New authentication mechanism using certificates for big data analytic tools

PAUL J. E. VELTHUIS
Abstract

Companies analyse large amounts of sensitive data on clusters of machines, using a framework such as Apache Hadoop to handle inter-process communication, and big data analytic tools such as Apache Spark and Apache Flink to analyse the growing amounts of data. Big data analytic tools are mainly tested on performance and reliability. Security and authentication have not been enough considered and they lack behind. The goal of this research is to improve the authentication and security for data analytic tools.

Currently, the aforementioned big data analytic tools are using Kerberos for authentication. Kerberos has difficulties in providing multi factor authentication. Attacks on Kerberos can abuse the authentication. To improve the authentication, an analysis of the authentication in Hadoop and the data analytic tools is performed. The research describes the characteristics to gain an overview of the security of Hadoop and the data analytic tools. One characteristic is that the usage of the transport layer security (TLS) for the security of data transportation. TLS usually establishes connections with certificates. Recently, certificates with a short time to live can be automatically handed out.

This thesis develops new authentication mechanism using certificates for data analytic tools on clusters of machines, providing advantages over Kerberos. To evaluate the possibility to replace Kerberos, the mechanism is implemented in Spark. As a result, the new implementation provides several improvements. The certificates used for authentication are made valid with a short time to live and are thus less vulnerable to abuse. Further, the authentication mechanism solves new requirements coming from businesses, such as providing multi-factor authentication and scalability.

In this research a new authentication mechanism is developed, implemented and evaluated, giving better data protection by providing improved authentication.

**Keywords:**— Cloud Access Management, certificate on demand, Apache Spark, Apache Flink, Kerberos, transport security layer (TLS), Authentication, Multi Factor Authentication, Authentication for data analytic tools, certificate based Spark authentication, public key encryption, distributed authentication, short valid authentication

**Publication date:** August 4, 2017
Acknowledgements

I am grateful for the help and support that is given by my parents and girlfriend. I am thankful for the support that is given by Fraunhofer Institute for Secure Information Technology (SIT), here in special Dr. Marcel Schäfer. The support provided by Christian Winter from Fraunhofer. I am appreciative for the pleasant lunches with Fraunhofer and the friendly chats with them and my other colleagues. I would like to thank the University of KTH, here in particular Dr. Jim Dowling for his valuable insights and being the Professor. I would like to thank Antonios Kouzoupis from KTH for providing valuable insights. Jim Dowling works within KTH at SICS, SICS performs research in the field of communication and applied information technology.

Fraunhofer SIT is a leading expert in the realm of IT security, and the institute offers a range of services and solutions for companies, such as assistance with crafting an effective IT security management strategy and auditing assistance to products and systems for potential vulnerabilities [1]. Fraunhofer helps to analyse and evaluate products. It gives recommendations on where to invest money and helps to benchmark the impact of such decisions. Fraunhofer helps to avoid implementation faults, by providing methodologies and identify indicators for the performance of security activities [1]. In the field of cloud computing, Fraunhofer SIT has build software like OmniCloud. This is a software solution, to transfer and store existing or new backups securely and economically to the cloud [2]. The software encrypts the data to be backed up in the cloud. To minimise the cost OmniCloud prevents duplication. A solution like this is aimed at small and medium sized companies.

[2] Jim Dowling works within KTH at SICS, SICS performs research in the field of communication and applied information technology.
### List of Acronyms and Abbreviations

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACID</td>
<td>Atomicity, consistency, isolation and durability</td>
</tr>
<tr>
<td>ACL</td>
<td>Access control list</td>
</tr>
<tr>
<td>API</td>
<td>Application programming interface</td>
</tr>
<tr>
<td>AM</td>
<td>Application Master</td>
</tr>
<tr>
<td>BLESS</td>
<td>Bastion lambda ephemeral SSH service</td>
</tr>
<tr>
<td>CAM</td>
<td>Cloud access management</td>
</tr>
<tr>
<td>CA</td>
<td>Certificate authority</td>
</tr>
<tr>
<td>CN</td>
<td>Common name</td>
</tr>
<tr>
<td>CRL</td>
<td>Certificate revocation list</td>
</tr>
<tr>
<td>DAG</td>
<td>Directed acyclic graph</td>
</tr>
<tr>
<td>HDFS</td>
<td>Hadoop distributed file system</td>
</tr>
<tr>
<td>Hops</td>
<td>Hadoop open Platform-as-a-Service</td>
</tr>
<tr>
<td>HOTP</td>
<td>HMAC-based one-time password</td>
</tr>
<tr>
<td>IAM</td>
<td>Identity and authentication management</td>
</tr>
<tr>
<td>IoT</td>
<td>Internet of things</td>
</tr>
<tr>
<td>JDK</td>
<td>Java development kit</td>
</tr>
<tr>
<td>JVM</td>
<td>Java virtual machine</td>
</tr>
<tr>
<td>KDC</td>
<td>Kerberos domain controller</td>
</tr>
<tr>
<td>MFA</td>
<td>Multi factor authentication</td>
</tr>
<tr>
<td>NM</td>
<td>Namenode Manager</td>
</tr>
<tr>
<td>NTP</td>
<td>Network time protocol</td>
</tr>
<tr>
<td>OS</td>
<td>Operating system</td>
</tr>
<tr>
<td>OTP</td>
<td>One-time password</td>
</tr>
<tr>
<td>PaaS</td>
<td>Platform as a service</td>
</tr>
<tr>
<td>PAM</td>
<td>Pluggable authentication modules</td>
</tr>
<tr>
<td>PA management</td>
<td>privileged access management</td>
</tr>
<tr>
<td>Acronym</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>PKI</td>
<td>Public key infrastructure</td>
</tr>
<tr>
<td>RDBMS</td>
<td>Relational database management system</td>
</tr>
<tr>
<td>RDD</td>
<td>Resilient distributed dataset</td>
</tr>
<tr>
<td>RM</td>
<td>Resource Manager</td>
</tr>
<tr>
<td>SSH</td>
<td>Secure Shell</td>
</tr>
<tr>
<td>SSL</td>
<td>Secure Sockets Layer</td>
</tr>
<tr>
<td>SQL</td>
<td>Structured query language</td>
</tr>
<tr>
<td>TLS</td>
<td>Transport layer security</td>
</tr>
<tr>
<td>TOTP</td>
<td>Time-based one-time password</td>
</tr>
<tr>
<td>2PC</td>
<td>Two phase commit protocol</td>
</tr>
<tr>
<td>TTL</td>
<td>Time to live</td>
</tr>
<tr>
<td>UI</td>
<td>User interface</td>
</tr>
<tr>
<td>UML</td>
<td>Unified modelling language</td>
</tr>
<tr>
<td>URI</td>
<td>Uniform resource identifier</td>
</tr>
<tr>
<td>VM</td>
<td>Virtual machine</td>
</tr>
<tr>
<td>U2F</td>
<td>Universal second authentication</td>
</tr>
<tr>
<td>UUID</td>
<td>Universally unique identifier</td>
</tr>
<tr>
<td>YARN</td>
<td>Yet another resource negotiator</td>
</tr>
<tr>
<td>YOTP</td>
<td>Yubikey one-time password</td>
</tr>
</tbody>
</table>
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Hadoop environment</td>
<td>5</td>
</tr>
<tr>
<td>2.2</td>
<td>YARN cluster mode in Spark architecture</td>
<td>7</td>
</tr>
<tr>
<td>2.3</td>
<td>Spark cluster [3]</td>
<td>8</td>
</tr>
<tr>
<td>2.4</td>
<td>Sequence diagram Kerberos for Apache Spark job</td>
<td>15</td>
</tr>
<tr>
<td>2.5</td>
<td>SSL handshake [4]</td>
<td>17</td>
</tr>
<tr>
<td>2.6</td>
<td>TOTP architecture [5]</td>
<td>18</td>
</tr>
<tr>
<td>2.7</td>
<td>U2F [5]</td>
<td>19</td>
</tr>
<tr>
<td>2.8</td>
<td>HDFS authentication [6]</td>
<td>23</td>
</tr>
<tr>
<td>2.9</td>
<td>Authentication path [6]</td>
<td>24</td>
</tr>
<tr>
<td>2.10</td>
<td>BLESS flow diagram[7]</td>
<td>29</td>
</tr>
<tr>
<td>5.1</td>
<td>Sequence diagram status job</td>
<td>46</td>
</tr>
<tr>
<td>5.2</td>
<td>Sequence diagram standalone cluster mode job</td>
<td>49</td>
</tr>
<tr>
<td>5.3</td>
<td>Sequence diagram job YARN cluster mode</td>
<td>51</td>
</tr>
<tr>
<td>5.4</td>
<td>Overview YARN certificate renewal</td>
<td>59</td>
</tr>
</tbody>
</table>
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Performance SSL Kafka [8]</td>
<td>18</td>
</tr>
<tr>
<td>2.2</td>
<td>Netflix SSH &amp; RSA certificate</td>
<td>28</td>
</tr>
<tr>
<td>4.1</td>
<td>Overview TLS and Kerberos</td>
<td>36</td>
</tr>
<tr>
<td>5.1</td>
<td>Certificate design</td>
<td>43</td>
</tr>
<tr>
<td>5.2</td>
<td>Spark Configuration options</td>
<td>53</td>
</tr>
<tr>
<td>5.3</td>
<td>Spark submit options</td>
<td>53</td>
</tr>
<tr>
<td>5.4</td>
<td>Administration table</td>
<td>56</td>
</tr>
<tr>
<td>5.5</td>
<td>Resources table</td>
<td>56</td>
</tr>
</tbody>
</table>
## Listings

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Spark example</td>
<td>9</td>
</tr>
<tr>
<td>2.2</td>
<td>Spark submit</td>
<td>9</td>
</tr>
<tr>
<td>5.1</td>
<td>Rsync command</td>
<td>60</td>
</tr>
<tr>
<td>6.1</td>
<td>Spark Python job</td>
<td>65</td>
</tr>
<tr>
<td>6.2</td>
<td>Setting cores</td>
<td>65</td>
</tr>
<tr>
<td>6.3</td>
<td>System property test</td>
<td>67</td>
</tr>
<tr>
<td>6.4</td>
<td>Spark local Python job</td>
<td>68</td>
</tr>
<tr>
<td>6.5</td>
<td>Spark status Standalone</td>
<td>69</td>
</tr>
<tr>
<td>6.6</td>
<td>Spark status YARN</td>
<td>69</td>
</tr>
<tr>
<td>6.7</td>
<td>Spark cluster mode command</td>
<td>69</td>
</tr>
<tr>
<td>6.8</td>
<td>Spark YARN Job command</td>
<td>71</td>
</tr>
<tr>
<td>6.9</td>
<td>Yarn testsuite</td>
<td>71</td>
</tr>
<tr>
<td>6.10</td>
<td>Compile Spark</td>
<td>72</td>
</tr>
<tr>
<td>6.11</td>
<td>Production version Spark</td>
<td>72</td>
</tr>
</tbody>
</table>
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract</td>
<td>i</td>
</tr>
<tr>
<td>Acknowledgement</td>
<td>i</td>
</tr>
<tr>
<td>List of Acronyms and Abbreviations</td>
<td>ii</td>
</tr>
<tr>
<td>List of Figures</td>
<td>iv</td>
</tr>
<tr>
<td>List of Tables</td>
<td>v</td>
</tr>
<tr>
<td>Listings</td>
<td>vi</td>
</tr>
<tr>
<td><strong>1 Introduction</strong></td>
<td>1</td>
</tr>
<tr>
<td>1.1 Problem statement</td>
<td>2</td>
</tr>
<tr>
<td>1.2 Approach</td>
<td>2</td>
</tr>
<tr>
<td>1.3 Outline</td>
<td>3</td>
</tr>
<tr>
<td><strong>2 State of the Art</strong></td>
<td>4</td>
</tr>
<tr>
<td>2.1 Apache Hadoop environment</td>
<td>4</td>
</tr>
<tr>
<td>2.1.1 Hadoop distributed file system</td>
<td>5</td>
</tr>
<tr>
<td>2.1.2 Apache Kafka</td>
<td>5</td>
</tr>
<tr>
<td>2.1.3 Apache yet another resource negotiator (YARN)</td>
<td>5</td>
</tr>
<tr>
<td>2.1.3.1 Scheduling</td>
<td>6</td>
</tr>
<tr>
<td>2.1.3.2 Architecture</td>
<td>6</td>
</tr>
<tr>
<td>2.1.4 Apache Spark</td>
<td>7</td>
</tr>
<tr>
<td>2.1.4.1 Programming languages</td>
<td>7</td>
</tr>
<tr>
<td>2.1.4.2 Model</td>
<td>8</td>
</tr>
<tr>
<td>2.1.4.3 SparkContext</td>
<td>8</td>
</tr>
<tr>
<td>2.1.4.4 Job examples</td>
<td>9</td>
</tr>
<tr>
<td>2.1.4.5 Development tools</td>
<td>10</td>
</tr>
<tr>
<td>2.1.5 Apache Flink</td>
<td>10</td>
</tr>
<tr>
<td>2.1.5.1 Streaming</td>
<td>10</td>
</tr>
<tr>
<td>2.1.5.2 Scheduling</td>
<td>11</td>
</tr>
<tr>
<td>2.1.6 Hadoop open Platform-as-a-Service (Hops)</td>
<td>11</td>
</tr>
<tr>
<td>2.1.6.1 Hadoop open Platform-as-a-Service with Spark</td>
<td>11</td>
</tr>
<tr>
<td>2.1.6.2 Project based multi-tenancy</td>
<td>12</td>
</tr>
<tr>
<td>2.2 Authentication with Kerberos</td>
<td>12</td>
</tr>
<tr>
<td>2.2.1 Overview</td>
<td>12</td>
</tr>
<tr>
<td>2.2.2 Kerberos advantages</td>
<td>14</td>
</tr>
<tr>
<td>2.2.3 Kerberos challenges in Hadoop environment</td>
<td>14</td>
</tr>
<tr>
<td>2.2.4 Kerberos process model for Apache Spark</td>
<td>15</td>
</tr>
<tr>
<td>2.3 Transport layer security (TLS)</td>
<td>15</td>
</tr>
<tr>
<td>2.3.1 Overview</td>
<td>16</td>
</tr>
<tr>
<td>2.3.2 Chain of trust</td>
<td>16</td>
</tr>
</tbody>
</table>
2.3.3 Infrastructure ........................................ 17
2.3.4 Encryption overhead costs ............................... 18
2.4 Multi-factor authentication ................................. 18
2.5 Distributed databases ........................................ 19
2.6 Certificates on demand ...................................... 21
2.7 Apache Hadoop environment Security ..................... 22
2.7.1 Security layers ........................................... 22
2.7.2 Hadoop environment authentication and security ....... 22
2.7.3 Hadoop environment data transport security .......... 24
2.7.4 Kafka Security ........................................... 25
2.7.5 YARN Security ............................................ 25
2.7.6 Spark authentication and security ...................... 25
2.7.7 Flink Security ............................................ 26
2.8 Bastion lambda ephemeral SSH service (BLESS) ........... 27
2.8.1 BLESS SSH access ....................................... 27
2.8.2 BLESS authentication and certificates ................ 27
2.8.3 BLESS log and track access ............................. 28
2.8.4 BLESS process model .................................... 28

3 Research Methods and methodologies ....................... 30
3.1 Requirement Analysis ....................................... 31
3.2 Unified modelling language ................................ 31

4 Analyses .......................................................... 32
4.1 Long-running streaming job analysis ...................... 32
4.2 Resource management with sensitive data analysis ...... 32
4.3 Analysis bastion lambda ephemeral SSH service (BLESS) . 33
4.4 Flink and Spark similarities and differences ............. 34
4.4.1 Flink and Spark similarities ............................. 34
4.4.2 Flink and Spark differences ............................. 34
4.5 Data transport security and Kerberos comparison ...... 34
4.6 Certificate on demand analysis ............................ 36
4.7 Cloud Access Management analysis ...................... 37
4.8 Summary ..................................................... 37

5 Contribution ....................................................... 39
5.1 Requirements ................................................ 39
5.2 Requirement analysis Kerberos ............................ 41
5.3 Authentication process design .............................. 41
5.3.1 Certificate design ........................................ 41
5.3.2 User authentication process design ................... 44
5.3.3 User authentication process design for Spark cluster . 47
5.3.4 User authentication process design for Spark with YARN cluster 50
5.4 Authentication Configuration options .................... 52
5.5 Implementation .............................................. 54
5.5.1 Certificate authority ..................................... 54
5.5.1.1 Trust in Certificate authority ....................... 54
5.5.1.2 Certificate integrity ................................... 54
5.5.1.3 Process .................................................. 55
5.5.2 Access control database .................................. 57
5.5.3 Auditing system .......................................... 57
5.5.4 Spark cluster ............................................. 57
5.5.4.1 Process .................................................. 57
Chapter 1

Introduction

Data becomes more crucial for businesses. The data analyses requires clusters of machines. The need for server clusters is driven by the fact that the computation requirements are growing at a faster rate than the advances in single computer performance. The cluster of machines is an solution that allows to scale the performance. However, this introduces new problems, for example communication overhead. To manage the overhead several frameworks have appeared [9]. These frameworks have low-level implementations to handle the inter-process communication [10]. A popular framework is Hadoop. The Hadoop environment is consisting of all the projects related to the Hadoop framework. The Hadoop environment enables the distributed processing of Petabytes of data for analysis [10]. This data comes by the term "Big data" and has four characteristics [11]. The first is volume, the amount of data. The second is variety, the many formats under which the data is stored. The third is velocity, the speed at which the data arrives. The fourth is veracity, this relates to how accurate and trustworthy the data is that is measured.

Business value is created by analysing the data. Hadoop supports big data analytic tools to analyse large amounts of data. There is an evolution going on and big data analytic tools are coming up [12]. These tools help to achieve better insights in marketing, business insights, automated decisions for real-time processing, and Fraud detection [12]. The most pervasive tool providing a magnitude of functions and application programming interfaces (APIs) is Apache Spark [13]. Spark is applicable for batch processing operations as well as stream processing and integrates machine learning libraries [13]. The Spark framework is an open source clustering framework [13]. A promising competitor is Flink which has a lower latency [14]. While both Flink and Spark are tested and compared regarding performance and flexibility [15][16], security and moreover authentication have not yet sufficiently been thought of [17]. The three most important aspects of choosing a data analytic tool are security, performance, and reliability [18].

Companies are starting to work with cloud solutions and tools to save money [2]. Cloud solutions work on clusters with multiple users and applications [19]. This demands for a data separation and knowledge protection [1]. The sharing of data might do harm to company assets [1]. That data can include sensitive information, such as the privacy of individuals, sensitive corporate data, or sensitive customer data [20][21]. If the attackers gain access to the sensitive data they are able to compromise the data in 60% of the cases before being discovered. Further, there are laws that require regulatory compliance and keeping the data safe [21]. This regulated data can contain health, personal, or payment data. The access to the data needs to be controlled so that wrong entities have no possibility to tamper with or access the data. When improving the security and authentication to sensitive data it can give companies new business opportunities [21]. For the data security, the data requires protection during Transport. In Hadoop, there is a transport layer security (TLS) responsible for the data protection during transport [22]. Currently, Kerberos is
1.1 Problem statement

The problem is that security and authentication for the analysis of big data are currently not considered well enough. In order to first understand the depth of this problem, a look is taken into the Hadoop environment its main components and security. To investigate the problems in authentication and security mechanisms there is intensive research in the State of the Art, the authentication with Kerberos and the transport security works with TLS. In this research there is tried to find an answer to the following research question:

RQ1 What are the characteristics of the current security and authentication mechanisms for data analytic tools in the Hadoop environment?  

After doing some research and having several discussions there is realised that the current authentication with Kerberos is challenging for many administrators. Using Kerberos in Hadoop is challenging and it has some insurmountable limitations. This leads to a new research question.

RQ2 Can Kerberos be replaced to improve the security and authentication in data analytic tools?  

During this research, there is realised that Kerberos can be replaced. A new mechanism is discovered that can perform authentication this mechanism has several improvements over Kerberos. Further, there is realised that the data analytic tool Apache Spark would be a good point of start for the implementation. In the analysis, several reasons are found why it is better to implement in Spark over Flink. Due to time constraints for making the implementation, there is chosen to implement the new mechanism in Spark. First the requirements that this new authentication mechanism should fulfil are created, then the implementation is started. This gives the last research question:

RQ3 Does the new authentication mechanism implemented in Spark fulfil the authentication requirements?

1.2 Approach

In the approach, there is defined how the research questions from the “Problem statement” are being solved. To answer question RQ1 the security and authentication in Hadoop are researched. Here there is specifically looked into the data analytic tools. The data analytic tools are becoming more popular and here the authentication and security seem to be insufficient developed. The security and authentication of the data tools are closely related to Hadoop. In this research the main components of Hadoop are shown in order to help the reader understand the relatively complex Hadoop environment. The current state of the security and authentication is shown with its important characteristics.

To answer question RQ2 there is analysed why Kerberos is so challenging. Further, there is analysed what the limitations of Kerberos are. This lets to some severe limitations which can not be easily solved with Kerberos. For this reason, the question is how to replace Kerberos. To replace Kerberos there is investigated how the data tools inner workings are. To answer this question there is looked into new security mechanisms that can be used. During the research, there is found recent research on using the mechanism for transport security as authentication. This mechanism is public key cryptography. To answer this question there is looked further into recent developments in public key cryptography for authentication.
To answer the third question, the goal is to design and implement a new security and authentication that improves the current authentication of Spark. Requirements are defined for what the new authentication mechanism should achieve. To achieve this analysis is done in the current security and authentication for Hadoop. Another analysis is done in new authentication techniques. This resulted in the new requirements. To evaluate the concept Spark has to be implemented with respect to the requirements. This authentication mechanism serves as a proof of concept to show that the authentication mechanism works. Spark is chosen because of its popularity, Spark is often used in combination with YARN. YARN is a popular resource scheduler which improves the performance of Spark. To answer this research question properly there should be looked Spark can run for its most popular use cases, for this reason in this research Spark is evaluated with YARN. This research question results in an answer whether the new authentication mechanism can be used for Spark.

1.3 Outline

The outline is presented in this section. In Chapter 2 the State of the Art provides the background information for the thesis. In the State of the Art contains information about the major components of Hadoop, such as YARN, the distributed file system, Apache Spark and Flink. The authentication with Kerberos and the data transport security TLS is explained in detail. New mechanisms for authentication are explained. The security and authentication of the Hadoop environment are explained, with the current security status of Spark and Flink. In Chapter 3 the research methods and methodologies used in this research are highlighted. In Chapter 4 the State of the Art is analysed. An analysis made between the current techniques used and how they can be improved with new techniques. The characteristics of the current authentication and security are analysed, in combination with the earlier explained security details of the Hadoop environment an answer to is provided. The similarities and differences between Flink and Spark are explained. In the end of this section, a summary is given with key points for the new solution. The analysis of a new mechanism and the summary in the end of Chapter 4 provide an answer to whether Kerberos can be replaced. This answers.

In Chapter 5 the new solution is proposed. The Chapter “Contribution ” contains the new requirements for the authentication system. The new authentication design with after that an explanation of the required new configuration. The new implementation is explained. The “Contribution ” is ended with programming scripts that make the implementation possible. In Chapter 6 “Evaluation” the implementation is evaluated. To evaluate there is a set up to test the implementation on. Then there are different test cases to perform the testing. There is evaluated how the implementation can be deployed in a continuous developing data analytic tool like Apache Spark. The important modules to make the implementation possible are evaluated. In the end, there is evaluated if the requirements are satisfied. The new requirements and implementation in Chapter 5 and the evaluation in Chapter 6 answers the third research question. There is evaluated at the end of the “Evaluation” Chapter whether the requirements are fulfilled, and a new solution is provided. In the Chapter 7 “Discussion” the research question is answered and the findings are discussed. Further, the risk consequences and ethics of this project are discussed. Chapter 8 concludes this thesis and gives directions for future work.
Chapter 2

State of the Art

This chapter provides the background information. The Hadoop environment consists of all the projects related to Hadoop. An explanation is given in Section 2.1. The Hadoop environment processes more and more sensitive data, the security has become more of a concern. For authentication of data and users, Kerberos is within the Hadoop environment. Detailed information of Kerberos is given in Section 2.2. For the security of data during transport is a TLS, the details are in Section 2.3. Multi-factor authentication (MFA) has gained in the recent year attention, providing an additional authentication mechanism over static passwords. The reason for this is that static passwords alone have several security concerns [24]. MFA introduced in Section 2.4 uses extra authentication factors, MFA uses two or more secrets instead of one, in order enhance the security and reduce the chance of disallowed user access. A database needs to store authentication secrets. A distributed database provides consistent and available authentication. Details of the distributed databases are in Section 2.5. In Section 2.6 is the new upcoming technique validating certificates for a short time. An explanation of the security components of the Hadoop environment in Section 2.7 gives clarification about the general security and the implementation of security for the data transport in Hadoop. Section 2.8 explains a distribution of Hadoop that improves the access control by using project multi-tenancy, which makes isolation of projects possible. Section 2.8 gives details of a new access control mechanism called Bastion Lambda Ephemeral SSH service. This new security mechanism uses on demand certificates and MFA.

2.1 Apache Hadoop environment

The Hadoop environment is an open source framework. The goal of the framework is to process and store large datasets. In the environment are storage, a messaging system, and tools to organise resource management. There are a data processing engine and an application to perform the analysis on the data. The overview is shown in Figure 2.1. Further, there are distributions build on top of the Hadoop environment. The storage system organises the storage of the files. The messaging system allows handling big data volumes that are coming in at the same time. The resource scheduler takes care of the resource management in the system. The data processing engine allows running many different workflows. The data processing engines, Spark and Flink are the application the user uses so that the user can execute the workflow.

