Towards a Virtual Domain based Authentication on MapReduce

DOI:
10.1109/ACCESS.2016.2558456

Document Version
Accepted author manuscript

Link to publication record in Manchester Research Explorer

Citation for published version (APA):

Published in:
IEEE Access

Citing this paper
Please note that where the full-text provided on Manchester Research Explorer is the Author Accepted Manuscript or Proof version this may differ from the final Published version. If citing, it is advised that you check and use the publisher's definitive version.

General rights
Copyright and moral rights for the publications made accessible in the Research Explorer are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

Takedown policy
If you believe that this document breaches copyright please refer to the University of Manchester’s Takedown Procedures [http://man.ac.uk/04Y6Bo] or contact uml.scholarlycommunications@manchester.ac.uk providing relevant details, so we can investigate your claim.
Towards a Virtual Domain based Authentication on MapReduce

Ibrahim Lahmer  
School of Computer Science  
The University of Manchester  
Manchester, M13 9PL, UK  
ibrahim.lahmer@manchester.ac.uk

Ning Zhang  
School of Computer Science  
The University of Manchester  
Manchester, M13 9PL, UK  
ning.zhang@manchester.ac.uk

Abstract— This paper has proposed a novel authentication solution for the MapReduce (MR) model, a new distributed and parallel computing paradigm commonly deployed to process BigData by major IT players such as Facebook and Yahoo. It identifies a set of security, performance and scalability requirements that are specified from a comprehensive study of a job execution process using MR and security threats and attacks in this environment. Based on the requirements, it critically analyses the state-of-the-art authentication solutions, discovering that the authentication services currently proposed for the MR model is not adequate. The paper then presents a novel layered authentication solution for the MR model and describes core components of this solution which includes the Virtual Domain based Authentication Framework (VDAF). These novel ideas are significant, because, firstly, the approach embeds the characteristics of MR-in-Cloud deployments into security solution designs, and this will allow the MR model be delivered as a Software as a Service (SaaS) in a public cloud environment along with our proposed authentication solution; secondly, VDAF supports the authentication of every interactions by any MR components involved in a job execution flow, so long as the interactions are for accessing resources of the job; and thirdly, this continuous authentication service is provided in such a manner that the costs incurred in providing the authentication service should be as low as possible.


I. INTRODUCTION

MapReduce (the MR model) is a new parallel programming paradigm. It is proposed to process large volumes of data. Data processing is carried out in two phases: map and reduce. The map phase takes a set of data and converts it into another set of data called key/value pairs to produce the intermediate results of the MR computation. The reduce phase then takes these intermediate results as its input and combines these data to produce an output and this output is the final result of the MR computation. More details as how MR works can be found in [1]-[3]. To carry out the two-phase MR computation, a set of distributed nodes (hereafter referred to as MR components) are used. Figure 1 shows a Generic MapReduce Computational (GMC) model that we have constructed based on the most recent MR application framework [1]-[3]. From the figure, it can be seen that a distributed set of MR components interact with each other and collaboratively execute a client’s job. The entire process for this job execution, i.e. from when the job is submitted to when the final computational result is ready for collection, is referred to as a job execution flow (or a job work-flow). The MR components can generally be classified into two main categories: master nodes and slave nodes. The Resource Manager and Name Node, shown in Figure 1, are examples of master nodes, and the rest are slave nodes. In this version of the MR model implementation, a client submits his job to the Resource Manager. The Resource Manager assigns the tasks of the job to a set of slave nodes that contains containers to run the Map and Reduce Tasks. However, in the classic MR model implementation [1] [2], a client submits a job to the Job Tracker directly and the Job Tracker then assigns Map and Reduce Tasks to a set of slave nodes (indicated in Figure (1) by using dash-dot lines labeled as ‘(3), (4), and (7)’). The two sets of MR components, respectively run on two large clusters of nodes are typically referred to as the Processing Framework (PF) cluster and Distributed File System (DFS) cluster [3]. The GMC model, shown in Figure 1, is derived to capture the interactions among different MR components in the newer MR model implementation (although what has been captured can also be applied to the classic MR model implementation). More details about the MR components, and their functionalities, of both versions of the MR model implementations (i.e. MR application frameworks) are available in [2].

The MR model, owing to its scalability, robustness and simple to use as a parallel and distributed programming framework, is becoming more and more widely used [4] [5]. Hadoop, an implementation of the MR model, has been adopted by many companies including the major IT players in the world such as Facebook, eBay, IBM and Yahoo. These implementations are largely done in their respective private clouds. However, recently there are efforts to implement the MR model in public clouds [6] [7].

A major concern of using the MR model in a public cloud is its inadequate security provision, such as authentication. The MR model was initially intended for use in private networks, so the issue of security was not a design consideration [8]. Since its introduction, lots of efforts have been made to improve the performance of this model making it more efficient rather than making it more secure. Deploying the MR model in an open environment, such as public clouds, without adequate security provisioning would put the clients’ jobs and their data at risks. This is because, in such an environment, different jobs submitted by different clients typically share the same set of physical nodes and software resources. The clients have very little control over (1) on which nodes their MR components (assigned to their respective jobs) are executed, and (2) on which DFS nodes the data associated to their jobs are stored. These could make the jobs and the data more vulnerable to security threats and attacks [1] [9]-[11].
Our work focuses on addressing identity related threats and attacks in deploying the newer version of the MR model in an open environment. To understand the security issues in this context and to capture the requirements necessary to address the issues, in this paper, we categorise the MR components involved in a job execution flow into two categories, MR Infrastructure (MR-Inf.) Components and MR-Job Components. MR-Inf. Components are the MR components that serve every job submitted by any clients. These components are not job specific. Examples of MR-Inf Components are Resource Manager and Name Node. MR-Job Components are the MR components that are invoked specially for a particular job submitted by a client. This set of components is job specific and their invocations and existence are purely for serving this particular job. Examples of MR-Job Components are Job Tracker (also called Application Master), Task Tracker (i.e. Node Manager) and Map and Reduce Tasks (i.e. Containers). As indicated in Figure 1, running the MR model in an open environment, three observations can be made: (1) clients typically access the MR application remotely via the Internet, (2) each client’s input data are partitioned and stored on a set of distributed and shared Data Nodes, (3) the MR components that are involved in executing the tasks (MR-Job Components) sprawled for a single client’s job are executed by multiple nodes, and these nodes may also host the tasks sprawled for other jobs submitted by other clients. An
authentication solution designed to secure the jobs and their data in such an environment should consider three aspects, and these are: (a) the authentication of a Client to the MR application (i.e. Client-to-MR authentication), (b) the mutual authentication among MR components (i.e. MR-Comp-to-MR-Comp authentication), and (c) data authenticity (i.e. Data-Authenticity, which covers both origin authentication and integrity protections). Client-to-MR authentication is to guard the entry gate to the MR application making sure only authorised users (i.e. the clients of the MR application) could submit jobs to the MR application. In other words, the authentication solution should be able to verify that a client who seeks to submit a job to the MR application is indeed whom he claims to be. MR-Comp-to-MR-Comp authentication is to make sure that an MR component seeking to retrieve any resources associated to a client’s job is whom it claims to be. The third aspect, Data-Authenticity, is to protect the authenticity (i.e. origin and integrity) of data generated in both map and reduce phases, making sure that any unauthorised access of, and/or alterations made, to the data can be detected.

The importance of addressing the above authentication issues and the requirements that should be satisfied by an authentication solution designed for the MR model have been discussed in literature [1] [9]. However, so far, little has been done in term of designing such a solution. As part of our effort on designing a secure and effective authentication solution for the MR model, in this paper, we critically analyse the state-of-the-art MR authentication methods. The purpose of this critical analysis is to examine the suitability and effectiveness of existing authentication methods (proposed for the MR model) taking into considerations of the features and characteristics of an MR application in an open environment such as a public cloud, so as to identify areas for improvement. It should be mentioned that our analysis of existing authentication methods proposed for an MR application has been previously published in [24]. However, this paper extends this analysis by (i) specifying design requirements for such an authentication solution, and analysing existing authentication methods proposed for the MR model against these requirements, (ii) further analysing what are missing in these methods in light of the features and characteristics of the MR model being deployed in an open environment, and (iii) providing a high level analysis of the MR model and its components in executing a client’s job, highlighting the functionalities of, and the interactions among, the MR components, (iv) proposing a novel approach to MR authentication, a layered authentication solution to the MR model that supports the newer version of the MR implementation. This solution is proposed to tackle the missing bits we have identified in existing authentication solutions designed for MR.

