JWTAMH: JSON Web Tokens Based Authentication Mechanism for Hadoop

Anish Gupta[1], Annu Sharma[2], Britto Raj S[3] , Manish Kr. Gupta [4,*]

[1]Department of Computer Science & Engineering, Chandigarh Engineering College, Jhanjeri, Punjab, India, 140307.
[2]Department of Computer Science & Application, Rajarajeshwari College of Engineering, Bengaluru, India, 530068.
[3]Department of Computer Science & Engineering, RRASE College of Engineering, Chennai, India, 601301.
[4]Department of Information Technology and Computer Application, Madan Mohan Malaviya University of Technology, U P, India, 273010.

Abstract

The Hadoop platform has become a widely adopted distributed computing platform for handling large-scale data processing tasks. However, the security of the Hadoop platform has become a major concern due to the increased risk of cyber-attacks. To address this concern, various security mechanisms have been proposed for the Hadoop platform, including authentication and access control. This research paper proposes a token-based authentication mechanism to enhance the security of the Hadoop platform. The proposed mechanism utilizes a combination of Kerberos and JSON Web Tokens (JWT) for secure communication between Hadoop components. The experimental results demonstrate the effectiveness of the Single point of failure, Guessing attack, Replay Attack, Brute force attack, and Dictionary attack. The proposed model has better performance in terms of average processing time and accuracy of authentication than other models.

Keywords : Hadoop, Token-based Authentication, Kerberos, JWT, Security, Access Control

Received on 11 02 2024, accepted on 12 06 2024, published on 16 07 2024

doi: 10.4108/eetsis.5429

*Corresponding author. Email:manish.testing09@gmail.com

1. Introduction

In recent years, big data has become a valuable asset for many organizations, providing insights that can drive business growth and inform strategic decision-making. To effectively harness the potential of this data, new technologies are required that can efficiently collect and analyze these enormous volumes of data over the network [1]. Hadoop, an open-source distributed computing platform, has become a standard for handling big data processing tasks. Hadoop is designed to handle large-scale data processing tasks by distributing them across multiple nodes in a cluster [3]. However, the distributed nature of Hadoop makes it vulnerable to security threats, such as unauthorized access, data breaches, and cyber attacks [5-6]. To mitigate these threats, various security mechanisms have been proposed for the Hadoop platform, including authentication and access control. Authentication and access control are critical components of any security system. Authentication ensures that only authorized users can access the system, while access control defines the level of access that each user has within the system. In the Hadoop platform, authentication and access control are implemented using various mechanisms, such as Kerberos, Lightweight Directory Access Protocol (LDAP), and Secure Shell (SSH) [2, 4, and 7]. Despite the availability of these mechanisms, the security of the Hadoop platform remains a major concern. This is due to the increasing sophistication of cyber-attacks, which can bypass traditional security measures and gain access to sensitive data. To address this concern, this research paper proposes a token-based authentication mechanism to enhance the security of the Hadoop platform.

Table 1: Symbols and their Definitions

Symbol	Definition	Symbol	Definition
Usn	User name	Pwd	Password
TGT	Ticket Granting Ticket	KAS	Kerberos Authentication Server
KSsk	Kerberos server's secret key	JWT	JSON Web Token
usid	user's identity	Ts	Timestamp
upwd	user's password	HAS	Hadoop Authorization Server.
Ssk	server's secret key	Puk	Public Key
SSk	Shared Secret Key		

The motivation behind the proposed token-based authentication mechanism is to enhance the security of the Hadoop platform and provide effective authentication and access control mechanisms to mitigate the risk of cyber-attacks. The Hadoop platform is widely used for critical business operations, and any security breach can result in the compromise of sensitive data and financial loss. The proposed mechanism contributes to the field of Hadoop security by utilizing a combination of Kerberos and JWT to provide a secure and efficient method for transmitting security information between Hadoop components. The use of the Kerberos authentication protocol ensures the secure authentication of users, while JWT provides a lightweight and efficient method for transmitting security information between Hadoop components.
The paper is structured in a manner that ensures easy comprehension of the methods in detail. Section 2 provides a background description, while related works are defined in Section 3. Section 4 presents the proposed methodology, and section 5 is dedicated to performance evaluation. Finally, section 6 elucidates the conclusion and outlines future work.

2. Background

The proposed model used the concept of Hadoop, Kerberos, and JWT. This section describes all these concepts in detail and also describes what are the roles of these concepts in the proposed model.

2.1. HADOOP

The Hadoop platform is a widely adopted distributed computing platform for handling large-scale data processing tasks. It consists of several components, including Hadoop Distributed File System (HDFS), Yet Another Resource Negotiator (YARN), and MapReduce. Hadoop's popularity stems from its ability to handle and process massive amounts of data, which traditional computing platforms cannot handle efficiently. Hadoop's distributed computing model allows it to break down large data sets into smaller chunks, which are then processed in parallel across multiple nodes in a cluster. However, the security of the Hadoop platform has become a major concern due to the increased risk of cyber attacks. As Hadoop deployments grow in size and complexity, the risk of security breaches also increases. Malicious actors can exploit vulnerabilities in Hadoop components to gain unauthorized access to sensitive data or disrupt critical business operations.

2.2. Kerberos

Kerberos is a widely used network authentication protocol that provides secure communication between clients and servers over an insecure network. It is primarily used for authentication purposes and is designed to provide a secure method for verifying the identity of users and services in a distributed computing environment. Kerberos was originally developed at the Massachusetts Institute of Technology (MIT) in the 1980s and has since become a widely adopted authentication protocol in enterprise networks and distributed computing environments. It uses a trusted third-party authentication server to verify the identity of users and services and issue secure authentication tickets that can be used to access network resources. Kerberos provides several benefits over traditional authentication mechanisms, including improved security, simplified user management, and reduced network traffic. It uses symmetric key cryptography to encrypt and decrypt authentication tickets, providing a secure method for transmitting sensitive information over an insecure network.
In the proposed token-based authentication mechanism for the Hadoop platform, Kerberos is used as the initial authentication mechanism to verify the identity of users and issue a ticket-granting ticket (TGT). The TGT is then used in conjunction with JWT to provide secure communication between Hadoop components. This approach enhances the security of the Hadoop platform and provides an efficient and

effective method for transmitting security information between Hadoop components.

2.3. JSON Web Tokens

JWT is an open standard for the secure transmission of information between parties over an insecure network. It is a compact, URL-safe means of representing claims to be transferred between two parties. These claims can be used to authenticate and authorize users in web applications, mobile applications, and other distributed systems. JWTs are structured as JSON object that contains a set of claims or assertions about a user or a service. These claims can include information about the user's identity, access rights, and other metadata relevant to the authentication and authorization process. The claims are digitally signed using a secret key or public key cryptography to ensure the integrity and authenticity of the information. JWTs are designed to be lightweight and easy to transmit between parties, making them an ideal choice for use in distributed systems and web applications. They can be used to transmit information between the client and server or between different services in a distributed computing environment.
In the proposed token-based authentication mechanism for the Hadoop platform, JWTs are used in conjunction with Kerberos to provide secure communication between Hadoop components. After the user's identity is verified by the Kerberos server and a TGT is issued, the TGT is used to obtain a JWT, which contains the necessary claims for authentication and authorization. The JWT is then used to provide secure communication between the Hadoop components, ensuring the integrity and confidentiality of the information being transmitted. This approach enhances the security of the Hadoop platform and provides an efficient and effective method for transmitting security information between Hadoop components.