In Section 2.1.1 the storage system of Hadoop is highlighted. The distributed messaging system of Hadoop is in Section 2.1.2. The most used resource scheduler YARN is explained in Section 2.1.3. In Section 2.1.4 the first data processing engine Apache Spark is explained and in the second data processing engine Apache Flink is in Section 2.1.5. There is an interesting distribution of Apache Hadoop called Hadoop Open Platform-as-a-Service. This Hadoop distribution is in Section 2.1.6.
The distribution improves the scalability of Hadoop and the security.

### 2.1.1 Hadoop distributed file system

This section covers Apache Hadoop its distributed file system, the Hadoop distributed file system (HDFS) [25]. This file system looks like any other file system the difference is that a file on Hadoop is split into small files, each of those files is replicated and stored on servers for fault tolerance constraints [25]. HDFS is the UNIX-based data storage layer of Hadoop. The HDFS follows a master-slave paradigm meaning that there is one master who coordinates the work of the slaves [25]. The term worker is used interchangeably with the term slave. Hadoop is inspired on the MapReduce programming paradigm for processing and handling large data sets [26]. It splits file requests into smaller requests which are sent to workers to be parallel processed [25]. As a result, the processing of massive datasets is fast. Hadoop can run on almost any commodity server. In Hadoop there is a guarantee write-once, read multiple times, it assumes that a file in HDFS once written will not be modified. The HDFS stores most of the data used by Apache Spark and Apache Flink.

### 2.1.2 Apache Kafka

Apache Kafka is a distributed messaging system [27]. Kafka allows handling big data volumes, streaming it to many services. For this reason, it is also known as a distributed streaming platform [27]. Kafka has three streaming capabilities. The first capability is the publish and subscribe mechanism to streams of records. The publish method presents the record to other services. By subscribing services retrieve the available records for the stream. The second capability is that the system stores streams of records in a fault-tolerant way. The third is the processing of streams of records as they occur. Kafka builds real-time streaming data pipelines that reliably get data between systems or applications. Kafka runs on a cluster of one or more servers. The server stores stream of records in categories called topics. Kafka works together with Spark streaming, where Kafka then creates the data stream. Kafka makes it possible that the consumer, such as Spark, can pull and use the data it needs.

### 2.1.3 Apache yet another resource negotiator (YARN)

Apache yet another resource negotiator (YARN) enables scheduling of resources [28]. YARN allows running multiple applications in Hadoop by sharing its common resource management system. YARN has two time interpretations Real and interactive time. Interactive time, is the time that happens in our time, human time. Real-time...
YARN is a global scheduler for Hadoop, the position of the resource scheduler is between the hardware and the framework, an example is between a server and Spark. YARN has a capacity scheduler that looks whether an application has enough resources available [9]. Mesos is an alternative for YARN and also aims at being a global resource manager for an entire cluster [29]. Mesos allows for an infinite number of scheduling algorithms to be developed. The scheduling algorithms are pluggable. In YARN are two mechanisms of resource distribution, pull and push-based [9]. Push-based means that the scheduler gives resources to the framework e.g. Spark. A Push-based scheduler has the advantage that it can self-define how fast it gets its resources. Pull-based resource scheduling waits for incoming requests. The YARN scheduler is pull-based and by this achieves reserved based scheduling. Meaning that when there are not enough resources available for a task the Resource Manager (RM) will reserve resources, when a task completes the manager gives the free resources to the task which reserved them. YARN can make use of a pull-based capacity scheduler, which only takes into account memory. There is another resource dominant scheduler, which takes into account memory and CPU. YARN has dynamic allocation to request resources based on the demands of an application. In general, the application started first gets the resources first. When using dynamic allocation, resource requests increase exponential until it suffices [30]. The dynamic allocation makes sure it dynamically scales down the resources not used. For the processing of the application, executors get requested when enough resources are available. An idle executor gets removed again. The next Paragraph 2.1.3.2 explains the architecture.

2.1.3.2 Architecture

YARN consists of multiple components and forms a cluster [28]. A YARN cluster running a Spark job is in Figure 2.2. YARN has node managers (NM) that sent a heartbeat to the resource manager (RM). The RM schedules the resources on each node.

In the process flow, a client submits the task, including specifications to launch the application master (AM). The specification includes how much memory and CPU cores are available for this job. The job arrives at the Spark Driver inside the AM. The tasks of the AM are, negotiate for new containers from the RM to process the work in, submit launch requests to run code in containers and handle notifications from RM. The AM communicates with YARN and is inside a YARN container. The AM registers with the RM. When the RM has enough memory, CPU and storage available, then another container can be deployed. The AM launches the container by providing the container launch specification to the NM. The application code executing within the container provides necessary information (progress, status, etcetera) to the AM. Spark starts executors within a container that perform the actual task. The NM reports the health of the containers. An application that has its
results closes all containers. The container of the AM deregisters with the RM, so the manager knows all the resources are free. Each executor is inside a container, the container is an encapsulation of resource elements, like memory, CPU, and storage. The containers give performance, and isolation of the data and the YARN version used[9]. Performance isolation is that jobs do not interfere with each other in terms performance. Data isolation is that users want to isolate their data using different instances of the framework, this improves the security of the data. Version isolation can allow end-users to migrate to a new version of the framework gradually.

2.1.4 Apache Spark

Apache Spark is a platform that provides Large-scale data processing engine supporting structured query language (SQL), streaming, machine learning & graph computation [13]. Spark can seamlessly combine these different processing models. Spark is an open source product originating from UC Berkeley. The community has 1000 contributors from 250 organisations [13]. Spark aims at speed, ease of use, extensibility, and interactive analytics [31]. To achieve processing speed, it can scale up from one to thousands of computation nodes. Spark runs various workflows useful for many data processing applications [31]. This section describes how Spark works. How Spark supports several programming languages is in Section 2.1.4.1. Section 2.1.4.2 describes the Spark model used to achieve its performance. The coordination of Spark jobs happens via a coordinator denoted as the SparkContext. The SparkContext is further described in Section 2.1.4.3. An example for a Spark job is in Section 2.1.4.4. The development tools are in Section 2.1.4.5.

2.1.4.1 Programming languages

Spark supports different programming languages, Python, Java and Scala, R and SQL. The source code is mainly in Scala and some Java. Spark uses Scala to compile code to bytecode and feeds it to the Java Virtual Machine (JVM). The JVM can communicate directly with Java and Scala, but not with Python. The Spark
community developed PySpark for Python. Python works because of a library called Py4J that enables communication with the JVM over sockets [32]. These sockets communicating with the JVM come at a small performance fee [33], making Python a little slower compared to Scala.

2.1.4.2 Model

Apache Spark uses a mini-batch model. mini batches are small batches collected in a buffer generated from the data coming in. The Spark engine processes the batches periodically and makes it a stream of small batches. Here Spark has the small performance decrease that for every mini-batch a complete batch processing job has to be scheduled since the batches are periodically processed by the Spark engine to make mini-batches. Batch processing is the execution of a job without manual intervention. It reduces the system overhead and avoids the idling of computer resources. A mini-batch is a batch that contains a few instances, so it contains a small amount of data, a subset. All the streaming information arrives in the form of events. When too many events arrive, Spark drops the events it can’t handle. To prevent this Spark uses Kafka, which is explained in Section 2.1.2. Together with Kafka, Spark can provide a the exactly-once guarantee. This guarantee makes sure that it is safe to receive duplicates so that it is not taken twice in a calculation. Spark uses resilient distributed datasets (RDDs). They are resilient in a way that Spark can rebuild them from a known state. Distributed because it can be distributed across multiple nodes, in Spark these are workers. Spark improves its performance by doing calculations in-memory and trying to not write to the disk.

2.1.4.3 SparkContext

The SparkContext coördinates the Spark application, the SparkContext runs inside the driver program. The SparkContext is part of a Spark cluster, visualised in Figure 2.3. The SparkContext can connect to cluster managers for job scheduling, and the managers include Hadoop YARN, Apache Mesos, and a simple cluster manager. The simple cluster manager is the pre-installed standalone scheduler for standalone cluster mode. To make the cluster manager recoverable and scalable Zookeeper can be used [34]. The Spark workers receive the application its tasks via the SparkContext.

![Spark cluster diagram](image)

Figure 2.3: Spark cluster [3]

The applications isolated on every worker have their workflow.
1. A standalone application starts and initiates a SparkContext instance (called a driver).
2. The driver program asks for resources to the cluster manager to launch execu-
tors.
3. The cluster manager launches executors. The driver process runs through the
user application. Depending on the actions and transformations over RDDs,
tasks are sent to executors.
4. Executors execute the tasks and save the results.
5. If any worker crashes, its tasks will be forwarded to different executors to be
processed again.

The cluster manager is responsible for the acquiring resources on the clusters. Here
using an advanced resource manager in the cluster manager can have a huge impact
on the performance of Spark jobs. By using static partitioning of resources, so no
resource manager, Spark holds the resources reserved by the application. Coarse-
grained scheduling distributes the resources over the applications. The isolation of
resources is enforced, by making sure that the application cannot use more resources
than it requested. Spark its scheduler YARN can give priority to a job by making
pools, each pool has its priority. To schedule the execution order in the job Spark
uses a Directed Acyclic Graph (DAG) scheduler. The DAG creates stages consisting
of several tasks. The tasks can then be scheduled to be executed by the executor as
shown in step 4.

2.1.4.4 Job examples

Many Fortune 500 companies have data that needs to be processed quickly, and use
streaming mechanism like Spark streaming. A stream is a unbounded sequence of
tuples, such tuples can processed by Spark efficiently. Spark does batch processing
in streaming, to do this it uses memory pinning. Memory pinning makes it impossible
to temporarily move the data, so it can reside in the memory and be processed fast.
Spark is used for fraud detection, this needs to be happening quickly. In Spark
programming stream is easy, a simple example of a stream application is shown in
Listing 2.1.

Listing 2.1: Spark example

```scala
val conf = new SparkConf().setMaster("local[2]")
val sc = new SparkContext(conf)
val lines = sc.textFile(pathToFile,2)
val words = lines.flatMap(_.split("\s"))
val pairs = words.map(words => (word,1))
val WordCounts = pairs.reduceByKey(_+_
wordCounts.print()
```

The usefulness of Spark is proven by the fact that it is very efficient in batch processing
and has won the yearly benchmark in 2015 for sorting 100 terabytes of data in the
Amazon Cloud (EC2).
2.1.4.5 Development tools

Apache Spark has two versions: a production version and a development version. The online production version is downloadable from the Apache Spark website [13]. Here the compiled version of Spark is obtained. To develop Spark requires the source code, this code is published on Github providing the latest development version of Spark [36]. In the Github repository are different branches each containing a different Spark development version. The master branch contains the most current version and is obtained by default, another branch can be selected when required. In the repository are different folders containing the source code. There is a core folder containing the core functions. Furthermore, there is a folder for the resource manager and several others. Each folder has a main file, where all the classes used in Spark are located, the classes are aggregated in the jar file. There is another folder test containing the classes for testing. Each testing folder might have different programming languages and has different testingsuites, to test different classes. Testingsuites are classes in which the testing happens, and these testingsuites mainly consist of assertion tests. How an assertion test happens is explained in Section 3. Assertions test the configuration variables and the software functions individually.

In Spark is a hidden REST application programming interface (API) for the user. The REST API is there to submit applications. The Spark REST API is not well documented. However, the API is useful for development and the testing of modifications to “spark-submit” [37].

2.1.5 Apache Flink

This section explains Apache Flink. The Apache Flink framework does processing and analysis of both batch and streaming data and optimises the iterative processes. After nine months at the Apache Foundation, Flink got the status of a top-level project, which is impressive for an Apache project [38]. Flink fuses the concept of Hadoop and SQL databases, the most important aspects are high performance, low latency, high concurrency and parallelisation [18]. The low latency makes it a useful data analytic tool for fraud detection [39], because the fraud detection is an iterative process that needs real-time analysis. Flink jobs are programmed in either the language Java or Scala. With the same Py4J library as in Apache Spark, Flink makes it possible to program in Python. The real-time low latency streaming is explained in section 2.1.5.1. Further, information about how job scheduling happens is in Section 2.1.5.2.

2.1.5.1 Streaming

To make streaming possible in Flink, Flink uses stream out of core algorithms. Flink achieves its low latency by scheduling a streaming job just once and continuously pipelines records through its operators. The optimisation is in the join algorithm to merge data by reusing the sorting and partitioning. The streams are moved as they arrive, allowing flexible time windows to process the data. Flink makes use of something called event time [40]. An example is a star wars movie, the fifth part of star wars came out in 1980, while the second part came out in 2002, and the first in 1999, meaning that the star wars story is not revealed in chronological order. The processing time is the year the movie came out. The event time is ordered chronological, so in event time it is star wars 1,2,3, et cetera. The benefits of using event time are that there is no dependence on processing time or arrival time. Event time can be used to get accurate results for data that is out of order. Each event has its own time. With event time it is possible to reorder events based on their event order, so when there are different events they rearrange by using the event time,
this prevents having wrong grouping at batch boundaries \cite{40, 41}. Flink transmits batches of records from the buffer over the network by using such a buffer. Flink improves the network efficiency compared to Apache Spark. There is a timeout sent to the buffer in case the stream is not fast to achieve low latency.

2.1.5.2 Scheduling

Flink makes use of scheduling to organise a job. The job manager is the coordinator of the scheduling system. The job manager sends the data to the task managers, which are the workers. Flink has a component stack consisting of three layers, the runtime layer, the optimiser, and the API layer. The runtime layer is responsible for receiving the program in a job graph. A job graph is a parallel dataflow with arbitrary tasks. The optimiser uses a directed acyclic graph (DAG) operators, examples are a filter, a map, and a reduce operator. The data can be of various types. The API layer implements the API’s that create the operator DAG. Each API provides the interaction via utilities, and an example is a comparator used for the comparison of the age of two persons.

2.1.6 Hadoop open Platform-as-a-Service (Hops)

Hadoop open Platform-as-a-Service (Hops) is a distribution for Apache Hadoop. Hops is a scalable and highly available architecture for Hadoop \cite{42, 43}. In this section a general impression of Hops is given. Section 2.1.6.1 explains how Hops works with Spark, then Section 2.1.6.2 explains project based multi-tenancy used for dataset access control.

Hops can host multiple sensitive datasets on the same cluster, providing dynamic role-based access control for both HDFS, and Kafka distributed streaming platforms. The platform is unified in an intuitive user interface and provides first-class support for Spark, Flink, and Kafka. Hops focuses on the Internet of Things (IoT) and Telecom markets as well as sensitive Big Data owners. Hops has a new HDFS implementation (Hops-HDFS). The metadata used to associate files resides in an in-memory distributed database, a MySQL cluster \cite{44}. YARN uses multiple namenodes and stores metadata in memory achieves higher scalability. In Hops the user and server share the same certificate authority and thus the same root certificate, to improve operation and ease of use. Hops does not have Kerberos, the username is in the Common Name field in the certificate, and the user stored in a database, in this way the user can be authenticated at different services. In Hops each project has a project specific users, this is done to achieve dynamic roles. A data scientist, which is allowed to write and run the code. A data owner can import and export data and share datasets and topics. Hops can contain more metadata than in Hadoop, this metadata gives the possibility to have attribute-based access control. With attribute-based access control is control over what a user does with a particular dataset.

2.1.6.1 Hadoop open Platform-as-a-Service with Spark

A Spark Job gets started with the normal Spark parameters. A Spark job that starts with YARN starts a builder which add local resources. These are the SparkJarPath, the log path and the metrics. It adds certificates obtained from Kafka. Every time a job in Spark starts Kafka distributes all the certificates. The keystore and truststore to authenticate a user are copied to the working directory in HDFS for this job. When the certificates are not there and the Uniform Resource Identifier (URI) to identify the working directories is not found, then an error will occur. The localisation process fails. This fails the creation of the YARN container, which fails the Spark Application.
The error that occurred is displayed to the user. If everything works well the job gets executed and the result returns to the user.

2.1.6.2 Project based multi-tenancy

Hops provides multi-project based tenancy. Meaning that projects are fully isolated from each other, people can be added or removed from projects with shared datasets. Every user can have a specific role in the project. Across projects datasets and topics can be shared. A Topic is a stream of records in a specific category initiated on Kafka. The project based multi-tenancy is achieved by using extensible metadata. Metadata contains logs, datasets, HDFS files, users, notebooks and access control. Isolation of highly sensitive data is important, especially financial information, that may be subject to government regulations or compliance policies that aim to protect security and privacy. Multi-tenancy requires a shared infrastructure, such an infrastructure has potential benefits in terms of efficiency, cost savings, and governance. This happens by better infrastructure utilisation, quicker starting of clusters. The infrastructure its CPU, memory and storage can be scaled more independently. The shared infrastructure eliminates the hassles and security risks of having to duplicate and store the same data for different user groups. The idea of multi-tenancy is a non-starter for most enterprises from an operational risk perspective. Before companies will even consider this approach to Hadoop multi-tenancy, they need to be confident that these tenants can share a common set of physical infrastructure without negatively impacting service levels or violate any security and privacy constraints. What’s needed is a secure multi-tenant Hadoop architecture that authenticates each user, “knows” what each user is allowed to see or do, and tracks who did what and when. Administrators are able to manage users and grant access to resources based on each user’s unique needs. With security of multi-tenancy in place, sensitive workloads are safe, because datasets can be restricted. Multi-tenancy provides data security and network isolation, multi-tenancy requires logging of what happens and permission management.

2.2 Authentication with Kerberos

In this section Kerberos MIT is explained. First with an overview in Section 2.2.1, then the advantages of Kerberos are discussed in Section 2.2.2. Further, Kerberos is implemented in the Hadoop environment. In Section 2.2.3 the current challenges Kerberos has in the Hadoop environment are explained. In Section 2.2.4 the process model of Kerberos is explained.

2.2.1 Overview

Kerberos is a venerable system for authenticating access to distributed services. The idea of Kerberos is that users can be authenticated using their systems credentials. Without Kerberos, services such as Apache Spark and Flink believe every username. This is an authentication challenge that Kerberos resolves. Kerberos resolves this by using a Secret-key distribution model. Kerberos authenticates by both encrypting and decrypting with the same secret key. This is better known as symmetric key cryptography.

Encryption = plain text + encryption key = ciphertext
Decryption = ciphertext + decryption key = plain text

In this model information needs to be passed to identify the author. The author is identified and verified, this is called authentication. Kerberos is there to make sure
that the username is being checked against a database to verify if the user is really
the person it claims to be. Kerberos is a trusted third party, the other two parties are
the users and the services [46]. Kerberos is trusted, because the clients and services
trust Kerberos that it accurately identifies the other clients and services.

An important actor in Kerberos is the principal, this is an identity in the system. To
log in as a principal there is a keytab, a keytab is a binary file containing the secrets
to log in as principal. The keytabs are used by services to authenticate themselves
with Kerberos. This also makes that every service is a principal. A principal is in
a specific realm, this can be seen as a separate part of the complete organisation
network. By using this realm the principal can only reach a certain domain, this
reduces the security risk. There can a Realm that can only access the data in Hadoop
and not in other places where data is stored. Every Realm uses a Kerberos Domain
Controller (KDC), the controller needed for every Realm and functions as a gate
inside the Realm [23]. Many people call this the gateway to the madness, because
it leads into Kerberos its limitations found in the book the Kerberos and Hadoop:
The Madness beyond the Gate [23]. The KDC is for example a Single Point of Failure.
The KDC have a centralised place to store the principals once this is hacked the hacker
has access. Once the KDC fails the user can’t access Kerberos anymore, and thus can’t
authenticate.

When a user logs in the KDC the users gets a ticket. Here it can happen that the
service granting the tickets is not found and there are no valid Credentials provided.
The obtained ticket is used to perform certain actions at a service. The ticket is used
to identify the user at the services. The ticket can be passed on to other services, and
thus makes an efficient distributed authentication mechanism, by which the user does
not have to be asked every time to authenticate himself. The tickets could potentially
be changed. The ticket is used for a finite time. To improve the security Kerberos has
the option to offer time limited tickets, if the time is not roughly consistent across
machines this won’t work. An error comes up which says that the clock skew is
too great [23], this means that the time in the machines differs too much from one
another. A stolen ticket can be used directly to do things on a service, since there is
no logging in the KDC, the administrator won’t notice if a ticket is changed. Another
problem is that a user might evade the authentication by reusing a Kerberos ticket
and impersonate someone, hereby it achieves obscurity [47], [48], [49]. If there is
access to a local administrator, the golden ticket can be obtained by which access to
everything is obtained, this can be seen as a forged KDC and basically means that the
whole Kerberos is compromised [47], [50]. The information can obtained slowly and
persistently by the attacker in order to remain uncovered [49]. When the attacker has
access, the attacker can put additional credentials in the system by which the attacker
can login again [49].

Microsoft focuses on making sure that Kerberos works with almost all the prod-
ucts. This causes that the security default can be relatively weak due to old products
supported and thus leaving some legacy risks. The configuration has to be carefully
checked and there cannot be assumed that the default is good enough. To configure
Kerberos is time-consuming, and it is difficult [48]. There are more legacy problems
and limitations, for example it might be that with only a single password there is
the possibility of a password guessing attack [51]. To strengthen the access Kerberos
supports pluggable authentication modules (PAM) to allow MFA. MFA is explained in
Section 2.4. Kerberos allows integrated PAM via Remote Authentication Dial In User
Service (RADIUS) a networking protocol for remote authentication and lightweight
access directory protocol (LDAP) [52]. There has to be a radius server which commu-
nicates with Kerberos to then authenticate the user using one authentication factor
from the PAM and the other from Kerberos [48]. In the future Kerberos will stay there
for on-premise solutions, but the community is now driving more for cloud solutions as well which support various authentication mechanisms.

On 11 July 2017 a new a bug called Orpheus’ Lyre is found. This is a mistake in the ticket system found in all the versions accept for the original version Kerberos MIT. The bug caused metadata, such as the ticket its expiration time to be taken form unauthenticated plaintext instead of the authenticated and encrypted KDC its response, by this attackers can gain the opportunity to impersonate services making Kerberos useless. Kerberos has some more general limitations, many limitations and advantages of Kerberos are found in the book, Hadoop Security: Protecting Your Big Data Platform by Ben Spivey. Kerberos does not address the data encryption, this is done by the Transport layer security discussed in Section 2.3. The user himself has to secure the data transport, otherwise it might be that tickets get intercepted, or communications forged. Applications and systems that rely on Kerberos often have many support calls and trouble tickets filed to fix problems. Kerberos lags behind in providing support for new features which causes problems. There is for example no fine-grained authorisation support. The limitations and troubles described often intimidates even experienced system administrators and developers.

### 2.2.2 Kerberos advantages

Kerberos has several advantages as an authentication mechanism. During the implementation period of Kerberos, there was an argument between using Kerberos its secret key model or by using public key cryptography. The public key cryptography is explained in the Section 2.3 about transport layer security, which makes use of the cryptography. Most of the arguments were in favour because of performance. The secret key model in Kerberos is a symmetric key operation, which is a little faster than public key cryptography. The authentication in Kerberos always happens before any sensitive data is exchanged. Kerberos separates the authentication away from the services that perform the work. Kerberos has simple user management, for example, revoking a user can be done by simply deleting the user from the centrally managed Kerberos KDC. To do the sign-in with Kerberos is easy, it only needs to happen at one place and then every service can be used. So the authentication is simplified. There is one central point for key storage, this means there is only one system to manage server access. This place is also the place where all access activity is logged, this is beneficial for auditing. Kerberos is supported by many operating systems and it is mature. Kerberos makes sure that the password is not transmitted over the network.

### 2.2.3 Kerberos challenges in Hadoop environment

This section explains the challenges in the Hadoop environment, and the process model of Kerberos for an Apache Spark application. In the Hadoop Environment Kerberos provides a special secret called a delegation token, to let the user access Hadoop. The token has a maximum time to live of 7 days. There can be up to several thousand of services requesting a token during startup, this causes lagging when starting up all the services. The KDC in Kerberos does not allow multiple concurrent log in of user accounts at a scale distributed applications need. The KDC can be replicated to allow more login request, but this has its own disadvantages. The Hadoop environment uses delegation tokens. The delegation token causes problems when you want to let the users gain access via a browser, since Kerberos will then limited browser access to prevent exposing the ticket.

To help you with resolving errors in the Hadoop environment there is a software called Ambari, that helps with automating and managing integration with a cluster. The Apache Hadoop environment is evolving fast, and the market demands
other authentication mechanisms to be available \cite{54}. Cloud services do not support external authentication via Kerberos, since it exposes password vulnerability \cite{56}. For example Microsoft Azure allows OAuth2.0 and OpenID. With the changes in the market, Microsoft the original developer of Kerberos is now driving for cloud identity standards \cite{56}.

