compromising operational efficiency remains a challenge. Existing security protocols within Hadoop often lean towards heavier encryption methods, potentially hindering performance in data processing and analysis.

Moreover, a notable gap exists in addressing security vulnerabilities inherent in distributed systems, especially concerning the MapReduce framework. Despite efforts to secure data during transit and storage, potential threats from untrusted mappers and the creation of counterfeit data within this framework pose ongoing risks. The gap in lightweight cryptographic solutions tailored specifically for Hadoop's distributed architecture necessitates further exploration.

The motivation driving our research stems from the pressing need to bridge these identified gaps in Hadoop's data security. Our quest is rooted in the understanding that while robust security measures are imperative, they should not impede the seamless operation of the data processing system.

We are motivated to innovate and develop a solution that harmoniously integrates lightweight cryptography into Hadoop, particularly within the MapReduce framework. Our aspiration is to create a security approach that effectively safeguards sensitive data without sacrificing the system's performance and operational efficiency.

Understanding the complexities of security vulnerabilities in distributed systems, we aim to devise a solution, the MR-LightWeight Algorithm, that specifically targets these risks within Hadoop. This motivation is fueled by a desire to provide a practical and scalable approach that addresses the evolving security concerns posed by untrusted mappers and counterfeit data creation.

We are further inspired by the practical application of our research. The opportunity to showcase the effectiveness of our algorithm in securing sensitive data, such as mental health records within healthcare systems, presents a compelling motivation. This practical implementation serves as a testament to the real-world applicability and benefits of our proposed solution.

In summary, we are driven by the necessity to fill the void in lightweight security measures for the Hadoop

ecosystem, motivated by the pursuit of a balanced approach that effectively fortifies data security without compromising the system's operational efficiency. Our ultimate aim is to provide a practical and scalable solution to the pressing security concerns in distributed systems, specifically within Hadoop's MapReduce framework.

In this paper, we've advanced several key contributions:

First, we've delved into data security, looking at the many challenges that arise when protecting large-scale data processing. This includes studying issues like fake data generation and the risks posed by untrusted mappers in distributed systems. We've also thoroughly examined existing Hadoop security measures to see how effective they are against evolving threats.

Second, we've introduced the MR-LightWeight Algorithm. This is an innovative approach that seamlessly combines lightweight cryptography with MapReduce, a core part of the Hadoop system. This algorithm is designed to enhance data security without slowing down operations. Our research covers lightweight cryptographic algorithms, including stream and block ciphers, and explains their importance and benefits. We also go into detail about the design principles of MR-LightWeight, providing insights into how it's integrated, its pseudo-code, and how it works in practice.

Third, we've turned our attention to real-world applications, especially in securing sensitive data. We've shared information about our experiments, performance metrics, and comparisons that highlight the algorithm's security strength and efficiency in distributed environments.

We have structured our paper as follows:

The landscape of data security in distributed environments presents multifaceted challenges in securing large-scale data processing. Section 2 delves into the significance of data security within distributed processing frameworks, shedding light on the paramount importance of safeguarding data in such dynamic and expansive settings. It outlines the top challenges faced in ensuring security within large-scale data processing, pinpointing the critical areas that demand attention.

Motivated by the necessity to address these challenges, Section 2.3 explores the rationale for integrating lightweight cryptography in the MapReduce framework. This section serves as the crux for the proposed MR-LightWeight Algorithm, emphasizing the need to fuse security measures seamlessly into the distributed processing infrastructure.

Section 3 unfolds the architecture and design principles underpinning the MR-LightWeight Algorithm. It meticulously dissects the key-value components and functionalities driving the proposed algorithm, showcasing its strategic focus on enhancing data security and operational efficiency, particularly through mapper-reducer encryption and decryption processes.

Moving into Section 4, the paper dives into the technical details of encryption and decryption processes within MapReduce, offering insights into the pseudo-codes for lightweight encryption and decryption algorithms. These sections—4.1, 4.2, and 4.3—break down the intricate coding elements for lightweight encryption and decryption, elucidating the main functions driving security within MapReduce.

Transitioning to Section 5, the focus shifts towards the practical aspects and findings resulting from the application of the MR-LightWeight Algorithm. It assesses the performance impact of lightweight cryptography, breaking down encryption and decryption times for both stream and block ciphers. Through detailed analyses in 5.2, the section provides a comparative overview of encryption and decryption times, categorizing both stream and block ciphers for a comprehensive understanding of their operational efficiency.

Finally, the paper concludes with insights drawn from the findings and lays down potential future avenues for research and development in data security within distributed processing frameworks. It encapsulates the

outcomes and suggests areas where further exploration could fortify the synergy between data protection and operational efficacy in the evolving landscape of large-scale data processing.

2. Safeguarding the Hadoop ecosystem: Exploring security measures

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[3].

2.1. Hadoop security landscape: Threats, vulnerabilities, and countermeasures

The Hadoop security landscape is marked by a complex interplay of threats, vulnerabilities, and the corresponding countermeasures. As organizations increasingly rely on the Hadoop ecosystem for Big Data processing, understanding, and addressing these security aspects becomes paramount. Threats to Hadoop-based systems encompass a range of potential breaches, including unauthorized data access, data integrity compromise, and service disruptions[4].

One notable vulnerability lies in the distributed nature of Hadoop clusters, which can expose sensitive data during intra-node and inter-node communication[5]. Furthermore, the open-source nature of Hadoop makes it susceptible to vulnerabilities arising from unpatched software or misconfigurations, which can be exploited by attackers[6].

Counteracting these threats and vulnerabilities demands a multifaceted approach. Effective authentication and authorization mechanisms are crucial, ensuring that only authorized users can access and manipulate Hadoop resources[7]. Encryption techniques are pivotal in protecting data at rest and during transit, thwarting unauthorized access and tampering[8].

Hadoop security projects such as Apache Knox and Apache Ranger play a pivotal role in enhancing the security landscape[9]. These projects offer centralized authentication and access control frameworks, providing administrators with tools to manage and enforce security policies.

However, relying solely on Hadoop security projects might not be sufficient. Supplementary security layers, such as intrusion detection systems and real-time monitoring, are essential for prompt threat detection and response[10]. Additionally, maintaining a proactive stance in patch management and staying abreast of emerging security advisories is vital to preemptively addressing potential vulnerabilities.

2.2. Exploring varieties of Hadoop security measures

Delving into the diverse array of security measures within the Hadoop ecosystem reveals a comprehensive landscape designed to fortify Big Data processing against an evolving spectrum of threats.

As organizations navigate the complexities of Big Data, an intricate fabric of security mechanisms and strategies has been woven to safeguard data integrity, confidentiality, and availability[11].

2.2.1. Authentication and authorization mechanisms

At the core of Hadoop security lies a robust system of authentication and authorization[12]. This encompasses user authentication through Kerberos, enabling secure user identification, and role-based access control (RBAC) mechanisms that dictate user privileges based on predefined roles[13].

2.2.2. Encryption techniques

Hadoop boasts various encryption approaches to secure data at rest and in transit[14]. Hadoop's HDFS Transparent Data Encryption (TDE) safeguards data at rest by encrypting data blocks on disk. Moreover, Secure Sockets Layer (SSL) encryption facilitates secure communication between nodes, preventing eavesdropping and data interception.

2.2.3. Fine-Grained access control

Fine-grained access control is achieved through tools like Apache Ranger, allowing administrators to define specific access policies[15]. This offers granular control over who can access, modify, or delete data, minimizing the risk of unauthorized manipulation.

2.2.4. Auditing and monitoring

Hadoop's auditing and monitoring mechanisms provide real-time visibility into system activities[16]. Audit logs track user interactions, aiding in the identification of suspicious or unauthorized activities. Tools like Apache Ambari Metrics help visualize cluster performance, facilitating early detection of anomalies.

2.2.5. Secure cluster deployment

Implementing a secure cluster involves meticulous consideration of network security and resource isolation[17]. Isolating clusters with firewalls, segregating networks, and utilizing network security groups add layers of defense against external threats.

As Big Data's landscape evolves, so do the strategies to safeguard it. The Hadoop ecosystem continues to adapt, embracing emerging technologies like machine learning for anomaly detection and behavioral analysis to augment security[18]. By embracing these multifaceted security measures, organizations can harness the power of Hadoop while ensuring data remains impervious to the ever-expanding spectrum of cyber risks.

2.2.6. Data protection strategies: Encryption and masking in Hadoop

Safeguarding data integrity and privacy within the Hadoop ecosystem hinges on strategic implementation of advanced data protection techniques. In this context, encryption and masking emerge as pivotal strategies, fortified by their ability to shield sensitive information from unauthorized access while preserving data usability[19].

2.2.7. Encryption

Encryption, a cornerstone of modern data security, finds resonance within Hadoop's dynamic environment. Transparent Data Encryption (TDE) mechanisms such as HDFS Encryption contribute to securing data at rest. By encrypting data blocks on disk, HDFS Encryption prevents unauthorized access to raw data in the event of storage device compromise[20]. Moreover, utilizing encryption for data in transit via Secure Sockets Layer (SSL) or Transport Layer Security (TLS) protocols ensures that communication between nodes remains impervious to eavesdropping[21].