3. Related works

Shen P et al. [8] propose a Kerberos-based approach to authentication and security for Hadoop clusters. It discusses the implementation of Kerberos technology in the context of Hadoop, and how it can be used to ensure secure communication and access control within the cluster. Jeong Y.S. et al. [9] present a hash chain-based authentication protocol for Hadoop systems that can be used to authenticate high-dimensional data. The protocol provides an efficient way to authenticate large volumes of data and is particularly useful in applications such as healthcare and finance where the accuracy of data is crucial. Zheng K. et al. [10] propose a token authentication solution for Hadoop based on Kerberos pre-authentication. The paper discusses how the token-based approach can be used to provide secure communication between nodes in the cluster, and how pre-authentication can be used to reduce the risk of credential theft. Chattaraj D et al. [11] propose a fault-tolerant authentication and key exchange protocol called HEAP for Hadoop-assisted big data platforms. The protocol is designed to provide secure communication between nodes in the cluster and to ensure that the cluster remains available in the event of a node failure. Mohamed H. et al. [12] propose a token-based approach to authentication for Hadoop platforms. The paper discusses how the token-based approach can be used to provide secure communication between nodes in the cluster, and how the use of tokens can help to reduce the risk of credential theft. AL-Rummana et al. [13] propose a user authentication technique for big data platforms that uses a combination of password-based authentication and one-time passwords. The paper discusses how the technique can be used to provide secure communication and access control within the cluster. Algaradi, T.S. et al. [14] present a knowledge-based authentication mechanism for Hadoop clusters that are based on Kerberos technology. The mechanism uses static knowledge as a means of authenticating users and is designed to provide secure communication between nodes in the cluster. Al-Rummana et al. [15] propose a robust user authentication framework for big data that is based on a combination of biometrics and password-based authentication. The framework is designed to provide secure communication and access control within the cluster. Chattaraj, D. et al. [16] propose a two-server authentication and key agreement protocol for accessing secure cloud services. The protocol is designed to provide secure communication between clients and servers, and Somu et al. [18] propose an authentication service for Hadoop using the one-time pad technique. The one-time pad is a symmetric key encryption technique that generates a random key for each encryption. The proposed method is claimed to be secure and reliable against various attacks, including replay and man-in-the-middle attacks. Sarvabhatla, M. et al. [19] propose a secure and lightweight authentication service for Hadoop using the one-time pad technique. This method is claimed to be more secure and lightweight compared to existing authentication services for Hadoop, such as Kerberos. The proposed method is also claimed to be efficient and scalable, making it suitable for large-scale Hadoop clusters. Wu T.Y. et al. [20] present a lightweight authenticated key agreement protocol that utilizes fog nodes in the context of the Social Internet of Vehicles (SIoV). The protocol aims to establish secure communication channels between vehicles and fog nodes, enabling efficient and secure information exchange. Hena M. et al. [21] propose a framework that aims to enhance the security of the Hadoop ecosystem by implementing a distributed authentication mechanism. It presents the design and implementation of the framework, along with its evaluation and comparison to existing authentication approaches. Honar P. et al. [22] introduce an IoT Big Data provenance scheme that utilizes blockchain technology within the Hadoop ecosystem. The scheme aims to ensure data integrity, traceability, and transparency in the context of IoT-generated big data. Marco Anisetti et al. [23] present a systematic approach to evaluating the quality and reliability of Big Data, considering various factors such as data sources, processing algorithms, and security mechanisms. M. Tall et al. [24] introduce a framework that aims to provide fine-grained access control mechanisms that consider multiple attributes and their associated sensitivities. Yin et al. [29] utilize a modality-aware graph convolutional network (MAGCN) to integrate entity attributes and graph connectivity features into a unified feature space, enhancing prediction performance. Yin et al. [30] use an integrated consecutive batch learning framework to predict exploitation times, combining features from vulnerability descriptions and the Common Vulnerability Scoring System. An Adaptive Sliding Window Weighted Learning (ASWWL) algorithm addresses dynamic multiclass imbalance, enhancing minority class performance. You et al. [31] propose an algorithm to build an access control knowledge graph from user and resource attributes and an online learning framework for access control decisions.

Guan et al. [32] provide a method that combines HDFS federation, HDFS high-availability mechanisms, and the Zookeeper distributed coordination mechanism to achieve dual-channel storage. It improves ECC encryption for ordinary data and uses homomorphic encryption for data requiring computation, utilizing a dual-thread encryption mode to enhance efficiency. Baig et al. [33] propose a framework for preserving privacy in data-at-rest within Hadoop, utilizing columnar data storage, data masking, and encryption techniques. Balaraju et al. [34] developed a specialized DNA algorithm for creating unique user IDs and dynamic passwords, enhancing security in Hadoop clusters. Each user accesses data through distinct nodes and services, prioritizing privacy against hackers.

Table 2: Features and Challenges of existing work

Citation & Year	Method Used	Advantage	Disadvantage
Guan et al. [32] 2024	Hadoop-based secure storage solution	Ensures secure storage of big data in cloud environments, high scalability and performance	Complexity in configuration and maintenance, potential overhead in encryption processes
Baig et al. [33] 2024	Column Encryption-Based Privacy-Preserving Framework	Enhances privacy and security of big data sets, specifically designed for Hadoop	May impact performance due to encryption overhead, potential challenges in key management
Balaraju et al. [34] 2024	Dynamic Password to Enforce Secure Authentication Using DNA	High level of security through dynamic password generation, innovative approach using DNA	Complexity in implementation, potential usability issues for users unfamiliar with the system
Anisetti et al. [23] 2023	Assurance process for Big Data trustworthiness	Provides a systematic approach for assessing and ensuring the trustworthiness of Big Data	Requires additional resources for implementing the assurance process
Tall et al. [24] 2023	Framework for Attribute-Based Access Control	Offers fine-grained access control in processing big data with multiple sensitivities	Complexity in managing and configuring access policies
Haggag et al. [12] 2023	Token-based authentication	Provides secure authentication method for Hadoop platform.	Limited information available on method
Hena et al. [21] 2022	Distributed authentication framework	Improves security in Hadoop-based big data environments	May have implementation challenges
Pajooh et al. [22] 2021	IoT Big Data provenance scheme	Ensures data integrity and traceability in IoT-generated big data	Blockchain implementation may introduce additional overhead
Wu et al. [20] 2021	Lightweight Authenticated Key	Enhances security in the Social Internet of Vehicles (SIoV)	The limited scope focused on SIoV

	Agreement Protocol		
Al-Rummana et al. [15] 2021	Robust User Authentication Framework.	Provides strong security features and authentication.	May have compatibility issues with certain applications
Al-Rummana et al. [13] 2021	Robust User Authentication	Provides strong security features and authentication.	May have compatibility issues with certain applications
Algaradi et al. [14] 2019	Static Knowledge-based Authentication	Provides secure authentication method.	Limited to static knowledge-based authentication
Chattaraj et al. [16] 2018	HEAP Protocol	Efficient and fault-tolerant authentication and key exchange protocol.	Requires a trusted third party for key exchange
Shen et al. [8] 2018	Kerberos Technology	Provides strong security features and authentication.	Requires configuration and setup
Chattaraj et al. [11] 2018	HEAP Protocol	Efficient and fault-tolerant authentication and key exchange protocol.	Requires a trusted third party for key exchange
Jeong et al. [9] 2016	Hash Chain based protocol	Efficient and secure method for authentication.	Limited to high-dimensional data

4. Proposed Methodology

The proposed methodology for the token-based authentication mechanism for the Hadoop platform consists of the three steps

1. User Authentication
2. Request for JWT
3. Access Request

4.1 User Authentication using Kerberos

The authentication process is initiated when a user requests access to a Hadoop component. The user's credentials are validated using the Kerberos authentication protocol, which generates a ticket-granting ticket (TGT).