In detail the remaining part of this paper is structured as follows. Section 2 specifies a set of authentication requirements based on our observations on, and security analysis of, the MR model being deployed in an open environment. In Sections 3 and 4, we critically analyse the existing work on MR authentication against the specified requirements. The analysis covers the authentication methods already adopted by the MR model (Section 3), and those recently proposed in literature for the MR model (Section 4). Section 5 gives a high-level analysis of the MR model, highlighting the functionalities of, and the interactions among the MR components in executing a client’s job, and this analysis leads to our novel proposal, a layered authentication solution to the MR model. Finally, Section 6 concludes the paper with further discussions and outline of our future work.

II. REQUIREMENTS for an MR AUTHENTICATION SERVICE

This section specifies a set of requirements for the design of an authentication service for an MR application implemented in an open environment. The specification of the requirements has taken into account of the characteristics of the implementation and the outcome of a threat analysis carried out on the MR model. Related work has been reported in [1][3][9].

(R1) Entity Identification and Credential Revocation

To authenticate clients, MR components and jobs submitted to the MR application, each of these entities (or components) should have a unique identifier. The names (acronyms) of the identifiers along with entities they each represent are given in the following:

(1) Clients IDs (Client-ID): a unique identifier for each client. This is usually a static ID, and it is typically the username of a user who has registered with the MR application and is running the client (i.e. MR-Client).

(2) MR-Inf. Components IDs (MR-Inf.Comp-ID): Each MR-Inf. Component should have a unique identifier and these identifiers are static ones.

(3) MR-Job Components IDs (MR-JobComp-ID): Each set of MR-Job Components serving a particular job should have a unique identifier to identify this set of MR-Job Components from the sets of MR-Job Components serving other jobs’. These IDs are dynamic ones.

(4) MR-Job Hosting Nodes IDs (MR-JobHostNode-ID): Each MR-Job-HostNode should be uniquely identifiable, and these IDs are also static identifiers.

(5) MR-Jobs IDs (MR-Job-ID): Each MR-Job should have a unique identifier to distinguish different jobs submitted by the same client or by different clients.

(6) Framework (cluster) ID: If there are two or more parties providing hosting nodes, then the hosting nodes provided by a single party may be treated as one cluster, and each cluster should be identified by a unique identifier.

Authentication is carried out by demonstrating (by a claimant), and verifying (by a verifier), the knowledge of a secret uniquely associated to an identity. Therefore there is a need for secure issuance, acquisition and revocation of an identity secret (which is also part of the corresponding credential). This leads to the following requirements:

(R1.2) Entity Identification (or. registration): There should be a secure method for a new client or a new MR component to be identified by the MR application and to establish secret associated to the identity.
(R1.2) **Credential Revocation:** There should be secure methods for revoking any credential(s) issued to the identity of an entity involved in a job execution at any point during the job execution or after the job execution. This may take place when a job is completed, or when a related MR-JobHostNode fails or is disconnected.

(R2) **Entity Authentications**

Entity authentication is to make sure that a communicating entity is the one that it claims to be. Multiple entities (i.e. components) in the MR model are involved in a job (MR-Job) execution. Some of these components are static components, while others are dynamic ones. The static components are identified by static identities that, once given, remain the same during the lifetimes of the components. The static components can be further classified into two groups: one is MR Clients and the other is MR-Inf. Components. MR-Inf. Components are shared by different MR-Jobs, Resource Manager and Name Node, shown in Figure 1, are MR-Inf. Components, so they are static components and their identities are static too. The dynamic components are identified by dynamic identities. A dynamic identity is assigned to a dynamic component when the component is assigned to an MR-Job. If this MR-Job is completed in which case the component may be assigned to another MR-Job, and if this is the case, this component will be assigned with a new identity. Job Tracker, Task Tracker and Map and Reduce tasks (i.e. Containers) are dynamic components and they are identified by dynamic identities. In an authentication solution designed for the MR model, all the components taking part, or being involved in a job execution, being static or dynamic, should be securely identified and authenticated. In detail, with reference to the MR model depicted in Figure 1, the authentication task should satisfy the following requirements:

(R2.1) **Mutual Authentication between an MR Client and an MR-Inf. Component:** This is to ensure that only an authorized client can connect to the MR application. Hereafter this is referred to as the Client-to-MR-App authentication and MR-App-to-Client authentication. More specifically, this should cover the mutual authentication between a Client and the Resource Manager and between a Client and the Name Node.

(R2.2) **Mutual Authentication between an MR-Job Component and an MR-Inf. Component:** This is to ensure that an MR-Job component involved in the execution of a client’s job is authenticated to the MR application, so as to ensure that any access to a client’s job input and output data can be granted in a secure manner. The mutual authentication between the Job Tracker of a job and the Name Node, and between a Reduce Task of the job and the Name Node are examples of this authentication requirement.

(R2.3) **Mutual Authentication between any pair of MR-Job Components:** This is to ensure that any access to a client’s job intermediate data can be granted in a secure manner.

(R2.4) **Mutual Authentication between an MR-Job Component’s Hosting Node and an MR-Inf Component:** This is to ensure that any new physical node assigned to hosting MR-Job Component(s) of a client’s job (e.g. a new hosting node of a Task Tracker) is authenticated to a MR-Inf. Component and vice versa. In this way, we can ensure that any two MR-Job Components’ Hosting Nodes can authenticate to each other. Hereafter this requirement is referred to as MR-Job-HostNode-to-MR-App and MR-App-to-MR-Job-HostNode authentication.

(R2.5) **Mutual Authentication between Domains (i.e. cross-provider authentication):** This authentication is needed when a third party is involved in an MR-Job, and it is to ensure that any new physical node which belongs to a third party domain and involved in hosting MR-Job Components is authenticated to the MapReduce domain where the job is submitted and mastered.

(R3) **Authenticity of Data and Protocol Messages**

(R3.1) **Data authenticity:** This is to ensure the origin authentication and integrity protection of data that are saved in, or produced by, the MR application. In other words, the protection should be applied to input data, intermediate data and output data of any job processed by the MR application.

(R3.2) **Authenticity of Protocol messages:** The origin authentication and integrity protection should also be applied to all the protocol messages facilitating the tasks of authentication in an MR application. The protocol messages are of two types: authentication requests and authentication responses.

(R4) **Confidentiality of Protocol Messages**

Confidentiality of authentication requests and replies: This is a protection of authentication requests and replies from any unauthorized disclosure. To counter eavesdropping attacks, the confidentiality of any such request or response sent between MR components throughout an MR-Job execution should be protected.

(R5) **Performance and Scalability Requirements**

(R5.1) **Minimizing Communication Overhead:** In accomplishing the task of authentication for an MR-Job, the communication overhead introduced should be as low as possible. This means that the number of authentication messages, and the length of each message should be as low as possible.

(R5.2) **Minimizing Computational Overhead:** the computational overhead incurred in accomplishing the task of authentication for an MR-Job should be as small as possible.

(R5.3) **Maximizing Scalability:** The MR application scales by simply adding new nodes (members) to the shared clusters. Any authentication solution designed for the MR application should scale similarly.

(R6) **Support for Updating of Authentication Credentials**

There may be cases where an execution of an MR-Job takes a long time, and, in such cases, for security reasons, authentication secrets or credentials may need to be renewed or updated. Therefore, any authentication solution designed for the MR application should support the renewal or updating of authentication secrets or credentials.
In the next section, we critically analyze authentication methods proposed for the MR model based on the requirements specified above. These authentication methods include those ever adopted by the MR model and also those published in literature.