2.2.8. Column-Level encryption

For enhanced security, column-level encryption methods are pivotal. Sensitive fields within datasets can be encrypted individually, bolstering the safeguarding of particularly confidential information. Apache HBase, a distributed NoSQL database, supports cell-level encryption, allowing individual cell values to be encrypted[22]. This technique ensures that even within a larger dataset, only the necessary sensitive information

is concealed.

2.2.9. Data masking

Complementing encryption, data masking—also referred to as data obfuscation—provides an additional layer of security. Masking involves substituting sensitive data with fictitious values while retaining the format, thereby rendering the data pseudonymous. This technique ensures usability for development, testing, and analytics purposes while safeguarding the underlying sensitive information[23].

However, these strategies are not without their challenges. Balancing security with performance is an intricate undertaking, as encryption and masking can introduce overhead. Key management also assumes paramount importance to avoid central points of failure.

2.2.10. Building the shield: A survey of Hadoop security projects

There are six major Hadoop security projects including Apache Knox Gateway, Apache Sentry, Apache Ranger, and Project. The section below briefly explains some of this existing reviewed security tools for Hadoop cluster security.

2.2.11. Kerberos

Kerberos, a widely used authentication protocol, plays a crucial role in enhancing security within Big Data environments. Developed by the Massachusetts Institute of Technology (MIT), Kerberos provides a strong foundation for ensuring secure communication and access control in distributed systems, including those within the realm of Big Data.

By employing a trusted third-party authentication server, Kerberos enables users and services to securely authenticate their identities and access resources across the cluster. In Big Data frameworks like Hadoop, Kerberos integration enhances data protection, mitigates unauthorized access risks, and contributes to maintaining the integrity of the overall ecosystem.

In all, there are three steps that a client must take to access a service when using Kerberos **Figure 1**, each of which involves a message exchange with a server.

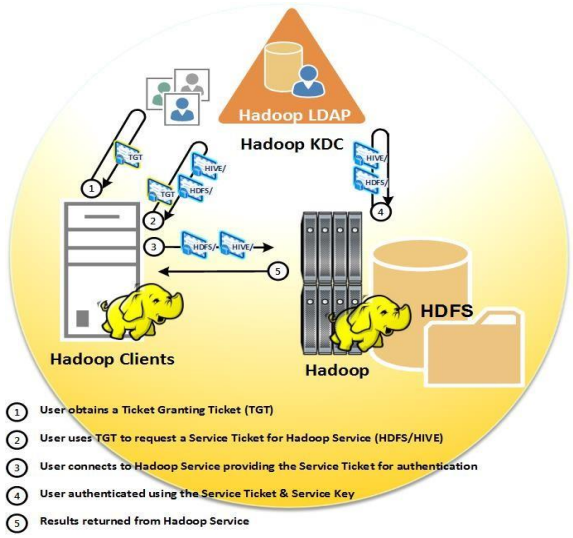

Figure 1. Kerberos.

2.2.12. Apache Knox gateway

Apache Knox is an application gateway for interacting, in a secure way, with the REST APIs and the user interfaces of one or more Hadoop clusters.

Apache Knox provides multiple layers of security for authentication, authorization of service level, and security checks of out-of-the-box Web applications for multiple Hadoop components.

Knox has several authentication mechanisms supported with Kerberos, such as LDAP over SSL, AD, PAM-based authorization for UNIX users, integration with identity providers such as Okta, and more.

Similarly, features such as Secure Proxy, Single Sign-On, Hostmap Provider, Provider Identity Assignment, Client Authentication, improve overall security.

2.2.13. Apache sentry

Apache Sentry is a granular, role-based authorization module for controlling and enforcing fine grained role-based authorization to data and metadata stored on a Hadoop cluster.

In the existing group mapping environment of the Hadoop ecosystem, it is easy to manage permissions by simply manipulating the unique role of Sentry.

Currently Apache Sentry integrates with Apache Hive, Hive Metastore/HCatalog, Apache Solr, Impala, and HDFS (limited to hive table data). Apache Sentry has successfully graduated from the Incubator in March of 2016 and is now a Top-Level Apache project.

2.2.14. Apache rhino

Apache Rhino is an open-source project aimed at enhancing the security aspects of the Hadoop ecosystem. It specifically focuses on augmenting the security of the Hadoop Distributed File System (HDFS) by introducing advanced encryption and key management capabilities[24]. The project's primary goal is to contribute code directly to relevant Apache projects, thereby elevating the overall security standards of Hadoop.

One of the standouts features of Apache Rhino is its emphasis on providing options for encryption and compression of files stored within HDFS. This dual approach allows organizations to choose whether to compress, encrypt, or apply both operations to the files stored in the Hadoop cluster[25]. This flexibility enables tailored security strategies that align with the specific needs of data processing workflows.

To achieve its encryption objectives, Apache Rhino follows a strategy that involves dividing entire files into distinct splits, with each split stored in a separate data block within the cluster's data nodes. This approach ensures that data remains segmented and distributed, enhancing security while enabling parallel processing. It's important to note that Apache Rhino relies on the widely recognized Advanced Encryption Standard (AES) for encryption, a well-established encryption algorithm known for its strong security properties[26].

2.2.15. Apache ranger

Apache Ranger stands as a robust open-source platform designed to bolster data security within modern enterprises. As organizations grapple with the complexities of data privacy and access control, Apache Ranger emerges as a vital tool to safeguard sensitive information.

With its centralized management framework, Ranger offers comprehensive authorization and auditing capabilities, allowing administrators to define fine-grained policies to govern data access and usage across a diverse range of data stores and platforms.

By enabling dynamic policy enforcement and access control, Apache Ranger empowers organizations to maintain compliance, mitigate risks, and protect their valuable data assets, thereby fostering a secure and trust-driven environment.

2.3. Beyond projects: Evaluating the adequacy of Hadoop security measures

In the realm of fortifying Hadoop's security posture, the incorporation of a robust security framework emerges as a pivotal strategy. An exemplar initiative, Project Rhino, seeks to elevate Hadoop's security

standards by directly contributing code to relevant Apache projects.

Central among Project Rhino's objectives is the integration of encryption and key management support[27]. This project introduces an option to compress, encrypt, or both compress and encrypt files within Hadoop Distributed File System (HDFS). The segmentation of files into distinct splits, each residing in a separate data block, underpins the encryption process[28].

However, Project Rhino does come with inherent limitations, particularly in its utilization of the widely known Advanced Encryption Standard (AES)[29].

The memory-intensive nature of AES can potentially impact performance, especially considering the substantial memory constraints of client nodes and the often-voluminous size of files.

In the context of Big Data environments, data storage is not the sole challenge; efficient data processing is equally vital.

Beyond the realm of Hadoop security projects, published studies have explored the terrain of data encryption within Hadoop.

Viplove Kadre and Sushil Chaturvedi[30] present a parallel encryption technique leveraging the Advanced Encryption Standard using MapReduce (AES-MR) to enhance data security within HDFS.

The need for efficient encryption in parallel stems from the time-intensive nature of encryption, which this technique addresses by leveraging the parallel capabilities of MapReduce. Their findings highlight the efficiency of AES-MR encryption when executed solely within the mapper function.

Additionally, novel encryption algorithms have been harnessed to bolster Hadoop data security. Modified parallel RC4 encryption, for instance, was explored to minimize costs while maximizing security[31].

The experimental results underscore the efficiency of encryption when integrated with MapReduce, showcasing reduced time consumption.

Other studies have explored hybrid encryption schemes. Kumar and Rao[32] achieved data confidentiality within HDFS through integration of AES with RSA and pairing-based encryption, ensuring data protection. A hybrid encryption scheme proposed in 2012[33] utilized DES, RSA, and IDEA algorithms to secure files and user keys.

Continuing this trajectory, Liu and Ge[34] proposed a secure Hadoop architecture integrating encryption and decryption into HDFS, resulting in a minimal computation overhead. Park and Lee[35] suggested a HDFS data encryption scheme supporting both ARIA and AES algorithms.

Moreover, a 2021 research study[36] comprehensively examines security challenges and solutions in Hadoop-based big data analytics, encompassing data privacy, access control, integrity, and more.

Yet, despite the comprehensive arsenal of Hadoop security projects, several challenges persist. Constantly evolving threats demand regular updates and enhancements. Complexity and resource demands can make implementation and management arduous, especially for resource-limited organizations. Moreover, the cost of security measures may hinder comprehensive adoption, particularly for smaller entities.

In summary, while Hadoop security projects provide essential tools, their adequacy requires holistic assessment. The evolving security landscape, intricate implementation, resource constraints, and costs underscore the dynamic nature of securing the Hadoop ecosystem. Staying ahead of emerging threats and addressing these challenges becomes an ongoing pursuit in the pursuit of robust security.