2.2.4 Kerberos process model for Apache Spark

In this section the process model of Kerberos is explained for an Apache Spark job. With Kerberos the user authenticates. There is a client. A Kerberos instance, which can authenticate the users. The Spark main application which wants to perform the job and the application its workers. Moreover, there is a logger, which is responsible for auditing authentication attempts. In the use case the user starts a job with Kerberos authentication. The authentication is either successful or it fails. The authentication attempts get logged with a logger, for auditing purposes. An unsuccessful authentication triggers an error for the user. To authenticate the user has to provide a principal and a keytab, one of the errors is that one of the arguments isn’t supplied. The authentication is successful the flow continues. This is displayed with the Alt, for alternative and the two guards. When the flow continues Spark can start the workers to execute the job. These workers return a result which Spark processes. The result is returned to the client.

\begin{center}
\includegraphics[width=\textwidth]{sequence_diagram_kerberos.png}
\end{center}

\textit{Figure 2.4: Sequence diagram Kerberos for Apache Spark job}

2.3 Transport layer security (TLS)

The data transport security is provided by the TLS. The goal of TLS is to provide privacy and data integrity between two communicating parties, achieved by a TLS handshake protocol. In Section \ref{2.3.1} an overview is given of TLS. In TLS data integrity is very important to provide this a mechanism called chain of trust is used, this is explained in Section \ref{2.3.2}. There is explained how the trust is used to make the
transportation is secure. The infrastructure required to set up TLS is in Section 2.3.3. At last, the encryption overhead costs of TLS is in Section 2.3.4.

2.3.1 Overview

Secure Sockets Layer (SSL) is the predecessor of TLS, a public key encryption technique. SSL tries to achieve privacy, integrity and trust. Privacy meaning that people can’t look directly at your password. Privacy in SSL and TLS is achieved by symmetric cryptography. integrity meaning that the data has not been altered when transported, trust meaning that you are who you say you are. It is based on the usage of certificates. SSL is often used to achieve non-repudiation, this means that the certificates are used to ensure that the data that is sent hasn’t changed. SSL makes use of a Public key infrastructure (PKI). The PKI is part of public key cryptography proposed by Diffie and Hellman in 1976 [57]. Public key cryptography uses a pair key. The public key to be distributed over the network to other parties and a private key that is protected and hidden from the network. The main idea here is that the data encrypted with the public key can only be decrypted with the private key and the other way around. Public key cryptography is used to authenticate communicating parties. To authenticate a text is encrypted with a private key, the receiver can then use the public key given by the sender as well. To encrypt the text the sender encrypts the text with the receivers its public key, only the receiver with the private key can then decrypt the text. TLS is built on the assumption that it’s computationally infeasible to calculate the original secret from the encrypted messages. Neither the secret nor any encryption keys are communicated over the wire. TLS prevents this by performing mutual authentication, both ends need to know the shared secret to be able to decrypt session data. Even if an attacker is able to insert a new malicious communication address, the attacker won’t be able to read any of the data exchanged between client and server, because the attacker does not have the secret to decrypt the message. This prevents that the transported data can be easily read by an attacker. Such an attack by the attacker is also called eavesdropping, which means the attacker is silently listening and understanding what you communicate. TLS uses a certificate which contains certain important information such as the start date and time, and end date and time. From the start end end time the valid time is calculated. A computer can be out of sync, so it can for example be that you obtained a certificate at 11 February 2017 at 11:30 and you present it to another computer on 11 February 2017 at 11:10. To overcome this problem TLS needs a Network time protocol (NTP), this protocol helps to synchronise the clocks of different computer systems. A certificate is like an ID card, the card contains information about the owner, and the purpose is clear. A certificate contains information with what it is meant for, for example a website name and the authority that granted it.

2.3.2 Chain of trust

A chain of trust is there to establish trust. For example, an SSL Authority has a Root an intermediate, there is a client that has a certificate from the intermediate. The intermediate trust the root, the client trusts the intermediate, and this creates a chain of trust. The intermediate trusts the client. This all can be verified, by verifying the chain, the chain of trust. The following example makes this more clear. Let’s say you want to buy a car from a dealer, you go to the dealer. The problem is you don’t trust him. You just don’t know him, the car might be stolen. He says the car is not stolen, but you have no idea. There is a police officer you know and trust implicitly, even though you never met him. Then there is your friend Bob, you trust him immediately. This is your intermediate, you trust it implicitly. Your friend can confirm who the
dealer is, so the first chain of trust is built. In this story the police officer is the root (certificate authority) he hands out the certificates and knows which cars are stolen or not. You are the client. The dealer is a server, and your friend is the intermediate.

2.3.3 Infrastructure

In a Public Key Infrastructure there are Certificate Authorities (CA’s). The CA signs and verifies certificates. The CA maintains a certificate revoke list (CRK), in this list certificate information is stored and certificates can be revoked. A revoked certificate cannot be used for verification by a user. In TLS once a new certificate revocation list is made, it has to be propagated and maintained on all the servers that allow access to users in this infrastructure. In the whole internet there are multiple public key infrastructures, and they may or may not communicate with each other. To start TLS and make sure non-repudiation is guaranteed an SSL handshake has to be performed, this handshake is presented in Figure 2.5.

![Figure 2.5: SSL handshake](image)

When the handshake cannot complete the connection is not established and the users cannot communicate. Once the handshake is complete the client and server establish a stateful connection. A stateful connection means that the state of the connection is maintained by TLS. The connection happens over secure sockets. Certificates are issued at CA for an identity. When a certificate is requested from a free Certificate Authority, for example your own created certificate authority, they are usually not automatically trusted. Using a self-signed certificate means the trust is gone, because trust is built by the public and thus public certificate authorities. When you hand out the certificate the change of trust changes, because it means you give the ID card to your self, so there is no public authority that says it is true. On the internet own signed certificates are in general not trusted. In web browser it will display a scary warning message telling your visitors that the certificate is not trusted. For this reason, Financial and e-commerce websites always use a trusted CA and most of them pay for this. This because the purpose of the SSL certificate is that it is assured by a trusted third party that the visitor is speaking to the right web browser. There is also a special type of certificate that has extended validation, they are called Extended Validation SSL certificates, and they are used by for example banks. They offer an extra form of validation. They are recognised in web browsers by a green bar. These certificates offer extra trust, because to obtain such a certificate from an important CA there is an extensive evaluation with several criteria, to see if the person can be trusted.
2.3.4 Encryption overhead costs

The addition of SSL causes some encryption overhead \[^{[54]}\]. How much overhead is caused can be seen in the Table 2.1. The performance test is with Kafka on an Amazon machine with 4 CPU cores, an 80 GB solid state disk and a network of about 90 MB per second. The network can be utilised almost fully without SSL, but with SSL about 10 MB is overhead for SSL also the CPU performance increases with at least 15%. How Kafka works is further explained in Section 2.1.2.

<table>
<thead>
<tr>
<th>Throughput MB/s</th>
<th>CPU on client</th>
<th>CPU on broker</th>
</tr>
</thead>
<tbody>
<tr>
<td>consumer (plaintext)</td>
<td>83</td>
<td>8%</td>
</tr>
<tr>
<td>consumer (SSL)</td>
<td>69</td>
<td>27%</td>
</tr>
</tbody>
</table>

2.4 Multi-factor authentication

Multi-factor authentication is important to prevent the account from being breached \[^{[58]}\]. In MFA it is important to have integrity, confidentiality and non-repudiation. Non-repudiation means that the author is not able to change the ownership. One of the most used second factor authentication mechanisms is the Time-based one-time password (TOTP). The TOTP is specified in RFC 6238 and can be used as a factor for two-step authentication \[^{[59]}\]. It is based on the one-time password (OTP) RFC 4226. The TOTP authentication requires six digits that are only one-time generated. Keyed-Hash message authentication code (HMAC) based OTP (HOTP) specified in RFC 4226 is an event based algorithm where the moving factor is a counter. The TOTP provides short-lived OTP values, this is desirable for good security. A TOTP password gets renewed on default every 30 seconds. When using a larger step size than 30 seconds there is a larger window for attack, however when the window is too short, then the usability highly decreases. The waiting time for the new token is part of the 30 second window. The one-time means that the verifier only accepts the password one time and after that refuses it. In the architecture below is explained how such a TOTP works.

![Figure 2.6: TOTP architecture \[^{[5]}\)](image)

The open source Google Authenticator is considered as one of the algorithms using TOTP \[^{[60]}\]. The Google Authenticator can be installed on a device. To attach it to a database or Kerberos a Pluggable Authentication Module (PAM) is required \[^{[60]}\]. Using a Google authenticator is a trade-off between security and usability. One of the platforms where a MFA is implemented with Google Authenticator is called BioBank.
In the scenario of using a Google Authenticator the user provides six digits, and gets then asked for a TOTP. The Google Authenticator TOTP generates six digits passwords every 30 seconds \(^{[61]}\). The user password is strongly encoded and this encoded password is in the credential store as well. The TOTP secret is a 128 bit secret that is base-32 encoded \(^{[61]}\). This secret is stored in for example a database. A Google authenticator requires a PC running Linux a smartphone and a server. This means that there is a separate database needed for storing the authentication data. Kerberos can also make use of MFA, by using LDAP \(^{[52]}\).

There is a push-based mechanism that can be used on a smartphone for authentication. Such a push-based notification mechanism sends when logging in a message to the phone. On the phone the message can then be verified and thus one authentication is provided. The second authentication can be your password. Not all users have a mobile device, another solution can be by using a Yubikey one-time password (YOTP). A Yubikey is a device that the user can carry and attach to their computer for authentication. It is a Universal second authentication (U2F). YOTP is an HOTP. A YOTP converts a received one time password to a byte string. This is decrypted using AES \(^{[61]}\), after decrypting the string checksum is checked. A non-volatile password will then be compared with the counter in the credential store. If the counter is bigger than the one in the credential store it will be validated. This device contains a private key can then be used to sign a shared challenge and the server can with a public key verify if the authentication is correct. A U2F uses an NFC or a USB device.

The challenge contains a user it’s private key as seen in Figure \(2.7\), this key is the user its password. Then and together with the OTP in Yubikey extracted from the device the user is verified against the public key. The Yubikey is a 44 digit code generated by the device for the specific counter. MFA has some disadvantages, MFA doesn’t secure when the endpoint is compromised, and there is one more process step to enter the second factor. The secret that is required for TOTP can be exposed during registration. The safest MFA mechanism is a private device with a private key like Yubikey. Even those devices might be exposed, so it is beneficial to use an authentication mechanism with multiple authentication factors.

### 2.5 Distributed databases

Databases are used for authentication. Many websites have a database with an username and a password to login on their website. In the database a secret and the username and password are stored. MFA discussed in Section \(2.4\) requires a database. Databases for authentication require consistency and availability. The
CAP theorem is important for distributed databases. The CAP theorem is useful when partitions happen, this is the moment where data is shared between databases to keep them up to date. In the theory three terms are important, consistency, availability and partition tolerance. **Availability** is that every request receives a non-error response, this is without the guarantee that it contains the most recent write. **Partition tolerance** is that the system continues to operate despite an arbitrary number of messages being dropped by the network between nodes. The CAP theorem states that it is impossible to have all three of the guarantees simultaneously, so one can only pick two. **Consistency** means that every read receives the most recent write or an error. In the CAP theorem only two of the three described terms can be chosen. In the absence of a network failure both consistency and availability can be tolerated. When network failures are tolerated one can choose for a system that is always available, but that does not guarantee it is the most recent write, so it can contain a not up to date message. When choosing consistency it can over time create a time-out, or it will return an error. To achieve consistency the two phase commit protocol (2PC) and foreign keys are used. A foreign key is to link two tables together. This is achieved by uniquely identifying a row of another table, or the same table. The 2PC is used to make sure that the data is distributed across different databases. 2PC achieves strong consistency and when the data can’t be committed across the databases a rollback is performed, during this time the database is blocked. The two phases are the commit request phase and the commit phase. In the commit request phase there is a coordinator which asks if the databases can commit or whether there is a problem. Here it is assumed that no database crashes forever, if there is a database crashing many times some data might be lost and a database administrator has to look at it. The commit phase starts if all databases say they can commit. In the commit phase the databases commit the data to each other, then when finished the locks are released and the database can continue the work. Ones all the databases are finished the coordinator is cleaned. When a database achieves availability and consistency it is most likely a relational database management system (RDBMS) [62]. A RDBMS uses a relational model, a relational model defines information as related entities with attributes across different tables. Two popular RDBMS to store 2fa solutions in are MySQL and PostgreSQL. They are popular, because they are both strongly consistent and highly available databases. MySQL is a database that is feature rich and has many open source versions. The database is used a lot for websites and applications, and there is a lot of information to be found about it online. MySQL has a lot of build in security features, to make sure that the database can be protected properly. MySQL is scalable, and works efficiently, to gain fast access to data. To be scalable MySQL has three different nodes, the data node is for storing the data, the management node is for the management, and the SQL node is for access and each has an identical view compared to the other SQL node. One of the disadvantages MySQL has is that it is bought over by Oracle that does not invest enough in the further development [63]. The community still supports the database and there are fully integrated databases that add value on top of MySQL. Another problem with MySQL is concurrent read and writes. PostgreSQL is an open source RDBMS capable of handling many tasks efficiently. PostgreSQL achieves concurrency, since it has no read locks as MySQL does. PostgreSQL achieves data integrity, this can be achieved, because it has atomicity, consistency, isolation and durability (ACID). In MySQL the data integrity is achieved by for example setting ranges to an age limit etcetera. PostgreSQL has a disadvantage that the replication is harder to set up compared to MySQL. For the PostgreSQL or MySQL database it is advisable to restrict the access granted to a database. This because it is dangerous if every user or application can change all the data in a database and can thus completely ruin a database which
is used for authentication. The privilege for an administrator is to view data, to delete data and to insert data and create tables. For an application administrator, the database access is different when it for example only has to view data, the privileges can be limited so that if hackers try to hack the application the hackers cannot easily change the database. Some users may even be allowed to only insert data in certain tables.

2.6 Certificates on demand

On demand certificates are for a short time valid, they have an expiration date, expiration dates are useful on for example food. A reason for this is trust, there is more trust in things that are sometimes renewed. The usage of certificates improves and distributes trust, since there is a trusted intermediary, both trusted by the user and the server providing the information, as explained in Section 2.3. Short-lived certificates can reduce the time of data compromised, by having the time window for certificates as short as possible [64]. The Certificate Revocation List (CRL) can be shorter, which is important for the management of certificates. There is a limited time access advantage, because when an attacker compromises a certificate’s private key, they may bypass revocation checks. They can then use this certificate until it expires. Shorter lifetimes decrease the time window and possible time being compromised, this prevents HeartBleed attacks [64]. With short living on demand certificates such attacks are prevented. Replication of certificates is difficult. In a browser world the machine clock is often wrong. This means that the time on one browser is an hour or even more ahead compared to the other. The irregular times on machines means that certificates have to be long valid, requiring the browsers to only be refreshed every three months and leaving an opportunity for attack. The aligning of time on machines is thus crucial to make it possible that certificates have a short validity period. A certificate is given a creation time from the certificate authority. This can be used to limit the time window, however keep in mind that not the time of every server that gives the certificate is the same. When the certificate authority is using the NTP it can be aligned with the servers that will use the on demand certificate, if they have NTP as well. Then the certificates can be valid for a couple of minutes which will highly improve the security in today’s world. The issuance capacity of certificate authorities gets higher with on demand certificates, because still many certificates are issued and then discarded, the certificate authority needs to store the data of the certificate when valid [65]. Another advantage of having short valid certificates being given by certificate authorities is, that it encourages the users of certificates to automate the issuance [65]. Automated issuance decreases accidental expiration, accidental expiration can be dangerous, because this is an opportunity for attack, and the users often receive annoying errors. The automated issuance is not used a lot yet, since there is not much support to automate this for popular web servers [65]. Another disadvantage is that IRC, mail and VPN require a restart to load the new certificates, so it is not in all scenarios useful. Another problem is that when the certificate authority is not available, the renewal can fail, when the renewal is not achievable quickly, then the service provided with the certificate can fail. Encouraging automation is nevertheless essential for ease-of-use especially since many web servers are moving to HTTPS, which requires certificate renewal once in a while. The renewing of certificates allows us to advance in internet security technology. For example the upgrading the certificates that are used for longer than a day from 2048 to 4096 bit encryption, because otherwise it is easy to decrypt by hackers. An on demand certificate can be used in Spark to authenticate every Spark job.
2.7 Apache Hadoop environment Security

The Hadoop environment is becoming more complex, and so as well the security. Hadoop has several distributed version, there are different versions of Hadoop, Spark, and Flink. The deployment of the infrastructure for security is complex, with the several security layers. The security layers are highlighted in Section 2.7.1. One important aspect of the security layers is the authentication and security. One of the security layers is the authentication layer which makes use of Kerberos and delegation tokens. In Section 2.7.2 there is more explained how this mechanism works. In Section 2.7.3 the data protection layer in the Hadoop environment is explained, this layer makes use of the TLS. The security for the resource manager YARN is in Section 2.7.5. Kafka has its own authentication mechanism based on TLS and this is explained in Section 2.7.4. The data processing engines its security is described in Section 2.7.6 for Spark, and Flink in Section 2.7.7.

2.7.1 Security layers

The Hadoop environment consists of several security layers. There is a operating system (OS), authorisation, authentication, data protection, and perimeter level security layer \([22]\). The OS security layer is there to protect individual nodes in a cluster at the operating level \([22]\). The OS layer protects data at rest, ensure administrators and applications cannot directly access files, and prevent information leakage. The authorisation layer in Hadoop is responsible for the authorisation of the Hadoop components. The authorisation layer is mainly secured by Apache Ranger. Apache Ranger tracks module configuration, and to set usage policies for fine-grained control over data access and thus authorisation \([66]\). Apache Ranger supports different authorisation methods, these can be role or attribute based access control. Apache Sentry is another system in the Hadoop environment to provide authorisation for the HDFS. The HDFS provides higher level abstractions. For example if you want to have the permission to read a table it means you have the permission to read the table files. If you want to have the permission to create a table you have a permission to write in the database’s directory. The authentication layer is used to validate nodes and client applications before admission to the cluster \([22]\). This is mostly being done by Kerberos. Further, there is Apache Oozie for authentication, it is a trust service. It can be used to authenticate a user for HDFS and MapReduce and is used by for example Yahoo \([6]\). Oozie is configured as a super user and may act in the name of other users. The perimeter security is important to limit the access to the Hadoop cluster. This is mainly solved by using a gateway, such as Apache Knox. Apache Knox is an Apache project for API and Perimeter security \([52]\). It enables integration with different authentication standards \([52]\). It supports LDAP, SSO, SAML and other authentication systems \([52]\). It provides a single SSL certificate, multi-cluster support, a single access point and Kerberos encapsulation. It protects network details and provides SSL for non-SSL services. A further explanation about how the data that is transported in the cluster is secured is discussed in Section 2.7.3.

2.7.2 Hadoop environment authentication and security

In the Hadoop environment the HDFS is more and more becoming a large silo where the data is stored. Hadoop stores sensitive data so it is essential that this is secured. In this section the further authentication and security are explained. For HDFS the clients authenticate themselves via the NameNode using Kerberos and delegation tokens. The client can then authenticate to Data Node via a block access token, the DataNode authenticates to NameNode. An overview is displayed in Figure 2.8.
The user authentication gives a general Hadoop token. The Hadoop token does the following, it tracks the individual services. These services are granted access by using authentication tokens, these are the block tokens and delegation tokens. The authentication tokens are to authenticate at another service. The Hadoop token can be renewed via an API. This happens by using the RPC and the HTTP protocol. The token is also revocable via the API. This is useful when the Hadoop token is stolen. Kerberos is used for the authentication and each Hadoop service must be configured with a Kerberos principal and keytab.

In Hadoop most of the communication happens over the RPC layer. RPC can encrypt data on the wire by using Kerberos. When wanting to access the Hadoop data in for example the internet this is not possible with RPC. HTTP can be used to transfer data to other services such as YARN. A danger of using HTTP is that it transfers data in clear text. An improvement is to use HTTPS which is achieved by using TLS. The HDFS works together with Kerberos to make sure the data is only accessible by the right person when using Apache Spark and Flink. The access to the filesystem and the Namenode is authenticated to prevent directory and file system manipulation. The joining of the nodes storing the data, the Datanodes is authenticated by Kerberos. This prevents malicious code from claiming to be part of HDFS and having blocks passed to it. The Namenodes are authenticated with the Datanodes, this prevents malicious code claiming to be the Namenode and granting access to data or just deleting it. The Namenode is responsible for generating the secret that is used to authenticate within the HDFS. The users are granted write access to data within the HDFS after they are authenticated with Kerberos.

Secure information about running and pending tasks in Hadoop is important. The task configuration and input splits are stored securely, the task location and secrets are stored in a secure place within Hadoop. There is a Job tracker that creates a random token, to connect it to a TaskTracker and authorising the HTTP to get the data for a shuffle. The jobToken inserted is for the task and has an URL that points to the Tasktracker. The authentication for the Name Node and Data Node happens with HTTP and RPC. For HTTP the Hadoop environment makes uses of Netty for secure data transportation. It is a network application framework for rapid development of maintainable high-performance protocol servers & clients. The data transfer in Netty is possible by a fast message event driven application Akka. An overview of the complete authentication for the jobs is in Figure 2.9. The third party messages and Oozie are not required to run a Spark job.
2.7.3 Hadoop environment data transport security

The data protection layer is there to secure the storage and the data being transported within Hadoop. For securing the storage there are authentication mechanisms explained in the previous Section 2.7.2. In this section the data protection during transport is discussed. For the data transport security the Hadoop environment makes use of the SSL keystore factory. For the transportation TLS is being used, which is made possible by the SSL keystore factory. The SSL Keystore factory is a service that manages core services that communicate with other services when running in a cluster. There is a keystore to provide credentials/keys and certificates, and the truststore, to verify credentials. The keystore is a kind of database. The truststore contains public keys from certificates, the private keys in the keystore can authenticate themselves against these public keys. For this reason the public key of the certificate authority has to be in there. A keystore database contains cryptographic keys, such as certificates, trusted certificates. The keystore protects the private keys with a password. Furthermore, it contains the certificates necessary to create the chain of trust. Each keystore has a unique alias, such an alias is a name that can consist of special characters like the underscore (“_”), but cannot consist of white spaces. The keystore factory can be used by services such as YARN and HDFS. When using for example Spark Standalone the keystore and truststore are stored with the use of the Java development kit (JDK), on which Spark runs. The JDK provides a library keytool that is used by Java for authentication. The keytool is a key certificate management utility. Allowing users to manage their private and public key pairs and allowing them to manage their certificates. The keytool makes it possible to authenticate users for the HDFS, Spark and Flink. This part of the system requires authentication with the keystore. The disadvantage of the JDK with no distribution mechanisms build in, like for Spark and HDFS, is that it requires a restart when the keystore or truststore is updated. The YARN and HDFS can act as clients and services. For this reason they must have access to truststores, so that each service can be authenticated together with the keystore. The keystore factory supports the possibility to use HTTPS instead of HTTP. There is a common location for the keystore and truststore that can be used by all Hadoop services. The certificates are with this factory centrally managed and distributed to all the nodes when needed.
2.7.4 Kafka Security

Authentication of Kafka can be done with using TLS together with an authentication mechanism [27], [71]. Furthermore, in Kafka Kerberos can be used to authenticate [27]. For the TLS authentication in Kafka the Common name (CN) or the subject alternative name (SAN) of the certificate must match the Domain name server, to make sure it connects to the desired broker [71]. Kafka works just as Spark and Flink with a keystore and a truststore to setup TLS. This keystore and truststore has to be set in Kafka its own configuration file [27].

2.7.5 YARN Security

The security for YARN is written in Scala. There is a credential manager to give the credentials for the Hadoop file system, this credential manager is discussed in Section 2.7.3. A useful option maximum attempts defines how many attempts and application/task may take before it has to fail. YARN allows job queues to be restricted to different users and groups with the use of Kerberos. It set’s the CPU limit and memory limit. With YARN there is an option “*” that gives everybody authorisation for that part, this function is misused a lot in the same way as the Hadoop root user is misused. YARN has a access control list that controls which user has access in YARN and which user not [72]. YARN applications can consume so many local resources that they might decrease the performance of other applications running on the same nodes. For this reason the YARN needs to be throttled, this can be set by changing the configuration of YARN or the application that uses YARN, for example Apache Spark.

2.7.6 Spark authentication and security

In this section there is explained how the security and authentication of Spark works. First there are the general settings and specifications of the Spark security, then how to secure Spark Streaming, and then how to set the Spark security for different cluster managers.

In Spark every connection can transmit sensitive data. Input data transmitted via broadcast, data during shuffles, data in serialised tasks and files uploaded with the job. In Spark the data can be "in-transit" or "at-rest". In transit is the data that is being transported over the network. At rest means that the data is stored persistently, so it remains after the power of a device is shut off. Spark has an own security backbone consisting of Kerberos, for authentication HDFS, for file storage and YARN for resource management [55]. To use the Spark security the Spark authentication and encryption have to be enabled. To have data encryption when data is being transported between the workers the TLS is used. The TLS works with a keystore and a truststore that the client has to set up himself. The SecurityManager inside Spark is responsible for verifying the client and truststore. It is responsible for the authentication as well. It uses the Java security keystore and x509 certificate module, the Java Authenticator, and Javax SSL module. The authenticator obtains the password and user information. The certificate module provides a standard way for accessing the certificates. In a keystore there are the trusted certificates, containing the certificates requested by the client and the client its private keys. There is a TrustManager factory that has all the instances of the keystore and truststore. The instances then get verified using a X509TrustManager. The instances then get verified using a X509TrustManager. The keystore and truststore need to be installed on every node [72]. For setting up security in Spark the configuration file of Spark, the spark-defaults.conf file has to be adjusted, instructions to set this up for both the encryption and security is found
in the following citation [73]. In this section the most important ones are provided. To enable SSL there is an option to enable it. When using the file system SSL also needs to be enabled for this system. The password for the keystore needs to be set. The path to the keystore needs to be set, in Spark the paths are absolute and start from the folder where the job is executed, so if the folder is the Spark folder, then this is taken as the base. The path to the truststore needs to be set. The truststore its password needs to be set. The protocol needs to be set, this is most likely TLSv1.2. The algorithms need to be set. To set up the security for the user interface Spark supports the authentication access control list (ACL) [55]. The ACL can be used to prevent for example that all users can access the user interface (UI). There is a ACL list for a view of the UI with all the running processes in Spark. There is another ACL list to specify who can modify Spark jobs. In the Spark user interface users can see logs, this can be useful for providing debugging and traceability. Debugging and traceability are essential parts to increase the security within the organisation. Spark has a special module log4J to log information. In the Spark configuration folder there is a template to turn on the Spark logging system. The Spark logger uses a web server to share for example the UI.