III. AUTHENTICATION METHODS EVER ADOPTED BY THE MR MODEL

Two authentication methods have been adopted by the MR model so far [1] [4] [13]. The first one [1] [4], adopted in the early generation of the model, assumed the use of an independent authentication service outside the MR model, e.g. an authentication service come with the host operating system (OS) running on a physical node. This is the so called OS-based authentication method. In other words, the MR model then did not have its own authentication service. Rather it relied on the use of an authentication facility provided by the OSES of the physical nodes in which an MR application is deployed.

The second method, used in the most recently deployed MR model was proposed by O. Malley et al. from the Yahoo Hadoop team (hereafter referred to as O. Malley method) [13]. This method is symmetric key based authentication and it is largely built on the Kerberos authentication protocol. At the time of writing this paper, the Kerberos authentication protocol is still a default mode of authentication for an MR application deployed in a private cloud. [1] [8], Figure 2 summarizes the authentication process using this method. As shown in the figure, a client or MR component first authenticates itself to the authentication server. Upon successful authentication, the MR component will obtain a Ticket Granting Ticket (TGT), which is then used to acquire a service ticket. The service ticket is then used by the MR component to access resources located on other MR components. This authentication process consists of six steps (steps 1 to 6, as shown in the figure), and is identical for all the MR components in the application. Assuming that a client is to write his job into the MR application as part of a job submission process, to authenticate himself to the application, the client first makes an authentication request to the Authentication Service (AS). The AS generates a response containing a TGT, which is encrypted using a key derived from the client’s password and sent to the client. Then the client uses this TGT to request a service ticket by sending the TGT along with an authenticator to demonstrate the secret in the TGT to Ticket Granting Service (TGS). Once the client receives this service ticket, he uses it to access the Name Node in DFS. The same steps are taken by any other MR components, such as Task Trackers, to get admitted to the cluster and to access other (remote) MR components for retrieving data or other resources used by the client’s job. Once a Task Tracker (or a Client) is authenticated, obtains a service ticket and is admitted to the Master Node in the MR application, both the Task Tracker and Master Node will use the shared key in the service ticket to authenticate to each other [1] [13].

This authentication method is a one-factor authentication method. The one factor used by a client to authenticate himself to the AS is the client’s password. Knowing the password would allow any entity to acquire a service ticket in the name of the client and to access any resources granted to the client. In other words, for an attacker to impersonate a legitimate component (e.g. a client or a Task Tracker), the attacker needs to obtain a service ticket. To access the ticket, the attacker needs to know the password of the (legitimate) client to whom the ticket has been issued. If a client’s password is compromised, then all the resources assigned to the client will be at risk. In addition, the attacker could use this compromised account to launch further attacks in the MR application. In other words, the security level offered by this one-factor authentication method is the same as that offered by the password chosen by a client. If a client chooses a weak password, then the risks imposed on the MR application will increase accordingly.

With regard to communication overheads introduced by this authentication method, we should work out how many protocol messages are generated and used per job submission (i.e. in each authentication instance), while assuming the length of each such message is approximately the same. For each authentication instance, three rounds of communications are required. Two of the three rounds are between a client (or an MR component, or MR-Req-Comp, for short) and the AS, and the third round is between an MR-Req-Comp and another the MR component that manages some resources (MR-Res-Comp). Each round consists of two messages (Msg), one request (Req) and one response (Res). Table 1 shows the number of communication rounds (along with the number of protocol messages exchanged) versus the numbers of MR components involved per job.
Table 1. Number of communication rounds for MR component/s authentication using Kerberos.

<table>
<thead>
<tr>
<th># of MR Comp. requests access to MR-Resource-Comp per Job</th>
<th>Number of communication rounds (R) and messages (Req and Res) for an authentication process/es.</th>
<th>Total # of Msg.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>AS MR-Req-Comp.</td>
<td>MR-Req-Comp MR-Resource-Comp</td>
</tr>
<tr>
<td>1</td>
<td>2xR (2x Req + 2x Res)</td>
<td>1xR (1x Req + 1x Res)</td>
</tr>
<tr>
<td>2</td>
<td>4xR (4x Req + 4x Res)</td>
<td>2xR (2x Req + 2x Res)</td>
</tr>
<tr>
<td>3</td>
<td>6xR (6x Req + 6x Res)</td>
<td>3xR (3x Req + 3x Res)</td>
</tr>
<tr>
<td>N</td>
<td>2NxR(2Nx Req + 2Nx Res)</td>
<td>NxR (Nx Req + Nx Res)</td>
</tr>
</tbody>
</table>


Deploying MR in a cloud environment is a shared computational environment, and, in such an environment, there are multiple possible usecases. For example, one client may submit a single job at any given time (hereafter referred to as the OneClient-OneJob usecase), one client may submit multiple jobs simultaneously (OneClient-MultiJobs) or multiple clients and each may submit one or more jobs simultaneously (MultiClients-MultiJobs). Table 2 shows the number of communication rounds and the total number of protocol messages generated for different numbers of MR-Req-Comp each job may require in each of the three usecases. The table uses the notation, yC/zJ, to indicate the different usecases, i.e. 1C/1J for Case-1, meaning one client y=1, and one job z=1; 1C/zJ for Case-2, one client y=1, and multiple jobs z>1, and yC/zJ for Case-3, where multiple clients y>1, and multiple jobs z>1.

Table 2. Number of communication rounds for authentication in three different usecases of the MR application.

<table>
<thead>
<tr>
<th>Cases (yC/zJ)</th>
<th># Of Comp. requesting access to MR Resource (service) for each case</th>
<th>Number of communication Rounds (R) in authentication process/es</th>
<th>Total # of Msg.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>AS MR-Req-Comp.</td>
<td>MR-Req-Comp MR-Resource-Comp</td>
<td></td>
</tr>
<tr>
<td>Case 1 yC/1J</td>
<td>1 (C) + n (Comp)</td>
<td>2 (R) + 2n (R)</td>
<td>1 (R) + n (R)</td>
</tr>
<tr>
<td>Case 2 yC/zJ</td>
<td>1 (C) + z + n (Comp)</td>
<td>2 (R ) + z * 2n (R)</td>
<td>1 (R) + z * n (R)</td>
</tr>
<tr>
<td>Case 3 yC/1J</td>
<td>y (C) + y * z + n (Comp)</td>
<td>y x 2 (R) + y * z + 2n (R)</td>
<td>n (R) + y + n * z</td>
</tr>
</tbody>
</table>

Example of: (i). One Client (C) has one Job (J) with n Components (Comp) per job. (ii) One Client has z Jobs with n Components (Comp) per job. (iii) y Clients, each client has z Jobs with n Component (Comp) per job.

Figure 3 plots the results for three example cases: 1C-1J-16Comp, 1C-6J-16Comp and 7C-4J-16Comp capturing different numbers of clients, jobs and MR components in each case. For Case-3, we assume that there are 7 clients and each client submits 4 jobs. The number of components involved in each job execution is 16 MR components. Detailed values with regard to the number of clients, the number of jobs per client, the number of MR components per job, and the number of protocol messages required for authentication in each of these cases are given in the figure. It can be seen from the figure that, for Case-3, the number of protocol messages generated for the authentication of these clients and the associated MR components used for the execution of the jobs submitted by the clients reaches more than 2700. If the number of clients, and/or the number of jobs submitted per client, goes up, this message number will increase sharply.