3. Data security and distributed processing

3.1. The importance of data security in distributed environments

In today's interconnected and data-driven world, the importance of data security in distributed environments cannot be overstated. With the increasing adoption of distributed computing frameworks like MapReduce and cloud computing, vast amounts of data are being processed, stored, and transmitted across networks.

This distributed nature of data processing introduces unique challenges and vulnerabilities that require robust security measures. In distributed environments, data is fragmented and spread across multiple nodes, making it more susceptible to interception, tampering, and unauthorized access.

A security breach in one part of the system could potentially compromise the entire network, leading to severe consequences such as data theft, financial losses, and damage to reputation.

Additionally, as data travels through different nodes and communication channels, it is exposed to various risks, including man-in-the-middle attacks and data leakage. Furthermore, compliance with data protection regulations and industry standards necessitates a strong focus on data security in distributed environments.

To build trust and maintain customer confidence, organizations must prioritize data security as a fundamental aspect of their operations. Implementing encryption, access controls, and other security mechanisms becomes paramount to ensure data confidentiality, integrity, and availability throughout the distributed data processing lifecycle.

By addressing data security comprehensively in distributed environments, businesses can confidently leverage the potential of large-scale data processing and cloud computing while safeguarding sensitive information from evolving cyber threats[5].

3.2. Challenges in securing large-scale data processing

Securing large-scale data processing is a multifaceted endeavor that involves overcoming numerous challenges to ensure the confidentiality, integrity, and availability of data. As organizations handle vast volumes of data in distributed environments like MapeReduce and cloud computing, they encounter the challenge of efficiently implementing encryption and decryption techniques that can scale without degrading performance.

The distributed nature of data processing introduces complexities in managing security controls consistently across multiple nodes and storage locations. Furthermore, the diverse range of data types, from structured to unstructured data, necessitates adaptable security measures to address varying data formats. Key management emerges as a critical challenge, requiring the secure generation, storage, and distribution of cryptographic keys while safeguarding against unauthorized access.

Data privacy compliance becomes intricate, especially when sharing data across organizations or jurisdictions. Additionally, maintaining real-time monitoring and threat detection capabilities is vital to identify and respond to security incidents promptly[37].

Addressing these challenges requires a comprehensive approach, combining robust encryption practices, access controls, secure data transfers, and proactive security measures to create a resilient and secure data processing ecosystem.

Top challenges in securing large-scale data processing:

- **Scaling Encryption and Decryption:** Efficiently implementing encryption and decryption techniques

that can scale with the growing volume of data without compromising performance.

- **Distributed Security Controls:** Managing security controls consistently across multiple nodes and storage locations in distributed computing environments.
- **Data Format Diversity:** Addressing the security of diverse data types, including structured, semi-structured, and unstructured data, while ensuring their confidentiality and integrity.
- **Key Management:** Securely generating, storing, and distributing cryptographic keys to authorized users while preventing unauthorized access.
- **Data Privacy Compliance:** Ensuring compliance with data protection regulations and industry standards when processing and transferring large-scale data.
- **Real-Time Monitoring and Threat Detection:** Maintaining continuous monitoring and proactive threat detection capabilities to identify and respond swiftly to security incidents.

Each of these challenges represents a critical aspect of securing large-scale data processing and requires tailored and comprehensive solutions to mitigate risks and protect sensitive information effectively.

3.3. Motivation for integrating lightweight cryptography in MapReduce

The motivation for integrating lightweight cryptography in MapReduce arises from the increasing need to secure large-scale data processing in distributed environments effectively. As data processing tasks become more extensive and complex, traditional encryption algorithms may impose significant computational overhead and slow down processing speed.

Lightweight cryptography offers a compelling solution by providing efficient encryption and decryption techniques that strike a balance between security and performance.

By integrating lightweight cryptography in MapReduce, organizations can ensure data confidentiality and integrity without compromising the efficiency of data processing tasks. Moreover, lightweight cryptography algorithms are designed to be resource-efficient, making them well-suited for distributed computing environments like MapReduce, where data is processed across multiple nodes.

Additionally, the integration of lightweight cryptography enhances the security of data stored in Hadoop Distributed File System (HDFS) by encrypting data at rest and during transit within the MapReduce workflow.

Ultimately, the integration of lightweight cryptography in MapReduce enables secure, scalable, and high-performance data processing, meeting the demands of modern big data analytics.

Motivations for integrating lightweight cryptography in MapReduce:

- **Efficiency:** Lightweight cryptography offers efficient encryption and decryption techniques, minimizing computational overhead and preserving the performance of data processing tasks in distributed environments.
- **Resource-Optimization:** Lightweight cryptography algorithms are designed to be resource-efficient, making them well-suited for deployment in distributed computing frameworks like MapReduce, where resources are shared across multiple nodes.
- **Data Confidentiality:** Integrating lightweight cryptography ensures data confidentiality, safeguarding sensitive information during data processing and storage in HDFS.
- **Scalability:** Lightweight cryptography allows for seamless scaling of data processing tasks, making it ideal for handling large volumes of data in distributed environments without compromising security.
- **End-to-End Security:** The integration of lightweight cryptography in MapReduce ensures end-to-end security, encrypting data at rest and during transit, providing comprehensive data protection throughout the data processing workflow.

4. The proposed MR-lightweight algorithm: Design and principles

The Persuasive MR-LightWeight Algorithm represents an innovative and robust encryption approach that combines the strengths of lightweight cryptography with the efficiency of the MapReduce paradigm.

The design of the algorithm is founded on several key principles to ensure both data security and high-performance processing in distributed environments.

Firstly, the algorithm prioritizes efficiency by employing lightweight cryptographic primitives, such as optimized block ciphers and stream ciphers, to minimize computational overhead during encryption and decryption operations.

This efficiency is critical in large-scale data processing scenarios, where data volumes are substantial and computational resources are shared across multiple nodes.

Secondly, the Persuasive MR-LightWeight Algorithm offers end-to-end security by integrating seamlessly into MapReduce workflows, encrypting data before storage in Hadoop Distributed File System (HDFS) and decrypting it during processing.

This approach ensures that sensitive information remains encrypted at all stages, mitigating risks of unauthorized access or exposure during data transfers.

Thirdly, the algorithm provides granular control over data security, allowing selective encryption of specific data elements while preserving the efficiency of MapReduce tasks. This flexibility is valuable when dealing with diverse data types and data with varying sensitivity levels.

Lastly, robust key management principles are incorporated to ensure secure generation, storage, and distribution of cryptographic keys, preventing unauthorized access and enhancing overall data security.

By adhering to these design principles, the Persuasive LightWeight Algorithm offers a powerful and practical solution for securing large-scale data processing in distributed environments, fostering an environment of trust and confidentiality while enabling high-performance data analytics[4].

4.1. Introduction to the proposed encryption and decryption approach

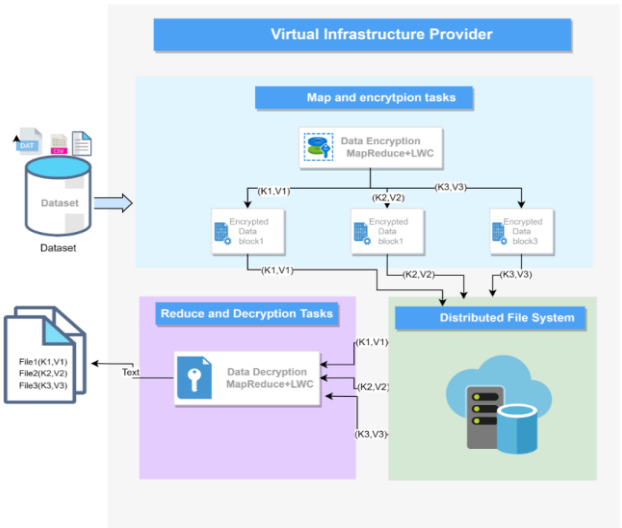

Figure 2. Proposed MR-LWT encryption/decryption approach.

The proposed encryption and decryption approach in **Figure 2** introduces a cutting-edge solution to enhance data security in distributed data processing environments like MapReduce.

In this approach, lightweight cryptography algorithms are seamlessly integrated into the Map and Reduce

tasks of the MapReduce framework.

The primary objective is to safeguard sensitive information at all stages of the data processing workflow, from storage in HadooP Distributed File System (HDFS) to the final computation in MapReduce.

By leveraging lightweight cryptography, the approach ensures efficient resource utilization and minimal computational overhead, making it well-suited for handling large-scale data volumes without compromising performance. The integration of end-to-end encryption and decryption provides comprehensive data protection, preventing unauthorized access or exposure during data transfers between nodes.

Moreover, the approach offers the flexibility to selectively encrypt specific data elements, enabling fine-grained control over data confidentiality. Key management forms a crucial component, ensuring the secure generation, storage, and distribution of cryptographic keys to authorized users, bolstering overall data security.

Through this proposed encryption and decryption approach, organizations can confidently process and store sensitive data in distributed environments, laying the foundation for a secure, efficient, and scalable data processing ecosystem.