Spark has different cluster managers. Apache Spark now supports three cluster managers, YARN, Mesos and Standalone. Standalone is just a simple cluster manager with no sophisticated managing. YARN has a resource manager, how this resource manager works is explained in Section 2.1.3. In Spark when enabling secure data transportation for the cluster manager the key- and truststore have to be set. How this are set differs when having a cluster of multiple nodes, because then there is a possibility to use different cluster managers. YARN is set in the configuration files. Mesos is a more general cluster manager. This section focuses on how to enable the data transportation for Standalone and YARN. For Spark Standalone the user needs to provide the key- and truststore configuration options for every node. The configuration can be set by adding the options in the spark-defaults.conf file, or by giving adding them as Java system properties in the spark-env.sh file. YARN organises its resources by using a resource manager. For YARN the keystore can be prepared on the client side, then it can be distributed as part of the application to all the executors. This is possible, because the user is able to deploy files before the real execution of the application is started. For long-running applications it is required to write to the HDFS, because the in-memory is not suitable for this, so the for example streaming data needs to be stored on storage. To be able to write to the HDFS the principal information needs to be given once in a while. The principal is in HDFS responsible for renewing the delegation tokens.

2.7.7 Flink Security

Flink security is achieved by using Kerberos for authentication and SSL for data protection [74]. Flink currently only supports Kerberos for authentication, even though they are aware that other arbitrary authentication methods, like TLS or username/password security are required in the future [74]. For example to assure data integrity and data privacy TLS has to be enabled on all components [75]. Flink currently has still some security errors. Currently improvement in the configuration files are needed for the SSL to be updated automatically. The Mesos environment for Flink is currently also missing security. Mesos is another resource scheduler compared to YARN.
2.8 Bastion lambda ephemeral SSH service (BLESS)

Bastion lambda ephemeral SSH service (BLESS) is used as an authentication mechanism by Netflix [7]. BLESS is implemented in 2016 and presented it at the OSCON conference. BLESS is released under the Apache 2.0 licence. The Github repository for BLESS is in the citation [76]. BLESS authenticates users for SSH services by using certificates. To make the security of Netflix more independent, the mechanism hosted on Lambda from Amazon. Every user has a separate account, each user can issue an access certificate, each user has specific permissions. The golden rule for permissions is that, too many permissions nobody complains until there is an accident. Too few and you can’t access. Lambda makes it possible to accept requests from servers only with the right keys. BLESS uses AWS Lambda from Amazon to run a small part of their code [7]. The users are stored in the identity and authentication management (IAM) of Amazon. IAM is a service to manage permissions and users. Lambda has several advantages, it automatically adds and drops servers based on consumption, it is fault tolerant and automatically receives the latest infrastructure updates [77]. It also automatically scales to support the rate of incoming authentication requests [77]. The next Section 2.8.1 explains the SSH access that is obtained in BLESS. Section 2.8.2 highlights the authentication and the SSH certificates. The Section 2.8.3 describes the logging and tracking within BLESS. The process flow of BLESS is explained in Section 2.8.4.

2.8.1 BLESS SSH access

BLESS is used to gain the Secure Shell service (SSH) access. The employees use BLESS to gain access to the company infrastructure [7]. This is needed for running and maintaining production systems. SSH has several dangers, there can be a malicious insider with SSH access, they can then "hack" into the system and steal the data [7]. The Keys used for gaining access to parts of the infrastructure can be stolen or are accidentally published in a public repository on for example Github. To resolve this Netflix makes sure that the keys can be easily updated with new keys. The infrastructure of BLESS enables SSH in such a way that it is hard for malicious insiders to enter everything. SSH keys can be distributed through special hardware such as a dongle, but for thousands of employees this might not be the best. Servers with SSH should be checked for undesirable SSH access. It is advised by Netflix to use multiple authentication servers, to provide the SSH access for many developers. Lambda can provide this servers, the servers can scale and maintain consistency.

2.8.2 BLESS authentication and certificates

By using an online authentication mechanism users can self-manage the biggest part of authentication. The mechanism is built in Python and is open source. BLESS uses MFA to authenticate a user. MFA is earlier explained in Section 2.4. If this time is not logical for a employee to log in they can warn the employee. If employees expose the private key they get a notification to their device. In the login the location is stored as well, an login from an odd location causes a warning. The user authenticates at a central bastion this is shown in Section 2.8.4. This is a kind of login portal, the address is known to the user, and the user trusts the bastion as a login portal. The bastion redirects the user after the login to its instances.

MFA takes some time, to resolve this certificates are generated. Certificates are used as a single sign-on for authentication at services. The login authentication should be used for a limited time and then the user should authenticate again with MFA. There are two types of certificates, user and host certificates. Host servers are
used to trust Authorities. User certificates are to authenticate and using SSH. The
sessions that BLESS establishes stay after the certificate expires. BLESS constructs
signed certificates using a Python cryptography tool, located on Lambda. The tool
communicates with the database containing information about the user and a Certifi-
cate Authority to sign the certificate. Both the database and the Certificate Authority
are scalable. The information in this certificate is shown in Table 2.2. For the Netflix
Certificates a trusted certificate authority is required to sign the certificates.

Table 2.2: Netflix SSH & RSA certificate

<table>
<thead>
<tr>
<th>Table name</th>
<th>The kind of certificate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key-ID</td>
<td>Any ID information you want.</td>
</tr>
<tr>
<td>Valid</td>
<td>specify how long it is valid.</td>
</tr>
<tr>
<td>Principals</td>
<td>Can be an account, application, username or instance.</td>
</tr>
<tr>
<td>Critical option</td>
<td>Specifies from what Authority the certificate can come from.</td>
</tr>
<tr>
<td>Extensions</td>
<td>To control what SSH sessions is used for.</td>
</tr>
</tbody>
</table>

Currently, BLESS is currently used by other companies than Netflix, for example Lyft
did a variant of a BLESS installation [78]. The Lyft BLESS installs short-lived client
certificates on workstations and removes the bastion. The authentication of Lyft uses
MFA to enforce strong authentication.

2.8.3 BLESS log and track access

BLESS gives you the possibility to log access and track the authentication of users.
In BLESS CloudWatch is used to log all the authorisation requests [7]. Amazon
CloudWatch is a monitoring service for AWS [79]. It can for example monitor metrics
like CPU utilisation, data transfer and disk usage, when there is something not right
a warning is sent.

2.8.4 BLESS process model

The authentication in BLESS happens in several steps. In this section there is ex-
plained more how the process exactly happens. An overview of the process is found
in Figure 2.10.

S.1 In BLESS a user authenticates himself through the AWS tool. This happens
with session credentials. The Bless algorithm uses the session credentials on
the Certificate Authority to request a certificate.

S.2 The second step is that the Bless algorithm the private key decrypts. This
happens by using the Amazon KMS. A KMS is a Key Management System.

S.3 In the third step the Certificate Authority generates and signs the SSH Certifi-
cate. This certificate authority is also responsible for the certificate manage-
ment.

S.4 The request and the certificate are logged in a logging system called Cloud-
Watch.

S.5 The BLESS certificate authority returns a short-lived certificate to the bastion
from which the certificate is requested.

S.6 With the certificate and the SSH tool access is obtained to the instances on
which the work is performed.

S.7 The instance validates the certificate and stores the certificate
S.8 The authentication results of validating are sent back to a logging system.

Figure 2.10: BLESS flow diagram[7]
Chapter 3

Research Methods and methodologies

This project uses qualitative research methods. One of the qualitative research methods used is the requirement analysis. The requirement analysis is explained in Section 3.1. The requirements are obtained from the state of the art and the analysis. The project uses Unit testing and some testing with the command line, to see whether the system behaves as it should. Command line testing is performed, with the command line testing there can be observed if the distributed system works. The Unit tests are not yet build to be tested in a distributed way. Up to the knowledge also no other testing approaches except for the command line are known for a distributed Apache Spark environment. The distributed test is performed to check if the new approach works distributed. There are descriptive and conceptual research methods used, there is investigated and described how Kerberos and Netflix Bless work. Further, the characteristics of Spark and Flink are explained, together with the Hadoop security environment. There are analytical methods used, like critical thinking and decision making, to come up with a new security solution. During such a project errors can occur on the machine. For this reason system changes are logged, like adding paths to the bashrc, or installing new software like Spark, or editing configuration files. Further, there is a repository for all the script files that are created and a repository specifically for all the Spark configuration. The logging and the repositories make the thesis and the implementation reproducible.

For the methodologies a literature study, development, evaluation, report and presentation is performed.

A literature study is performed, the KTH database Primo is used and Google scholar. To search the database keywords are used, these are amongst others "SSL", "TLS", "Kerberos", "Spark", "Flink", "Spark security" and "Flink Security". Further, information is obtained by asking around in Fraunhofer and gaining information from the Supervisor. Moreover, information is obtained from the Professor of KTH. Further, Google is used to find information. To find specific information about programming Stackoverflow is used. To organise the relevant information for the thesis, the data is stored in separate folders and websites are stored with bookmarks. To find relevant information the abstract and conclusion of papers are scanned. The websites are quickly scanned and conclusions and abstracts are read. Before writing the code there is looked on the internet if solutions already exist.

For the development and evaluation, a look is taken in the source code of Apache Spark and Flink. In the development phase is extensive contact with the supervisor, to find a solution. During the literature study there was realised that Kerberos has some problems, so a look is taken into alternatives. One of the possible alternatives is to use certificates, the certificates are already used in Spark and Flink to protect the data transportation. The idea is gathered to use the concept of bastion lambda ephemeral SSH service (BLESS), this has advantages as logging mechanisms and strong authentication. The requirement analysis is used, this methodology is in Section 3.1. In the end unit testing and normal software testing is performed. The
unit testing happens mainly through assertion testing, this is a testing method, where the expected result gets compared to the actual result [80]. A boolean result will then be returned, with true when it passed. Assertion testing is used to detect errors in the code. The disadvantage is that the expected outcome on beforehand has to be specified, and it might be that the developer oversees some bugs. After the assertion testing and additional testing with a command line to find the bugs in the distributed environment, the results are extensively discussed with the supervisor.

The report is written in accordance with the guidelines of KTH. The finished draft report is handed in and a presentation is performed. Then another week follows to improve the report based on the presentation.

3.1 Requirement Analysis

Requirement analysis consists of tasks that determine the needs or conditions for a new or altered product[81]. Requirements analysis is critical to the success or a failure of a software project [82]. Requirement analysis is used to discover the bounds of the software and how it must interact with its environment. In requirement analysis the best requirements are actionable, testable and traceable. To execute a good requirement analysis first the problem is identified, this is currently done in the Section 4 analyses. Then the preliminary requirements are discussed, then different approaches are discussed with their important elements. The different approaches are discussed by using, different use cases [83]. This technique is powerful to attempt a solution based on the preliminary requirements, this is further explained in Section 3.2. The preliminary requirements are the requirements that are coming from the goal that the system tries to achieve. Moreover, there are more requirements to make every use case possible. After requirement analysis comes requirement validation. Requirement validation is performed to ensure that the requirements defined are defined for the right system, a system the user expects. A validation can be done by building a prototype. [82].

3.2 Unified modelling language

The Unified modelling language (UML) is a general purpose visual modelling language [83]. The UML has gained wide attention in the software industry. The language is powerful for describing the software. Besides that the language is powerful for describing software, it is powerful for testing as well [84]. In UML use cases can be used, this is a popular technique. Use cases can describe in various detail a scenario. A use case can be described with an interaction view or a use case diagram [83]. An interaction view is to display interactions. An example of an interaction view is a sequence diagram. In an interaction view there is a sequence of messages among different roles. Roles are also known as actors, an example is a client and a server. An actor is a human, computer or process. A sequence diagram is a diagram where the messages are arranged in time sequence [83]. Each actor has its own lifeline, this is a vertical line that represents the actor during the diagram time. There are loops and if and else statements in a sequence diagram. This happens with the use of guards and alternatives. Alternatives create a choice between two options called guards for example, an if and an else. If a guard is true in an alternative that part is executed. The sequence diagram is important to show the behaviour sequence in a use case [83]. A use case diagram can be used as well, but this describes only how the specific use cases interact with the actors. For the solution in this thesis the sequence and interaction are more important, since it describes what is happening at which moment. In the use case analysis the use case is described and then a sequence diagram is provided.
Chapter 4

Analyses

In this chapter the State of the Art is analysed. Here the analysis leads to the final approach in Section 5. Section 4.1 analyses the challenge of running jobs for a time longer than seven days, the maximum time of a Kerberos ticket. An analysis about the sharing of sensitive data is in Section 4.2. The insights gained from the authentication mechanism bastion lambda ephemeral SSH service (BLESS) is in Section 4.3. The similarities and differences between Flink and Spark are then analysed in Section 4.4. Transport layer security (TLS) and Kerberos are then discussed to highlight new possibilities and some advantages and disadvantages of both mechanisms in Section 4.5. The most important parts of on demand certificates are analysed in Section 4.6. The Section 4.7 examines the management of user access in the cloud. The analysis in this chapter is summarised in Section 4.8.

4.1 Long-running streaming job analysis

The goal of a long-running streaming job is that once it is submitted to run it is supposed to run forever until intentionally stopped. Streaming jobs are understood to run for long periods of time (many months) in production environments. If a long running job crashes, it can potentially cause data loss or duplicate data. A long-running streaming job usually runs on a yet another resource negotiator (YARN) cluster. Neither YARN, Spark, Flink, or Kerberos have been designed for running jobs for a long time. Spark and Flink now use Kerberos delegation tokens, explained in Section 2.2. A Kerberos delegation token is valid for a maximum of seven days [22]. For jobs running longer than seven days, requires the tracking of delegation tokens. Otherwise, the user can’t authenticate himself again to the job if the token is expired. Currently, this requires an own developed update mechanism for tracking the token and renewing it. This mechanism should take into account that there is authenticated multiple times throughout the life of the job, meaning there are multiple tokens for the same job since some jobs can run for months. To keep track of the tokens requires a database.

4.2 Resource management with sensitive data analysis

When using sensitive data, it is important to do resource management. Not all users should be able to access all the files and resources across the cluster. For example, a user might be allowed to only ask for a job status, while the other might only execute a job. It should be possible to restrict the user its access rights by using privileged access management (PA management). In Apache Spark and Flink resource management for a particular user is not yet thought of sufficiently. For example, it might be useful in a company that one user is allowed to run with more datasets than the other. By implementing PA management, this can be made possible. The first step can be to make it possible for administrators to manage the users and the resources in Flink and
Spark. In a database, the user and the information of which folders the user is allowed to access can be stored. Currently, in Hadoop open Platform-as-a-Service (Hops) it is possible to share datasets between users, this is observed in Section 2.1.6.2. Hops maintains metadata, to know which user has access. The sharing of sensitive datasets is not the main goal of this project, but is an important improvement made by Hops.

4.3 Analysis bastion lambda ephemeral SSH service (BLESS)

From the BLESS mechanism inspiration is obtained for a new implementation. The authentication happens with MFA and makes sure that only users with the right permissions can access a certain SSH. MFA has more authentication factors, making it harder for intruders to gain access to the system. The BLESS user does not have to authenticate for every separate service because a certificate is used to authenticate at multiple services. Authenticating for every service would demand much time for an authentication mechanism with multiple authentication factors.

To authenticate BLESS uses for multi-factor authentication (MFA) a user database, that contains the password and the Pluggable Authentication Modules (PAM). The PAM allows authentication from an authenticator device like a Yubikey, or a time-based one time password (TOTP). Unlike BLESS in Kerberos the user can first authenticate himself with a PAM, then authenticate with Kerberos and obtain a delegation token[85], two databases (Kerberos and a database to store the second authentication factor) are required to be maintained. In Kerberos is thus the weakness that only the password is used to obtain the token. A permission system with only an database for MFA is beneficial over using Kerberos alone. To prevent such a database for causing lags, or becoming a single point of failure, the database can be build distributed. The consequences of the lag can be that the user cannot be authenticated for a longer time, and can thus not access certain services. This can make data unreachable, making it for the user not possible to continue work.

The process flow of Netflix BLESS is explained in Section 2.8.4. The process makes use of certificates. The certificate given is only for the parts the user has permissions, ensuring that the user cannot reach sensitive instances. In Apache Spark and Flink something similar can be done. The access rights for each job could be specified, meaning that the some developers can observe a job and its output and others not. The certificates in BLESS contain information about the user and access time and where the request was coming from. This information can be logged and with that it is possible to track malicious insiders. The logging of the authentication attempts in BLESS happens through CloudWatch, which is a commercial application of Amazon to monitor and track log files. CloudWatch is important to track the mistakes that can happen by developers, and for logging the access to services. Logging the access helps to later identify unauthorised access. With this Netflix solution, Amazon and its CloudWatch maintains sensitive information that is inside the logs. Trust is important when using a third party as Amazon since the third party can form a security issue, in this case Netflix trusts Amazon in maintaining its users information.

BLESS builds an own Python mechanism to authenticate the user automatically. This mechanism is for SSH authentication, for the authentication for Flink or Spark this mechanism can be adapted. Flink and Spark both already use certificates to secure data transportation. The certificate can be relatively easy updated into a new certificate for authentication, this is also the case in Netflix BLESS. The main things a certificate should contain is the validation date, a user, and it provides information about the usage of the certificate. The certificates with a short validation time based on the validation end and start time are further analysed in Section 4.6.
4.4 Flink and Spark similarities and differences

This section analyses the similarities and differences between Flink and Spark. The analysis of similarities is in Section 4.4.1. The differences are analysed in Section 4.4.2.

4.4.1 Flink and Spark similarities

Flink and Spark have similarities. Flink and Spark both do in-memory computing. They both provide an exactly once-guarantee for processing, each record is only processed once. Both use the Java virtual machine (JVM). They can both do Graph processing, structured query language (SQL) queries, and they run both in standalone and cluster mode. Flink and Spark are part of the Hadoop environment and can make use of YARN. Flink and Spark both support the same programming languages, Python, Scala and Java. Flink and Spark schedule the execution of jobs with the use of a directed acyclic graph (DAG).

4.4.2 Flink and Spark differences

One of the biggest differences between Spark and Flink is the way streaming works in both data processing engines. Spark uses mini-batches and Flink uses real-time streaming. Streaming can use the data ad-hoc, so immediately as it arrives. Spark tries to stream the data by using mini-batches. These batches are so small that it for the user looks like a stream, the mini-batches have previously been highlighted in Section 2.1.4.2. The mini-batches have a small time gap, because it creates a small batch of data, instead of feeding it at once to the processor. The real-time streaming of Flink is previously introduced in Section 2.1.5.1. Flink makes it possible to achieve lower latency compared to Spark, since here data tuples, key-value pairs, are processed directly. The mini-batches of Spark have other advantages, like the API for batch processing, this makes it possible to combine the different Spark libraries, such as for example the machine learning library and streaming. The API makes it easier to translate a batch job into a streaming job. Spark also allows for interactive queries, which Flink doesn’t [86]. Flink is less popular than Spark. In the Spark community there is more information about how run jobs in Spark and more information for Spark developers. The Spark security is also more extended compared to Flink. Both Flink and Spark have some differences and similarities. Each having their own unique advantages and some disadvantages, currently, Spark is a little more advanced.

4.5 Data transport security and Kerbereros comparison

In this section Kerberos and the data transport security, which is provided by TLS, are compared. Kerberos and TLS are difficult to compare because of its different intended use. They are compared in this section to try to highlight that TLS can be used as authentication. The goal of this section is to explain similar mechanisms that can achieve authentication. Further important differences are discussed, to highlight the problems both mechanisms have. This section concludes with an overview of Kerberos and TLS. This analysis concludes with the possibility of using TLS for both authentication and data encryption.

Kerberos and TLS have similarities and differences. The first major difference is that Kerberos uses a symmetric encryption and TLS uses public key encryption together with symmetric encryption. In symmetric encryption the same secret is used for encryption and decryption. Public encryption safely exchanges keys to
secure the TLS connection. TLS uses a a public key and a private key to setup a safe connection that protects the data being transported. Kerberos is mainly used for authenticating and providing a central key management. TLS is used for the data transport protection. Kerberos is a trusted third party service, the clients and services are connected to Kerberos, they trust Kerberos in identifying, and allowing other clients and services into the system accurately. In TLS the certificate authority is a trusted third party. The user can check the certificate with the third party certificate to see if the certificate is valid. Once Kerberos is compromised the third party Kerberos and its authentication cannot be trusted anymore. The same is the case for once the Certificate authority is compromised. In Kerberos the token and the certificate are time limited. The token in Kerberos default valid time is 10 hours and many users tend to not change this setting to a shorter time [87]. The certificate can in theory be limited to only minutes as explained previously in Section 2.6 about on demand certificates. In TLS the certificate might not be valid if the machine time differs much between the Certificate Authority and the services requiring a certificate. Kerberos raises a clock error if the time difference of the machine using Kerberos and the service is substantial. This requires that the services using Kerberos should have almost the same time as the server running Kerberos. In TLS the services using the certificate authority should also have almost the same time. When a token should be not valid anymore, it can be revoked in Kerberos by the KDC. The certificate can be revoked in TLS by the Certificate Authority. Both Kerberos and TLS work on the trust that the user and service believe that they are correctly verified.

A disadvantage of both TLS and Kerberos is that MFA is troublesome to setup. This is because the PAM module is not supported with Kerberos and TLS. In Kerberos PAM can be supported by adding a Radius server. In TLS this can be solved by using a database that can support PAM mechanisms, this can be a MySQL database. Kerberos has a single point of failure, the KDC. When using a distributed MySQL database this can be resolved. Replicating the KDC also solves the problem, although this is more challenging compared to replicating a database. The Kerberos system has legacy risks from its legacy, to minimise these risks the settings of Kerberos have to be set correctly, this requires experience [48], [51], [88]. TLS does not have many known legacy risks. TLS has significant overhead costs when encrypting all the data as previously shown in Section 2.3.4. Using either Kerberos or a TLS mechanism offers an advantage that the authentication workflow is decoupled from individual services. Kerberos makes sure that the users are verified. TLS makes sure that the data transferred over a connection isn’t changed. Kerberos provides easy authentication with a single sign on, a centralised service to do the user management and auditing to see who logs in. TLS makes sure that the data isn’t forged that is being sent. It establishes a chain of trust which makes sure that the establishment of the connection can be trusted.

Spark and Flink are offering data protection for the transport of data with TLS. This is important to turn on when transferring sensitive data because this achieves a secure connection. This secure connection requires certificates. These certificates make it possible to achieve non-repudiation, and thus integrity. The certificates are also part of a chain of trust. The chain of trust makes sure that it can be verified where the certificate came from. This is the inspiration for using it as an authentication mechanism. This can be achieved by using a database, which can authenticate the users and support PAM. The certificate can be requested new for every authentication to build the chain of trust. As seen in the previous analysis of BLESS in Section 4.3 the certificates can be easily gained for a new authentication. This certificate can then be used to set up the TLS connection and complete the authentication. The only extra overhead costs that would then occur is the request for the certificate since
the connection is already there for the safe transport of data. The certificate can be used as a single sign on as Kerberos uses its delegation token for the single sign on. The certificates can achieve a single sign on as in BLESS in Section 4.3. The user here obtains access to multiple instances. In Spark and Flink it is possible to start the job with the certificate, this is further explained in Section 5.3. To conclude TLS can replace Kerberos by taking over its authentication. To achieve this a database is required. This would resolve some problems such as providing authentication using multiple authentication factors. In the next Section 4.6 more potential improvements of certificates are analysed.