To further examine the effects of different factors on the scalability of the solution, we have calculated the number of protocol messages required versus the number of MR components used per job, the number of job submitted per client and the number of clients submitting the jobs, respectively. Figure 4(A) shows the number of protocol messages generated versus the number of MR components used per job. The figure plots the results for further three cases by changing the number of clients (y) and the number of jobs submitted per client to {y = 1, z = 1}, {y=2, z=2}, and {y=3, z=3}, respectively. As can be seen from the figure that, if there are only three clients each submitting three jobs, then the total number of MR components required to execute these jobs is about 70, but the number of protocol messages required for authenticating the clients and the MR components are more...
than 4000. This is a significant increase in comparison with the number of clients and the number of jobs submitted by the clients, and could impose a significant risk on creating a performance bottleneck in the cluster. Figure 4(B) shows the number of protocol messages generated versus the number of submitted jobs. From the results shown in the figure, it can be seen that, when the number of clients (y) is fixed at 3, i.e. y = 3 and the number of MR components used per job at n = 30, as the number of jobs submitted per client increases from 1 to 4, the total protocol messages generated will increase from about 500 to over 2000. Figure 4(C) shows how the number of protocol messages increase as the number of clients accessing the MR application increases, where the number of jobs submitted per client and the number of MR components per jobs are fixed.

IV. AUTHENTICATION METHODS PUBLISHED IN LITERATURE

In addition to the authentication methods described above, there are also methods that have been proposed for the MR model in the research domain. These methods can largely be classified into two groups, symmetric key based and asymmetric key based. The authentication methods proposed by N. Somu [14] and S. Rubika [15] are symmetric key based, and their focus is on verifying the identities of clients requesting to access an MR application. On the other hand, the methods proposed by W. Wei et al [16], A. Ruan [18] are an asymmetric key based. They focus on verifying the authenticity of an MR component. In addition, the method proposed by A. J. Zhao et al [19] is also asymmetric key based, but this method provides both clients’ authentication and MR components’ authentication. In this section, we give an overview of these methods.

A. Somu and Rubika Authentication Methods

Somu et al. [14] proposed an authentication method (hereafter referred to as the Somu method) for the Hadoop MR model. This method is symmetric key based. It is similar to the O. Malley method in that both methods use a single authentication factor, relying on the use of a client’s username and password, to authenticate the client to the MR application. However, unlike the O. Malley method, the Somu method uses
two further ideas to strengthen the security level of the authentication service. The two ideas are: (1) the introduction of a one-time pad key (session valid only), and (2) the use of the principle of the separation of duties. The ciphertext of a client’s password, encrypted using the client’s one-time pad key, is stored in the Registration Server (one of the two servers used to implement the authentication service) and the ciphertext of the client’s one-time pad key, encrypted using the client’s password, is stored in the other server, a Backend Server. The two ideas are used in such a manner that no passwords or encrypted passwords are sent over the channel and no cleartext passwords are stored in any of the two servers, thus minimize the exposure of clients’ long-term credentials, i.e. the passwords.

Figure 5 depicts the authentication process using the Somu method. As shown in the figure, two servers (the Registration Server and the Backend Server) are involved in an authentication process. The verification makes use of three ciphertexts, Ciphertext-1, Ciphertext-3 and Ciphertext-4. Ciphertext-1 is the client’s password encrypted using a one-time pad key belonging to the client, and it is stored in the Registration Server. Ciphertext-3 is the one-time pad key encrypted with the user’s password and it is pre-stored in the Backend Server. Ciphertext-4 is generated by the Registration Server each time when an authentication request is received. It is generated by encrypting the one-time pad key using the user’s password. Figure 5 shows the steps of the Somu authentication method. First, the client sends an authentication request to the Registration Server and this request contains the client’s username. The Registration Server forwards this request to the Backend Server. The Backend Server uses the username to fetch and return Ciphertext-3 (it is pre-stored) to the client through the Registration Server. The client decrypts Ciphertext-3 using his password, and sends the pad key back to the Registration Server. These steps are indicated by messages 1, 2, 3, 4 and 5, in Figure 5. The Registration Server then uses the pad key to decrypt Ciphertext-1 to obtain the password and then uses the password to encrypt the pad key to generate Ciphertext-4. The Registration Server then sends Ciphertext-4 to the Backend Server, as indicated by messages 6, 7, and 8. Finally as indicated by messages 9, 10, 11 and 12, the Backend Server compares Ciphertext-4 with Ciphertext-3 and if the two are equal, the Backend Server will send a positive notification to the Registration Server, which contains the client’s Username. The Registration Server compares the Username received from Backend Server with the one received from the user. If they match, then the login process is successful.

The Somu authentication method supports client authentication with a stronger level of protection of clients’ long-term credentials (passwords) than the methods discussed earlier. This protection involves the use of a symmetric one-time pad key and two authentication servers. A client’s password is encrypted with the one-time pad key, the one-time pad key is encrypted with the password, and the two encrypted items are, respectively, stored on two different servers. To impersonate a client, an attacker needs to guess or obtain the client’s password. Getting hold of the client’s password by stealing the ciphertext stored on either of the two servers is computationally difficult. For example, if the attacker can steal Ciphertext-1 (the encryption of the password using the one-time pad key) from the Registration Server, to access the password, the attacker will need to guess the pad key or to use a dictionary attack to guess the password. However, this is computationally difficult as the pad key used is valid for one session only. Once the client logs off a session, a new pad key will be generated and used to reencrypt the password [14]. The dictionary attack is also subject to the difficulty brought by the use of the one-time pad key. If an attacker could steal Ciphertext-3 (i.e. the encrypted pad key using the password) from the Backend Server, then only a dictionary attack could be used to guess the password, as the encryption is not reversible here. Another advantage of this authentication method is that, similar to the O. Malley method, the Somu method does not require any transmission of clients’ long-term credentials (e.g. password) over the channel.

![Authentication steps of the Somu method](image)

However, against our requirements detailed in Section II, the Somu authentication method has two limitations. Firstly, it only supports gate-level authentication. In other words, it only supports the client’s authentication to the MR application; it
S. Rubika [15] has also proposed an authentication method (hereafter referred to as the Rubika method) for the MR application. This method uses three servers for authentication, an Authentication Server, and two backend servers, Backend Server 1, and Backend Server 2. Figure 6 shows the registration and authentication processes of this method. To register, a client submits his username and password to the Authentication Server (or a password is created for the client). The server divides the password, a set of ASCII letter, into three values, m1, m2, and m3, and it also generates three random numbers, c1, c2, and c3. Then the Authentication Server uses the two sets of values, {m1, m2, m3} and {c1, c2, c3}, to generate a new set of values called angles that are denoted as {θ1, θ2, θ3}. The Username and the random numbers {c1, c2, and c3} are stored in Backend Server 1 and the Username and {θ1, θ2, and θ3} are stored in Backend Server 2. These two sets of values are used to authenticate the client when the client makes an access request to the MR application.

As described above, the Rubika method uses three servers for authentication, but only one of the three servers, the Authentication Server, is exposed to the public (i.e., accessible to users). The other two servers, the backend servers, are used to store password-verifiers. In other words, with this approach, there is nothing related to the clients’ passwords that are stored in the server accessible by the public. In the Somu method, on the other hand, clients’ encrypted passwords are stored in the registration server which is accessible in the public. In addition, with the Rubika method, to compromise a password by stealing the password verifier, an attacker would have to compromise two servers, as each password verifier is divided into two portions and each portion is stored on a different server. These two measures make the Rubika method more secure than the Somu method. The authors also claimed that, by using the two-portion password verifiers and alienate passwords, their method is robust against replay and password guessing attacks. Additionally, although the Rubika method uses three servers, rather than two as in the case of the Somu method, the communication overhead incurred in the Rubika method is lower than the Somu method. The Rubika method only needs three rounds of requests and replies for one client authentication instance. This is one round less than the Somu method.

Figure 6. Registration and authentication processes of the Rubika method.