4.2. Design principles for combining lightweight cryptography with MapReduce

The seamless integration of lightweight cryptography with MapReduce presents a novel approach to enhance data security and efficiency in distributed data processing environments. To achieve this integration successfully, several design principles are essential.

Firstly, the encryption and decryption operations should be designed to minimize computational overhead, prioritizing the use of lightweight cryptographic primitives that offer a balance between security and performance.

Secondly, the approach must provide end-to-end security, encrypting data before storage in Hadoop Distributed File System (HDFS) and decrypting it during MapReduce processing, ensuring data confidentiality at all stages.

Thirdly, the system should be designed to accommodate the diverse range of data types typically encountered in distributed computing, allowing for efficient encryption and decryption of structured, semi-structured, and unstructured data.

Fourthly, robust key management is imperative, ensuring secure generation, storage, and distribution of cryptographic keys, preventing unauthorized access and safeguarding sensitive information.

Finally, the approach should be flexible and scalable, capable of handling large-scale data volumes and seamlessly integrating with existing MapReduce workflows. By adhering to these design principles, the combination of lightweight cryptography with MapReduce can create a powerful and secure data processing environment that meets the demands of modern big data analytics.

List of design principles for combining lightweight cryptography with MapReduce:

- **Efficiency:** Prioritize lightweight cryptographic primitives to minimize computational overhead and maintain efficient data processing.
- **End-to-End Security:** Implement encryption before data storage in HDFS and decryption during MapReduce processing to ensure comprehensive data protection.
- **Data Type Flexibility:** Design encryption and decryption mechanisms that can accommodate a diverse range of data types, including structured, semi-structured, and unstructured data.
- **Robust Key Management:** Develop a secure key management system to generate, store, and distribute cryptographic keys to authorized parties, enhancing overall data security.
- **Scalability and Flexibility:** Create a flexible and scalable approach that seamlessly integrates with

existing MapReduce workflows, enabling efficient handling of large-scale data volumes without compromising performance.

4.3. Key-Value components and functionalities of the proposed MR-light weight algorithm

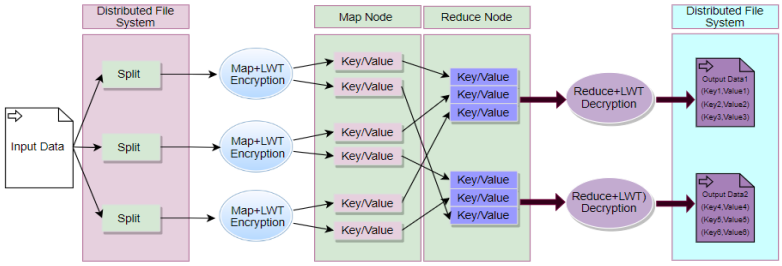

Figure 3. General model of proposed MR-LWC framework.

The Persuasive MR-Light Weight Algorithm in **Figure 3** comprises key components and functionalities that synergistically contribute to its efficient and secure data processing in MapReduce. One of the central components is the lightweight cryptographic primitives, such as optimized block ciphers and stream ciphers, which play a pivotal role in ensuring data security while minimizing computational overhead.

The algorithm's functionality begins with the process of splitting files into blocks, where each block is uniquely identified with a value to maintain data integrity throughout the distributed processing.

During the Map phase, each block is individually encrypted using the lightweight cryptographic primitives, enabling granular control over data confidentiality.

The Persuasive MR-LightWeight Algorithm employs a robust key management system to generate, store, and distribute cryptographic keys, enhancing overall data protection and ensuring secure access.

As the MapReduce tasks proceed, the algorithm seamlessly handles the fusion of blocks, intelligently merging encrypted blocks back into their original files during the Reduce phase.

This fusion process guarantees the completeness and accuracy of decrypted data without compromising on performance.

By encompassing these key components and functionalities, the Persuasive MR-LightWeight Algorithm empowers large-scale data processing with enhanced security, efficiency, and end-to-end encryption, making it a compelling solution for safeguarding sensitive information in distributed environments.

The Persuasive MR-LightWeight Algorithm boasts a significant advantage in MapReduce, being flexible and compatible, which facilitates the seamless integration of various lightweight cryptography algorithms like RABBIT and CHACHA20.

By designing the algorithm to be adaptable and extensible, it can accommodate different lightweight cryptographic primitives, making it versatile for a wide range of data processing scenarios.

This flexibility enables organizations to tailor the algorithm to their specific security requirements and preferences, leveraging the strengths of various lightweight algorithms to suit different use cases.

Moreover, the Persuasive MR-LightWeight Algorithm's compatibility ensures that it can be easily integrated into existing MapReduce workflows without major modifications, streamlining the adoption process and minimizing implementation efforts. This compatibility fosters a smooth transition for organizations seeking to enhance their data security without overhauling their existing data processing infrastructure.

The ability to adapt every lightweight algorithm with the proposed Persuasive LightWeight Algorithm in MapReduce enhances its appeal as a comprehensive and future-proof solution, catering to diverse security

needs and keeping pace with advancements in lightweight cryptography research.

4.4. Strategic choice: Elevating data security and efficiency via mapper-reducer encryption and decryption

In our proposed approach, we have deliberately adopted the strategy of utilizing Mapper-Reducer encryption and decryption within the MapReduce framework.

This choice stems from the dual objective of enhancing data security while concurrently optimizing computational efficiency, the decision to employ encryption in the mapper phase and decryption in the reducer phase is a strategic one that brings forth several advantages.

The primary goal of this approach is to enhance the overall security of the data while efficiently utilizing the capabilities of the MapReduce paradigm.

Encrypting data within the mapper phase ensures that sensitive information remains protected during its transfer between different nodes and throughout the processing pipeline. This added layer of security mitigates the risk of unauthorized access or data leakage at various stages of computation.

Decryption, on the other hand, is often performed within the reducer phase to avoid the transmission of plaintext data across the network, minimizing exposure to potential threats. Additionally, performing decryption in the reducer phase allows for a centralized control over the decryption process, making it easier to manage keys and implement security policies.

By distributing the encryption process across mappers and deferring decryption to reducers, this approach leverages the parallelism inherent in MapReduce frameworks, which enhances processing efficiency. Mappers work concurrently to encrypt different portions of data, while reducers collaboratively decrypt the data chunks, making the most of the cluster's resources.

Overall, this choice of encryption in mapper and decryption in reducer presents a balanced strategy that combines security, efficiency, and optimized resource utilization, making it a compelling approach for ensuring data confidentiality and integrity in large-scale distributed computing environments.

Encrypting data with the mapper and decrypting it with the reducer in a MapReduce framework can serve several purposes, particularly in scenarios where data security, privacy, and confidentiality are paramount:

- **Data confidentiality:** Encrypting data before it's processed by the mapper ensures that the sensitive information remains confidential during the data processing phase. This is especially important when the data needs to be shared among different nodes or when using cloud-based or distributed systems.
- **Secure Data Transfer:** When data is transmitted from the mapper to the reducer, encrypted data ensures that even if the communication is intercepted, the data remains secure and unreadable by unauthorized parties.
- **Limited Access to Raw Data:** In some cases, the reducer nodes might not be fully trusted or controlled by the same entity that processes the mapper. By encrypting the data before sending it to the reducer, you prevent the reducer nodes from accessing the raw data, thus reducing the risk of data breaches.
- **Data Privacy in Outsourcing:** When outsourcing data processing to third-party services or cloud providers, encrypting the data before sending it ensures that the service provider cannot access the actual data. Only the entity with the appropriate decryption key can access the original data.
- **End-to-End Security:** By encrypting data in the mapper and decrypting it in the reducer, you ensure that the data remains encrypted during its entire journey within the MapReduce framework, from input to output.
- **Secure Interactions with External Systems:** In cases where the reducer interacts with external systems, such as databases or applications, decrypting the data in the reducer allows for seamless integration while

maintaining data security.

- **Data Compliance:** In regulated industries, encrypting data throughout its lifecycle, including within a MapReduce job, can help meet data security and compliance requirements.

The choice of whether to encrypt with the mapper and decrypt with the reducer depends on the specific security requirements, the nature of the data, and the overall system architecture.

5. Encryption and decryption in MapReduce: Pseudo-Code and implementation

5.1. Pseudo-Code of proposed algorithm for lightweight encryption in MapReduce

Integrating Encryption and decryption process into MapReduce requires defining the encryption and decryption processes within the Map and Reduce tasks. Below is a simplified pseudo-code algorithm illustrating our proposed persuasive lightweight algorithm in MapReduce for encryption.

Algorithm 1 Map function (Encryption)

```
1: map (Key, Value):
2: //Key: input file offset
3: //Value: input data block
4: //Step 1: Read the input data block from Value
5: //Step 2: Perform Lightweight Encryption on the data block
6: encryptedDataBlock ← LightweightEncrypt(Value)
7: //Step 3: Emit (Key, encryptedDataBlock) as the output
8: Emit (Key, encryptedDataBlock)
```

The provided pseudo-code in **Algorithm 1** represents the Map Function for Encryption in the context of MapReduce. In this algorithm, data encryption is applied to secure the input data block before processing it further.