Algorithm 1: Kerberos Authentication

```
Input: User Credentials (usn, pwd)
Output: TGT
Start
1. User sends the usn and pwd to the KAS.
   User → KAS || usn, pwd
2. The KAS verifies the user's credentials by checking
   against its user database or external authentication service.
3. If the credentials are valid, the KAS generates a TGT for the user.
   KAS |-- TGT || used, Ts
      a. The TGT typically contains the user's identity, a
         timestamp, and other relevant information.
         TGT || used, Ts
      b. The TGT is encrypted using the KSsk.
         TGT |-- E(KSsk(TGT))
4. The KAS sends the TGT back to the user.
   KAS → user || TGT
      a. The TGT is encrypted using the user's password
         as the encryption key.
         TGT |-- E(upwd(TGT))
      b. The user must have a secure channel (e.g., SSL/TLS)
         established with the Kerberos server to protect the
         TGT during transmission.
5. The user stores the TGT locally for further authentication requests.
End
```

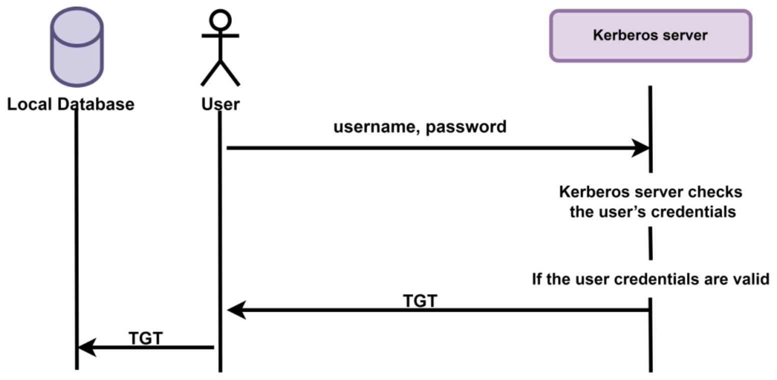

Kerberos server
Local Database
User
username, password
Kerberos server checks the user's credentials
If the user credentials are valid
TGT
TGT

Figure 1: Kerberos Authentication Process

```
class KerberosServer:
    def __init__(self, username, password):
        self.username = username
        self.password = password

    def check_credentials(self, username, password):
        return username == self.username and password == self.password

    def generate_tgt(self):
        return "Ticket Granting Ticket (TGT)"

class User:
    def __init__(self):
        self.tgt = None

    def send_credentials_to_kerberos(self, username, password):
        kerberos_server = KerberosServer("admin", "password")

        if kerberos_server.check_credentials(username, password):
            self.tgt = kerberos_server.generate_tgt()
            print("TGT received successfully.")
        else:
            print("Invalid credentials.")

# Example usage
username = input("Enter username: ")
password = input("Enter password: ")

user = User()
user.send_credentials_to_kerberos(username, password)

# Check if the user has a valid TGT
if user.tgt:
    print("User has a valid TGT:", user.tgt)
else:
    print("User does not have a valid TGT.")
```

Figure 2: Kerberos Authentication Process Code

In this code, we have two classes: “KerberosServer” representing the Kerberos server and “User” representing the user. The “KerberosServer” class has methods to check the user's credentials and generate a TGT. The “User” class has a method to send the credentials to the Kerberos server and store the TGT locally. After running the code, the user is prompted to enter their username and password. The code then checks if the credentials are valid by comparing them with the predefined credentials (“admin” and “password” in this example). If the credentials are valid, the user receives a TGT, which is stored in the User object. Finally, the code checks if the user has a valid TGT and prints the result accordingly.

4.2 Request for JWT

The TGT is then used to request a JWT from the Hadoop Authorization Server. The JWT contains the user’s identity and access permissions, which are validated by the Hadoop component before granting access.

Algorithm 2: Request for JSON Web Token (JWT)

```
Input: TGT
Output: JWT
Start
1. User sends the TGT to the Hadoop Authorization Server.
        User →HAS|| TGT

2. The Hadoop Authorization Server validates the TGT:
        TGT |-- validate (HAS)
    a. Decrypts the TGT using the server's secret key
       to obtain the user's identity.
       uid --| D(Ssk(TGT))

    b. Verifies the TGT's integrity by checking the
       encryption and timestamp.
       Verify (TGTid) --| E, Ts

3. Checks if the user's identity and TGT are authorized to
   request a JWT.

    a. If the TGT is valid, the Hadoop Authorization
       Server generates a JWT:

    b. The JWT contains the user's identity, access
       permissions, and other relevant claims.

    c. The JWT is signed by the Hadoop
       Authorization Server using its private key or a
       shared secret.

4. The Hadoop Authorization Server sends the JWT back to
   the user.

    a. The user must have a secure channel (e.g.,
       SSL/TLS) established with the Hadoop
       Authorization Server to protect the JWT during
       transmission.

5. The user stores the JWT locally for future access requests.
End
```

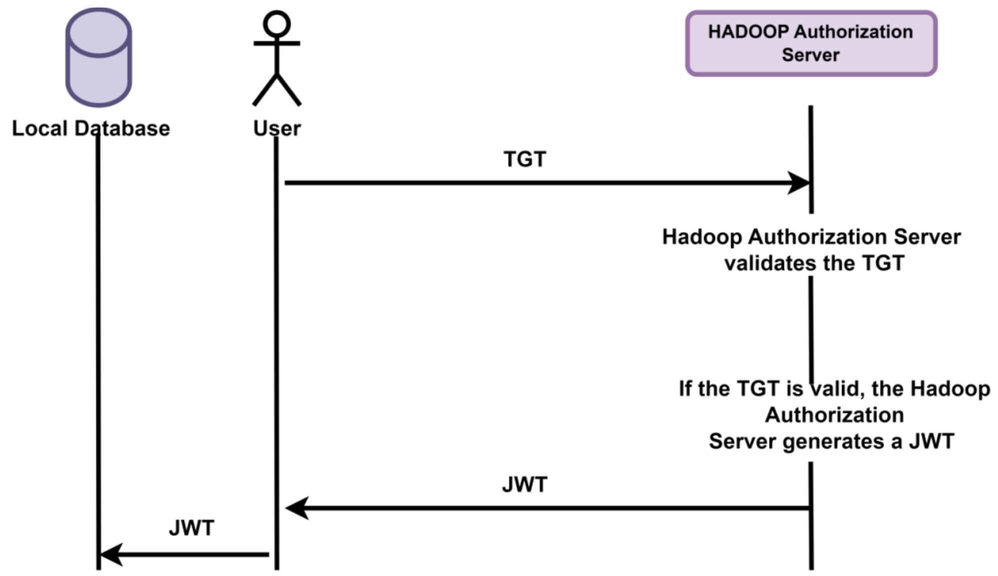
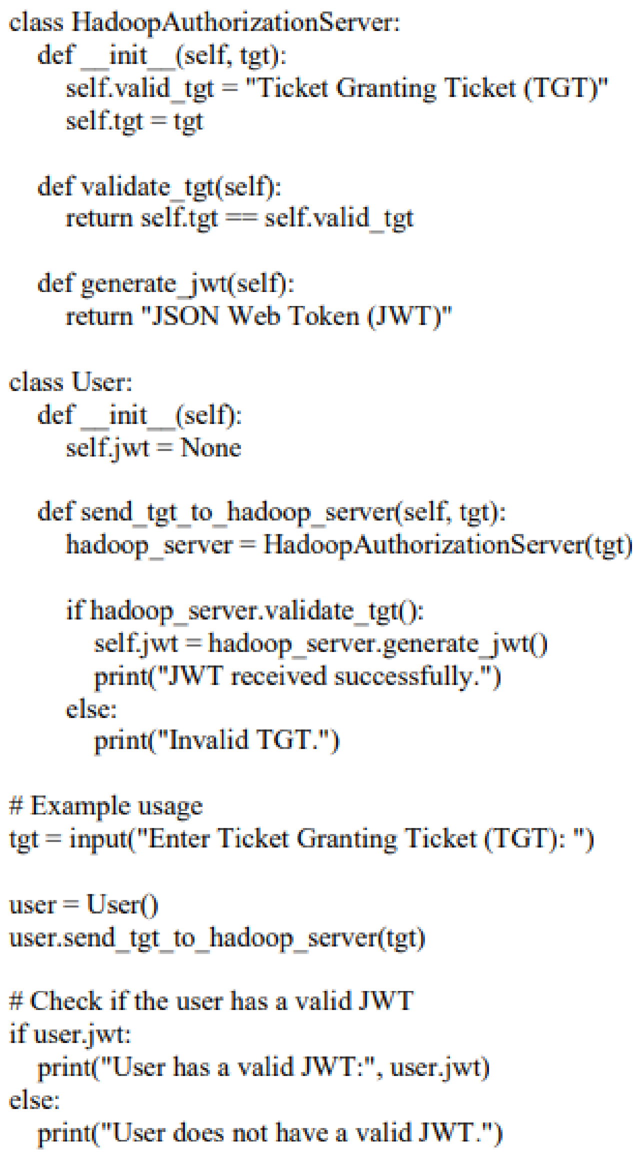