Table 4.1: Overview TLS and Kerberos

<table>
<thead>
<tr>
<th>Kerberos</th>
<th>TLS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Symmetric encryption</td>
<td>Public key encryption</td>
</tr>
<tr>
<td>Kerberos is trusted third party service</td>
<td>Certificate authority is trusted third party service</td>
</tr>
<tr>
<td>Token time limited</td>
<td>Certificate time limited</td>
</tr>
<tr>
<td>Key revocation at KDC</td>
<td>Key revocation at Certificate Authorities</td>
</tr>
<tr>
<td>Delegation token</td>
<td>Secret</td>
</tr>
<tr>
<td>Centralised key management</td>
<td>Used for data transport protection</td>
</tr>
</tbody>
</table>

Disadvantages

- Does not support PAM
- Legacy risks
- No fine-grained authentication
- Requires replication of KDC to work at scale
- No easy management of Certificate Authorities

Advantages

- Preventing access to wrong users
- Single sign on
- Simple user management and auditing
- Chain of trust

4.6 Certificate on demand analysis

The certificate on demand technique is becoming popular. These certificates have a short time that they are valid. The analysis of Kerberos and TLS in Section 4.5 and the analysis about BLESS in Section 4.3 interpret that certificates are promising for authentication. The companies currently working with Hadoop applications like Spark and Flink are demanding for new authentication mechanisms [54]. They would like to have the possibility to authenticate with certificates, and use third party solutions like OAuth2.0 over the internet [89][54]. Here they are interested in a solution that supports fine-grained authorisation in a multi tenant environment that authenticates users based on the and the resources they are allowed to use [54]. OAuth2.0 (RFC 6749) is an authorisation framework to give a user limited access to an HTTP service [90]. With this framework the user can grant an application or service access to its information, without giving them authentication details such as a password. The authentication happens by sharing an access token. Access over HTTP and support for OAuth2.0 cannot be achieved with Kerberos, because it requires breaking the existing authentication mechanisms in place [54]. When Kerberos is kept there would be two tokens, an OAuth2.0 and a Kerberos token.
both for the same authentication, this makes the authentication complex and should thus be averted. In a database such as MySQL this authentication can be simplified, since the OAuth2.0 token can be easily linked to the credentials in the database. On demand certificates allow authentication over a short time period. This is an improvement of trust over Kerberos, because Kerberos provides authentication over a longer time period, in general 10 hours, during the time period possible attacks could be performed to misuse the token. This can be traced by letting each service have its own audit log, to see which user performs which actions on the specific service. The trust is improved with TLS because the time period of a certificate can be a couple of minutes. Both Kerberos and TLS use NTP to keep the time synchronised between the different systems. Without time synchronisation it might be that a certificate or token is invalid on one machine and valid on the other, this gives the problem of an inconsistent authentication across the network. Especially with on demand certificates time synchronisation across machines is important to keep the valid time of certificates limited, this is achieved by NTP.

With the use of on demand certificates the login attempts can be managed. When using Kerberos the user uses the principal and keytab to log in. With on demand certificates the management of certificates becomes easier, since only the active certificates have to be managed. Since these certificates are only active for a short time there are fewer certificates to be managed. When the valid period of the certificates expires they have to be logged, to keep auditing possible. Auditing is important in the case something went wrong. In this section the use of on demand certificates for the authentication of Spark and Flink is explained. Opportunities are seen to modernise the authentication. It can allow for stronger authentication, better logging, and a shorter opportunity of attack.

### 4.7 Cloud Access Management analysis

An Identity and authentication management (IAM) system is used in BLESS as described in Section 2.8. The IAM system is provided by Amazon. In the current implementation of Spark and Flink Kerberos is used to provide IAM. Kerberos is a cost-free software. The new solution should utilise only cost-free software. In Section 4.2 there is analysed that PA management is important when working with sensitive data on a shared infrastructure, such as in the cloud. MFA is becoming more used, since the authentication with only a single password is more vulnerable to get compromised. MFA is poorly implemented in Kerberos, due to the setup that is required to use another system for the other authentication factors than the username and password. A new database can make the authentication happen without requiring to store other authentication factors in both a Kerberos and a normal database. Recently, IAM, PA management and MFA combined coined a new term **cloud access management (CAM)** [91]. CAM becomes more important for the authentication and privilege management in cloud systems [91].

### 4.8 Summary

This section summarises the chapter. In Section 5 the new contribution is based on the points mentioned in the summary. In this section is argued for which framework is chosen to start the implementation with, and which authentication techniques are used. Then the management of the certificates is analysed. At the end of the summary the most important points are summed up for the new solution.

The new framework can be implemented in either Spark or Flink. Spark has more popularity than Flink. Spark has more information available about how to develop for
the platform and test the code. The general security in Spark is also more developed and documented compared to Flink. For these reasons above, there is chosen to implement the new contribution in Spark.

Section 2.4 discusses the authentication mechanisms used for MFA. The best authentication mechanism is U2F together with a user password. However, using the U2F factor is out of reach for this project. A U2F device, for example a Yubikey cost around 45 Euro. For a testing environment such a price is not reachable. The TOTP solution which is a little less secure but costs no money is considered as a better option. TOTP is considered to be more secure as HOTP or as a push-based authentication mechanism.

Using certificates modernises the authentication infrastructure. To manage the access to Apache Spark or Flink a database can be used. For the management a database can be used for management together with certificates. To have cloud access management the privileges of a user can be set in the database in order to achieve the multi-tenancy as analysed in Section 4.3. In the database can be set which authentication mechanisms are used, for example TOTP. For the usage of a distributed database it has to be sure that the database is consistent and available, from the information in Section 2.5 there can be concluded that the best choice for this is a MySQL database. This database offers consistency and availability, further it is widely supported and offers good security. With the use of multiple authentication mechanisms, Hadoop can get up to date with the latest demands from the business. In this way Hadoop can join the trend of certificates on demand analysed in Section 4.6. The data transport security in Section 2.7.3 is a possible mechanism which can implement the on demand certificates. Here there is a keystore for the certificates and a truststore for the root certificates of the certificate authority. With this mechanism the data protection for transportation gets established. This can also be done with the certificates created for authentication. The keystore can then verify the connection with this certificate. This verification needs to happen to establish the secure data connection over the sockets. If this verification fails, the connection cannot be established. The establishing of the data transport can be used to see if the user is authenticated. There are several key points taken from this summary into a new approach.

1. Limit the valid time to achieve non-repudiation and thus ensure that the short valid certificate is not forged.
2. Support multiple factor authentication, to make it more difficult for an unauthorized person to access.
3. New cloud access management, with privilege and authentication management scalable to thousands of users.
4. Realise a distributed authentication that can authenticate thousands of users.
5. A database and certificates is an improvement for the auditing and scalability.
6. TLS can both be exploited for authentication and security.
7. Non-repudiation is gained by using certificates
8. Single sign-on to authenticate at multiple services

There is a distributed authentication required with multiple authentication factors that supports single sign-on. These above mentioned key points are new authentication demands. These points help to protect the data analytic tools such as Spark against modern attacks. The concept of BLESS is used to achieve better auditing. This can be achieved with certificates that also ensure that the data sent over the network is safe, the on demand certificate provide non-repudiation, because this certificate is a hashed certificate with a short valid time. Building a new approach with these key points will likely improve the authentication of Spark.
Chapter 5

Contribution

In the Chapter “Contribution”, the new concept is explained. First, the requirements are defined in Section 5.1. After defining the requirements, there is analysed in Section 5.2 if Kerberos fulfils the requirements. A new authentication process design is described in Section 5.3. The authentication process design consists of a new process design and a certificate design. The certificate design shows the information stored in the certificate. Section 5.4 explains the configuration. The configuration is required to make the authentication process work. During the design process is assumed that the current authentication does not break. The new contribution leaves the user with the choice to choose Kerberos, the new authentication or no authentication. When using Spark inside a company, it is advisable to use authentication. A caveat is that to oblige authentication requires significant code changes breaking the old code. It is not clear if the community wants to break the old code to implement this feature, therefore this new feature is not implemented. Section 5.5 highlights the implementation and the implemented features. In the implementation are new components for the new solution and programming scripts.

5.1 Requirements

The authentication system of Apache Spark needs an upgrade. The information in Chapter 4 “Analyses” gives the foundation for the requirements. In this section the requirements are presented. Each requirement is identified by R and then a number, for example, R1, R2, ... etcetera.

R1 One of the upgrades needed is multiple factor authentication (MFA). Make it more difficult for an unauthorized person to access by using multiple authentication factors. For MFA at least two of the following three authentication possibilities are required [92].

(a) Something you know (knowledge). For example a password.

(b) Something you have (possession). For example a phone.

(c) Something you are (inheritance). For example a voice or a fingerprint.

Providing biometric data such as a fingerprint is still considered as unsafe [92]. Biometric data is irreplaceable, and it can get stolen from a compromised database [93]. For this reason, the first two authentication options (1,2) are the best. Most people use passwords. However, more than 80% of the hacks and data breaches happen because of static passwords, and thus users require more authentication factors [93]. To support MFA requires a pluggable authentication module (PAM). As noted in Section 4.3 PAM is used by many authentication mechanisms.

R2 Section 4.5 analysed the importance of Public key encryption. Public key encryption is achieved with a public key infrastructure to generate a public certificate. Using such an infrastructure achieves privacy, integrity and trust. Public key encryption prevents eavesdropping, by safely exchanging the keys to
encrypt the messages. The certificate authority in public key encryption establishes the trust in authentication and creates the certificate for data protection.

**R3** An infrastructure for authentication should be scalable. The requirement is therefore scalability, this is extended with consistency and availability, which is requirement [R11].

**R4** Auditing, a logging system to log everything about login and authentication attempts. Logging ensures that every authentication can be traced and possible errors can be easier resolved. The auditing makes sure that if a possible hacker enters the system, there can be traced what information has been accessed.

**R5** Short valid time. The short-lived certificates can be achieved by using certificates on demand, meaning that the time to live (TTL) of the certificate is set to short time, hence limiting the possibility for an attacker to misuse the user its certificate. The certificates should be logged everytime it is requested and this short valid time improves the auditing. These enhancements limits the possibility of a golden ticket attack as in Kerberos. The short time gives less time to change and abuse the certificate.

**R6** The separation of authentication and instances. In the concept are instances for authenticating the user, such as a database and the certificate authority. Further, there are instances to perform the job execution and store the information, such as the Spark master, its workers and a Hadoop distributed file system. The instances that provide the authentication and that perform the job execution should be separated. The separation makes that the authentication still functions if an instance executing a job or storing data shuts down. The system becomes better recoverable from failures, because there is a clear separation between authentication and performance instances.

**R7** The new solution should fulfil the three mechanisms in Cloud access management (CAM). For authentication, the solution should have an identity and authentication management (IAM), a privilege access management and MFA.

**R8** central access to the IAM, IAM achieves that permissions of users can be revoked quickly in case of an emergency.

**R9** Fine-grained authorisation, the new database should provide the possibility to have different access permissions to Spark for each user. Here there can be thought of various authorisation rights.

(a) A user only allowed to ask the status and results of specific projects in Spark.

(b) A user allowed to run its projects in Spark

Giving users different permission rights minimises the chances of being hacked.

**R10** Resource permission for users. In the design, it should be possible that one user can use more resources than the other. The resources are for example Memory and CPU.

**R11** Consistency and availability of the authentication service. The authentication should be consistent across the system, as a result users are correctly authenticated. The authentication should be available most of the time.

**R12** Encrypt the data at rest for a user. In an environment with multiple users sharing files, the users are only to access the data for which they have access rights. The files should be encrypted to prevent them being opened by a user with no access rights.

**R13** The data needs to be encrypted during transit so that sensitive data cannot be gained when sending data over the network.

**R14** Minimise authentication overhead. The goal of Spark and Flink is to be fast, and thus all overhead has to be minimised. The authentication needs to be done as efficiently as possible.
The **authentication is automated.** The administrator should have as less work as possible to authenticate the users. The login should be automated.

**R16 Non-repudiation** implies that the data integrity can be proven. The data transferred across the network thus has not been modified.

**R17 Resource multi-tenancy,** meaning that the resources in the cluster are shared by multiple users. Each user gets its share of the resources.

**R18 role based access control and sharing projects** as in Hops earlier described in Section 2.1.6.2.

**R19 Single sign-on,** in is Kerberos one of the main advantages that when the user is authenticated, this authentication can be used at multiple services as described in Section 2.2.2. The ability to have a single sign-on should be kept in the new design.

### 5.2 Requirement analysis Kerberos

Kerberos does not meet several requirements. These are R2, R3, R5, R7, R9, R10, R12, R13, R16 and R18. Kerberos cannot achieve fine-grained authorisation for Spark. An additional Hadoop environment component Apache Ranger can be used to achieve fine-grained authorisation. Apache Ranger identifies if the user is allowed to read a database. R11 requires that the data is consistent and available, the data inside Kerberos is consistent. However, consistent replication of a Kerberos database is hard. With many users, it might be that the database becomes unavailable. Evaluation of the requirement R14 minimised authentication overhead is in the “Evaluation” chapter Section 6.6. Kerberos together with YARN supports resource management and achieves requirement R17 resource multi-tenancy. The requirements met are R1, R4, R6, R8, R15 and R19. Kerberos does provide authentication using more than one factor by using for example RADIUS or LDAP to provide one factor as pre-authentication, then the password can be provided after the pre-authentication [85]. Kerberos does thus provide MFA. Nevertheless, this is complicated to achieve. Only the password as an authentication factor is used to obtain the ticket, this means that factual only one factor is used by Kerberos [85].

### 5.3 Authentication process design

The new authentication requires a new process design. Each process obtains and uses the certificates differently. The certificates have a uniform design which is in Section 5.3.1. Section 5.3.2 describes the authentication for receiving a status or starting a local job. The design explains how a user authenticates to gains access to the job results. Further, it gives an explanation for a local job with authentication. When running in a cluster or YARN, the design is slightly different. Section 5.3.3 explains the authentication process design of the Spark cluster. Section 5.3.4 describes the cluster design that supports YARN. The design for YARN and the Spark cluster requires certificates for data transportation. The certificates set up safe data transportation in the cluster over a secure connection.

#### 5.3.1 Certificate design

This section describes the design for certificates. The certificate is signed by a certificate authority with an intermediate key which is created by the root key. Together with the root and intermediate certificate the chain of trust can be build as explained in Section 2.3.2. With a certificate sign request, a request to sign a new certificate is created by using a Java keystore and its private key. The certificate is signed by
the certificate authority and contains a chain of trust to verify the certificate. The retrieved certificate is verified and added to the keystore. The type certificate being used is X509 version 3 [94]. An extra option in this version and type of certificate is the keyUsage. The keyUsage specifies the usage of the certificate it can be a digital signature or key encipherment. There is a SubjectKeyIdentifier which is a way of identifying the certificate in version 3 [94]. Certificates are encrypted they begin with "—–BEGIN CERTIFICATE—– encrypted certificate " and end with "—–END CERTIFICATE—– " and end with "—–END CERTIFICATE—– ". The root and the intermediate certificate have a stronger hash encryption. The stronger hash takes longer to generate but is more secure. The certificate has a "name.cert" and an alias name stored inside the certificate. The name is unique for each certificate. It is used to identify the user and the job. Tracking and storing the certificates provides auditing. The signed certificate can be verified by the root and intermediate certificate from the Certificate Authority (CA). The certificates are generated on demand and have a short valid time. The new certificate design is in Table 5.1.
### Table 5.1: Certificate design

<table>
<thead>
<tr>
<th>Label</th>
<th>Input</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alias name</td>
<td><code>username-UUID</code></td>
<td>the username with a universally unique identifier (UUID) as the name of the certificate. The UUID and username are used to identify the unique job.</td>
</tr>
<tr>
<td>Creation date</td>
<td><code>date</code></td>
<td>Shows the creation data of the certificate.</td>
</tr>
<tr>
<td>Entry Type</td>
<td><code>trustedCertEntry</code></td>
<td>The <code>privateKeyEntry</code> are the private key and certificate for this keystore signed by the certificate authority. The <code>trustedCertEntry</code> in the keystore is a public certificate signed by the certificate authority. The intermediate and the root certificate from the certificate authority are “TrustedCerts” in the keystore.</td>
</tr>
<tr>
<td>Certificate chain length</td>
<td><code>3</code></td>
<td>The length of the certificate chain, a certificate signed by an intermediate and thus the root has a length of three.</td>
</tr>
<tr>
<td>Owner</td>
<td><code>username-UUID</code></td>
<td>The owner is the one who owns the certificate, this can potentially be different to the one that issued it, and it can have the same information fields as the issuer field.</td>
</tr>
<tr>
<td>Issuer</td>
<td><code>CN=username-UUID, O=organisation, L=locality</code></td>
<td>The common name of the issuer has the same name as the certificate. The issuer can have more identification fields such as organisation and locality. In this design the Organisation is Fraunhofer and locality is Darmstadt.</td>
</tr>
<tr>
<td>Valid from</td>
<td><code>Wed May 17 09:37:06 UTC 2017</code> until: <code>Wed May 17 09:47:06 UTC 2017</code></td>
<td>Here the valid time of the certificate, in this implementation using on demand certificates the validity time is short.</td>
</tr>
<tr>
<td>Certificate fingerprints</td>
<td><code>SHA2:BE:32:3A:3C:76</code></td>
<td>The hash of the certificate. In this case, the cryptographic hash function is SHA2. The fingerprint is used to identify the longer public key inside the certificate.</td>
</tr>
<tr>
<td>KeyUsage</td>
<td><code>Digital Signature, Key encipherment</code></td>
<td>The digital signature is used for authenticating the user, together with the key encipherment to make safe data transportation possible.</td>
</tr>
<tr>
<td>Version</td>
<td><code>3</code></td>
<td>The x509 certificate version used</td>
</tr>
<tr>
<td>SubjectKey Identifier</td>
<td><code>KeyIdentifier [0000:DA0DE131F5E97FB2]</code></td>
<td>The subject key identifier extension provides identification of a certificate that contains a particular public key.</td>
</tr>
</tbody>
</table>
5.3.2 User authentication process design

In Spark, there are different mechanisms to run a job and obtain the status of a job. This job is being authenticated by using the bastion lambda ephemeral SSH service (BLESS) mechanism. The retrieval of the job status envisions the following steps:

1. Request certificate for obtaining the status
2. Client can obtain status, or receives an error
3. Client receives status report
4. Remove certificate

Each of the certificates is unique for every status request. There are several actors involved to get the job status. In this section, each actor is explained. The actors are the client, database, certificate authority, logger, Spark and Spark worker. The first actor is the **Client**. The client wants to safely run a Spark Job and later wants to see results. The second actor is the **database**. The database is there to perform the MFA. The database also checks if a user is authorised to run a certain job and for example check results of a specific Spark Job. The **CA** is there to create certificates and validate them. There is a **Logger** to log all the certificate actions. With the logging of certificates, there can be figured out who is responsible for a mistake. The user can be tracked based on its certificates. **Spark** is there to orchestrate the commands. It can be seen as a controller that communicates with the other parties involved. There is a Spark-defaults.conf file where the configuration for the actors is set. How to set the new configuration is explained in Section 5.4 in Table 5.2. For the process visualisation, a sequence diagram is drawn based on the BLESS flow diagram in Section 2.8.4. Every S.* refers to a step in the BLESS flow diagram. In sequence diagram Figure 5.1 this gives the following steps:

1. Receive a status.
2. Confirm that all the client certificates are removed.
3. check if the password and username given by the user are correct with the password and username in the database.
4. return the result of the password verification
5. Log the password attempt for auditing purposes.
6. When the password is incorrect an error is sent back to the client with that the password is incorrect.
7. When the password is correct the second factor for authentication is being checked. A second factor is in this design TOTP. This factor can be entered wrong due to being entered just after the valid time window. For this reason, there are three attempts. This together with step 3 corresponds to BLESS step S.1.
8. The attempt is verified with the database together with a script for translating the second factor, and the result is sent back. This corresponds to step S.2.
9. Each attempt is logged for auditing.
10. When after three attempts there is still an error, an error is sent back to the client saying that he can’t authenticate himself.
11. The certificate is requested. This is similar to S.3.
12. The certificate is logged and audited. The logging is comparable to the step S.4 in BLESS.
13. The certificate is received. The certificate receiving is similar to S.5.
14. The status is obtained. Either the status can be obtained either from Spark directly, or there is a short communication with YARN to get the status.
15. The status is returned to the client.

The status is obtained at the Spark master. To get the status is first a certificate is obtained via Spark. The certificate has to be verified before the status is returned.
and otherwise the access is denied like in BLESS. Then the status can be obtained. In the loop used in the sequence diagram Figure 5.1 a user has multiple attempts for the second authentication. A second factor is in this design TOTP. This factor can be entered wrong due to being entered just after the valid time window. The TOTP 6 digits change exactly at the moment the client sends them, requiring the authentication to be done again. For this reason, there is chosen to give the user three attempts. Information about how TOTP and other authentication mechanisms work is given earlier in Section 2.4. The MFA factors used in this design are, (1) a factor that you know, and (2) something that you have. Here there is used a password and on the device you have is a six digit generated. These two examples are the most easy to implement. In an organisation other authentication can be used. For example, OAuth2.0 or a push based authentication can be used to authenticate yourself.

For running a job locally, the process is the same as getting the status. After acquiring the certificate, the status retrieval is started. For obtaining the certificate, three components have to be implemented. The components are the certificate authority, the database, and the auditing system. These are discussed in the implementation. The design for obtaining of the job status and running a local job aims to fulfil several requirements. The requirements are earlier proposed in Section 5.1. For every requirement, there is explained whether or not it is fulfilled. The first requirement R1 MFA is accomplished. The user now enters his password and another secret. R2 is achieved, the certificates are a part of the public key encryption. R4 is fulfilled, the certificate and the password attempts are recorded. The Spark job is logged, better auditing is achieved by tracing the authentication process and the jobs. The short valid time of the certificate, needed to obtain the status and run the job, fulfils R5. R7 is mostly fulfilled, the database does the identity access management. The database can potentially do privilege access management, and by using the second factor for authentication, the MFA is fulfilled. The privilege access can be achieved by storing in the database the amount of resources that the user can use and to which results the user has access. The database fulfils R8 it gives a central access to the aim, and R9 is fulfilled because there is a fine-grained authorisation. The resources a user is allowed to use can be managed at the start of the job. This is further explained in Section 6.2.1. R10 is the requirement that each user can use a different amount of memory or CPU. This requirement should be handled by the implementation the management of resource usage is further explained for this design in Section 5.5.2. R11 is fulfilled by having a database and a CA that gives a high availability. The high availability is achieved by having the database and CA distributed across multiple servers. The consistency can be achieved by using a MySQL database which can achieve consistency and availability amongst multiple servers. There is a safe connection between the client and Spark which fulfils R13. The authentication overhead with this design is a little higher compared to Kerberos. For authenticating the user is a message exchange with the database. The additional communication cost is the certificate request with the CA. However, this new authentication is more secure compared to Kerberos, since the whole authentication is better audited and the MFA happens with a single database. The implementation thus minimises the extra authentication overhead, fulfilling the requirement R14. The authentication here is organised in such a way that the authentication can be automated, and thus R15 is fulfilled. By using the certificates non-repudiation is achieved, so R16 is fulfilled. The status is obtained at the master, and a job is running on a single machine, for this reason, R17 is not relevant for this design.

Some criteria are not fulfilled. R3 is not fulfilled, since running a local version of Spark does not provide any scalability. The authentication can make use of a
Figure 5.1: Sequence diagram status job
distributed database and multiple CAs. Section 5.5.2 specifies the implementation of the database, and Section 5.5.1 describes the implementation of the CA. R6 is about whether the authentication and Spark are separated. When this design is on one server, the instances are not separated. However, this criterion is fulfilled when the user builds the database and the certificate authority on another server than Spark. R12 is the encryption of data at rest is not satisfied by this design. The problem here is that the user itself defines where the data is stored, making it difficult to know by Spark which data belongs to the user. R18 is not yet fulfilled because the role based access control has to be achieved by a database which knows where the data is stored. The access to the data can then be permitted based on the role. Currently, when using Hops such a requirement would be achieved. R19 the single sign-on works for the Spark worker and Master. However, it is not yet working in the complete Hadoop environment. For example, when using the Hadoop distributed file system (HDFS) access should be gained via Kerberos. In the future work Section 8.2 the idea is to remove Kerberos and replace it with certificates. This new design with certificates ensures MFA, extended auditing and scalability.

5.3.3 User authentication process design for Spark cluster

In Spark is a standalone mode to run a Spark cluster. The standalone mode has a basic cluster manager, the standalone scheduler, that manages the work and sends the work to the workers in a first in first out (FIFO) mechanism. The cluster manager is inside the Spark master, and the rest of the cluster consists of workers. A requirement of the new approach is that there is proper auditing. During the authentication of the user with the database, information is logged about the authentication. The logging is essential to catch potential intruders that try for example brute force attacks. Actions can be taken on an attack with a clear pattern, such as an attempt to attack a single user, this provides auditing. Moreover, logging helps to debug a software error.

For the design, there are the following steps to authenticate a user and request the certificates.

1. Check if all the certificates are deleted from the keystore, if not, delete them.
2. Authenticate the user and request the certificate at every worker.
3. Verify the certificate for a specific user on all workers.
4. Establish a connection between the workers.
5. If the connection is setup successfully the job is started. If the connection is not successful, a warning is raised.
6. Start the job for this user on all workers.
7. At the end of the job remove the certificates from the keystore.

The first step checks if all the certificates belonging to other users are deleted. If there is still a certificate in the keystore, it might be that the connection gets established with a wrong certificate. This certificate might be chosen because the technical details of the keystore specify that the transport layer security (TLS) algorithm automatically picks the first valid certificate in the list. The certificate provides improvements as described in Section 4.8. The data encryption is earlier described in Section 2.7.3. The user is authenticated at a database and multiple authentication factors are used. The authentication with two authentication factors is shown in the sequence diagram Figure 5.2. If the authentication fails the user sees an error message. After the successful authentication, Spark is allowed to obtain certificates. To get a certificate there are three steps, there is a certificate requested, the certificate is obtained at the certificate authority, and the certificate is verified. The certificate is verified in the keystore, and the connection can be set up. An unsuccessful verification, or an invalid certificate in the keystore fails the setup of the TLS connection for this particular job.
The certificate might not be obtained at all the workers. It is not possible to kill, or not use a worker for a single job in standalone mode. Thus this worker would be killed for all the jobs running in the cluster. In the worker, the jobs run isolated from each other. Therefore, there is chosen to raise a warning and tell the user that the connection is insecure. After the warning the there is decided not to start the process, because the goal of this research is to have a safe security for Spark. To give a clear insight into what steps are performed, an overview is in sequence diagram Figure 5.2. In this diagram there is an extra actor compared to the Diagram 5.1 in Section 5.3.2. The Spark worker's are there to execute the Spark job in a cluster. There can be multiple workers working on a single job. The workers report to the Spark master when they finish. The worker needs certificates to make a secure connection. The sequence diagram is based on the BLESS flow diagram in Section 2.8.4. Every S.* refers to a step in the flow diagram.