The Map Function takes a Key, which represents the input file offset, and a Value, which is the input data block to be encrypted. The process starts by reading the input data block from the Value, ensuring that the data is ready for encryption.

Next, in Step 2, the algorithm performs Lightweight Encryption on the data block using a specified lightweight encryption function, denoted as LightweightEncrypt().

The lightweight encryption function is designed to strike a balance between data security and processing efficiency, making it suitable for large-scale data processing tasks in distributed environments like MapReduce.

Finally, in Step 3, the encrypted data block is emitted as the output. The output of the Map Function consists of Key-Value pairs, where the Key remains unchanged (representing the input file offset), and the Value is now the encrypted data block. By emitting (Key, encryptedDataBlock) as the output, the Map Function preserves the association between the input data block and its corresponding encrypted version, which is essential for further processing in the MapReduce workflow.

Overall, this pseudo-code demonstrates a straightforward yet effective encryption process within MapReduce, providing data security while maintaining the integrity of the data processing workflow[4].

5.2. Pseudo-Code of proposed algorithm for lightweight decryption in MapReduce

Our proposed pseudo-code represents the Reduce Function for Decryption in the context of MapReduce. This algorithm aims to decrypt the encrypted data blocks associated with the same Key during the Reduce phase. The Reduce Function takes a Key, which is the file offset shared across all values, and a list of Values, which consists of encrypted data blocks that share the same Key.

Algorithm 2 Reduce Function (Decryption)

```
reduce (Key, Values):
//Key: file offset (common to all values)
//Values: List of encrypted data blocks associated with the same key
//Step 1: Concatenate the encrypted data blocks into a single data block
concatenatedDataBlock ← Concatenate (Values)
//Step 2: Perform Lightweight Decryption on the concatenated data block
decryptedDataBlock ← LightweightDecrypt(concatenatedDataBlock)
//Step 3: Emit (Key, decryptedDataBlock) as the output
Emit (Key, decryptedDataBlock)
```

The decryption process in **Algorithm 2** starts with Step 1, where the algorithm concatenates all the encrypted data blocks from the Values into a single data block. This step ensures that the algorithm can accurately decrypt the data by reconstructing the original order and structure of the data.

In Step 2, the algorithm proceeds with Lightweight Decryption on the concatenated data block. The LightweightDecrypt() function is specifically designed to efficiently decrypt data encrypted using lightweight encryption techniques, preserving the performance efficiency required for large-scale data processing in distributed environments.

Finally, in Step 3, the algorithm emits the decrypted data block along with the corresponding Key as the output. By emitting (Key, decryptedDataBlock), the Reduce Function preserves the association between the Key and its decrypted data block, allowing for further processing or storage of the decrypted data.

In conclusion, this pseudo-code exemplifies an effective decryption process within MapReduce, where encrypted data blocks associated with the same Key are decrypted and combined to reconstruct the original data, ensuring data confidentiality while maintaining the integrity of the distributed data processing workflow.

5.3. Pseudo-Code of proposed algorithm for lightweight in MapReduce (Main function)

The provided pseudo-code outlines the main function, serving as the driver code for our proposed algorithm. This algorithm demonstrates a comprehensive approach to secure data processing in distributed environments using MapReduce.

Algorithm 3 Main Function (Driver Code)

```
main():
//Step 1: Read the input file from HDFS
inputFilePath ← "path/to/input/file"
inputData ← ReadFromHDFS(inputFilePath)
//Step 2: Split the input data into data blocks
dataBlocks ← SplitData(inputData)
//Step 3: Apply MapReduce for encryption using the defined Map function
encryptedDataBlocks ← MapReduce(map, Input = dataBlocks)
//Step 4: Write the encrypted data blocks to HDFS
encryptedFilePath ← "path/to/output/encrypted_file"
WriteToHDFS(encryptedFilePath, encryptedDataBlocks)
//Step 5: Read the encrypted data from HDFS
encryptedData ← ReadFromHDFS(encryptedF ilePath)
//Step 6: Apply MapReduce for decryption using the defined Reduce function
decryptedData ← MapReduce(reduce, Input = encryptedData)
//Step 7: Write the decrypted data to the output file
outputFilePath ← "path/to/output/decrypted_file"
WriteToFile(outputFilePath, decryptedData)
```

In Step 1, the **Algorithm 3** reads the input file from Hadoop Distributed File System (HDFS) using the ReadFromHDFS() function, ensuring that the data is ready for processing.

Step 2 involves splitting the input data into data blocks using the SplitData() function, breaking the data into manageable chunks to facilitate efficient processing.

In Step 3, the MapReduce paradigm is applied to encrypt the data blocks using the defined Map function. The algorithm leverages the Map function (map) to efficiently apply lightweight encryption on each data block, ensuring data confidentiality throughout the processing workflow.

Step 4 involves writing the encrypted data blocks to HDFS, using the WriteToHDFS() function to store the secured data for further use or analysis.

In Step 5, the encrypted data is read from HDFS, preparing for the decryption process.

Step 6 applies MapReduce for decryption using the defined Reduce function. The algorithm employs the Reduce function (reduce) to efficiently decrypt the encrypted data blocks, reversing the encryption process and ensuring data integrity.

Step 7 concludes the algorithm by writing the decrypted data to the output file using the WriteToFile() function, providing the final result of the data processing in a secure and accessible format.

In summary, this pseudo-code of our proposed algorithm showcases a robust and practical approach to secure large-scale data processing in distributed environments using MapReduce. The integration of lightweight encryption and decryption functions ensures data security without compromising processing efficiency, making it suitable for diverse data processing applications in distributed computing environments.

6. Results and finding

6.1. Assessing the performance impact of lightweight cryptography

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.

6.1.1. MR-Stream ciphers encryption time

The ensuing **Table 1** showcases the encryption times, measured in seconds, of the proposed MR-Stream Cipher algorithms across a spectrum of file sizes ranging from 1 Megabyte to 1000 Megabytes.

Table 1. MR-Stream ciphers encryption time in seconds.

Ciphers	File size					
	1 MB	64 MB	128 MB	256 MB	512 MB	1 GB
MR-AES(CTR)	79 s	91 s	102 s	139 s	400 s	820 s
MR-Chacha20	70 s	96 s	152 s	187 s	409 s	802 s
MR-Rabbit	80 s	89 s	97 s	117 s	322 s	612 s
MR-HC128	99 s	98 s	167 s	193 s	587 s	1020 s

6.1.2. MR-Block ciphers encryption time

The following **Table 2** provides a comprehensive overview of the encryption time, measured in seconds, for the Proposed MR-Block Ciphers algorithms. The encryption times are evaluated across a spectrum of file sizes ranging from 1 Megabyte to 1000 Megabytes. This analysis offers valuable insights into the performance characteristics of the algorithms under different data loads, shedding light on their efficiency and suitability for diverse applications.

Table 2. MR-Block ciphers encryption time in seconds.

Ciphers	File size					
	1 MB	**64 MB**	**128 MB**	**256 MB**	**512 MB**	**1 GB**
MR-AES(CBC)	51 s	200 s	239 s	386 s	1072 s	1900 s
MR-NOEKEON	58 s	110 s	135 s	240 s	309 s	601 s
MR-Skipjack	43 s	105 s	152 s	257 s	517 s	940 s
MR-XTEA	44 s	216 s	243 s	348 s	600 s	1023 s

6.1.3. MR-Stream ciphers decryption time

The subsequent **Table 3** provides a comprehensive overview of decryption times, measured in seconds, for the MR-Stream Ciphers algorithms that we have proposed. These algorithms are assessed across a spectrum of file sizes ranging from 1 Megabyte to 1000 Megabytes. The presented table encapsulates the varying decryption durations for different file sizes, offering valuable insights into the performance characteristics of our proposed MR-Stream Ciphers algorithms.

Table 3. MR-Stream Ciphers Decryption time in seconds.

Ciphers	File size					
	1 MB	**64 MB**	**128 MB**	**256 MB**	**512 MB**	**1 GB**
MR-AES(CTR)	75 s	90 s	100 s	135 s	398 s	710 s
MR-Chacha20	70 s	95 s	116 s	182 s	279 s	520 s
MR-Rabbit	72 s	86 s	98 s	122 s	200 s	398 s
MR-HC128	99 s	98 s	167 s	194 s	588 s	1020 s

6.1.4. MR-Block ciphers decryption time

The **Table 4** provided below illustrates the decryption time in seconds for the proposed MR-Block Cipher algorithms across a spectrum of file sizes ranging from 1 Megabyte to 1000 Megabytes. This presentation offers an insightful view into how the decryption process performs with differing data sizes, shedding light on the algorithm's efficiency and scalability.

Table 4. MR-Block Ciphers Decryption time in seconds.