B. Wei’s Authentication Method

Both Somu and Rubika authentication methods are designed to support client authentication only. They do not consider the authentication issues between different MR components. Wei et al. addressed this gap by proposing a SecureMR Framework [16]. The Framework (hereafter referred to as the Wei method) is aimed at protecting the integrity of MR data processing services, namely the messages sent by Map and Reduce tasks, and the data processed or generated by the tasks. For the latter, both intermediate data and final computational results from an MR job execution are protected. For example, a Reduce task (Reducer) verifies the authenticity of intermediate data produced by a Map task (Mapper), and a client should verify the authenticity of the final result generated by a Reducer. The method also supports consistency checks of intermediate data and final results from an MR job execution. This is done by replicating some Map and Reduce tasks and assign them to different workers. At the end of the computation, the master compares the results produced
by different sets of tasks. If the results are identical, then the consistency of the results (both intermediate results or final results) is assured.

The verification process is carried out collaboratively between the Master and a worker (i.e., Mapper). Two protocol messages, Assign and Commit, are used to authenticate and verify the authenticity of both the task and data produced by the task. For example, as shown in Figure 7, to assign a Map task to a Mapper, the Master sends the Mapper an Assign message containing the ID of this Mapper, MapperID, and the location of the data, DataLocation. The Master signs the message using his private key and then encrypts the message with the Mapper’s public key. When the Mapper receives the Assign message, the Mapper decrypts the message by its private key and verifies the signature using the Master public key. Upon positive verification, the Mapper executes the task assigned. After the task execution is completed. The Mapper hashes each partition of the computational result (intermediate data) and signs the hashed values by his private key, and then constructs and sends a Commit message to the Master. The Commit message contains the signed data partitions of the result. Upon the receipt of this message, the Master verifies the Commit message using the Mapper’s public key. If the Master receives more than one Commit message from different Mappers but for the same map task (replicated task), the Master will compare the signed values contained in the different Commit messages to see if they are consistent with each other [16].

The above method is also used to ensure the authenticity of any intermediate data assigned to a Reducer by a Mapper. The Reducer verifies the authenticity of the intermediate data which are produced by the Mapper using the Mapper’s public key. However, the method used to verify the authenticity of the final result produced by an MR job execution is different from the one discussed above. In the latter case, a secure verification component is installed into the MR client application, the Master and client verify the authenticity of the output data by using an additional phase, called Verify phase, [16].

In addition to achieving message and data authenticity, the Wei method also protects the confidentiality of protocol messages (i.e. Assign and Commit messages). This is done by encrypting the entire protocol message with the recipient’s public key (after signing the message with sender’s private key).

The major difference between the Wei method and the Soma and Rubika methods is that the Wei method ensures the authenticity of messages sent from one MR-Job Component to another and from an MR-Job component to an MR-Inf Component and the data or results produced by the MR-Job components. These protections are provided by using digital signatures, so the method also provides the property of non-repudiation of origin protecting against false denial of having generated or transmitted a message. However, as discussed in [12] and [17], a public key cryptosystem is computationally more costly in comparison with a symmetric key cryptosystem, especially when it is applied to a large-scale computational environment such as a Cloud environment where a large number (possibly hundreds or thousands [5]) of jobs may need to be processed and a large number of distributed components are involved. Furthermore, the Wei method has an extra phase (verification phase) in addition to the map and reduce phases. This extra phase is used to verify the authenticity of the final result produced by an MR job execution. The performance evaluation presented in the paper has not considered the costs as introduced by this extra verification phase; it has only considered the communication costs of these scenarios: Master-to-Mapper, Master-to-Reducer, and Mapper-to-Reducer.

C. Ruan’s Authentication Method

Ruan et al have proposed a trust-based authentication solution for the MR application, called a Trusted MapReduce (TMR) Framework [18]. The TMR Framework uses the notion of trust and a public key cryptosystem based authentication method to facilitate the authentication between MR components. The authentication process is carried out in two phases. The first phase is for initial trust (attestation) establishment, and is carried out when an MR component (e.g., a worker) sends a connecting request to another MR component (e.g., a master). The second phase is for periodical trust updates between the worker and the master, and it is carried out regularly during the lifetime of the job execution. When a worker first registers with a master, it generates a pair of public and private keys, and this pair of keys is called an Attestation Identity Key (AIK) pair. The worker then sends the public key to the master.

This TMR Framework is similar to the Wei method in that it uses a public-key cryptosystem based authentication and it is different in that it can provide continuous authentication between different MR components. However, the TMR Framework design has not considered the authentication of a client to the MR application, nor the issue of secure distribution of public keys. It assumes that the AIK public key should either be certified by a trusted third party (e.g. Privacy-CA) [18] before run-time, or sent in a secure channel from one MR component (i.e., worker) to another. In addition, with this method, the master has to keep the public keys of all the workers to provide continuous authentication between the master and each worker.

D. Zhao’s Authentication Method

J. Zhao et al have proposed an authentication method to support the authentication of a client to an MR application and authentication between a pair of MR components [19]. A user logs into the master node (of the MR application) using his username and password. The master node has a Database that contains users’ login information along with their access rights.
The master node verifies the password submitted by the user. If the verification is positive, the user will be allowed to submit a job to the MR application and a user instance is created for the user to indicate that the user has an active job. The subsequent authentication between the MR components associated to the user instance (i.e. job) is achieved by using two types of certificates, proxy and slave certificates. The proxy certificates is used to authenticate the Job Tracker (the master node), linked to this user instance, to Task Tracker (the slave node, i.e. the worker), while the slave certificate is used to authenticate the slave node to the master node. The proxy certificate contains the public key of the master node and CA-ID (Certificate Authority Identity). The slave certificate contains the public key of the corresponding slave node along with the CA-ID. When the master applies for a proxy certificate for a user instance, a secure connection is set up between the master node and a Certificate Authority (CA) using the Secure Socket Layer (SSL) protocol. In this way, both the CA and the master node can be authenticated to each other by using this protocol. Then the master node generates a pair of public and private keys (Mpub and Mprv) for the user instance. The master node keeps the private key and sends the public key to the CA through the secure channel just established. The master node also generates a user session which will be used for later communication with the allocated slave nodes. The CA adds some information such as key life time to form the first part of the proxy certificate and signs it with CA’s private key. The same generation and certification process is also applied to the corresponding slave certificate. The proxy certificate is sent to all the slave nodes that are involved in the user instance (job), and the slave certificates are sent to the master node.

This method provides mutual authentication between a master and a set of slave nodes involved in a user’s job. This certificate based mutual authentication can mitigate a number of threats such as Man-In-The-Middle (MITM) attack between the master and slave nodes. A handshaking protocol is used to facilitate the mutual authentication. Also, as a secure channel is used between the CA on one side and the master or a slave node on the other, the messages sent in the channels are confidentiality and integrity protected.

To evaluate the performance of this method, the authors have implemented the authentication method assuming the following usecases: (1) one master with one slave, (2) one master with two slaves and (3) one master with three slaves, and 20 jobs were submitted. The results show that the execution time taken by the master node to authenticate three slave nodes is about the double of the execution time taken to authenticate two slave nodes. This means that the execution time may be excessively high if the number of nodes increases to hundreds or even thousands. The high level cost is mainly due to the use of the asymmetric key based cryptosystem, the use of a third party (CA), and the need to issue and distribute proxy and slave certificates securely.

### E. Quan’s Authentication Method

Q. Quan et al. have extended the work presented in [13] and [16] (the Malley and Wei methods), focusing on file authenticity protection and key exchange [20]. The authors believed that the authentication methods proposed in [13] and [16] mainly provide user identity and service integrity verifications, while the most needed method to secure the MR model is to provide a mechanism to protect the data itself. Based on this belief, they proposed a method to protect data confidentiality and integrity in the MR application. This method makes a hybrid use of the public and symmetric key cryptosystems, i.e. a pair of MR components use a public key cryptosystem to securely exchange a shared symmetric key and then use this symmetric key to encrypt the data. The following steps summarize this method.

1. **Shared key exchange:**
   - An MR component, A, generates a symmetric key, encrypts it using another MR component, B, public key, and then sends the ciphertext to B.
   - B decrypts it using its own private key.
   - Now both A and B share the same secret key which is used to encrypt and decrypt any data (file) sent between the two components.