1. Start the job.
2. 2a, 2b remove the old client certificates in every worker. The certificates are removed from the keystore. The working of the keystore is explained in Section 2.7.3.
3. check if the password and username given by the user are correct with the database.
4. Return the result of the password verification.
5. Log the password attempt for auditing purposes.
6. When the password is incorrect, an error is sent back to the client with that the password is incorrect.
7. When the password is correct the second factor for authentication is being checked. A second factor is in this design TOTP. This factor can be entered wrong due to being entered just after the valid time window. For this reason, there are three attempts. This together with step 3 corresponds to BLESS step S.1.
8. The attempt is verified with the database together with a script for translating the second factor, and the result is sent back. This is comparable with S.2.
9. Each attempt is logged for auditing.
10. 10a, 10b if after three attempts there is still an error, an error is sent back to the client saying that he can’t authenticate himself.
11. Request the certificate at every worker. This is similar to the step S.3.
12. At the certificate authority, the certificate is logged. This happens for each worker and corresponds to S.4 in BLESS.
13. The certificate is sent back to the worker, and it is inserted in the keystore. The certificate is for a specific user. This is step S.5 and S.7.
14. 14a,14b The workers and the Spark master establish a connection over TLS. This is comparable with step S.6.
15. The result of the connection setup is returned. This is comparable with the S.8.
16. The connection setup returned false and an error is returned.
17. The job is started across the workers.
18. 18a,18b If it is not a long running job, the result is returned to the user.
19. At the end of the job remove the certificates from the keystore.
20. The result is returned to the client.

In Spark Standalone there are several requirements fulfilled and not fulfilled. In Section 5.3.2, there is already an extensive discussion on the requirements. Some of these requirements are in the same way fulfilled or not fulfilled as in that section. The requirements fulfilled are the following: R1, R2, R4, R5, R8, R9, R11, R15, and R16. Some requirements are not fulfilled by the design. They can be fulfilled when having the right implementation. These are R6, R10, R12 and R18.
Figure 5.2: Sequence diagram standalone cluster mode job
R3 is the requirement of that the infrastructure should be scalable. In the earlier design in Section 5.3.2, the authentication could scale, now by using a cluster the resources for running a job can scale as well. However, when having a large cluster, the authentication can still be improved. A cluster manager could then be used that can send the certificate over the network instead of requesting one for each worker. This can be achieved with a YARN cluster, shown in the next Section 5.3.4. R7 is as for the same reasons as in Section 5.3.2 mostly fulfilled. The certificates make a safe connection between the workers and the master, making it possible to transfer data safely achieving R13. Each worker has to obtain a certificate which increases the authentication overhead, because every worker has to request this certificate. This makes that R14 is not fulfilled. R17 is the resource multi-tenancy, meaning that the resources are shared. In the Standalone mode, the resource scheduling happens by using a first-in-first-out scheduler. The sharing of resources is not optimised, since it does not look into the CPU or memory scheduling among tasks. However, when information is required from the HDFS Kerberos should be used to authenticate within the HDFS.

### 5.3.4 User authentication process design for Spark with YARN cluster

In the YARN cluster mode, is a YARN cluster manager which distributes resources. The YARN scheduler has unlike the has an even workload distribution based on the memory load of a worker. Moreover, YARN has the possibility to distribute files before a job is started. At the beginning of a Spark job with YARN, the resources are sent to the resource manager. This resource manager manages the distribution of files and resources. A file can be a key store containing the certificates. The certificate is used for the authentication and setting up the secured connection. The resource manager can distribute the key store containing the certificate added. The possibility to distribute the key store improves the scalability and minimises the overhead compared to the standalone mode described in Section 5.3.3. In general, when using a large cluster, it is not advised to run Spark standalone. This is because of the overhead caused by requesting many certificates. In YARN cluster mode the following steps are performed.

1. Check if all the certificates are deleted, if not, delete them.
2. Authenticate the user and request the certificate for the key store.
3. Verify the certificate for the user.
4. Distribute the new key store over all the workers with the resource manager.
5. Establish the TLS connection with the workers.
6. Start the job for this user on all workers.
7. At the end of the job remove the certificates from the key store.

When the certificates are correctly removed, the resource manager can request a new certificate from the certificate authority. In this step also the user needs to be verified. The new certificate can then be inserted in the key store. The key store is then distributed to all workers, and the TLS connection with the workers is initialized. An overview of the process is given in the sequence diagram drawn in Figure 5.3. The sequence diagram is very similar to the Figure 5.2 in the Spark cluster design. The main difference is that YARN is now responsible for verifying the authentication. In this design only the actor **YARN**. YARN distributes the certificate in a key store to every worker before the TLS connection is established.
The YARN sequence diagram is presented in Figure 5.3 and consists of the steps below. The S.* to a step in the Netflix Bless flow diagram in Section 2.8.4.

1. Start the job.
2. 2a, 2b Remove the client certificates in every worker. The certificates are removed from the keystore. The working of the keystore is explained in Section 2.7.3.
3. Check if the password and username given by the user are correct with the database.
4. Return the result of the password verification.
5. Log the password attempt for auditing purposes.
6. When the password is incorrect an error is sent back to the client with that the password is incorrect.
7. When the password is correct the second factor for authentication is being checked. A second factor is in this design TOTP. This factor can be entered wrong due to being entered just after the valid time window. For this reason, there are three attempts. This together with step 3 corresponds to BLESS step S.1.
8. The attempt is verified with the database together with a script for translating the second factor, and the result is sent back. This is S.2.
9. Each attempt is logged for auditing.
10a, 10b If after three attempts there is still an error, an error is sent back to the client saying that he can’t authenticate himself.
11. Request the certificate at every worker. This is similar to the step S.3 in BLESS.
12. At the certificate authority, the certificate is logged. Every worker has its certificate logged. This corresponds to S.4.
13. The certificate is sent back to the worker, and it is inserted in the keystore. The certificate identifies the user and job. This is step S.5 and S.7 in BLESS.
14. The certificates are distributed to one or multiple workers.
15a, 15b The workers and the Spark master establish a connection over TLS. This is comparable with step S.6.
16. The result of the connection setup is returned. This is comparable with the S.8.
17. If there was an error, it gets sent back to the client. The establishing of the connection failed in the cluster failed, an error gets returned.
18. The secure connection is set up. Start the job for this user on all workers.
19a, 19b If it is not a long running job, the result is returned to the user.
20. At the end of the job remove the certificates from the keystore.
21. The result is returned to the client.

The requirements are compared with the requirements in Section 5.3.3. The requirements fulfilled are R1, R2, R3, R4, R5, R8, R9, R11, R13, R15, R16, and R17. Requirements that can be fulfilled when having the right implementation R6, R7, R10, R12, and R18. R14 is now met since the authentication overhead is minimised by sharing the single certificate across the cluster. R19 is achieved in the same way as in the earlier Section 5.3.3. The output is stored in the HDFS, but the user specifies where the output has to go. Setting the access rights for accessing this output is challenging. In the future, an improvement could be to use this database, to let the users specify in which folders they have stored the output. In this way, the results can be shared with other users.

5.4 Authentication Configuration options

To enable authentication with certificates, it is necessary that the user can provide information for authentication. When using Kerberos, the information given by the user are the keytab and the principal. For TLS there are some basic configuration options to turn it on, the options are set in the Java System properties within Spark [72]. The user needs to provide a keystore and a truststore with the intermediate and root certificate from a Certificate Authority. The TLS location for the keystore and truststore is initialised in the Spark-defaults.conf before startup of the cluster when running Spark in standalone mode [72]. The truststore is always initialised. For authentication with TLS, there is thought of additional options. The Table 5.2 illustrates the extra options.
Table 5.2: Spark Configuration options

<table>
<thead>
<tr>
<th>Option</th>
<th>Summary</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>spark.ssl.authSSL</td>
<td>Option to turn on the authentication for Spark over TLS.</td>
<td>true</td>
</tr>
<tr>
<td>spark.ssl.auth.certificateAuthority.enabled</td>
<td>Option to enable that you have your own certificate authority that can support automatic certificate generation scripts.</td>
<td>true</td>
</tr>
<tr>
<td>spark.ssl.certificateAuthority.ip</td>
<td>The IP address of the certificate authority.</td>
<td>127.0.0.1</td>
</tr>
<tr>
<td>spark.ssl.auth.dir</td>
<td>The directory where the certificates are located, and the certificate requests are stored. This directory is built to gain a clear overview of the certificates in Spark. This directory its path is absolute from where Spark is launched. So if Spark is launched from the Spark Home folder, then the certificate folder is there in the folder specified by the user.</td>
<td>dir/cert</td>
</tr>
<tr>
<td>spark.ssl.authUser</td>
<td>The username that is used for the authentication.</td>
<td>Paul</td>
</tr>
<tr>
<td>spark.ssl.auth.password</td>
<td>The password that the user uses to log in.</td>
<td>test123</td>
</tr>
<tr>
<td>spark.ssl.auth.DB.ip</td>
<td>The IP where the database is located.</td>
<td>127.0.0.1</td>
</tr>
<tr>
<td>spark.ssl.auth.secret</td>
<td>The second authentication factor. This is a TOTP six digit secret.</td>
<td>123456</td>
</tr>
</tbody>
</table>

The certificate authority can be set by providing the IP address in the spark-defaults.conf, for registering a user. The script is defined in Section 5.5.6.4, the script uses the database and verifies the user. The authUser is used as a new username for authentication, this user can provide its password and a second factor TOTP secret. The authentication options, username, secret and password can be given via the Spark submit command as well. The Spark submit command is earlier illustrated in Section 2.1.4.4 in Listing 2.2. In Table 5.3 is a description of the additional Spark submit options.

Table 5.3: Spark submit options

<table>
<thead>
<tr>
<th>Option</th>
<th>Summary</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>–auth-user</td>
<td>The username that is used for the authentication.</td>
<td>Paul</td>
</tr>
<tr>
<td>–auth</td>
<td>Option to turn on the authentication for Spark over TLS</td>
<td>true</td>
</tr>
<tr>
<td>–password</td>
<td>The password that the user uses to log in.</td>
<td>test123</td>
</tr>
<tr>
<td>–secret</td>
<td>The second authentication factor. This is a TOTP six digit secret.</td>
<td>123456</td>
</tr>
</tbody>
</table>
5.5 Implementation

In this section of the thesis is the implementation described. The implementation has different important actors, such as the Certificate Authority (CA). This actor signs and distributes certificates. The implementation of the CA is in Section 5.5.1. The implementation of the database is in Section 5.5.2. The database handles permission and authentication. The implementation of the Spark standalone cluster is described in Section 5.5.4. The implementation of the Spark YARN is in Section 5.5.5. The implementations require uniquely identifiable certificates, the certificates are identified with a UUID and a username. The unique identifiable certificates can be tracked. For the logging, there is the Spark logging system. The logging system is improved and is described in Section 5.5.3. To make the implementation work with the different actors requires programming scripts. The programming are in Section 5.5.6.

5.5.1 Certificate authority

For reasons mentioned in the analysis Section 4.5, Section 4.6, Section 4.8, and the requirement Section 5.1, short-lived certificates are proposed to authenticate users. On the contrary, for the root and intermediate certificate long living certificates of one year are chosen. The root and intermediate certificate require thus a stronger encryption than the user certificates. The certificate for the user, the intermediate and root certificate form a chain of trust. The certificates are stored in the truststore. Building the truststore up for every connection would take too much time. In the next Section 5.5.1.1 is explained what to do if a CA is not trusted anymore. In Section 5.5.1.2 the certificate integrity is described. In the Section 5.5.1.3 the certificate signing process inside the CA is explained.

5.5.1.1 Trust in Certificate authority

The administrator setting up Spark has to define which CAs can provide the certificates. Here a caveat is that for our solution there is a connection between a CA and Spark. The connection is over a safe SSH connection. Over this encrypted connection, the certificates are obtained. The certificates are thus only requested at the CA by which the IP is known. If a CA is not trusted anymore, the certificate requests should not be sent to this CA. The certificate authority should be removed as a trusted authority. In this way, the chain of trust is not established anymore. The chain of trust is earlier described in Section 2.3.2. The chain of trust is maintained in the truststore. When a certificate is overdue, it cannot be used for authentication anymore, and the certificate needs to be refreshed.

5.5.1.2 Certificate integrity

The certificate is verified on whether it is still valid. It is important to make sure that the certificate has not been tampered with in order to trust it. To prevent tampering, and thus provide data integrity and non-repudiation, the certificates are hashed and made valid for a short time. In the case of this certificate, the certificate is hashed in general over SHA-2, a technique to create a digital signature, better known as a certificate, with a hash [95]. A hash is irreversible and required to prevent modifications to a certificate. SHA-2 is a specific family that can be obtained in different bit-sizes. At the beginning of TLS SHA-1 was used, but this encryption is now too weak [95]. SHA-2 is used in this solution to create the certificates.
5.5.1.3 Process

The process for the certificate authority is the following; Spark asks for a certificate at the CA. The CA looks if this request is justified. The certificate forms a chain of trust with the signed certificate when the keystore has the root certificate and intermediate of the CA. These certificate are the public key that is shared by a CA to Spark, this public key is described in Section 2.3.1. When the root or intermediate certificate is not apparent in the keystore, the chain of trust is broken, and there is no trusted authentication. These certificates need to be supplied by the user in the certificate directory. The user needs to create this directory and give the pathname in the configuration. This additional configuration option is explained in Section 5.4. The CA has to generate the certificate. In a self-signed CA, a script can be used to sign the certificate, this script is described in Section 5.5.6.1. The self-signed CA is safe for internal use. For a business using this implementation a proprietary CA is recommended, because of improvement of trust and extra security as explained in Section 2.3. A proprietary free CA like Let’s Encrypt is not suitable for this solution, since those free solutions have their limitations. For example, Let’s Encrypt has a limitation of only 20 certificates a week, which means that you can only run 20 jobs a week with this newly proposed solution [96]. An important field in the certificate is the "common-name" because this is the common name by which a user and application can be identified in the new design. When the certificate is requested with the "common-name", the certificate returns signed with the same "common-name" to identify the user and the job. This job launches with the certificate for the specific user and application. For the auditing, every certificate activity is logged. With the certificate the Spark workers and master are validated, this is like the instance validation in BLESS. More information about the auditing is given in the implementation of the logging system in Section 5.5.3.

5.5.2 Access control database

The database is the actor responsible for storing authentication, and access control data about the users. In this way, it is possible to verify the user. The database does the permission management. It checks if a user is to execute a job. It can be that a user may temporarily not execute jobs and has to be blocked. The resource permissions are managed by the database as well. For example memory usage or CPU. One user might be provided with more cores than the other. This can be important when the performance of some applications should be higher than others. The Spark resources manager tries to use all resources. the database is a MySQL database. MariaDB is an extension on top of MySQL that improves MySQL, and it provides an open source MySQL cluster, the original MySQL cluster is owned by Oracle and proprietary [97]. it can’t be used for commercial purposes. For this reasons, MariaDB with MySQL is chosen [98]. The multi factor authentication happens in the following steps:

1. The username and password are verified in the database.
2. The user receives a request to enter its six digit pin.
3. The user enters his six digit pin.
4. The user is verified, the attempt gets logged.
5. The now obtained certificate is added to the keystore.

In the use cases in Section 5.3.3, 5.3.4, and 5.3.2 the user needs to be authenticated. The authentication of the user with the database works as described in the above steps. The described approach assumes that the user is already registered in the database. When the user is not yet registered, the user needs to be added. A small Python script is written to register the user. The script is explained in Section 5.5.6.4.
Apache Spark has only read rights on the database to ensure that a job can’t manipulate the data in the database.

In the database is an administration table with an identification (ID), username, password and a secret for the MFA. The secret is 16 characters long. The character length of the username and password is up to the user.

Table 5.4: Administration table

<table>
<thead>
<tr>
<th>Table name</th>
<th>Example value</th>
<th>ID</th>
<th>username</th>
<th>password</th>
<th>secret</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>1</td>
<td>Paul</td>
<td>test123</td>
<td>65CQQZIEWTPC6AJZ</td>
<td></td>
</tr>
</tbody>
</table>

The database is created as a cluster, to test if it can work in a consistent and available manner. To connect to the cluster a structured query language (SQL) node can be used, this node has an ip address which can be used to redirect to the database that has the “Administration table”. There can be multiple nodes, to ensure that a user can connect to the database. In this implementation, only one SQL node is used.

Table 5.5: Resources table

<table>
<thead>
<tr>
<th>Column name</th>
<th>Example value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>user_ID</td>
<td>1</td>
<td>ID linked to the user name.</td>
</tr>
<tr>
<td>driver_cores</td>
<td>1</td>
<td>Amount of CPU cores used by the driver.</td>
</tr>
<tr>
<td>driver_memory</td>
<td>1g</td>
<td>Amount of memory to be used by an driver.</td>
</tr>
<tr>
<td>executor_memory</td>
<td>1g</td>
<td>Amount of memory to be used by an executor.</td>
</tr>
<tr>
<td>python_worker_memory</td>
<td>1g</td>
<td>Amount of memory to be used by Python during aggregation.</td>
</tr>
<tr>
<td>cores_max</td>
<td>2</td>
<td>Number of CPU cores used for an application.</td>
</tr>
<tr>
<td>task_max</td>
<td>1</td>
<td>Number of cores that is used for each CPU task.</td>
</tr>
<tr>
<td>executors_max</td>
<td>10</td>
<td>Maximum executors that can be used with dynamic allocation.</td>
</tr>
</tbody>
</table>

The Table 5.5 is linked with the user ID to the Table 5.4. The resources of each user can be managed, an illustration of what resources can be managed by the database is in Table 5.5. The resources are based on the Spark configuration that can be used to configure the amount of processing power for an application [73]. In the Table 5.5, is the column name, an example of how to set it, and a short explanation of the table. Setting the resources is helpful to make sure that users can’t use more than their assigned resources. This can be valuable in a company where multiple developers use the Spark cluster. In the database the resources of the developers can be limited, to control the resource usage. Future work is implementing that another user than the one that executed the job can view the result. A group could be made for people that are allowed to see the results.

5.5.3 Auditing system

In Spark, every worker has its own logger that keeps track of the log files for the jobs. This logger needs to be set up by Spark. There is a file called log4j.properties where
the settings can be filled out. In the log4j file is set whether to display the logs on a
terminal console or to save it to a file. In standalone every worker that performs some
work has a log file about the job. In Spark YARN there is an option to get all the logs
from the workers and master [99].

In the implementation of the design, the audit logs are coming from three parts,
the CA, Spark Master, and the workers. The master and workers have their own log
folders for each job. The CA maintains a list of the certificates that are handed out.
If there is a duplicate, it will not be added to the list, and an error will occur. Further
implementation of the certificate is described in Section 5.5.6.2. When something
went wrong with the job, the certificate and the log trace are used to verify what went
wrong. The information is also logged if the authentication went successful. The
three components CA, Spark master, and worker make adequate auditing possible.

5.5.4 Spark cluster

In Spark the authentication is until now achieved with Kerberos. The data trans-
portation security is enabled by TLS [100]. The configuration of TLS has to be set
according to the guidelines of encryption in the attached source [73]. The keystore
and truststore have to be placed both on the worker and the master to make TLS
work. With the truststore a certificate in the keystore is verified. The Keytool of Java
provides us with the possibility to import a signed certificate into the keystore. The
keystore and truststore have the root certificate from the CA so that the certificates
from the CA can be verified and imported. This script is described in Section 5.5.6.1. If
the keystore sends a certificate that has the same chain the connection is established.
The keystore will work as described in the use case of Section 5.3.3. The user can be
individually identified. The next paragraph explains the process with the keystore.

5.5.4.1 Process

The steps taken in Spark Standalone are the following.
1. Check keystore and truststore are set correctly, and the configuration of Spark
   is correct.
2. Check if all other possible valid certificates are deleted from the keystore except
   for root certificates.
3. Request and receive a certificate for every worker and put it in the keystore
4. Verify the certificate with the chain of trust.
5. Make sure the resources for Spark are available.

In the solution, certificates are generated on demand. The certificates have to be
easily added and removed. In Standalone the certificate request happens in the
SparkSubmitArguments. The SparkSubmitArguments file in the core folder. Here
is a the scala deploy folder containing SparkSubmitArguments. During this submit
the certificates are requested to set up the connection. The author is authenticated
according to the design earlier discussed in Section 5.3.3, and the script in Section
5.5.6.3. The certificate is requested by the use of the script in Section 5.5.6.1
The created request is sent to the CA to be signed. The certificate request is stored
in the certificate directory. The Certificate Authority signs the certificate by using
the script in Section 5.5.6.2. When the certificate is received back, it is added to the
keystore. Every worker now has a certificate.

5.5.4.2 Setup

The settings of the system are set before launching the Spark job in order to have a
safe connection and authentication. The path of the keystore and truststore can be
absolute or relative to the directory where in which the job starts \[73\]. When the keystore is located in the Spark home folder, then the command should be submitted from Spark home. For the request of the certificates later it is important to set the configuration for certificate authority its IP address and the directory. These configurations are earlier described in Section \[5.4\]. This setup of the SecurityManager happens at every worker. This is realised by looking at the code and editing it in such a way that it prints something. This happens at the startup and the initialization of the job. The setup of the TLS connection requires that there are certificates in the keystore at the start of a job.

### 5.5.5 Spark with YARN cluster

The implementation of Spark with Yarn can be used to distribute a keystore containing the certificates. The keystore is used together with the truststore to turn on the TLS wire encryption. The advantage of YARN is that files can be transported to workers before the start of the job. The master can thus send the keystore before the start of the job to the workers. In the next Section \[5.5.5.1\] there is shown how YARN achieves distribution of the keystore before starting the job. The process of authentication is explained in Section \[5.5.5.2\].

#### 5.5.5.1 Localisation and isolation

The YARN cluster provides localisation \[101\]. Localisation means that the remote resources on for example a Hadoop system are copied to the local machine, from which they can be accessed. YARN does this localisation before the start of the job. YARN can distribute files before the job starts. The files are given to each of the Spark executors. The resources and sensitive data is protected in YARN by using isolated containers. The resource isolation is earlier explained in Section \[2.1.3\]. The keystore has to be sent before every Spark Job to all the workers. A bigger keystore creates more overhead than a smaller keystore, therefore the file size matters. The size of the truststore is 3128 bytes. The file size of the keystore with one certificate for the client one key and two verification certificates is 7893 bytes. Distributing such a file size is noticeable, but does not cause significant delay. The delay is less than a second. An important caveat is that the workers get added and dropped dynamically when dynamic allocation is on. Currently, this is not automatically implemented. A solution here might be to guess the process time the job needs to run and ask for this time a certificate. However, if the work runs for weeks the certificate should be renewed several times during these weeks. Therefore a better solution might be to update the certificate automatically after a certain time period. To help understanding the solution, an overview of this solution is given in Figure \[5.4\]. The keystore is distributed by the YARN ”–files” option to every Spark worker and thus executors. The workers receive the latest keystore version. As long as the job is not finished the keystore is updated with new certificates. When building a new worker the connection needs to be setup again. It might be that the certificate is by then expired and a new one needs to be requested. One way would be to check 1 minute before the certificate expires if a the job is finished. If it is not finished a new one should be requested and added to the keystore. In YARN there is a function to check whether a job is finished \[102\]. Further, the database should be updated keep track of the job, there should be kept track which certificates belong to the job. Apache Spark should be given the possibility to only insert data in this table. If this concept is implemented YARN should be able to support a resilient dynamic allocation with authentication.
5.5.5.2 Process

The implementation of YARN is located in the "resource-managers/yarn/src" folder. Inside the main folder is the code for Scala. In the Scala deploy folder are the files located to set up Spark before the job starts. The Client.scala is the file changed in this implementation. Here in YARN the certificate is requested, inserted into the keystore and sent to the workers. The authentication of the client happened before the resource manager is initialized at the same place where the authentication for standalone happens. This is in the SparkSubmitArguments document. The process follows the design in Section 5.3.4. The password and the user its second factor are verified according to the use case, for this there is a script explained in Section 5.5.6.3. A certificate request is created and sent by the request script explained in Section 5.5.6.1. The CA signs a successfully created certificate sign request with the script in Section 5.5.6.2. When the certificate is received back, it is added to the keystore. If that is successful the certificates are distributed with YARN’s its option to distribute files, the job execution of the job is then started. After this, the result is sent back to the user either over a console or by asking the result logs via the terminal.

5.5.6 Programming scripts

For the implementation, various programming scripts have been developed. In this section, the scripts are discussed that are used for the certificate authority. There is a script to request and receive the certificate in Section 5.5.6.1. The script to request and receive the certificate is located in the Spark folder so that it is accessible by the other Spark classes. The Certificate Authority has a script to sign the certificates, this is explained in Section 5.5.6.2. The user is authenticated with the database over a script. The script is described in Section 5.5.6.3. The users that want to use Spark need to exist in the database, to create the user there is a script in Section 5.5.6.4.
5.5.6.1 Request and receive certificate

When requesting the certificates from the certificate authority, a script is required to send the request and receive the certificate. This script is created in Python. In Python are several modules that can be reused to make it possible to send and receive certificates. These modules are discussed in the next Paragraph 5.5.6.1.1. The process of sending and receiving the certificates is discussed in the last Paragraph 5.5.6.1.2.