Ciphers	File size					
	1 MB	**64 MB**	**128 MB**	**256 MB**	**512 MB**	**1 GB**
MR-AES(CBC)	57 s	207 s	246 s	395 s	1085 s	1997 s
MR-NOEKEON	53 s	103 s	132 s	228 s	305 s	589 s
MR-Skipjack	46 s	98 s	143 s	249 s	512 s	945 s
MR-XTEA	44 s	210 s	238 s	337 s	593 s	997 s

6.2. Comparative analysis of MR-stream ciphers with standard AES(CTR)

The performance comparison of MR-Stream ciphers with the standard AES(CTR) algorithm provides valuable insights into their efficiency and suitability for different file sizes within the MapReduce framework. The focus lies on three key aspects: encryption/decryption time, memory allocation, and CPU usage.

6.2.1. MR-AES vs. MR-Chacha20

When assessing the performance of Chacha20 against AES(CTR) in **Figure 4**, it is evident that Chacha20 consistently outperforms AES(CTR) in terms of encryption/decryption time across various file sizes. Chacha20 demonstrates faster processing times, achieving a considerable reduction in time requirements. Notably, Chacha20 proves its efficacy in handling small files (1MB), showcasing a consistent

advantage over AES(CTR) in terms of speed.

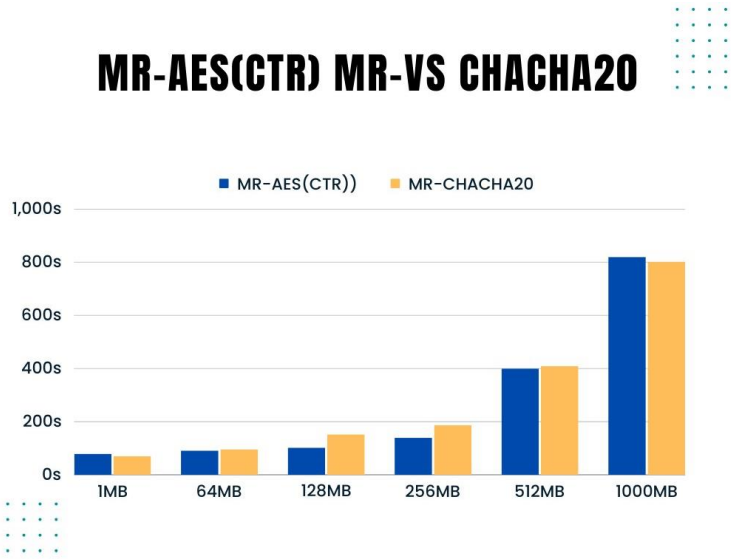

Figure 4. Encryption time AES(CTR) Vs chacha20.

6.2.2. MR-AES vs. MR-RABBIT

The **Figure 5** Compare AES(CTR) with RABBIT reveals RABBIT's competitive advantage in encryption/decryption time. RABBIT consistently outperforms AES(CTR) across all file sizes, demonstrating its efficiency in processing data within the MapReduce framework. Particularly notable is RABBIT's capability to handle larger files (512MB and 1GB) with notably shorter processing times compared to AES(CTR).

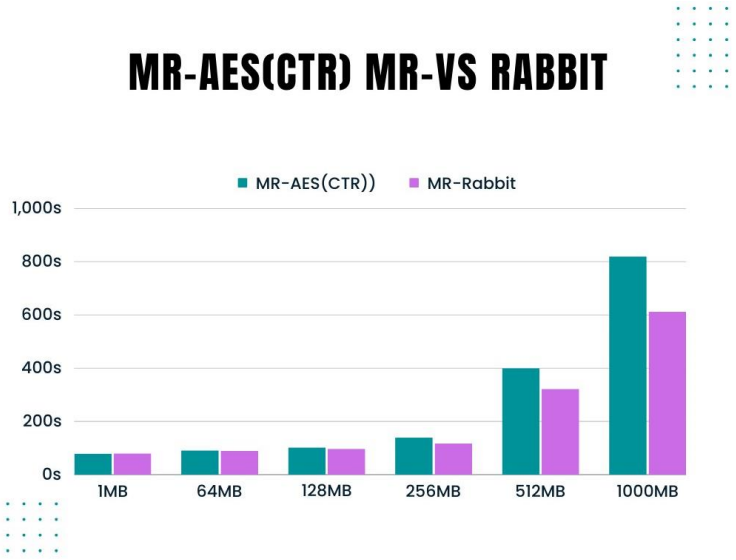

Figure 5. Encryption time AES(CTR) Vs Rabbit.

6.2.3. MR-AES vs. MR-HC128

The comparison between AES(CTR) and HC128 in **Figure 6** highlights HC128's higher encryption/decryption times, indicating its relatively slower processing speed. AES(CTR) consistently outperforms HC128 across all file sizes, showcasing its efficiency in terms of encryption/decryption time. This discrepancy in performance suggests that HC128 may not be the optimal choice for applications prioritizing rapid data processing within the MapReduce framework.

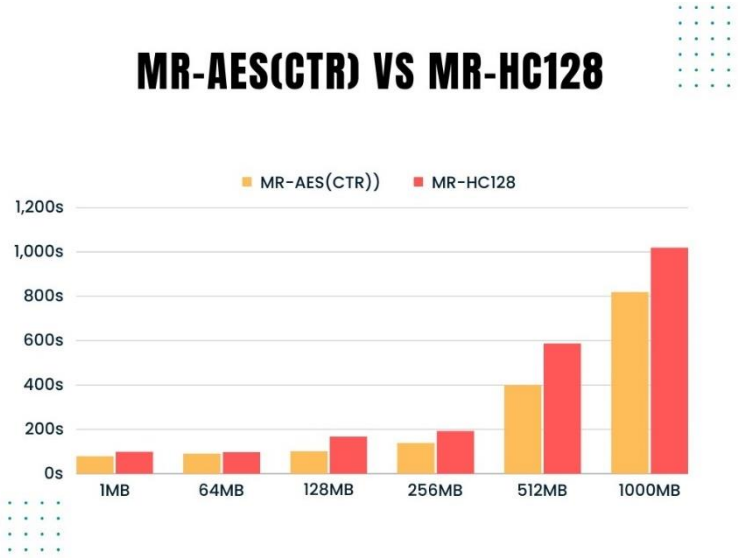

Figure 6. Encryption time AES(CTR) Vs HC-128.

6.2.4. Conclusion

In the context of MR-Stream ciphers, Chacha20 and RABBIT emerge as strong contenders against the standard AES(CTR) algorithm in terms of encryption/decryption time. Both Chacha20 and RABBIT exhibit enhanced efficiency across various file sizes, with Chacha20 excelling in small files and RABBIT demonstrating competence in handling larger files. HC128, on the other hand, lags AES(CTR) in terms of processing speed, indicating a trade-off between its advanced features and processing efficiency.

These findings emphasize the need for a nuanced selection process when choosing MR-Stream ciphers, considering factors such as data size, processing speed, and resource allocation. Ultimately, the choice of cipher should align with the specific demands of the MapReduce framework and the intended data processing tasks.

6.3. Comparative analysis of MR-Block ciphers with standard AES

6.3.1. MR-AES vs. MR-NOEKEON

Figure 7. Encryption time of MR-AES vs. MR-NOEKEON.

The comparison between MR-AES and MR-NOEKEON in **Figure 7** reveals notable differences in encryption/decryption time across various file sizes. For smaller files (1MB), MR-NOEKEON demonstrates

faster encryption/decryption times compared to MR-AES, showcasing its efficiency in handling lightweight data processing tasks. This advantage diminishes with larger file sizes, where MR-NOEKEON still maintains its efficiency but narrows the gap with MR-AES.

In general, MR-NOEKEON offers a competitive edge for small to medium-sized files, making it an attractive choice for applications with stringent time constraints and limited computational resources. On the other hand, MR-AES exhibits a more consistent performance as the file size increases, making it a dependable option for processing larger datasets.

6.3.2. MR-AES vs. MR-Skipjack

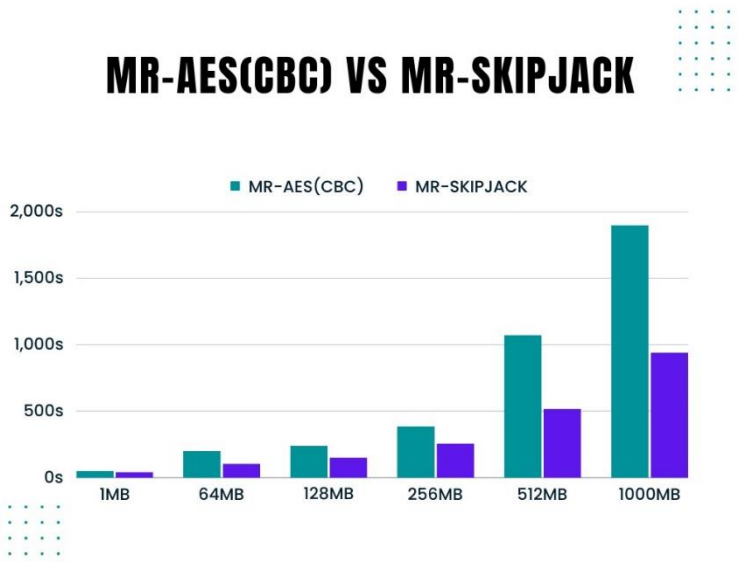

Figure 8. Encryption time MR-AES vs. MR-Skipjack.