2. **Data confidentiality and integrity protections:**
   - A’s file content (data) is hashed using a hash function such as MD5.
   - A signs the hashed value of the file content along with other items (that form the file header), such as file ID, file name, and time stamp, using A’s private key, and then sends the lot to B.
   - B verifies the signature using A’s public key.
   - B calculates the hash value of the file content (data) after decrypting it using the shared key. It then compares the hash value with the hash value sent within the encrypted (signed) header.

The merit of this method is that it does not use asymmetric key cryptosystem for encrypting and decrypting the data itself (as the data could be big), rather it uses it to encrypt and decrypt a symmetric key and the file header which has a small size in comparison to the size of the data itself (file content). This is because of the high computational cost of using the asymmetric key cryptosystem for a big data [12] [17]. This method ensures the authenticity and confidentiality of the client data as well as any data sent between any two MR components. However, this method does not provide the authentication of data if the data are not already read by an authenticated MR component; it assumes that any MR component, that reads, or encrypts and decrypts the data itself, has already been authenticated.

Tables 3 and 4, respectively, summarize the related works against the requirements specified for an MR authentication service and the properties specified in Section 3 based on the analysis conducted on the MR model in [1], [3] and [9].
Table 3. Related works versus design requirements for authentication services for the MR model.

<table>
<thead>
<tr>
<th>Item</th>
<th>Requirements</th>
<th>O. Malley and et al</th>
<th>N. Somu and et al</th>
<th>S. Rubika and et al</th>
<th>W. Wei and et al</th>
<th>A. Ruan and et al</th>
<th>J. Zhao and et al</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1.1</td>
<td>(R1) Entities Authentication</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>(R1.1) Entity Identification (or registration)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>1.2</td>
<td>(R1.2) Identity Secret Revocation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.1</td>
<td>(R2) Mutual Entities Authentication between</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>(R2.1) An MR-Client and an MR-Inf Components</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
<td>x</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>(R2.2) An MR-Job Component and an MR-Inf Component</td>
<td>Ø</td>
<td>x</td>
<td>x</td>
<td>✓</td>
<td>x</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>(R2.3) Any pair of MR-Job Components</td>
<td>✓</td>
<td>x</td>
<td>x</td>
<td>✓</td>
<td>x</td>
<td>Ø</td>
</tr>
<tr>
<td></td>
<td>(R2.4) An MR-Job Components’ Hosting Node and an MR-Inf Component</td>
<td>✓</td>
<td>x</td>
<td>x</td>
<td>✓</td>
<td>x</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>(2.5) Cross Domains (i.e. providers)</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>3</td>
<td>(R3) Authenticity of Data and Authentication Messages</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>(R4) Confidentiality of Authentication Messages</td>
<td>✓</td>
<td>x</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>5.1</td>
<td>(R5) Performance and Scalability requirements</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>(R5.1) Minimizing computational overhead</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>(R5.1) Minimizing communication overhead</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>5.3</td>
<td>(R5.3) Maximizing Scalability</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>6</td>
<td>(R6) Support for Updating of Authentication Credentials (secrets)</td>
<td>Ø</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>Ø</td>
</tr>
</tbody>
</table>

✓ - addressed and achieved; x – not addressed or mentioned; Ø – partially addressed or have rooms for improvement.

Table 4. Protection against some security threats for proposed methods

<table>
<thead>
<tr>
<th>Protection against</th>
<th>O. Malley and et al</th>
<th>N. Somu and et al</th>
<th>S. Rubika and et al</th>
<th>W. Wei and et al</th>
<th>A. Ruan and et al</th>
<th>J. Zhao and et al</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identity Spoofing and Impersonation Attack</td>
<td>✓</td>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
</tr>
<tr>
<td>Data Disclosure and Unauthorized Modifications threat</td>
<td>✓</td>
<td>x</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Non-Compliance Threat (Cheating Behavior)</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Denial of Services (DoS) Attacks</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>Ø</td>
<td>Ø</td>
</tr>
</tbody>
</table>

✓ - addressed and achieved; x – not addressed or mentioned; Ø – partially addressed or have rooms for improvement.
V. WHAT IS MISSING

The critical analysis of the existing authentication methods, presented in section IV, shows that some methods are designed to support gate-level authentication (i.e. the authentication of users or clients to the MR application), while others only protect the integrity and origin authentication of protocol messages and data sent among different MR components. Though there are efforts on supporting mutual authentication between different MR components, these efforts are largely based on the use of public key credentials. Public key (i.e. asymmetric key) based solutions require the involvement of a third party (CA) for credential issuance and distribution. The costs incurred in such solutions are usually high. In addition, these methods have not considered mutual authentication between an MR-Job Component and an MR-Inf. Component (Name Node). Mutual authentication between an MR-Job Component and an MR-Inf. Component is necessary and important as the former need to request for data (e.g. input files) or other resources from the latter during a job execution.

The lack of an adequate authentication service specifically designed for the MR model will make the model vulnerable to security threats and attacks. The threats and attacks are not just those in relation to identity thefts, impersonation or replays attacks. A successful compromise of a client’s account with an MR application will give attackers a better chance to launch other attacks, gaining unauthorized access to data and/or interrupt other job executions. This is particularly the case if the MR model is deployed in a shared environment.

To address this open issue, the next section presents a high-level analysis of the MR model, highlighting the functionality of its components and the interactions among the components when executing a job submitted by a client. This analysis will lead to our initial idea of using a layered approach to authentication in the MR model being deployed in a shared environment.

VI. HIGH LEVEL ANALYSIS AND IDEA

From the GMC model shown in Figure 1, we can see that, when executing a job (i.e. in a job execution flow), multiple MR components are involved. Each component executes a well-defined function and the multiple components interact with one another to collaboratively accomplish the job execution [2] [3] [9]. The MR components, either of one job or multiple jobs submitted by a single or multiple clients, are hosted in two shared clusters, Processing Framework (PF) and Distributed File System (DFS). Each interaction between a pair of MR component is a client-server interaction and must be authenticated. Examples of these client-server interactions are the reading and writing requests made to the Name Node. Each request is a procedure call. These calls could be for (i) reading data (job resource) (typically initiated by a Job Tracker or a Task Tracker), or (ii) writing data submitted by a client or produced from a job execution (e.g. input and output data of a job) (typically initiated, respectively, by clients and Reduce Tasks). Other examples of the client-server interactions are those initiated to and from the Resource Manager. When a client submits a new job, he needs to make a new job-submission request. Upon the receipt of the job-submission request, the Resource Manager needs to make a resources-allocation request to a Job Tracker. All these requests are interactions and are made by using procedure calls. In other words, these calls could be for (i) submitting a new job (typically initiated by clients), or (ii) allocating a Job Tracker to master the Map and Reduce Tasks assigned to different Task Trackers related to a job execution.

Depending on their functionalities, the interactions involved in a job execution can be classified into three groups: (i) those for submitting a job, (ii) those for allocating resources for the execution of the job, and (iii) those for reading or writing data related to the job execution.

The first group (Group-1) of interactions takes place when a client submits a job to the MR model. To submit a job, the client makes the job submission via the Resource Manager, and writes the data for the job execution into the Name Node. The Resource Manager is the master node in the PF cluster, and the Name Node is the master node in the DFS cluster. In other words, the first group of interactions is between a client and the two master nodes, one in each cluster. It should be emphasized that one client may submit multiple jobs, and there will be multiple clients submitting jobs. Hereafter we shall refer the interactions taking place for the submission and execution of a single job as one set, and one such set of interactions consists of the interactions from all the three groups, i.e. \( \{ \text{the set of interactions for the execution of one job} \} = \{ \text{a subset of Group-1 interactions} \} + \{ \text{a subset of Group-2 interactions} \} + \{ \text{a subset of Group-3 interactions} \} \), where all the subsets are all related to the submission and execution of a particular job.