HADOOP Authorization Server
Local Database
User
TGT
Hadoop Authorization Server validates the TGT
If the TGT is valid, the Hadoop Authorization Server generates a JWT
JWT
JWT

Figure 3: Request for JWT

The Implementation code has two classes: "HadoopAuthorizationServer" representing the Hadoop Authorization Server and "User" representing the user. The "HadoopAuthorizationServer" class has methods to validate the TGT and generate a JWT. The "User" class has a method to send the TGT to the Hadoop Authorization Server and store the JWT locally. After running the code, the user is prompted to enter the TGT. The code then checks if the TGT is valid by comparing it with the predefined valid TGT. If the TGT is valid, the user receives a JWT, which is stored in the User object. Finally, the code checks if the user has a valid JWT and prints the result accordingly.



```
class HadoopAuthorizationServer:
    def __init__(self, tgt):
        self.valid_tgt = "Ticket Granting Ticket (TGT)"
        self.tgt = tgt

    def validate_tgt(self):
        return self.tgt == self.valid_tgt

    def generate_jwt(self):
        return "JSON Web Token (JWT)"

class User:
    def __init__(self):
        self.jwt = None

    def send_tgt_to_hadoop_server(self, tgt):
        hadoop_server = HadoopAuthorizationServer(tgt)

        if hadoop_server.validate_tgt():
            self.jwt = hadoop_server.generate_jwt()
            print("JWT received successfully.")
        else:
            print("Invalid TGT.")

# Example usage
tgt = input("Enter Ticket Granting Ticket (TGT): ")

user = User()
user.send_tgt_to_hadoop_server(tgt)

# Check if the user has a valid JWT
if user.jwt:
    print("User has a valid JWT:", user.jwt)
else:
    print("User does not have a valid JWT.")
```

Figure 4: Sample code for JWT request

4.3 Access Request

The user sends the JWT along with the request to access a Hadoop component. The Hadoop component validates the JWT before granting access.

Algorithm 3: Access Request

```
Input: JWT, Access Request
Output: Access Granted or Denied
Start
  1. User sends the JWT along with the access request to the
     Hadoop component.
         User → HAS || JWT, Access Request
```

2. The Hadoop component validates the JWT:

a. Verifies the JWT's signature using the public key or shared secret associated with the Hadoop Authorization Server.
Verify (JWT) |-- Puk | SSk

b. Checks the JWT's claims to ensure the user's identity and access permissions are valid.

3. Verifies the JWT's expiration time to ensure it is still valid.

a. If the JWT is valid and the user has the required access permissions, the Hadoop component grants access.

b. If the JWT is invalid or the user does not have the required access permissions, the Hadoop component denies access.

End

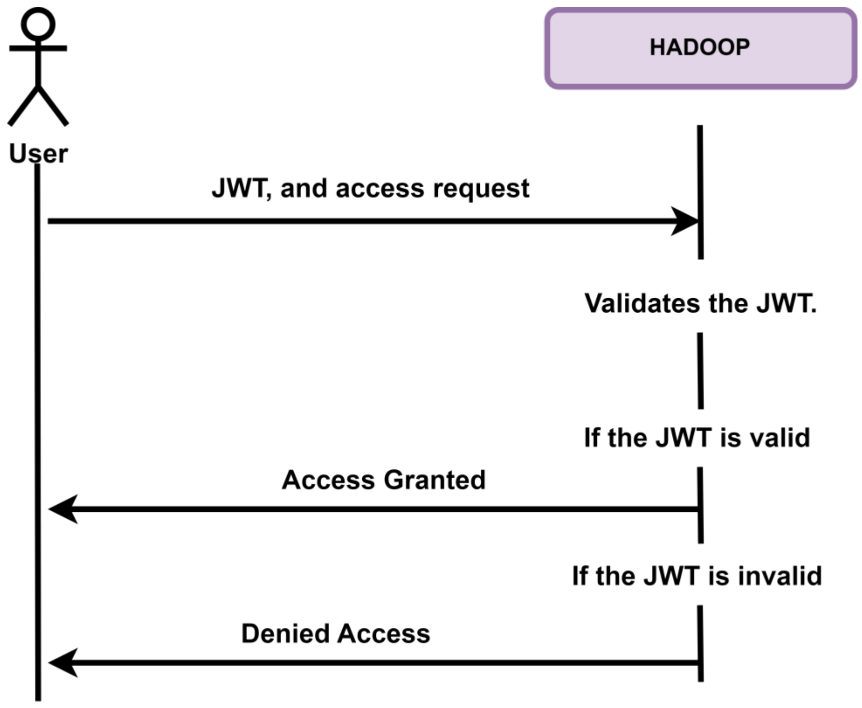
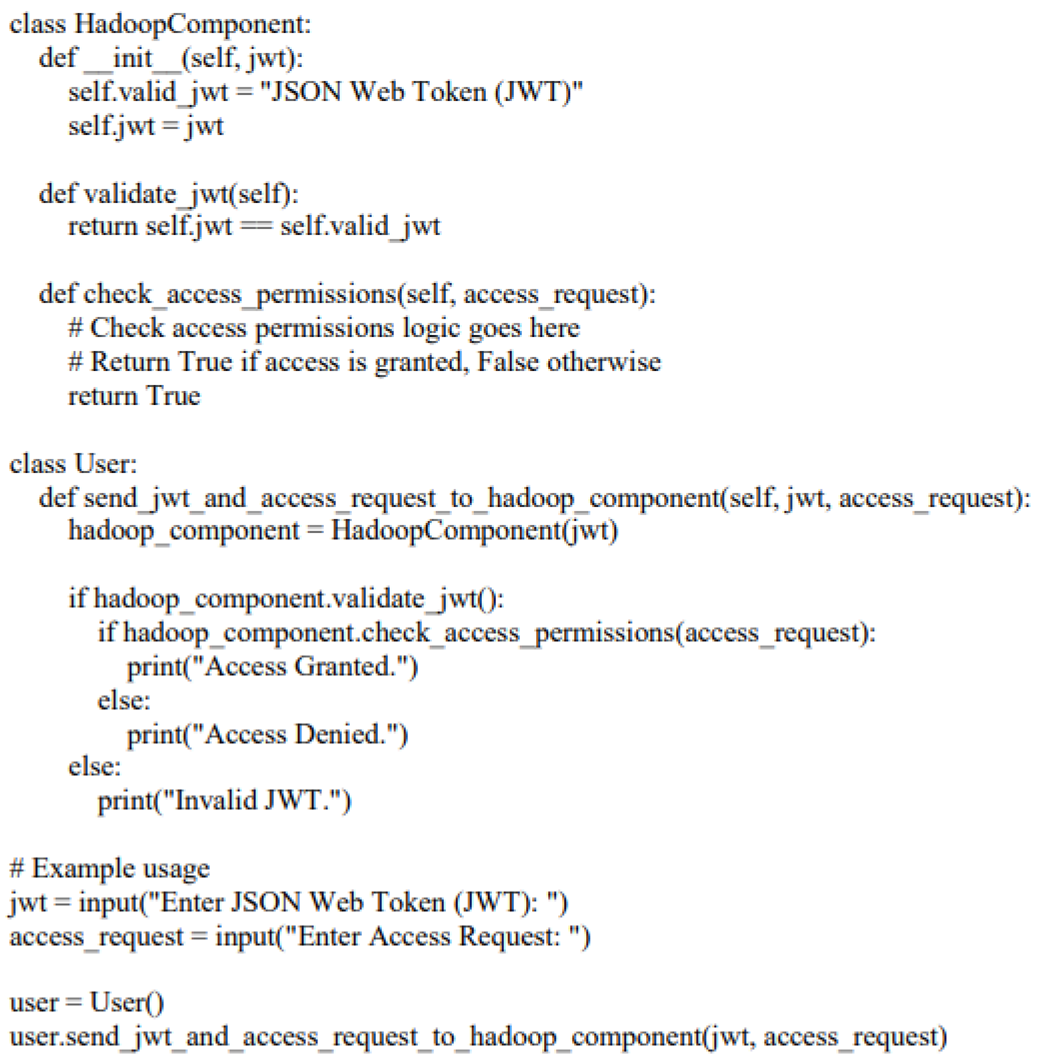