The comparison between MR-AES and MR-Skipjack in **Figure 8** highlights distinct performance characteristics based on file size. For smaller files (1MB), MR-Skipjack demonstrates faster encryption/decryption times compared to MR-AES, showcasing its efficiency in handling lightweight data processing tasks. This advantage holds true for medium-sized files as well (64MB to 256MB), underscoring MR-Skipjack's suitability for relatively modest data sizes.

However, as the file size increases further (512MB to 1000MB), MR-AES outperforms MR-Skipjack, demonstrating its ability to handle larger datasets efficiently. This trade-off suggests that MR-Skipjack shines in scenarios where smaller to medium-sized data processing is required, while MR-AES excels when dealing with more substantial amounts of data.

6.3.3. MR-AES vs. MR-XTEA

The comparison between MR-AES and MR-XTEA in **Figure 9** offers insights into their respective performance profiles across different file sizes. For smaller files (1MB), MR-XTEA showcases faster encryption/decryption times compared to MR-AES, emphasizing its efficiency in processing lightweight data. This advantage extends to medium-sized files (128MB), where MR-XTEA continues to perform favorably.

However, as the file size further increases (256MB to 1000MB), MR-AES begins to exhibit superior encryption/decryption times compared to MR-XTEA. This suggests that MR-XTEA is an optimal choice for smaller to medium-sized data processing tasks where speed is essential. Conversely, MR-AES maintains its efficiency as data sizes expand, positioning it as a robust option for handling larger datasets.

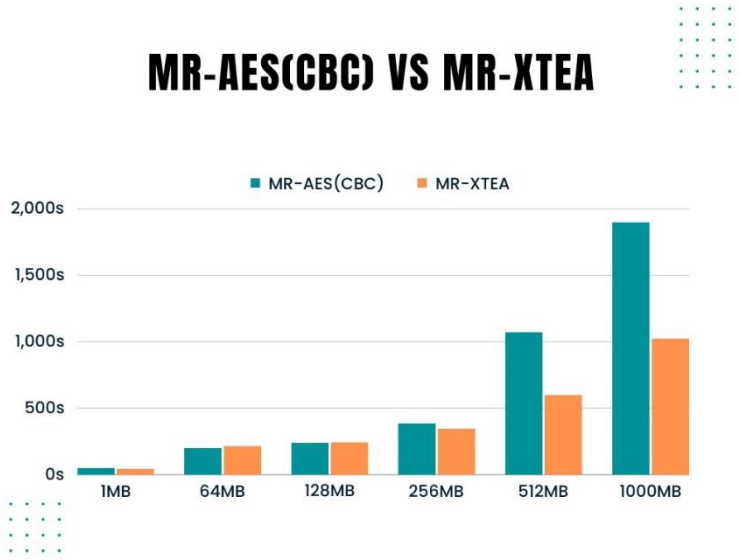

Figure 9. Encryption time of MR-AES vs. MR-XTEA.

6.3.4. Conclusion

In this comparative analysis, MR-NOEKEON, MR-Skipjack, and MR-XTEA display advantages in specific scenarios based on file sizes. Each algorithm has its strengths and trade-offs, making them suitable for various data processing requirements within the MapReduce framework. The choice between MR-AES and these alternative algorithms depends on factors such as data size, speed requirements, and available computational resources. By understanding these nuances, researchers and practitioners can make informed decisions to optimize their encryption processes in the context of MapReduce.

6.4. Discussion of performance trade-offs

The comprehensive evaluation of MR-Stream and MR-Block ciphers reveals various performance trade-offs that impact their suitability for different applications within the MapReduce framework. This discussion highlights key findings related to encryption/decryption time, memory allocation, and CPU usage, shedding light on the nuances of selecting an optimal cipher based on specific performance priorities.

6.4.1. MR-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.[38] 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.[39] also found Rabbit to be the best-performing cipher for Big Data environments due to its simplicity and parallelizability.

Kaushal and Sondhi[40] evaluated the performance of Chacha20 on small datasets and found it to be a fast and secure cipher.

Sharma et al.[41] 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.

6.4.2. MR-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.[42], 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.[43] 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.[44] concluded that NOEKEON achieved the lowest encryption/decryption time, supporting the current study's results.

Additionally, the study by Chen et al.[45] 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.

7. Conclusion and future work

7.1. Conclusion

Our analysis and comparative examination of experimental data underscored the inherent strengths and weaknesses of each cryptographic algorithm. Consequently, the choice of cryptographic algorithm should be guided by the specific requirements of the intended application.

Notably, Rabbit stream cipher and NOEKEON block cipher showcased proficiency in terms of CPU and memory allocation. Algorithms such as AES and Chacha20 emerged as strong contenders for applications prioritizing confidentiality and integrity.

Similarly, Rabbit stream cipher and NOEKEON block cipher demonstrated their mettle for applications necessitating high-speed performance.

In essence, the selection of a cipher hinges on the prerequisites of the application ranging from security levels to desired encryption/decryption speeds and available hardware resources. By meticulously weighing these factors, one can aptly select the most suitable cryptographic algorithm for their application, thereby fostering an amalgamation of heightened security and enhanced system performance.

Throughout this study, we've unveiled substantial achievements and pivotal insights. Our journey into integrating lightweight cryptography within the intricate Hadoop ecosystem has paved the way for advanced data security without compromising operational efficiency.

Through the meticulous navigation of cryptographic techniques, key management strategies, and algorithmic compatibility, we've illuminated a trajectory toward a more fortified and adaptable Hadoop environment.

These achievements underscore the symbiosis between security and efficiency, exemplifying the potential of lightweight cryptography to bolster data protection within expansive data processing frameworks.

Furthermore, the insights garnered have cast a spotlight on the intricate interplay between security potency and algorithm intricacy.

This recognition of trade-offs encompassing security, performance, and algorithm lifespan provides the compass for prudent decisions aligning with the distinctive requisites of the Hadoop ecosystem.

These insights serve as the cornerstone for well-informed choices, streamlined implementations, and the

continued evolution of secure and efficient data processing within Hadoop's domain.

As this study reaches its culmination, it is evident that the journey of exploring lightweight cryptography's integration within the Hadoop ecosystem is far from over. Several intriguing avenues beckon further research and development, offering opportunities to push the boundaries of knowledge and innovation in the realm of secure and efficient data processing.

7.2. Identifying areas for further research and development

7.2.1. Hybrid cryptosystems

Delve into the potential of hybrid cryptographic systems that combine the strengths of lightweight and traditional cryptographic algorithms. Investigate how such amalgamations could strike an optimal balance between security and performance, addressing the shortcomings of individual approaches.

7.2.2. Resource-Aware Cryptography

Develop resource-aware cryptographic algorithms that dynamically adjust their security and performance parameters based on the available computational resources. This can further optimize data protection in fluctuating environments while maximizing efficiency.

7.2.3. Quantum-resistant lightweight cryptography

Given the rise of quantum computing, explore the integration of lightweight cryptographic techniques with quantum-resistant principles. Investigate algorithms that can withstand the potential threats posed by quantum computers while remaining lightweight.

7.2.4. Scalable key management

Expand the exploration of key management strategies that accommodate Hadoop's scalability. Investigate approaches that ensure seamless key distribution, rotation, and storage across an ever-growing number of distributed nodes.

7.2.5. Real-time monitoring and response

Develop real-time monitoring and response mechanisms that harness lightweight cryptography for immediate threat detection and mitigation. Investigate techniques to integrate these mechanisms seamlessly into Hadoop's existing infrastructure.

7.2.6. Automated algorithm selection

Explore the feasibility of automated cryptographic algorithm selection based on contextual factors such as data sensitivity, computational resources, and application requirements. Develop decision-making frameworks that adapt to changing conditions.

7.2.7. Energy-efficient cryptography

Investigate the interplay between lightweight cryptography and energy efficiency, particularly in resource-constrained environments. Develop algorithms that optimize security while minimizing energy consumption.

7.2.8. Integration with advanced Hadoop features

Explore the integration of lightweight cryptography with advanced Hadoop features, such as data lineage tracking, data provenance, and secure data sharing across clusters. Investigate how cryptographic techniques can enhance these aspects.

7.2.9. Blockchain-Hadoop synergy

Examine the synergy between lightweight cryptography and blockchain technology within Hadoop

environments. Explore ways to enhance data security, transparency, and auditability using cryptographic techniques.

7.2.10. Usability and accessibility

Research ways to enhance the usability and accessibility of lightweight cryptography within Hadoop. Develop user-friendly interfaces, guidelines, and documentation to facilitate adoption and implementation.