The interactions in the second group (Group-2) are for allocating job resources. Here, for each job, three nodes are involved in this group of interactions, the Resource Manager, the Name Node and a Job Tracker allocated for the job. When a job is admitted, the Resource Manager will allocate a Job Tracker for the job. This Job Tracker will be assigned (i.e. allocated) multiple Task Trackers. The Map and Reduce tasks of this job will be executed on these Task Trackers. The Group-2 interactions also include the interactions carried out by the Name Node to manage and maintain a set of Data Nodes. The Data Nodes host data for all the jobs that are submitted. As can be seen from our discussions here, the functionalities of the Resource Manager and Name Node are for managing the executions of all the jobs that are submitted by the same client or by different clients. The Resource Manager and the Name Node serve all the jobs submitted. They are shared by different jobs and are identified by static identities. These components are not invoked because of a job submission; they are there to serve every job submitted by any client. It is for this reason, the two components are called MR Infrastructure components (i.e. MR-Inf. Components, for short)

As mentioned above, the Group-2 interactions are for providing resources for the executions of jobs. As shown in the GMC model (Figure 1), for a single job execution, these interactions are those of Resource Manager to Job Tracker (RM-JT), Job Tracker to Task Tracker (JT-TT), and Name Node to Data Node (NN-DN). These interactions are different from Group-1 interactions, as these are performed for accomplishing cluster functions and for serving the execution
of a particular job. This is due to the following observations: (i) both Job Trackers and Data Nodes are, respectively, the slave nodes of PF and DFS clusters, (ii) any interactions initiated by a cluster master node (i.e. RM or NN) towards a cluster slave node (i.e. RM to JT, or NN to DN), or from a slave node of the PF cluster to another slave node in the same cluster, do not involve any access (read, write or retrieve) to any data of a job. Also, the RM and the NN are the only RM and NN, which initiate Group-2 interactions, and the JT is the only JT that initiates Group-2 interactions for a particular job. In other words, there is no other RM or NN to initiate such interactions in the cluster, and there is no another JT to initiate such interactions for the same job [2].

The third group (Group-3) interactions are for executing a job submitted by a client. For each job, four types of components are involved in this group of interactions (i.e. these components, assigned to a particular job, initiate this group of interactions): the Job Tracker (JT), Task Trackers (TTs), Map Tasks (MTs) and Reduce Tasks (RTs). In executing the job, the JT retrieves the input splits of the data (the data of the job submitted by a particular client) from the DFS. The JT can then start managing the tasks (MTs and RTs). TTs also retrieve the data (for the job execution) from the DFS. TTs can then start executing the MTs and RTS assigned to them. In executing the tasks, RTs read the intermediate data, which is produced by MTs, from the respective Task Trackers. RTs also write the output results of their computations into the DFS. Group-3 interactions can actually be seen as different subgroups (subsets) of interactions, each subgroup of them is performed by a set of components invoked for a particular job. In other words, the set of components are a JT that is invoked for a particular job, and TTs, MTs and RTs, all of which are associated to the JT. This set of components are created or invoked when a job is submitted, and they are terminated or reassigned when the job execution is completed. The existence of this set of components is purely for serving this job. Therefore these components are identified by dynamic identities, and the identities are short-lived and so are the secrets issued to them. For this reason, we refer them as MR-Job Components.

We can further use an example to explain the Group-3 interactions. As shown in the GMC model (Figure 1), during the execution of this particular job (i.e. in a job execution flow), this set of Group-3 interactions are for executing this job and can be identified as follows: Job Tracker to Name Node (JT-NN), Task Tracker to Name Node (TT-NN), Reduce Task to Task Tracker (RT-TT), and Reduce Task to Name Node (RT-NN). These interactions are for executing/processing the job, i.e. they perform (belong to) jobs’ functions. Group-3 interactions may be invoked concurrently by MR-Job Components serving different jobs. Because of this we need to distinguish the interactions based on Job IDs, i.e. which job a particular interaction, or a set of interactions, actually serve.

Three groups of interactions and the MR components performing the interactions take part in executing a client’s job, but neither the interactions nor the operations (or functionalities) of the components are in the control of the client. The components perform their functionalities and interactions to execute the client’s job (or process the client data) on behalf of the client. Therefore, there is an open issue here, i.e. how could a client trust such a shared computational environment? This issue is particularly important if the data processed by the job are privacy or security sensitive.

To achieve effective authentication in an MR environment, the authentication solution should capture the characteristics of this environment. The characteristics that should be captured in the design of an authentication solution for MR can be summarized as follows: (1) this is a shared environment with one or more clusters; (2) each cluster hosts a set of distributed MR components, and these components can be classified into MR-Inf. Components and MR-Job Components; (3) the MR-Job Components are job-dependent, i.e. they are invoked for a particular job submitted by a client; (4) multiple jobs submitted by the same client or by different clients may be hosted by the MR environment, or, in other words, the MR environment typically executes multiple jobs submitted by the same client, and/or different clients, at any one given time. Based on these observations, we can single out the set of MR components (MR-Inf. Components and MR-Job Components) that are involved in executing a particular job and give this set of component a name, an MR-Job Domain. In other words, each job will have a unique identity and this identity is also used to index an MR-Job Domain that refers to the set of MR components involved in serving a particular job.

We here propose a domain-based authentication approach for the newer MR implementation. The novel idea behind this approach is that the MR components that serve a particular job is singled out as a MR-Job Domain and the components in this MR-Job Domain are responsible for authenticating themselves to each other. This is actually an idea of isolation, i.e. we isolate the MR-Inf. Components and MR-Job Components, which are involved in executing a given job, into one set, and require this set of components to authenticate to each other, so that only the components in this set are allowed to access the resource belonging to (or owned by) this particular job and any component outside this domain is not allowed to access to the resource. To implement this idea, we here propose a novel framework, named as the Virtual Domain based Authentication Framework (VDAF). This framework is said to be virtual domain based, because (1) MR-Job Components are dynamic, (2) more than one MR-Job Domain may co-exist at any one given time in the two clusters (PF and DFS), and (3) these MR-Job Domains work on the top of another group of entities, master and slave nodes of the two clusters. From this point on, this latter group of entities, i.e. master and slave nodes of the two clusters, is referred to as the shared cluster infrastructure (Inf.) domain (i.e. MR-Inf. Domain). The MR-Inf. Domain should have its own authentication method, referred to as MR-Inf. Authentication. The MR-Inf. Authentication method is likely to be different from VDAF, the authentication method we propose for a MR-Job Domain. The classification of MR components into different groups, the structure of the groups into different layers, and use of different authentication methods for different layers indicate a layered approach to MR authentication. The next section describes a layered
authentication model we propose for MR, i.e. the MR Layered Authentication Model.

A. MR Layered Authentication Model

Based on the analysis and discussions in the section above, we propose to use the MR Layered Authentication Model (MR-LAM) to realize the whole task of authentication for MapReduce. As shown in Figure 8, MR-LAM consists of three authentication layers. These are MR-Inf. Domain Authentication Layer (Layer-1), MR-Job Domain Authentication Layer (Layer-2), and MR Components Authentication Layer (Layer-3).