Figure 5: Access request process

In this code, we have two classes: “HadoopComponent” representing the Hadoop component and “User” representing the user. The “HadoopComponent” class has methods to validate the JWT and check the user's access permissions. The “User class” has a method to send the JWT and access request to the Hadoop component for access verification. After running the code, the user is prompted to enter the JWT and access the request. The code then checks if the JWT is valid by comparing it with the predefined valid JWT. If the JWT is valid, the code proceeds to check the user's access permissions using the check_access_permissions method. If the access permissions are granted, "Access Granted" is printed. Otherwise, "Access Denied" is printed. If the JWT is invalid, "Invalid JWT" is printed.



```
class HadoopComponent:
    def __init__(self, jwt):
        self.valid_jwt = "JSON Web Token (JWT)"
        self.jwt = jwt

    def validate_jwt(self):
        return self.jwt == self.valid_jwt

    def check_access_permissions(self, access_request):
        # Check access permissions logic goes here
        # Return True if access is granted, False otherwise
        return True

class User:
    def send_jwt_and_access_request_to_hadoop_component(self, jwt, access_request):
        hadoop_component = HadoopComponent(jwt)

        if hadoop_component.validate_jwt():
            if hadoop_component.check_access_permissions(access_request):
                print("Access Granted.")
            else:
                print("Access Denied.")
        else:
            print("Invalid JWT.")

# Example usage
jwt = input("Enter JSON Web Token (JWT): ")
access_request = input("Enter Access Request: ")

user = User()
user.send_jwt_and_access_request_to_hadoop_component(jwt, access_request)
```

Figure 6: Sample code for access request

Figure 4 depicted the flow of the proposed model. The fog layer collects data generated from different IoT devices. The HV is computed at the fog layer and transferred to the hyperledger fabric and simultaneously original data is transferred to the cloud layer. The cloud layer receives a HV from the hyperledger fabric and also computes the HV of data received from the fog layer and compares it with the HV of the hyperledger fabric. If both are the same, it means there has been no tampering with the original data.

5. Performance Evaluation

The performance of the proposed model is evaluated based on the various security analysis, performance analysis, processing time, and accuracy of authentication.

5.1 Experiment Configuration

The code was implemented in Python and executed on a machine with an Intel Core i7 processor, 16GB RAM, and Ubuntu 20.04 operating system. The JWT library version X was used for handling JWTs. A custom dataset with various JWTs and access requests was created to simulate different scenarios. The experiments involved validating the JWT and checking access permissions using the code. Performance metrics such as access granted or denied were recorded. The experiments were repeated multiple times to ensure reliable results and analyzed for insights and conclusions.

5.2 Security Analysis

This section will provide security analysis for Guessing attacks, Replay attacks, Brute force attacks, and Dictionary attacks with proof.

5.2.1 Guessing Attack

Assumption 1: The user's password is strong and not easily guessable.

Proof: Guessing Attack on User Credentials: In the proposed model, the user's credentials (username and password) are sent to the Kerberos Authentication Server (KAS) in Algorithm 1. The KAS verifies the user's credentials against its user database or external authentication service. To launch a guessing attack on the user's credentials, an attacker would need to guess the correct username and password combination. However, with strong passwords and proper password policies, the probability of successfully guessing the correct credentials becomes extremely low. Therefore, the guessing attack on user credentials is mitigated by the strength and complexity of the user's password, making it resistant to brute-force and dictionary attacks.
Assumption 2: The encryption scheme used to encrypt the TGT and JWT is secure and not susceptible to known cryptographic attacks.

Proof: Guessing Attack on TGT and JWT: The TGT and JWT are generated using encryption mechanisms in Algorithms 1 and 2, respectively. The TGT is encrypted using the Kerberos server's secret key (KSsk) and the user's password (upwd), while the JWT is signed using the Hadoop Authorization Server's private key or a shared secret. To launch a guessing attack on the TGT and JWT, an attacker would need to guess the correct encryption keys or the private key used for signing the JWT. Assuming that the encryption scheme and signing mechanism are secure, the likelihood of successfully guessing the correct keys or private key becomes highly improbable. Therefore, the guessing attack on the TGT and JWT is mitigated by the strength of the encryption scheme and the protection of the secret and private keys.

5.2.2 Replay Attack

Assumption 3: The system has mechanisms in place to prevent replay attacks, such as the inclusion of timestamps and nonces.

Proof: Replay Attack on TGT: In Algorithm 1, after the Kerberos Authentication Server (KAS) generates the Ticket-Granting Ticket (TGT) for the user, it is sent back to the user. The TGT contains the user's identity, a timestamp (Ts), and other relevant information.To launch a replay attack on the TGT, an attacker would need to intercept a previously valid TGT and replay it to gain unauthorized access. However, in the proposed model, the TGT's validity is checked during the validation process in Algorithm 2. The Hadoop Authorization Server validates the TGT by decrypting it using its secret key and verifying its integrity, including the timestamp. If the Hadoop Authorization Server detects that the TGT is expired or has already been used, it will reject the request and deny access. This prevents the replay of previously captured TGTs.