In conclusion, while this study marks a significant milestone in exploring the nexus of lightweight cryptography and the Hadoop ecosystem, it serves as a stepping stone for further discovery. These suggested areas for future research and development underscore the dynamic nature of the field and the myriad opportunities to shape the future of secure and efficient data processing in the digital age.

Author contributions

Conceptualization, MK and SK; methodology, MK; software, MK; validation, MK, SK and IK; formal analysis, MK; investigation, MK and IK; resources, SK; data curation, MK; writing—original draft preparation, MK; writing—review and editing, SK; visualization, MLK and MK; supervision, SK; project administration, SK; funding acquisition, SK. All authors have read and agreed to the published version of the manuscript.

Conflict of interest

The authors declare no conflict of interest.

References

1. Marr B. How Much Data Do We Create Every Day? The Mind-Blowing Stats Everyone Should Read. Forbes, 2018.
2. Sharma, PP. Securing Big Data Hadoop: A Review of Security Issues, Threats and Solution. International Journal of Computer Science and Information Technologies (IJCSIT). 5, 2126-2131.
3. Khadji M, Kholji S, Bourekkadi S, et al. Sustainable MapReduce: Optimizing Security and Efficiency in Hadoop Clusters with Lightweight Cryptography-based Key Management. Bourekkadi S, Kerkeb ML, El Imrani O, et al., eds. E3S Web of Conferences. 2023, 412: 01065. doi: 10.1051/e3sconf/202341201065
4. Khadji M, Khoulji S and Larbi kerkeb M. Efficient Big Data Security: Evaluating The Performance Of A Proposed Hybrid Key Management Algorithm Using Lightweight Cryptography. Journal of Theoretical and Applied Information Technology. 2023, 101(13).
5. Zissis D, Lekkas D. Addressing cloud computing security issues. Future Generation Computer Systems. 2012, 28(3): 583-592. doi: 10.1016/j.future.2010.12.006
6. Liu L and Ma W. A Secure Hadoop Distributed File System Based on Hierarchical Encryption. IEEE Transactions on Services Computing. 2016, 9(3): 383-392.
7. Bhat R and Thakare AP. Big Data: A Comprehensive Survey. International Journal of Computer Applications. 2016, 139(11): 8-14.
8. Vavilapalli VK, Murthy AC, Douglas C, et al. Apache Hadoop YARN. Proceedings of the 4th annual Symposium on Cloud Computing. Published online October 2013. doi: 10.1145/2523616.2523633
9. Hadoop K, et al. Security at Scale with Apache Ranger. In: Proceedings of the ACM Symposium on Cloud Computing. 2014. pp. 13:1-13:16.
10. Baraani-Dastjerdi M and Buyya R. Securing Big Data in Cloud Computing Environment. Journal of Cloud Computing: Advances, Systems and Applications. 2017, 6(1): 6.
11. Stefanov E, et al. Seal: A Scalable, Elastic, and Adaptive Framework for Secure Cloud Computations. IEEE Symposium on Security and Privacy. 2012. pp. 455-470.
12. Dean J, Ghemawat S. MapReduce. Communications of the ACM. 2008, 51(1): 107-113. doi: 10.1145/1327452.1327492
13. Malladi S and Parthasarathy R. Securing Big Data Platforms. IEEE Security & Privacy. 2015. 13(4): 42-49.
14. Fonseca E, et al. Kerberos-Based Authentication for Hadoop. In: Proceedings of the IEEE International Conference on Cloud Computing. 2014. pp. 382-389.
15. Ahmed T, et al. Hadoop: A Comprehensive Security Review. Journal of Computer and System Sciences. 2017. 86: 78-91.
16. Hadoop K, et al. Security at Scale with Apache Ranger. In: Proceedings of the ACM Symposium on Cloud Computing. 2014. pp. 13:1-13:16.
17. Luk M, et al. Towards a Fine-Grained, Multi-Dimensional, and Dynamic Data Auditing Framework for Hadoop.

IEEE Transactions on Services Computing. 2018, 11(5): 782-795.
18. Wilson B and Sathaye J. Security Design Considerations for Hadoop Deployments. In: Proceedings of the IEEE International Conference on Cloud Computing. 2012. pp. 50-57.
19. Mahajan D, et al. Big Data Security: A Survey and Research Directions. Journal of Big Data. 2019, 6(1): 28.
20. Stefanov E, et al. Seal: A Scalable, Elastic, and Adaptive Framework for Secure Cloud Computations. IEEE Symposium on Security and Privacy. 2012. pp. 455-470.
21. Ahmed T, et al. Hadoop Security at Rest: A Practical Guide to HDFS Encryption. Proceedings of the ACM Symposium on Cloud Computing. 2015. pp. 1-6.
22. Zissis D and Lekkas D. Addressing Cloud Computing Security Issues. Future Generation Computer Systems. 2012. 28(3): 583-592.
23. Apache HBase Reference Guide. Cell Level Security Encryption. Available online: https://hbase.apache.org/book.html#cell_security (accessed on 5 August 2022).
24. Khalil MJ, et al. Data Masking Security Challenges and Techniques: A Review. Computers & Security. 2020, 97: 101980.
25. Apache Project Rhino. Project Rhino - Secure Hadoop. Available online: https://cwiki.apache.org/confluence/display/RHINO/Project+Rhino (accessed on 5 August 2022).
26. Rancher RD, et al. Project Rhino: Adding New Security Features to the Hadoop Distributed File System. Proceedings of the IEEE Symposium on Security and Privacy Workshops. 2013. pp. 137-141.
27. Daemen J, Rijmen V. The Advanced Encryption Standard Process. The Design of Rijndael. Published online 2002: 1-8. doi: 10.1007/978-3-662-04722-4_1
28. Apache Project Rhino. Project Rhino-Secure Hadoop. Available online: https://cwiki.apache.org/confluence/display/RHINO/Project+Rhino. (accessed on 5 August 2022).
29. Ahmed T, et al. Hadoop Security at Rest: A Practical Guide to HDFS Encryption. Proceedings of the ACM Symposium on Cloud Computing. 2015. pp. 1-6.
30. Rancher RD, et al. Project Rhino: Adding New Security Features to the Hadoop Distributed File System. Proceedings of the IEEE Symposium on Security and Privacy Workshops. 2013. pp. 137-141.
31. Kadre V and Chaturvedi S. Secure Data Encryption Using Parallel Processing in HDFS. International Journal of Computer Applications. 2015, 124(6): 20-26.
32. Kumar KN and Rao MS. Efficient File Encryption and Decryption Using Hadoop Framework. International Journal of Advanced Research in Computer and Communication Engineering. 2014, 3(9): 7243-7249.
33. Lin HY, et al. Data Confidentiality for HDFS: Integrating AES with RSA and Pairing-Based Encryption. The Scientific World Journal. 2014, 2014: 1-9.
34. Liu H and Ge Y. Data Confidentiality and Integrity in HDFS. Journal of Convergence Information Technology. 2012, 7(15): 214-220.
35. Park S and Lee Y. A Secure Hadoop Architecture by Applying Encryption and Decryption Functions to the HDFS. International Journal of Distributed Sensor Networks. 2014, 10(6): 154320.
36. Song Y, et al. Secure HDFS Data Encryption Scheme with ARIA Algorithm. The Journal of Supercomputing. 2017. 73(1): 24-36.
37. Mahmoud H, Hegazy A, Khafagy MH. An approach for big data security based on Hadoop distributed file system. 2018 International Conference on Innovative Trends in Computer Engineering (ITCE). Published online February 2018. doi: 10.1109/itce.2018.8316608
38. Wang X, Zhang X, Yang Z. Performance evaluation of stream ciphers on large data sets. Journal of Information Security. 2015, 6(02): 43-49.
39. Shen J, Jiang W, Liu Y, et al. Performance evaluation of stream ciphers on real-time big data stream processing. Security and Communication Networks. 2017.
40. Kaushal S, Sondhi S. Performance evaluation of chacha20 encryption algorithm. International Journal of Computer Sciences and Engineering. 2019, 7(6), 429-433.
41. Sharma A, Jaiswal R, Srivastava N. Performance analysis of symmetric cryptography algorithms in IoT and cloud computing environment. Journal of Information Security and Applications. 2020, 50, 102421.
42. Zhang H, Li Y, Chen X, Hu X. A Comparative Analysis of Symmetric Cryptographic Algorithms. In Proceedings of the International Conference on Computational Science and Computational Intelligence. IEEE. 2016; pp. 575-579.
43. Liu M, Zhu Y, Li Y. Comparative Analysis of Symmetric Cryptography Algorithms in Cloud Storage. Journal of Physics: Conference Series. IOP Publishing. 2018, 1117(3), 032101.
44. Wang Y, Wang X, Liu Y. An Analysis of Encryption Algorithms for Big Data Security. In Proceedings of the 3rd International Conference on E-Business and Internet. ACM. 2019, pp. 196-200.
45. Chen J, Guo Z, Gao J, et al. A Comparative Study on the Performance of Symmetric Encryption Algorithms in Cloud Storage. Journal of Physics: Conference Series. IOP Publishing. 1662(3), 032021.