The first layer, the MR-Inf. Domain Authentication Layer, serves the authentication of both the clusters’ server nodes and the MR components (MR-Inf Components and MR-Job Components). Layer-1, the MR-Inf. Domain Authentication Layer, is responsible for the authentication of any new physical node joining in a cluster of the MR application and the mutual authentication between any pair of physical nodes in the cluster. In other words, an Authentication Service (AS) for this layer should support two authentication tasks. The first task is the initial authentication of any new server nodes wanting to join the MR-Inf. Domain. A node should only be admitted to becoming a member of the MR-Inf. Domain if the node has been successfully authenticated. For example, considering the case shown in Figure 1, if many jobs are submitted and the Resource Manager in the PF cluster is running out of resources on the slave nodes and/or if the IT administrator, looking after the cluster, has decided to bring in a new slave node into the cluster, then the new node should be authenticated by the AS before it is allowed to become a member of this cluster. The second task performed by Layer-1 authentication service is to support a continuous mutual authentication among the server nodes in the MR-Inf. Domain. These two authentication tasks can be achieved using the Kerberos authentication solution. The Kerberos solution is a preferred authentication method for the MR-Inf. Domain, because the Kerberos is still the default mode of authentication service already deployed for the cluster infrastructure in private clouds, and many operating system (OS) of the clusters’ server nodes, such as Microsoft windows OS and Red Hat OS [21]-[23], already support this authentication method. Basically, if the Kerberos solution is used as the default mode of the authentication service in a cluster, all the slave nodes (i.e. all the members of the MR-Inf. Domain) in the cluster should support this mode of authentication service. They will use Kerberos to authenticate themselves to the master node in the cluster and to establish shared secret keys between the services hosted by the slave and master nodes (i.e. MR components). In this case, the MR-Inf. Domain authentication service is provided through the use of the Kerberos solution. However, the clients who have their jobs admitted into the MR cluster are not members of the MR-Inf. Domain themselves, but they should be authenticated before their jobs could be admitted. As a client will be part of a MR-Job Domain at Layer-2, a client should be authenticated by the authentication service provided at Layer-2. In other words, the second layer of the authentication model, the MR-Job Domain Authentication Layer, should be able to provide means for clients’ authentication.

For Layer-2, i.e. the MR-Job Domain Authentication Layer, the authentication task is for mutual authentication among the MR components that are involved in serving a particular job. In other words, for this layer of authentication, an authentication method, different from the one used in Layer-1, may be used. As mentioned earlier, we propose to use the VDAF to support this layer of authentication. With VDAF, the MR components that serve a particular job are collectively referred to as a MR-Job Domain. The components in a MR-Job Domain perform authentication among themselves. At this layer, each client is registered with an AS. Upon successful registration, the AS issues a long-term access credential to the client so that the client can use this credential to submit jobs to the MR application. During the execution of this job, the client would be able to make use of the MR components (MR-Inf Components and MR-Job Components) or resources provided through these components, so long as these components and resources are assigned to the job. Also, at Layer-2, the authentication method and protocols used by each MR-Job Domain are expected to be the same, but the secrets used in each such domain are different and they should be protected against exposure to other domains. As mentioned earlier, each MR-Job Domain has its own MR-Job Components involved in the execution of a job submitted by a particular client. The client generates and manages the credentials (authentication secrets and other data) used to authenticate the MR-Job Components in this domain. In other words, this layer is responsible for providing the identification and authentication service by which the MR-Job Components assigned to each job can be securely identified and authenticated and do so at every interaction among themselves throughout the execution cycle of the job. Also, the Resource Manager, which manages the resources of the MR-Inf. Domain, is also involved in the authentication of all the MR-Job Domains. For the MR-Job Domain level (i.e. Level-2) authentication, the Resource Manager works as a relay to deliver the access credentials of each MR-Job Domain 2 to the client and the MR-Job Components in that Domain (more details to follow in our future work).

The third layer (Level-3) is the MR Components Authentication Layer. As mentioned earlier, some of the MR components are shared by more than one job (these components will carry static and semi-permanent IDs), while others are exclusively used by the tasks created for a particular job submitted. These components will carry dynamic IDs – they are created when the job is created, but discarded when the execution of the job is completed. Hence, at Level-3, it is assumed that each of the MR components (either an MR-Inf Component or an MR-Job Component) has an authentication module. These authentication modules, depending on the hosting MR components, can be respectively named as Job Tracker Authentication (JT-AuthN) Modules, Task Tracker Authentication (TT-AuthN) Modules, Map Task Authentication (MT-AuthN) Modules, Reduce Task Authentication (RT-AuthN) Modules, Name Node Authentication (NN-AuthN) Module, Data Node Authentication (DN-AuthN) Modules, and Resource Manager Authentication (RM-AuthN) Modules.

2 The access credentials of an MR-Job Domain are the access credentials of both the client and the MR-Job Components of the MR-Job Domain.
Authentication (RM-AuthN) Relay Module. By embedding these authentication modules into their respective MR components, we can provide MR-Inf. Components and MR-Job Components with authentication services supporting the authentication among themselves and preventing unauthorized access to data or resources assigned to (or owned by) a particular job domain. The implementation of the Layer-2 and Layer-3 authentication services will be described in more details in our future paper.

![MR layered authentication model](image)

**Figure 8. MR layered authentication model (A layered approach to authentication)**

**VII. CONCLUSION AND FUTURE WORK**

This paper has critically analyzed existing authentication methods designed for the MR model. It has also presented a high-level analysis of how an authentication service may be provided for the MR model and given a high-level idea of using a layered approach to the authentication in this context. The analysis of existing authentication methods has indicated that providing an inadequate authentication service to the MR model or deploying an authentication service that fails to capture the characteristics of the MR model would put clients’ jobs and the resources hosted in an MR application at a high level of risks.

Providing an adequate authentication service for the MR model is a challenging task. This is due to the characteristics that the MR model is usually deployed in a shared infrastructural environment, and in such an environment, it is difficult to distinguish between a compromised and a trustworthy MR component. In addition, the hosting nodes in this environment are distributed, and possibly provided by multiple providers.

The VDAF facilitates authentication on per job basis (i.e. Job Authentication (Job-AuthN)) and do so during the entire execution cycle of the job. It covers a chain of authentication tasks, namely, (i) from a user to the user’s client running on the user’s machine, (ii) from the client to the MR application (using the AS), and (iii) messages sent by the MR components; these are transactions sent from an MR-Job Component to an MR-Inf Component and from one MR-Job Component to another MR-Job Component. Among these authentication tasks, from (i) to (iii), data authentication should be provided. In other words, an authentication solution for MR should not only guard the gate into the system, but also guard every resource access during the execution of a job, as the execution involves multiple MR components distributed across multiple nodes of both PF and DFS clusters. Implementing a simple but effective authentication solution for the MR model is needed. So as part of our work to design such a solution, this paper has also given a high-level overview of how this solution may be designed. Our proposed idea is to use a layered approach to tackle the complex task of authentication in MR. This approach takes into account of the authentication requirements of all the components of the MR model. The authentication service should be provided at multiple levels and do so securely and efficiently. To satisfy the authentication requirements as detailed in R2.1 to R2.3, we have analyzed and identified all the possible interactions among the MR components. The interactions of C-RM, C-NN, JT-NN, TT-NN, RT-TT and RT-NN have been considered in our design of VDAF. We will also address the requirement of R2.4 and R2.5, taking into account of the interactions of RM-JT, NN-DN and JT-TT, in the design of an authentication solution for the MR-Inf. Domain Authentication Layer.

The way we design the VDAF can also make the MR model be delivered as a Software as a Service (SaaS) in a public cloud environment. This is due to two factors. First, the MR-Inf. Domain is likely to be managed remotely from a central location by an MR provider and not by a client, and the client is not involved in issuing and managing (i.e. is not in control of) the authentication credentials or secrets of the members of the MR-Inf. Domain (i.e. the MR-Inf components which are actually the master nodes of both PF and DFS clusters) and MR-Job Component’s Hosting Nodes (which are actually the slave nodes of both clusters). In other words, a client does not have to install or configure the master and slave nodes of both shared clusters, the client is not involved in issuing and managing the authentication credentials or secrets of the MR-Inf components and the MR-Job Components’ Hosting Nodes, rather the client controls the authentication credentials or secrets of the MR-Job Components that are involved in executing his job (i.e. his own MR-Job Domain). Therefore, a client only needs to make a new job-submission request which includes uploading the data of his MR-Job Domain. Secondly, an MR application delivers its computing services in a “one-to-many” manner. This means that one MR-Inf. Domain hosts multiple MR-Job Domains and each MR-Job Domain serves the execution of a particular job submitted by a particular client. Each client has his own MR-Job Domain secrets (i.e. isolated from other MR-Job Domains), and the client controls the credentials or secrets of his own MR-Job Domain.

The detailed design of the VDAF for MR-Job Domains will be presented in our future work.

**REFERENCES**