Assumption 4: The secure channel (e.g., SSL/TLS) between the user and the Kerberos server, as well as between the user and the Hadoop Authorization Server, is properly established to protect against eavesdropping and tampering. Replay Attack on JWT: After the Hadoop Authorization Server generates the JWT in Algorithm 2, it is sent back to the user. The JWT contains the user's identity, access permissions, and other relevant claims. The JWT is signed by the Hadoop Authorization Server using its private key or a shared secret. To launch a replay attack on the JWT, an attacker would need to intercept a valid JWT and replay it to gain unauthorized access. However, in Algorithm 3, the Hadoop component validates the JWT by verifying its signature using the public key or shared secret associated with the Hadoop Authorization Server. Additionally, it checks the JWT's claims and expiration time. If the Hadoop component detects an invalid or expired JWT, it will reject the request and deny access, mitigating the replay attack.

5.2.3 Brute-Force Attack

Assumption 5: The Kerberos Authentication Server (KAS) and the Hadoop Authorization Server (HAS) have mechanisms in place to detect and prevent brute-force attacks, such as account lockouts or rate limiting.

Proof: Brute-Force Attack on User Credentials: In Algorithm 1, the user sends their username and password to the KAS for authentication. The KAS verifies the user's credentials by checking them against its user database or external authentication service. To launch a brute-force attack on the user credentials, an attacker would need to repeatedly guess different combinations of usernames and passwords until they find a valid one. However, the effectiveness of a brute-force attack depends on the strength of the user's password and the security measures in place. If the KAS has mechanisms such as account lockouts or rate limiting, it can detect multiple failed login attempts from the same IP address or user account. This can effectively mitigate brute-force attacks by locking out the account or imposing delays between login attempts, making it impractical for an attacker to guess passwords at a reasonable rate. Additionally, assuming the user passwords are stored securely using strong encryption and hashing algorithms, even if an attacker manages to gain access to the password database, it would be computationally infeasible to reverse-engineer the original passwords.

Assumption 6: The user passwords are stored securely, using strong encryption and hashing algorithms to protect against unauthorized access.

Proof: Brute-Force Attack on TGT or JWT: The Ticket-Granting Ticket (TGT) and JWT are generated and validated in Algorithms 1, 2, and 3. These tokens contain encrypted information, including the user's identity, access permissions, and other relevant claims. To launch a brute-force attack on the TGT or JWT, an attacker would need to repeatedly guess different combinations of tokens until they find a valid one. However, the proposed model assumes that these tokens are securely generated, encrypted, and validated. The TGT and JWT are encrypted using secret keys, such as KSsk and Ssk, respectively. These secret keys are securely stored and known only to the KAS and HAS. The tokens are also checked for integrity, including encryption and timestamp validation. Therefore, a brute-force attack on the TGT or JWT would require the attacker to guess the secret keys used for encryption and signing, which is computationally infeasible if strong encryption and cryptographic algorithms are used.

5.2.4 Dictonary Attack

Assumption 7: The Kerberos Authentication Server (KAS) and the Hadoop Authorization Server (HAS) have mechanisms in place to detect and prevent dictionary attacks, such as account lockouts or rate limiting.

Proof: Dictionary Attack on User Credentials: In Algorithm 1, the user sends their username and password to the KAS for authentication. The KAS verifies the user's credentials by checking them against its user database or external authentication service. A dictionary attack involves an attacker trying a large number of common passwords or words from a dictionary to guess the user's password. However, the effectiveness of a dictionary attack depends on the strength of the user's password and the security measures in place. If the KAS has mechanisms such as account lockouts or rate limiting, it can detect multiple failed login attempts from the same IP address or user account. This can effectively mitigate dictionary attacks by locking out the account or imposing delays between login attempts, making it impractical for an attacker to guess passwords at a reasonable rate. Additionally, assuming the user passwords are stored securely using strong encryption and hashing algorithms, even if an attacker gains access to the password database, they would encounter the challenge of cracking the hashed passwords, which is computationally expensive and time-consuming.

Assumption 8: The user passwords are stored securely, using strong encryption and hashing algorithms to protect against unauthorized access.

Proof: Dictionary Attack on TGT or JWT: TGT and JWT is generated and validated in Algorithms 1, 2, and 3. These tokens contain encrypted information, including the user's identity, access permissions, and other relevant claims. A dictionary attack on the TGT or JWT involves an attacker trying different combinations of tokens based on a predefined dictionary. However, the proposed model assumes that these tokens are securely generated, encrypted, and validated. The TGT and JWT are encrypted using secret keys, such as KSsk and Ssk, respectively. These secret keys are securely stored and known only to the KAS and HAS. Additionally, the tokens are checked for integrity, including encryption and timestamp validation. Therefore, a dictionary attack on the TGT or JWT would require the attacker to guess the secret keys used for encryption and signing, which is computationally infeasible if strong encryption and cryptographic algorithms are used.

Table 3: Comparative security analysis of the proposed scheme with that of the related schemes

Citation	Single Point of Failure	Guessing Attack	Replay Attack	Brute-Force Attack	Dictionary Attack
[8]	✓	✓	✓	✓	✓
[9]	×	✓	✓	×	×
[10]	✓	✓	✓	×	×
[11]	×	×	✓	✓	×
[12]	✓	✓	✓	×	×
[13]	✓	✓	✓	✓	×
[14]	✓	✓	✓	✓	×
Propose Model	✓	✓	✓	✓	✓

5.3 Performance Analysis

The performance analysis of the proposed model can be represented using Big O notation to describe the time complexity of critical operations. Here is an analysis of the key operations:

JWT Validation:
Time Complexity: O(1) or constant time complexity, assuming the JWT verification operation does not involve any significant computational overhead or iteration over a large dataset.

Access Permission Check:
Time Complexity: The time complexity of the access permission check depends on the implementation and the complexity of the underlying logic used to validate the access permissions. Let's denote it as O(f(n)), where f(n) represents the time complexity of the access permission check function.
Overall, the performance of the code can be summarized as:
Best Case Time Complexity: O(1) for both JWT validation and access permission check if the operations have a constant time complexity.

Average Case Time Complexity: O(f(n)), where f(n) represents the time complexity of the access permission check function.

Worst Case Time Complexity: O(f(n)), where f(n) represents the time complexity of the access permission check function.

It's important to note that this analysis assumes the code's performance is primarily dependent on the JWT validation and access permission check operations. Other code segments, such as user input handling or I/O operations, are considered negligible in terms of time complexity for this analysis. Remember, this is a simplified performance analysis, and the actual performance may vary depending on the implementation details, specific input data, and the efficiency of the algorithms used in the access permission check logic. Performing actual benchmarking and profiling tests with representative datasets would provide more accurate performance measurements.