5.5.6.1.1 Modules

Python has several modules to send and receive information safely. These are for example the modules Paramiko, or Rsync [103, 104]. Paramiko and Rsync are modules to send and receive files. Installing Paramiko requires 150 MB extra space. Here it is assumed that the software for using Spark is already installed. The 150 MB is significant. Rsync is pre-installed with Python and therefore on most operating systems does not require additional data. However, Rsync requires an SSH connection. This SSH connection establishes a safe connection to send the files [105]. In this research, an SSH connection can be established. Rsync can be used when it is not possible to establish an SSH connection Paramiko should be used. The use of Rsync makes that the password does not have to be transported over the network since keys secure it. In Paramiko to access the other server the password for the server has to be sent over the network. This can potentially be intercepted, so in general, when SSH is possible SSH is advised. To enable SSH a public key can be distributed, so two servers can connect with each other and then send in a secure way the data. In BLESS for the SSH is a certificate created, to keep track of who sets up an SSH connection with a server. This situation assumes that there is only one person can make the SSH connections. For providing safe SSH access for multiple users BLESS can be implemented to handle certificates for setting up an SSH connection with more auditing. Rsync can copy files over a network. It is optimised for fast file transfer [104]. There is a transfer algorithm in Rsync that will make sure only differences between file A and B are sent over the wire.

5.5.6.1.2 Process

The script is used for a certificate request or to receive the signed certificate. The script can be used to either send or receive files, making the distribution of files possible between the CA and Spark. In this project, the workers, master, and certificate authority are set up by an administrator. The administrator can set up the SSH and only has to do this once. For this thesis Rsync is chosen for transferring the files. Rsync has command line options, allowing the user to fine tune its behaviour. This allows us to change the default parameters which can be changed for every application. A Rsync command looks like the following:

Listing 5.1: Rsync command

rsync originIP:fileLocation destinationIP:fileDestLocation

For the sending and receiving of the data the structure described in Listing 5.1 is used. The IP address of the origin and destination are specified, and the file is sent to the destination.
5.5.6.2 Sign certificate

For the implementation of a script to sign certificates in the CA a cryptography library is required to sign certificates. OpenSSL is a cryptography library which can be used for signing certificates [106]. The OpenSSL library is standard installed on most Linux servers. The OpenSSL library can sign the certificates. These certificates can then be used inside the keystore. The certificate authority has a root certificate, root key, intermediate certificate and intermediate key. The keys are the private keys to sign the certificate. With the root key, the intermediate certificate is signed, this forms a chain of trust. The Root key is in general kept away hidden from the internet to make sure that nobody has access to it. If the root key is compromised, the certificates are as well because with the root key every certificate can be signed and is trusted. For this reason, an intermediate key which can be revoked if compromised. When comparing to BLESS, this is the key used in step S.3. The intermediate key and certificate are used to request new certificates for jobs. The use of an intermediate is earlier explained in Section 2.3. The root certificate with the intermediate certificate creates a chain of trust. Each keystore on a Spark worker or master has a chain of trust. When adding the certificate the integrity gets verified with the chain of trust. The chain of trust is in the connection setup used to verify if the certificate is given out by the right authority. If the certificate is given out by the right authority and the certificate is valid, the connection is established. When a certificate is generated there is the possibility to add data to the certificate to later recognise it, this is earlier explained in Section 5.3.1. The Common Name (CN) in the certificate represents the name in the certificate. This CN gets a unique name to identify the user and the job. For the generating of certificates is decided to write the code in Python. In the Python script, the username with a UUID is added to the certificate for the CN. More information can be added as well to identify the certificate, such as country name, state or province, locality name, and email. Currently, the location of the script to generate the certificates has to be manually set in the code. The YARN script path is set, in YARN Client.scala. The script path is set in SparkSubmitArguments for running locally. The cluster script path is set in the SecurityManager for standalone mode. The script is called by python in the following way: "python script.py name-ID.csr" the output should be then in the form name-ID.cert . This gives back the unique certificate for the user.

5.5.6.3 Database user verification

The user is verified by the password and second factor authentication. A script is developed in Python where the password gets checked by comparing it to the password in the database. To compare connection with the database has to be made. This is achieved with the MySQLdb library [98]. This library provides a Python API to a MySQL database called "mysql-connector". The "mysql-connector" connects to the database over the IP address. The IP address needs to be set manually in the configuration. This configuration option is previously explained in Section 5.4. If the password is compared successful, meaning there is no MySQL error, or the user obtained an error that its password is wrong, then the second factor is compared. For the second factor there is chosen to implement TOTP, TOTP is open source and does not require additional hardware or software to be bought. TOTP is earlier explained in Section 2.4. To use the second factor in this implementation, the user has to enter a six digit phrase. The six digits are generated by a library for generating TOTP. This can be a Google Authenticator. Google Authenticator offers a plugin where the user can store its secret, and the six digits for authentication are generated. The phrase gets compared with the six digits generated by the server. The PAM library used by the server to generate the six digits is PyOTP [107]. This library uses the secret
stored in the database for a specific user. By the secret stored the six digits are generated for a specific time frame. The system of the user and the server require time synchronisation in order for the six digits to match, this can be achieved by using a protocol such as NTP. On the phone, a user can install Google authenticator and store the secret. For phone time synchronisation the phone uses the time given by the network provider, this is most of the time using the NITZ protocol\(^\text{[108]}\). When the six digits match in a specific time frame, the TOTP authentication is completed successfully.

5.5.6.4 Database user registration

In this section a script is explained that can create new users. In Paragraph 5.5.6.4.1 there is discussed how privileges can be used to manage the database access. Then the process this script performs is explained in Paragraph 5.5.6.4.2. For the creation of a user a Python script in the external folder, this can be executed in the following way:

```
scala insertUserPass username password
```

In this script, the database password needs to be entered. One has to be very careful with running such scripts. For the reason that the database password required can insert users and other data in the database. For static passwords for both the users and the administrators, it is advised to follow best practices, such as hashing and salting\(^\text{[109]}\). The hash makes sure that it is hard to reverse the hash into the real password, this ensures that the real password is not revealed\(^\text{[110]}\). Salting is then used to make it harder for a hacker to perform a dictionary attack. In such a attack the hacker tries common passwords. The salt prevents this by randomising the hash string, making a dictionary attack impossible\(^\text{[110]}\). The salt should differ per password, and there should be a list of salts maintained in the database\(^\text{[110]}\).

5.5.6.4.1 Setting privileges

According to the computer ethics identity theft should be prevented\(^\text{[111]}\). In the design, identity theft is taken into consideration, and the privileges are managed in the database by having different roles for a user. In a database, it is possible to give the administrator a specific privilege. This can be done with ALL PRIVILEGES, then the administrator has all the privileges, with the SELECT privilege the administrator can only read through the database. When authenticating a user information has to be read from the database, so then the SELECT privilege can be used. In Section 5.5.6.3 Spark accesses the database to authenticate a user, Spark has SELECT privileges to achieve this. To insert new data into a database an INSERT privilege is given to an administrator. This script for registering a new user requires inserting new data about the user in the database, so the INSERT permission has to be granted. The administrator of the database maintains the database. This includes the possibility to add users with new privileges. The administrator has ALL PRIVILEGES and is for example allowed to change the privileges of another user. To further improve the security the IP addresses allowed to access the specific administrator can be added in SQL.

5.5.6.4.2 Process

The script is programmed in Python. The ”mysql-connector” is used to connect with SQL. This connector is previously used by the verification script. The database
"authDB" is accessed over the script. In this database, there is a table to store details about the users. There is an adminTable, in this table the user details are presented. The table is organised in the same way as in Table 5.4 in Section 5.5.2. This script for registering the user first creates a username and password. The database is updated with a secret generated by the PyOTP library [107]. Now the user is added with all the details. When the script finishes a secret is displayed, the user can use the secret for generating the six digits for authentication.
Chapter 6

Evaluation

In this chapter, the new concept is evaluated. The implementation is tested, the setup used to test is explained in Section 6.1. Here the hardware, programming language, and the software that is required for this project are explained. The different test cases are evaluated in Section 6.2. Spark is under a continuous development, during this development Spark went from 2.1.0 to 2.1.1. Users demand usability and thus easy installation. In Section 6.3 is explained how the implementation can be compiled to a version that can be easily distributed. In Section 6.4 the Certificate authority module is evaluated, and in Section 6.5 the database is tested and recommendations are given to setup a database. At the end of this chapter in Section 6.6 the requirements are validated.

6.1 Setup

Spark is select as the environment to implement the new concept in, the Spark implementation is earlier described in the sub sections of Section 5.5. This section describes the setup. This setup consists of several Linux containers, the hardware in these containers is described in Section 6.1.1 and the software in Section 6.1.2. In Section 6.1.3 is described in which programming language the implementation is written. The evaluation environment is set up and tested in Section 6.1.4.

6.1.1 Hardware

The evaluation environment is to test the different designs from Section 5.3. The evaluation environment of this project consists of four Linux containers. One Certificate Authority for handing out certificates. One Spark Master orchestrates the work and distributes the work among the workers. Two workers that execute the Spark job demanded by the user. A Linux container provides isolation of the applications within the server. Each container gets a certain amount of CPU, memory and storage. Every container has 2 CPU cores assigned. The biggest container has 2 GB of RAM. This container is going to run the Spark master. The three smaller ones for the Certificate Authority and the two Spark workers have each 1 GB of RAM. The Spark master has 15 GB of storage space, and the other three have 5 GB of storage space. In the next Section 6.1.2 the software installed in each container is described.

6.1.2 Software

Certain software is required to be installed for this project in each container. All the Linux containers run an operating system called Linux Ubuntu 16.10. Spark requires having either Windows, MAC OS, or Linux as an operating system. For this development Linux is chosen because of the familiarity by the author and because it is open source and has a lot of freedom [112]. To run Spark Java is installed. For running Spark version 2.0 or higher, there is required that Java version 1.8.0
or higher installed. In this project, Java version 1.8.0 is installed. Java is owned by Oracle. Companies who work with Java from Oracle are required to pay a license fee to use their software [167]. Paying a fee is a problem for this project since this project tries to utilise free software. Therefore, an open source version of Java is used. This open source version of Java is the Openjdk-8-jre. For running Spark in a cluster, the nodes have to know each other so that data can be exchanged. The file "/etc/hosts" is changed, in the file each node its IP address is specified. The design in Section 5.3.4 requires the installation of the resource scheduler yet another resource negotiator (YARN). YARN comes installed with Hadoop. The version used for this project is Hadoop 2.7.3. Hadoop is required is because of its configuration settings that are required to run YARN. The configuration for YARN is inside the directory of Hadoop. YARN has to be set in the environment variables of the computer as well. The environment variables make sure that Spark knows where YARN is located and can thus be used as a resource manager. In the "Hadoop.version/etc/Hadoop" folder the configurations are located. In order to let YARN work across the cluster, it has to be installed on all nodes.

### 6.1.3 Choosing programming language

Apache Spark performs best in Scala since the algorithms coded on Spark are optimised for Scala. For compiling Spark uses Java. For Java, Python and Scala there is good documentation available. At Fraunhofer SIT, Spark is used most of the time with Python. Hence, the most significant part of the implementation is in Scala, and a small part is written in Python.

### 6.1.4 Setup for test cases

To make sure that the test environment is set up correct, a test is performed in Spark. The test is performed locally, in the cluster, and in YARN. For each of the configurations, basic Spark jobs are tested. For the testing there is decided to test with Python jobs for Spark. Py4J is standard installed to run the Python jobs. An example Spark Python job is given here:

```
Listing 6.1: Spark Python job
$SPARK_HOME/bin/spark-submit \
  --master spark://host:6066 \
  examples/src/main/python/pi.py \
  1000
```

During the tests, the transport layer security (TLS) is turned on by following the configuration instructions from the Spark website [73]. The logs can be viewed if Log4J is turned on. In a log file can be seen if the SecurityManager correctly sets the TLS connection. The implementation is tested incremental first there are small manipulations made to see if it would be possible to authenticate with Spark. The test cases for authentication are highlighted in Section 6.2.

The CPU settings in the Java virtual machine (JVM) have to be properly set. The JVM needs configuration with a maximum and minimum CPU usage. An example is in listing 6.2, here the number of cores is set during start-up.

```
Listing 6.2: Setting cores
$SPARK_HOME/sbin/start-all.sh --cores 2
```

In Spark, the number of CPU cores can be more than there are available for the virtual machine (VM). During the setup, it occurred once that the test took 48 cores (all
available on the server), instead of the two cores assigned to the container. For this reason, the manager of the server kills the process. The error is resolved by setting the number of cores manually.

6.2 Test cases

In this section, different test cases are highlighted. The test cases are performed by testing from terminal and assertion testing. The Spark test folder is organised following the test assertion guidelines [80]. Assertion testing verifies the correct working of functions. The assertion testing for authentication has tests for authenticating local jobs, and authentication tests for obtaining the Spark status. The authentication testing for a local job is described in Section 6.2.1. The authentication for the Spark cluster is tested in section 6.2.2. The authentication for the YARN cluster is tested in section 6.2.3.

6.2.1 Authentication testing

In this section this test case is the new mechanism for authentication in Apache Spark tested. This test case consists of two parts. One part is executing a job locally, this is explained in Section 6.2.1.1. The other part is obtaining the status of a job explained in Section 6.2.1.2.

6.2.1.1 Local job

To run a local job the implementation is inspired on the design in Section 5.3.2. For verifying if the implementation works there is the log file with information about the execution of the job. The importance of having such logging is earlier discussed in section 5.5.3. There is tested if the authentication with the database can be performed. Assertion testing is conducted to test if the functions work correctly in paragraph 6.2.1.1.1. The execution of the job over the terminal console is tested in paragraph 6.2.1.1.2.

6.2.1.1.1 Assertion testing

In the assertion testing, the functions of the SparkSubmitArguments script are tested. In this script is a function to verify a user with the database. Further, there is a function to test the six digits. In assertion testing is tested if the output is the same as the expected output. The testing for SparkSubmitArguments is located in the core "core/src/test", there is a folder deploy for deployment testing. The SparkSubmitSuite is where the assertion tests are for SparkSubmitArguments, and thus the test for submitting a job. In Spark, it is possible to test classes individually. The SparkSubmitSuite is adjusted to test the extra configuration options in the implementation. This testsuite is for debugging purposes to see whether everything worked. For testing are the following three steps:

1. launch sbt by running build/sbt in the Spark folder: sbt
2. switch to core project: >project core
3. Run the test: > testOnly *SparkSubmitSuite

Inside the SparkSubmitSuite there is a test case to test the authentication, this is called "User, password, secret test". To run the test case the authentication parameter "–auth" is in the test case to test the authentication. The parameter to setup the database and the certificate authority (CA) should be switched before the test to make testing possible. The default parameter setting for testing with authentication
is "false". This needs to set to "true" to run authentication tests. The reason reason
for this default setting is, that in general Spark users do not have the database set,
so the users cannot run this test case. These are additional configuration settings
implemented and are set as in 5.4. These configurations are required to test the
new functions for authentication. Code that is not tested properly can crash Spark.
The configurations are set by parsing a sequence of configuration options like the
following sequence Seq("–auth", "true", "–auth-user", "username", "–password",
"password", "–secret", "123456"). Here there is verified with the system properties
whether all settings are set, this is shown in Listing 6.3.

Listing 6.3: System property test
sysProps . keys should contain ("spark . ssl . authUser")

If the user is configured correct, the assert completes with true. In this way, every
configuration is checked. After the configuration is tested, the different functions are
tested. A function verifies the password with the user verification script in Section
5.5.6.3. To test the script, a database administrator is required on the localhost. There
are two functions tested, user verification and secret verification. There is an assertion
test to test the secret and the username and password in the SparkSubmitSuite.
An assertion compares the actual result with the expected result. The password and
username are verified. For the test case usernames with passwords have to be in the
database. The following scenarios are used for testing:
1. A right username and password, this gives the expected result true
2. A non-existing user, this gives the expected result false
3. A username and a wrong not existing password in the database, this gives the
   expected result false
4. A username and an existing password belonging to another user, this gives the
   expected result false.

The second factor assertion test is to test if a TOTP six digit secret can be correctly
obtained for a user and verified. There is checked if the secret can be obtained. To
test there is a user "username" in the database and this user has a secret. This secret
can then be used for generating the six digit authentication code. In this test, there
is checked if the six digits authentication code can be generated. In this test, the
generation of the six digits is successful via the secret. The six digits are also tested
against a wrong six digit number, to see if the comparison is correct. Another test
is whether the user can correctly enter the username, password and secret on the
terminal. This is performed in the next Paragraph 6.2.1.1.2.

In Kerberos, the authentication happens within seconds [113]. However, for Kerber-
ros to scale to multiple KDCs and keep consistency is challenging. Further, the KDC
that stays a single point of failure [23]. To test the performance of this authentication,
an assertion test is made in the SparkSubmitSuite. To perform a test cases to verify
the password, verify the other second factor and obtain a certificate takes 0.4 seconds.
From this there can be assumed that the authentication is relatively fast and sufficient
for a new implementation. If there are 100 concurrent user logins at the same time
in the database, the authentication takes longer. A login test is parallely started 100
times with the Future method in Scala. During the test, there was realised that no
more than 100 threads at the same time could be started. It is not found why the
number of threads is limited to 100. The authentication mechanism implemented
is scalable to handle more than 100 users. The authentication takes a little longer
than 0.4 seconds on 100 users at the same time. The longest authentication took 0.9
seconds, and the average is around 0.75.

As there is observed in this section replacing Kerberos has a high chance of solving
current authentication challenges in the Hadoop environment. Kerberos does not
have a certificate authority or a distributed database that can be easily replicated such as the new proposed solution has.

6.2.1.1.2 Terminal test with input

The new design has to be tested via the terminal. With terminal testing, the proper working of the authentication for the Spark job is verified. There is checked whether the user can correctly enter the username and password. It might be that the assertion test work, and the job authentication via terminal fails. The command that is used is presented in listing 6.4.

Listing 6.4: Spark local Python job

```bash
$SPARK_HOME/bin/spark-submit \
   --master spark://host:port \
   --auth true \
   --username username \
   --password password \
   --secret 123456 \
   examples/src/main/python/pi.py \
   1000
```

In this submit command first the master is given with the IP address and the port number to connect. In general, to commit a submit command it happens over port number 6066. There are four new options compared to a normal local job, the auth for turning the authentication on, the username, the password and secret. These additional authentication options are earlier explained in Section 5.4 in Table 5.3. The authentication is set with "--auth" to true. When the user enters the password or username wrong, an error is printed that the username or password is incorrect. When the secret is false an error printed on the terminal that the secret is false, there is then asked to enter the secret again. The secret can be entered three times wrong. The reason for this is that the secret is time based and it might be that the secret is submitted at exactly the wrong time. For this reason, to improve the usability, there is asked to enter the secret again. It is three times to prevent that a brute force attack is executed on the secret. After the user authenticated correctly, a certificate is obtained and the job is executed.

6.2.1.2 Job status script

In this project, the REST application programming interface (API) is used to obtain the Spark Status. The REST API is earlier discussed in Section 2.1.4.5. Each Spark mode has a different job status. There is a different job status for running Spark local, cluster and YARN. The status report is obtained at the master for local jobs, and cluster jobs. How the job is obtained is explained in Paragraph 6.2.1.2.1. YARN obtains the job status via a command this is explained in Paragraph 6.2.1.2.2.

6.2.1.2.1 Job status local and cluster

For the local and cluster job, the job status is received over the REST API. Before the user can obtain the status, it is verified whether the user is allowed to obtain the status. Once the user is allowed the attempt gets logged, and the user can obtain the status report. The approach is tested by a user who isn’t authorised and a user who is authorised. When a user is not authorised, the user is rejected access. In the REST API, the status is obtained through the application ID. When asking for the
application state, a submit command is sent to the Spark master. The status for a Spark job is obtained with the command in Listing 6.5.

Listing 6.5: Spark status Standalone
```
spark-submit --status [application ID] --auth true \n--auth-user username --password password --secret 123456
```

The user verifies himself just like with the Spark job, the authentication settings are entered on the terminal. The application ID for the job is obtained when starting a job. With the application ID and the option ”–status”, the status can be obtained.

6.2.1.2.2 YARN job status

In YARN the status can be obtained with the command in Listing 6.6. This command has not been implemented yet with the authentication, since it is inside of YARN, in the future the authentication can be added, by adjusting YARN. Adjusting the source code of YARN is out of scope for this project. The receiving of the status over YARN is future work.

Listing 6.6: Spark status YARN
```
yarn logs --applicationId <application_id>
```

6.2.2 Authentication testing on Spark cluster

This test for the Spark cluster contains a Spark master and workers. In the previous section is explained how the status is obtained. In this section is explained how a cluster job is executed. There is tested to launch a job in the terminal in Section 6.2.2.1. There is explained how the certificate is obtained and used in Section 6.2.2.2.

6.2.2.1 Terminal test with input

In this section, a job launch is tested via the terminal. In the terminal console the configuration arguments are given. In Listing 6.7 is the command that launches a job in the Spark cluster from the terminal. The Spark deploy mode is set to ”cluster”. Spark assumes that it is a local job if the deploy mode is not specified.

Listing 6.7: Spark cluster mode command
```
$SPARK_HOME/bin/spark-submit \n--class org.apache.spark.examples.SparkPi \n--master spark://host:port --auth true \n--username username --password password --secret 123456 \n--deploy-mode cluster --executor-memory 1g \n--executor-cores 10 \nexamples/jars/spark-examples_2.11-2.2.0-SNAPSHOT.jar
```

In cluster mode, Spark contains the master configuration in the form of spark://host:port. This configuration is used to connect to the given Spark master. The port must be set to the port listened by the Spark master. By default this port is number 6066 to send commands, the Spark REST API listens by default to this port. The Spark job is tested via Java, this is because in Standalone cluster it is not possible to run Python. In this test, a class and a jar file are committed in order to run the Java job. In the cluster mode, the memory and the executor cores can be defined. The memory is the amount of memory each node can use. For example, the memory set is 1g (1024 mebibytes). The number of cores is virtual cores. These cores are in general more
than the cores available on the cluster. The cores also specify the maximum amount of executors that can be used in this case 10. There are three added options, the authentication, username and secret. The authentication happens analogously to the local job in Section 6.2.1. In both the local job and this cluster job the certificate is obtained after authenticating the user. The certificate is used for setting up the TLS connection across the cluster for safe data transportation. In the next section 6.2.2.2 the certificate is tested.

6.2.2.2 Certificate testing

The certificates are requested at different places depending on the Spark mode. The reason is that the certificate is used to make a secure connection in cluster and YARN. As earlier discussed YARN has the possibility to distribute files, the cluster has not. The certificate for the cluster has to be requested for every worker. The certificate is requested and obtained by SecurityManager of Spark where the connection is set up for each worker and master. For YARN and the local job, the certificate is requested in the class to submit the arguments, the SparkSubmitArguments class. Before requesting the certificate, there has to be known where the CA is to obtain the certificate. In the Section 5.4 there is explained that there is an option to enter the IP address of the CA. There is tested if a certificate can be obtained from the CA. In the test, the certificate authority is hosted on a different IP address as Spark itself to simulate a case that is close to a real use case of Spark with the new authentication. To test if the certificates are obtained correctly, two tests are performed, assertion testing in Paragraph 6.2.2.2.1 and terminal testing in Paragraph 6.2.2.2.2.

6.2.2.2.1 Assertion testing

To test whether the certificates are obtained the SecurityManagerSuite had to be adjusted. There is a test case developed called "keystore with certificates". The authentication is earlier in Section 6.2.1.1.1 tested with SparkSubmitArguments. A CA is defined to test if certificates are obtained. If a certificate isn’t provided by the right authority, then the certificate can’t be added to the keystore. When the certificate can’t be verified, Spark can’t connect, and an error occurs saying that the connection can’t be established.

6.2.2.2.2 Terminal testing

To test whether the connection is setup correctly, a terminal test is performed. To test if the certificates work, there are three scenarios envisioned. One to test whether an empty keystore will give an error. An empty keystore cannot setup a connection. The second is tried with a wrong certificate from a wrong certificate authority. The certificate request isn’t generated by this keystore, and the keystore does not have the root and intermediate certificate from this certificate authority. Therefore, an attempt to add the certificate to the keystore will fail. The third test is with a valid certificate gained from the certificate authority. In the three different modes, this happens differently. In the local mode and YARN mode, the certificate can be obtained at the Spark master. After authenticating the user, the certificate is obtained and added to the keystore. The Spark cluster mode does not automatically distribute certificates. However, every worker needs a certificate. The SecurityManager of each worker obtains the certificate. If there is something wrong with the certificates, there can be find an error description in the debug log file of the job. In the logs are the authentication of the user, the request for the certificate, and the setup of the
connection. The user receives a proper error message in case something went wrong. By logging all the actions happening with the authentication obtaining of certificate and setting up the connection, there is verified that the system is used correctly. This helps to fix potential errors in the system. The logging also provides the possibility to audit the log files, and thus attackers can be caught.

Every certificate request is logged in the certificate authority. The certificates are stored in a folder, and there is an index list to keep track of the certificates. This list keeps track of certificates that are not valid anymore and the ones that are still valid.

6.2.3 Authentication testing on Spark with Yarn cluster

The implementation is tested by running a Spark job in YARN mode. The terminal command executed for the test is seen in Listing 6.8.