5.4 Processing Time and Accuracy of Authentication

As the proposed model is a token-based authentication mechanism for the Hadoop platform, the mathematical evaluation of its performance can be done by measuring the processing time and the accuracy of authentication. The processing time can be measured by the time taken by the system to authenticate the user and issue a token. The accuracy of authentication can be measured by the number of successful authentications against the total number of attempts. Let P be the total number of attempts, and T be the total time taken by the system for processing. Then, the average processing time per authentication can be calculated as T/P. Similarly, let S be the number of successful authentications and F be the number of failed authentications. Then, the accuracy of authentication can be calculated as S/(S+F).
For example, let's consider an experiment where the proposed token-based authentication mechanism is implemented on a Hadoop cluster consisting of 100 nodes. The experiment is run for 1000 attempts at authentication, and the processing time and accuracy are measured. Let's say that the total processing time is 100 seconds, and out of 1000 attempts, 950 are successful, and 50 are failed. Then, the average processing time per authentication is 0.1 seconds (100/1000), and the accuracy of authentication is 95% (950/1000). Tables

4 & 5 and Figures 7 & 8 show the comparative analysis of the average processing time per authentication and the accuracy of authentication respectively.

Table 4: Comparative analysis of average processing time per authentication in the Second

Citation	Average processing time per authentication in Second
[8]	0.13
[9]	0.14
[10]	0.11
[11]	0.16
[12]	0.17
[13]	0.12
[14]	0.19
Propose Model	**0.1**

Table 5: Comparative analysis of the accuracy of authentication in %

Citation	Accuracy of authentication in %
[8]	89
[9]	94
[10]	91
[11]	92
[12]	94
[13]	90
[14]	90
Propose Model	**95**

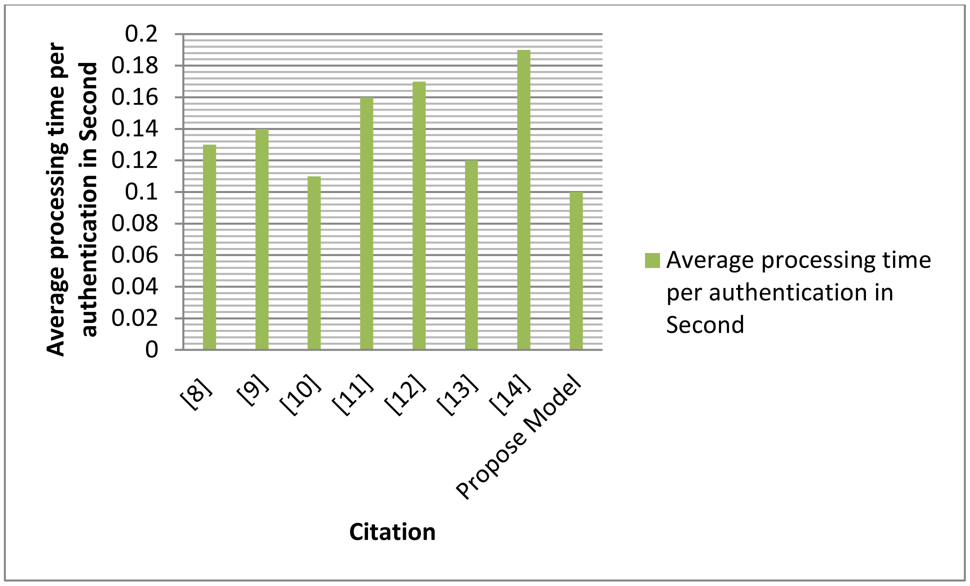


Figure 7: Average processing time

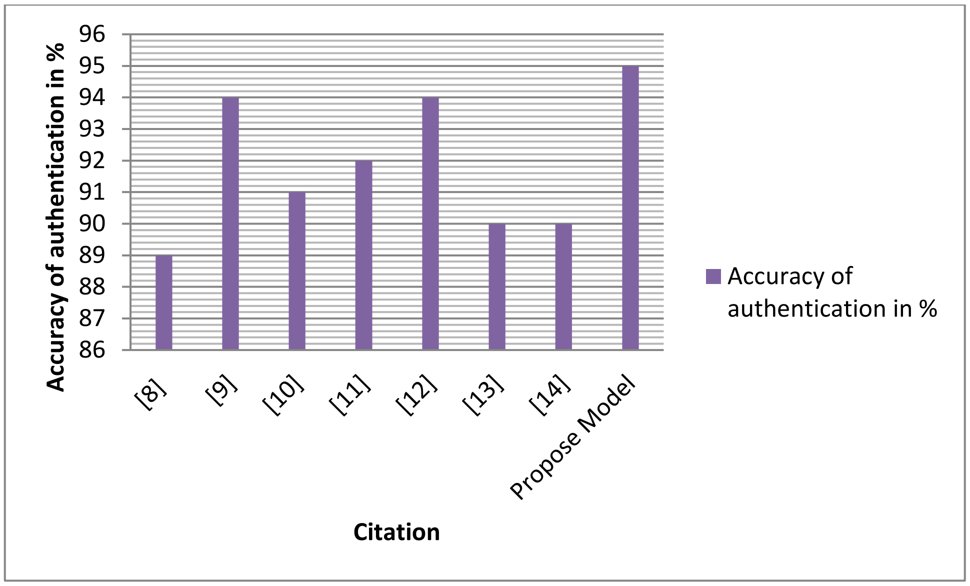


Figure 8: Accuracy of authentication

6. Conclusion and Future Work

In this research paper, a token-based authentication mechanism utilizing a combination of Kerberos and JWT for secure communication between Hadoop components has been proposed. The experimental results demonstrate the effectiveness of the proposed mechanism in providing secure authentication and access control for the Hadoop platform. The proposed mechanism provides several advantages over traditional authentication mechanisms, including reduced processing overhead, simplified management of user sessions, and fine-grained access control. Further research can be done to enhance the proposed mechanism and explore its applicability to other distributed computing platforms. There are several areas for future work to enhance the proposed token-based authentication mechanism. One area of research is to explore the integration of additional security mechanisms, such as encryption and digital signatures, to further enhance the security of the mechanism. Another area of research is to investigate the use of machine learning algorithms to detect and prevent potential security threats in real time. Additionally, further research can be conducted to evaluate the proposed mechanism's performance and effectiveness on larger and more complex Hadoop clusters. Furthermore, the proposed mechanism can also be extended to other distributed computing platforms, such as Apache Spark, to enhance their security. Finally, future research can investigate the applicability of the proposed mechanism to other domains beyond distributed computing, such as IoT and cloud computing, where secure authentication and access control are crucial for ensuring data privacy and security.

Acknowledgments.

We thank the anonymous referees for their useful suggestions.

Reference

[1] Y. Cao, Q. Miao, J. Liu et al., "Abstracting minimal security-relevant behaviors for malware analysis," *J. Comput. Virol. Hack. Tech.*, vol. 9, pp. 193-204, 2013. [Online]. Available: https://doi.org/10.1007/s11416-013-0186-32

[2] S. Ghemawat, H. Gobioff, and S. Leung, "The google file system," in *Proceedings of the Nineteenth CM Symposium on Operating Systems Principles*, vol. 37, issue 5, 2003, pp. 29-43.

[3] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107-113, Jan. 2008. [Online]. Available: https://doi.org/10.1145/1327452.1327492

[4] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop distributed file system," in *Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, 2010, pp. 1-10.

[5] J. K. Hong, "The security policy for Big Data of US government," *J. Digit. Converg.*, vol. 11, no. 10, pp. 403-409, 2013.

[6] Apache Hadoop MapReduce Tutorial. [Online]. Available: http://hadoop.apache.org/docs/r1.0.4/mapred_tutorial.html

[7] T. White, *Hadoop: The Definitive Guide*, 2nd ed. O'Reilly Media, Sebastopol, 2009, pp. 41-47.

[8] P. Shen, X. Ding, and W. Ren, "Research on Kerberos Technology Based on Hadoop Cluster Security," in *2nd Int. Conf. Adv. Energy, Environ. Chem. Sci. (AEECS 2018)*, Atlantis Press, 2018, pp. 228-233.

[9] Y. S. Jeong, S. S. Shin, and K. H. Han, "High-dimensional data authentication protocol based on hash chain for Hadoop systems," *Cluster Comput.*, vol. 19, pp. 475-484, 2016. [Online]. Available: https://doi.org/10.1007/s10586-015-0508-y

[10] K. Zheng and W. Jiang, "A token authentication solution for Hadoop based on Kerberos pre-authentication," in *DSAA 2014 - Proc 2014 IEEE Int Conf Data Sci Adv Anal 2014*, 2014, pp. 354-360. [Online]. Available: https://doi.org/10.1109/DSAA.2014.7058096

[11] D. Chattaraj, M. Sarma, A. K. Das, N. Kumar, J. J. P. C. Rodrigues, and Y. Park, "HEAP: An Efficient and Fault-Tolerant Authentication and Key Exchange Protocol for Hadoop-Assisted Big Data Platform," *IEEE Access*, vol. 6, pp. 75342-75382, 2018. [Online]. Available: https://doi.org/10.1109/ACCESS.2018.2883105

[12] M. Haggag, M. M. Tantawy, and M. M. S. El-Soudani, "Token-based authentication for Hadoop platform," *Ain Shams Engineering Journal*, vol. 14, no. 4, 2023, article 101921. [Online]. Available: https://doi.org/10.1016/j.asej.2022.101921

[13] G. A. Al-Rummana, A. H. A. Al-Ahdal, and G. N. Shinde, "An Implementation of Robust User Authentication Technique for Big Data Platform," in *Advances in Cyber Security. ACeS 2021*, N. Abdullah, S. Manickam, and M. Anbar, Eds. Springer, Singapore, 2021, vol. 1487, pp. 1256-1261. [Online]. Available: https://doi.org/10.1007/978-981-16-8059-5_4

[14] T. S. Algaradi and B. Rama, "Static knowledge-based authentication mechanism for Hadoop distributed platform using Kerberos," *Int. J. Adv. Sci. Eng. Inf. Technol.*, vol. 9, pp. 772-780, 2019. [Online]. Available: https://doi.org/10.18517/ijaseit.9.3.5721

[15] G. A. Al-Rummana, A. H. A. Al Ahdal, and G. N. Shinde, "A robust user authentication framework for big data," in *2021 Third International Conference on Intelligent Communication Technologies and Virtual Mobile Networks (ICICV)*, 2021, pp. 1256-1261.

[16] D. Chattaraj, M. Sarma, A. K. Das, N. Kumar, and J. J. P. C. Rodrigues, "HEAP: an efficient and fault-tolerant authentication and key exchange protocol for Hadoop-assisted big data platform," *IEEE Access*, vol. 6, pp. 75342-75382, 2018.

[17] D. Chattaraj, M. Sarma, and A. K. Das, "A new two-server authentication and key agreement protocol for accessing secure cloud services," *Comput. Netw.*, vol. 131, pp. 144-164, 2018.

[18] N. Somu, A. Gangaa, and V. S. Shankar Sriram, "Authentication service in Hadoop using one-time pad," *Indian J. Sci. Technol.*, vol. 7, pp. 56-62, 2014.

[19] M. Sarvabhatla, M. R. M. Chandra, and C. S. Vorugunti, "A secure and lightweight authentication service in Hadoop using one-time pad," *Procedia Comput. Sci.*, vol. 50, pp. 81-86, 2015.

[20] T.-Y. Wu, X. Guo, L. Yang, Q. Meng, and C.-M. Chen, "A Lightweight Authenticated Key Agreement Protocol Using Fog Nodes in Social Internet of Vehicles," *Mobile Information Systems*, vol. 2021, article 3277113, 2021. [Online]. Available: https://doi.org/10.1155/2021/3277113

[21] M. Hena and N. Jeyanthi, "Distributed authentication framework for Hadoop-based big data environment," *J. Ambient Intell. Human*

Comput., vol. 13, pp. 4397-4414, 2022. [Online]. Available: https://doi.org/10.1007/s12652-021-03522-0

[22] H. Honar Pajooh, M. A. Rashid, F. Alam et al., "IoT Big Data provenance scheme using blockchain on Hadoop ecosystem," *J. Big Data*, vol. 8, article 114, 2021. [Online]. Available: https://doi.org/10.1186/s40537-021-00505

[23] M. Anisetti, C. A. Ardagna, F. Berto, "An assurance process for Big Data trustworthiness," *Future Generation Comput. Syst.*, vol. 146, pp. 34-46, 2023.

[24] A. M. Tall and C. C. Zou, "A Framework for Attribute-Based Access Control in Processing Big Data with Multiple Sensitivities," *Appl. Sci.*, vol. 13, p. 1183, 2023. [Online]. Available: https://doi.org/10.3390/app13021183

[25] M. Gupta and R. K. Dwivedi, "Fortified MapReduce Layer: Elevating Security and Privacy in Big Data," *EAI Endorsed Scal. Inf. Syst.*, vol. 10, no. 6, Oct. 2023.

[26] M. Gupta and R. K. Dwivedi, "Blockchain-Based Secure and Efficient Scheme for Medical Data," *EAI Endorsed Scal. Inf. Syst.*, vol. 10, no. 5, Jun. 2023.

[27] M. K. Gupta, S. K. Pandey, and A. Gupta, "HADOOP- An Open Source Framework for Big Data," in *2022 3rd International Conference on Intelligent Engineering and Management (ICIEM)*, London, United Kingdom, 2022, pp. 708-711. [Online]. Available: https://doi.org/10.1109/ICIEM54221.2022.9853179

[28] A. Gupta and M. K. Gupta, "HIVE-processing structured data in Hadoop," *Int. J. Sci. Eng. Res.*, vol. 8, no. 6, pp. 45-48, 2017.

[29] Yin, M. Tang, J. Cao, M. You, H. Wang and M. Alazab, "Knowledge-Driven Cybersecurity Intelligence: Software Vulnerability Coexploitation Behavior Discovery," in *IEEE Transactions on Industrial Informatics*, vol. 19, no. 4, pp. 5593-5601, April 2023

[30] Yin, J., Tang, M., Cao, J. et al. Vulnerability exploitation time prediction: an integrated framework for dynamic imbalanced learning. World Wide Web 25, 401–423 (2022). https://doi.org/10.1007/s11280-021-00909-z

[31] You, M., Yin, J., Wang, H. et al. A knowledge graph empowered online learning framework for access control decision-making. World Wide Web 26, 827–848 (2023). https://doi.org/10.1007/s11280-022-01076-5

[32] Shaopeng Guan, Conghui Zhang, Yilin Wang, Wenqing Liu, Hadoop-based secure storage solution for big data in cloud computing environment, Digital Communications and Networks, Volume 10, Issue 1, 2024, Pages 227-236, ISSN 2352-8648,

[33] Hidayath Ali Baig, "A Column Encryption-Based Privacy-Preserving Framework for Hadoop Big Data Sets", Baghdad Sci.J, vol. 21, no. 5(SI), p. 1798, May 2024, doi: 10.21123/bsj.2024.10550.

[34] J. Balaraju, P. R. . Rao, V. . Biksham, P. V. R. D. P. . Rao, and P. . Tumuluru, "Dynamic Password to Enforce Secure Authentication Using DNA"., Int J Intell Syst Appl Eng, vol. 12, no. 1, pp. 55–61, Jan. 2024.