Listing 6.8: Spark YARN Job command

```
$SPARK_HOME/bin/spark-submit \
  --class org.apache.spark.examples.SparkPi \
  --master yarn --deploy-mode cluster --driver-memory 2g \
  --executor-memory 1g --executor-cores 1 --auth true \
  --auth-user username --password password --secret 123456 \
  --files keystore,truststore \n$SPARK_HOME/examples/jars/spark-examples_2.11-2.2.0-SNAPSHOT.jar
```

The command has the option "--files", this makes it possible to spread files across all the YARN workers. This option makes it possible to distribute the keystore and the truststore. The certificate is put in the keystore and submitted to the cluster. The certificate is obtained in Client.scala as described in section 5.5.5. To test the implementation run the following command:

Listing 6.9: Yarn testsuite

```
./build/mvn Pyarn -Dtest:none \
-DwildcardSuites=org.apache.spark.deploy.yarn.ClientSuite test
```

In Spark there is a testingsuite to verify if in YARN resources can be obtained correctly. This testingsuite is called by running the listing 6.9. In the testingsuite, is verified if the certificate can be obtained. The authentication is performed in another class. The class SparkSubmitArguments and the authentication are explained in Section 6.2.1.1. After authentication, the certificate is allowed to be obtained and added to the keystore. This keystore is then be successfully distributed across all workers to start the job.

6.3 Continuous development, building and deployment

The implementation build has to be compiled. The compiling makes it possible to run the new implemented version of Spark. It is useful if a compiled production version of Spark is built. The production version contains compiled classes in Java and Scala. The production version does not contain the development source code. In the production version also some python scripts, such as a script to create a new user. The idea of the production version is, that it isn’t used for development anymore and only contains compiled classes and python scripts. The production version requires simple distribution. Therefore, it has to be in a zip or tar format, so it is relatively small. To create a zip or tar the project has to be compiled, if the project doesn’t compile correctly, errors are displayed. To run this project on a YARN cluster, the
project is compiled with YARN. Since YARN is also part of Hadoop, Hadoop has to be compiled as well. This allows the project to run in YARN, cluster and local. The code for compiling is shown in Listing 6.10.

Listing 6.10: Compile Spark

```
$SPARK_HOME/build/mvn --Pyarn --Phadoop-2.7
-Dhadoop.version=2.7.0 --DskipTests clean package
```

After compiling there is displayed that the build is a success. With the compiled build a new production version is made by using the command in Listing 6.11. This production version is distributed among other users. In this project the production version is used for testing the Spark terminal.

Listing 6.11: Production version Spark

```
$SPARK_HOME/dev/make-distribution.sh --name auth --tgz
--Phadoop-2.7 --Pyarn
```

6.4 Certificate authority evaluation

In the CA the certificates are signed. In this implementation the certificates are auto-signed, this has some consequences discussed in the following section. Autosigning has a security impact, because of the trust relation with the CA. The CA might be hacked unseen, and the certificates are still being signed. For this reason, the request and signing of certificates need to be logged \([114]\). The certificate authority keeps track of which certificates are requested. The transportation of sensitive data has to be secure. Instead of using self-signed certificates, licensed certificate can be used to improve the security. A challenge here is that companies might not have the opportunity to request an unlimited amount of licensed certificates. To handle this challenge, a proposed solution is having licensed certificates for the transport of data, and have the second certificate without a licence to authenticate the user. The user can then authenticate with the Spark master by checking if the certificate can be added to another second keystore. If the certificate does not come from the specified certificate authority, then the certificate cannot be added. The use of a third party certificate authority to obtain licensed certificates has some more implications shown in the next paragraph.

There are several challenges for using a third party certificate authority that provides licensed certificates. The first is the communication of the certificate request. This communication can be over SSH, HTTP with TLS or another protocol. Another challenge is that the certificate has to be automatically received within a short period of time. It is not clear which company can provide these demands, therefore the licensed certificates are moved to future work. Currently, for the self-signing CA, there is a script implemented over SSH, this is earlier explained in Section 5.5.6.1. The script can be activated with an option for using a self-signing CA as specified in 5.4. The certificate request and receive are now automated and received within a second.

6.5 Access control database evaluation

The processes concerning accessing the database are evaluated. The database contains sensitive information about the users, such as their authentication information. For the testing of the database, a MariaDB cluster is set up. To protect the database there is permission management. The database has three administrators. One
administrator has all rights concerning the database. A second administrator has the right to read the database. The third administrator is able to add users to the database. The fourth administrator is able to delete users from the database. The second administrator that can read the database is used to run Spark applications. The Spark application requires to authenticate users and thus to read the database. The third administrator uses the script in Section 5.5.6.4 to create users. The creation of a user needs to be separately tested. This process is independent from Spark since it has nothing to do with the Spark process. The script returns a 16 digit code back to generate a six digit authentication pin by using the TOTP algorithm. The user is added to the database by an administrator that has INSERT privileges in MySQL. This administrator is created by the administrator who has all rights. Earlier in Section 5.5.6.4 there is more information about the privileges.

The password or the six digit pin should not be easily guessable by a hacker. An easy brute-force attack on the password or six digits should also be prevented. Before entering the six digits, it has to be verified that the user exists, otherwise a general error is sent that the user or password does not exist. Kerberos is vulnerable for a password guessing attack as shown in Section 2.2.3. To prevent an attack on the password it should not be allowed to do many false attempts, after 10 attempts the account should temporarily be locked and the user should be warned. The IP range where this attempt is coming from could be blocked if it is found to be an attacker. After the password and username are entered correctly, the six digits are requested. Only when the user is authenticated, the certificate is requested. Otherwise, it can never connect with the script to the certificate authority. In this example, the database and the certificate are located on the same instance, in the configuration of the Spark master and workers, the IP addresses are set.

An important caveat is that the database is a single point of attack. Kerberos has a single point of attack and a single point of failure. A single point of attack means that there is a single point where the system is vulnerable. A single point of failure is a part of a system that can fail and will then stop the entire system from working. The database does not have a single point of failure since it can be replicated safely. Further, a backup can be made of the database after a certain time period. In Kerberos a backup can be made as well. The distributed database precludes the single point of failure. In the database, the single point of attack is the password of the administrator who has all privileges or the password of a administrator who can delete users, the administrator who can add users. These three administrators are important because they expose the integrity of the database. With the password of the fourth administrator, the database can be destroyed because this administrator can do everything on the database. For this reason, the fourth administrator should only be used for deleting users. Further the passwords of the administrators should be changed regularly, once every halve year, or once a year, to keep the possibility on a password guessing attack small. For the auditing all the requests made to the database are logged with a general query log to catch potential attackers.

6.6 Requirements evaluation

In Chapter 5 “Contribution” requirements are made for this project. The requirements from Section 5.1 are evaluated. The requirements are listed, and there is evaluated whether they are either achieved, possible, or not achieved.

R1 Multi factor authentication is achieved with this implementation. There is a password and a six digit TOTP implemented to authenticate the user. The implementation is described in Section 5.5.2.
There is public key encryption, this is achieved by setting up a TLS connection. This is described in Section 6.2.1 and 6.2.2.2.

To build the infrastructure for a CA in a scalable way is possible. First, the database should be built scalable as is explained in Section 2.5, and implemented in 5.5. Second, multiple CAs are setup. These multiple CAs sign certificates and are located on different servers.

The auditing is achieved with multiple logging systems. The Spark master logs everything that happens on Spark, the separate workers log their work as well. The CA logs the certificates. Each job has its own log. These logging systems allow consistent auditing.

The short valid time is achieved with short-lived certificates created by the CA. The implementation is explained in Section 5.5.1.

In the implementation of this thesis is shown that the separation of authentication and the instances that perform the job is possible. The implementation should be setup in such a way that the database and certificate authority are on a different server compared to the Spark master.

Cloud access management (CAM) is possible. In this implementation, identity and access management (IAM) and multi-factor authentication (MFA) and privileged access management (PA management) are used. The PA management is not implemented for each Spark user. The privilege management is implemented for the management of the database, however, not for the access to data. For this reason, CAM is not completely implemented.

The central access to the IAM is achieved. The IAM system uses a database and a certificate authority. All the users are stored in the database, and the CA keeps track of the certificates that can be revoked to deny access. The implementation of the database is in Section 5.5.2 and the Certificate Authority in Section 5.5.1.

In this implementation fine-grained authorisation for the database is possible, each user can be given different permissions in the database, by using Apache Ranger.

The resource permission means that the user only has permission to use a certain amount of computation resources. This is possible with this implementation, but is not implemented yet.

The consistency and availability of the authentication services is achieved. The database can be replicated consistently. The database can at the same time achieve availability. The certificates are requested at the CA. The CA can be multiplied to achieve higher availability. A management system like Lemur can be used to manage the certificates, together with the information stored in the database and the logs, this achieves consistency.

Encryption of the data at rest is not achieved. Currently, each user has no separate privileges to access folders containing data. In this implementation, there is thought of how to make this possible. For the management of privileges for data access, inspiration can be taken from Apache Sentry [115]. More information about Apache Sentry is given earlier in Section 2.7.1. Apache Sentry manages the access fine grained authorisation and thus access to files in the HDFS.

Encryption during transit is achieved with TLS. TLS uses the keystore and truststore to set up a secure connection. The data transported over the connection cannot be read by other jobs or hackers. The implementation is in Section 5.5.4 and 5.5.5.

The overhead cost to authenticate users is minimised. The user is authenticated with the database and certificate authority. The TLS connection
requires certificate given by the certificate authority to establish a secure connection. The authentication time takes 0.3 seconds this is comparable with a Kerberos authentication \[113\]. The evaluation of the authentication overhead is in Paragraph 6.2.1.1. The limited time required to authenticate a user implies that this requirement is achieved.

**R15** The automated authentication is achieved with a script that requests and receives the certificate automatically from the CA. The script is in Paragraph 5.5.6.1.

**R16** Non-repudiation is achieved with certificates. The certificates are hashed and are therefore protected from changes. By using hashing and a short valid time for the certificates non-repudiation is achieved. The non-repudiation is explained in Section 5.5.1.2.

**R17** When using Spark cluster or a cluster with YARN, multi-tenancy is achieved. The Spark cluster and YARN make sure that the resources are shared. The implementation of multi-tenancy is described in Section 5.3.3.

**R18** There role based access control and the sharing of projects is not achieved. This role based access control is difficult to achieve since a lot of metadata needs to be kept about the user. A possible solution to achieve this is to use Hops, for which this is already implemented.

**R19** Single sign-on is possible for Apache Spark. The certificate can be reused to authenticate across the Spark masters and workers. However, the Hadoop system cannot yet be accessed with the certificates, and thus Kerberos is still used to access the data. Future work is to replace Kerberos in the Hadoop system as well. In BLESS Section 2.8.2 there is a bastion to manage the login and redirect to the instances. When implementing other services, such as Apache Flink and access to the Hadoop data storage, it is beneficial when there is a bastion built which then distributes the certificate.

In this evaluation most of the requirements are implemented. Some requirements are possible to be implemented. They are possible to be implemented, because it mostly depends on how the user implements this solution. This thesis gives some suggestions to the user on how to implement it.

The requirements **R18**, and **R12** are not achieved. These requirements could be achieved with a lot of future work. A new database could be designed to keep metadata about the users its projects and Kafka streams. Hops as described in Section 2.1.6 has such a solution. The encryption of data at rest is difficult, because of two reasons. The first reason is that, to track data information for a specific user it has to be known where the user stores the data. The second reason is that the encrypted data requires fast decryption to provide quick data analyses.
Chapter 7

Discussion

In this discussion, the Risks, consequences, and ethics of running such a project are discussed in Section 7.1. To discuss the ethics and the risks of this project is important to highlight some ethical issues that come up and explain how on ethical issues is acted. After the ethical issues the findings are in Section 7.2, the findings represent the results found during the research. In the Chapter “Introduction” research questions are proposed. The findings and limitations provide an answer to those research questions in Section 7.3. The goal of this research question is to find the current characteristics of the current security and provide new solutions.

7.1 Ethical, economical, and environmental risks and consequences

This research project updates the security of Apache Spark and Apache Flink. When updating security, there are several risks, and these can have consequences. This investigation is important to realise how the author looks to the subject and with what ethical care the author approaches this project. Ethical considerations are important to establish trust, fairness and accountability. This section analyses these risks, consequences and ethics. The analysis is to show that this research is performed safely and tries to improve the security instead of weakening it.

The first risk is that it is possible to introduce new security threats in Apache Spark and Flink. The code should be cross-checked by other people, preventing potential new breaches. The security is therefore improved, or it stays the same. In this project, Fraunhofer SIT verified the contribution to help find any mistakes. There were feedback sessions to discuss the design and new implementation and evaluation. The second risk is that the current security status does not improve. What can be done is a discussion why the improvement does not help the current Apache Spark or Flink solution. Another risk is that a hardware failure occurs on the testing machine, and thus causes economical damage, to minimise this the configuration is carefully setup, and the hardware used in this approach is limited. This research tries to make the authentication better and prevent data breaches. Data breaches might cause economical damage. The research also tries to minimise the authentication overhead cost and take into account the environment.

There is an ethical dilemma about how trustworthy the computer is that provides the authentication or data security. It might be that the system gives us wrong information back, such as it can be the case with the golden ticket seen in Section 2.2. Such an ethical dilemma comes down to how much people trust the computer. This research assumes that most people believe that the current security measurements taken in Apache Spark and Flink are trustworthy. Some security mechanisms are reused, for example, the TLS mechanism earlier discussed in Section 2.7.3. This security mechanism might have a security leak now or in the future. The assumption of trust in the computer programs and their security is necessary since people rely daily on systems of which they are not 100% sure know how safe it is. When designing
a new computer program, the property rights of other software used should be taken into account [116], if not it can lead to potential lawsuits. This project uses only open source third party software which licensed in such a way that the new solution can have a commercial license. A commercial license is beneficial when selling the solution to other companies.

7.2 Findings

This section presents the “Findings” of the research. The findings discovered in the analysis are in Section 7.2.1, the findings for the contribution are discussed in Section 7.2.2. The findings of the evaluation are discussed in Section 7.2.3. At the end of this section limitations are presented. The limitations are important to give the constraints on this research. The findings and limitations make it possible to answer the research questions.

7.2.1 State of the Art and analyses

In the “Analyses” Chapter 4 and some of the findings in Chapter 2 are made. For the data analytic tools, the most important components are TLS for security and Kerberos for authentication. Authentication is confirming that the user is really who the user presents to be by another entity. transport layer security (TLS) uses public key encryption, TLS prevents eavesdropping from other users on the network. To encrypt the data going over the network a consequence is extra security overhead cost. The authentication used also has some overhead costs and other limitations. For example, Kerberos uses symmetric encryption, which is weaker than public encryption. In the Chapter “Analyses” is discussed to replace Kerberos with TLS and on demand certificates. The Hadoop environment is complex, in the Hadoop environment users have problems setting up Kerberos [23, 22]. The Hadoop environment consists of many components, for processing of data, authentication, and security. In data analytic tools the data to be analysed is stored and secured by for example the Hadoop distributed file system (HDFS). The data analytic tools can communicate with HDFS storage systems by communicating via data nodes.

This research identified several security layers for Hadoop environment. These security layers have characteristics. There is an operating layer to isolate processes from each other and set file permissions. An authorisation layer is there to authorise users. Apache Ranger can set different usage policies and fine-grained control of data access. For the authentication of services and users, Kerberos is responsible. Kerberos authenticates a user, making it possible for a user to access different services in the Hadoop environment. For the authentication, several messages come in over different protocols, such as RPC, HTTP, a block access protocol, and there are third parties that send messages. An overview is given earlier in Figure 2.9. The data analytic tools can communicate with HDFS storage systems by communicating via data nodes. In Spark is a security manager with a keyfactory that makes sure that the TLS connection between the workers and Spark is setup correctly. The keystore has a certificate given by the certificate authority (CA), and the truststore contains the public certificate of the CA. The keystore and truststore set up the TLS connection. The certificates inside the stores form a chain of trust. Kerberos does not cover all use cases for authentication, such as multi-factor authentication (MFA). Businesses demand MFA for authentication. The authentication should be more fine-grained than it currently is. The user can use Kerberos by giving its username and password. Kerberos uses a complex protocol to set up MFA, over RADIUS or LDAP [52].

77
BLESS provided new inspiration for authentication. The authentication happens via a third party that provides certificates. The certificates are valid for a short time to improve security. The authentication happens in this mechanism over the internet and using Amazon. It allows for using MFA by using pluggable authentication modules (PAM). Further, it does careful management of which user has access to what services and data. BLESS uses certificates that can be used to authenticate at different services. The authentication is in the same way as Kerberos uses its tokens to authenticate the user at services. BLESS watches the computation usage as well via CloudWatch. A new inspiration is to have the possibility to set resource usage for each user, such as how much CPU the user is allowed to use. To manage the users BLESS uses a database. A database can keep easier track of which user is authorised to access a job compared to Kerberos. The reason for this is that in Kerberos a user can have many tokens. Kerberos does not keep track of the old tokens, and so this data is lost. A database could keep track of which user can access the job and thus possibly further allow for long-running streaming jobs.

The short lived certificates work differently compared to Kerberos. In Kerberos, the chance exists on a so-called golden ticket attack [50]. This ticket allows to impersonate anybody and for a time of up to 10 years. The usage of on demand certificates prevents the golden ticket attack since on demand certificates are valid for a short amount of time. Certificates could be given for as short as 10 minutes, whereas in Kerberos the token the default valid time default is 10 hours and many users tend not to change this setting to a shorter time [87]. So even if an attacker obtains access, the attacker will only gain the access for a short amount of time. The golden ticket attack is possible because in Kerberos a ticket can be forged [50]. The certificates are build to have non-repudiation and are being hashed. The certificates are signed by a public instance making the certificates harder to forge compared to Kerberos. On demand certificates can be easily managed, due to the short revocation time and the easy storage. The certificate can both be used for data transport protection as for authentication. The usage of certificates could allow for cloud access management. Cloud access management (CAM) is the management of identity, privilege and multi-factor authentication.

The security of Spark turned out to be on a higher level compared to the security of Flink. Spark has a bigger community compared to Flink. The other advantage of Spark is that it has more documentation compared to Flink. The streaming in Spark and Flink happens differently. Spark can because of the usage of mini-batches have transparent API. Flink can provide lower latency because of the more efficient use of tuples.

The resource management of sensitive datasets in Hadoop is complex, specialised distributions like Hops come up, they use metadata to store information about the user. A database could store such metadata, to make sharing of datasets possible. The user has to analyse data on servers with different resources. There is becoming a need for real distribution not only for the resources but also the authentication should scale. There should be the possibility to log into multiple accounts at the same time requiring the authentication system to run on multiple servers. In distributed systems, time management is essential. Time management happens mostly with network time protocol (NTP). Time management is important to make it possible that each system agrees on that it is around the same time. By knowing the time the authentication with certificates and Kerberos tokens is limited.
7.2.2 Contribution

This section discusses the findings made in Chapter 5. Firstly, the requirements are identified, here there is a need for new requirements to improve the authentication. The new requirements make that it is beneficial to replace Kerberos. Kerberos does not use public key encryption. Public key cryptography in combination with symmetric cryptography is considered safer than only symmetric. The implementation uses a database and a certificate authority to make this possible. In Kerberos the auditing of the delegation tokens is not happening correctly, this makes that an attack could go unnoticed [50]. Kerberos does have some excellent characteristics, such as centralised access to the user database and single sign-on. MFA is in Kerberos not easily possible. Users of Spark want a user and developer friendly authentication mechanisms. The list of new requirements for the new solution is in Section 5.1.

For the implementation a new design is made, to show the possibility of authenticating with certificates. The three scenarios for authenticating are local, cluster, and YARN explained in Section 5.3. These scenarios achieve the authentication easier compared to Kerberos. The contribution contains a certificate design. Local is used on a single machine and thus does not require TLS to secure a connection. Here the user is authenticated with two factors and the certificate is then obtained. For the cluster the user is first authenticated, then the certificate is requested for each worker. For yet another resource manager (YARN) the certificate can be easily distributed by an extra configuration option to distribute files, this makes that not every worker has to request one for itself. YARN limits the overhead costs by distributing the certificate. The new design introduces several configuration options in Section 5.4, making it possible for the user to authenticate. Here configuring options such as the second authentication factor and the location of the database.

The new implementation requires a database to for the user information. The database stores new information such as the secrets required for authentication. TLS requires certificates to establish the connection. The certificates are used for authentication as well. The CA needs a secure connection with the Spark master to provide certificates. The easiest way to achieve this is over SSH. During the implementation is realised that it is hard to remove a worker for a particular job. When the authentication at a worker fails, the worker is insecure, to make the user aware the user gets a warning. The secure MFA TOTP is implemented in Spark together with a password. TOTP is easy to setup gives little overhead and is freely available. There is a database to keep track of the two factors for authentication.

For a CA it is important that the certificates are signed by a third-party to improve the security. Netflix uses Amazon to sign the certificates. A free third-party certificate authority that provides enough certificates is not found. A self-signed authority is built, this CA is a little less reliable since the authority is self-owned and thus a developer in the company has access to the authority and could manipulate certificates. An advantage of on demand certificates is that they are valid for such a short time that they don’t require much management.

During this implementation, security of data at rest is hard to implement. One of the main reasons is that the user is free to store the data anywhere in Hadoop. The user configures the retrieval and storage of the data inside the job application. Therefore is it difficult know where the user might store its data, making it hard to protect it from unauthorised access of other users. A solution is to use Hops which keeps the location of the data for each user inside metadata. In the implementation, the user can access Spark without authenticating. When using such a project in a company, the authentication should be required. An option can be made where the developer switches a boolean so that the authentication is always required. When
not enough licensed certificates can be ordered, and the transport security should have a licensed certificate, then a second keystore should be used. It might be that companies require having a licensed certificate for the data transportation. The certificate requests and retrieval are over SSH. If SSH is not possible the company should implement another technique to request and receive certificates.

7.2.3 Evaluation

In the evaluation, several findings are made, in the setup, the test cases, the deployment, the evaluation of important modules and the evaluation of the requirement. This section highlights the most important findings. For the setup, the finding is that most of the Spark environment is written in Scala. The solution is thus mainly written in Scala as well. The implementation uses freely available open source software that can be commercialised. In the research first, the software and hardware are tested to make sure the setup works before any made adjustments. After, the new implementation is tested. During the testing, the logging of Spark is turned on. Spark is in this test running inside a Linux container with a limited number of cores CPU cores available. Spark it tries to consume all cores when not setting the number of cores. The number of CPU cores assigned to the container is not observed by Spark and has to be set to prevent the Spark application from crashing the server.

For local jobs, cluster and YARN the implementation can be tested. Each with assertion testing and the testing via a terminal. The assertion testing can happen through suites in which test cases can be created. Assertion testing is used to test different functions designed to make the new authentication possible. For example, if the authentication of a user works. The testing of the different scenarios is successful, making the new authentication technique promising. An assertion test for the password of the user, and the secret TOTP verifies that the authentication works, the password and secret are validated with a database. Another test requests the certificate to validate if the CA works. With the assertion tests completed and the findings that the authentication works, tests from the terminal are performed. The Spark application can run with the new authentication. In the cluster and YARN mode, a secure connection is established over TLS. The results when running an application in either local, cluster or YARN are obtained.

During the evaluation, the continuous development, building and deployment are important. Spark is developing at a rapid pace, and this contribution can be implemented in new versions. The code can compile to production code. To make distribution possible the production the new user should share the production version.

Two components are necessary for running the new authentication mechanism, The CA and the database. The CA is tested successfully with the SSH connection setup. In this research, there has not implemented a different way to obtain the certificates from the authority. The findings are that there are other ways to connect, for example using HTTPS. Since there is no information about how the certificate is exchanged between a third-party Certificate Authority that provides third-party certificates. These licensed certificates might also be limited to such an amount that they can’t be used for Authentication. A licensed certificate authority called Let’s Encrypt could only provide 20 certificates a week for free. There is an authentication test with the database. The access rights of the administrators to the database is limited. Limiting the access rights reduces the opportunity of attack. To achieve this the privileges of the administrators have to be set correctly. A user can be authenticated in 0.3 seconds. In Kerberos, the authentication also happens within seconds \[113\]. However, consistently scaling Kerberos is challenging because of
the replication technique and the KDC, that follows a master slave architecture. Kerberos. Therefore, is a single point of failure [23]. It does not have a CA or a distributed database that can be easily replicated such as the new proposed solution has.

The new contribution meets most of the requirements. Some requirements depend on how the user implements this new authentication mechanism. For example, making the authentication happen on a different service compared to the service that executes the Spark job. Two requirements for encrypting datasets when in the database, and making it possible that only certain users have access to it are not achieved. These depend on how the user stores the information when running a Spark application.

7.2.4 Limitations
In the research, the single point of failure turned out to be very important. The single point of failure for Spark is not yet discussed. Spark can use Zookeeper to recover the master, resolving the issue of Spark being a single point of failure [24]. The scalability of the Spark authentication is discussed. The scalability of Spark is not discussed. The scalability of Spark itself is achieved by using more workers, to make Spark masters scalable multiple masters can be used. When using YARN Hops can be used to achieve more scalability [2.1.6]. The scalability of the CA is also not well discussed. Using more CA’s demands for the management of certificates across the different authorities. There is a tool called Lemur [117] which can achieve this.

7.3 Answers research questions
This section provides the answers to the research questions.

RQ1 What are the characteristics of the current security and authentication mechanisms for data analytic tools in the Hadoop environment?
To answer this question information is used from the Chapter 2 “State of the Art” and Chapter 4 “Analyses”. The findings of these Chapters are discussed earlier in Section 7.2.1. The analyses and State of the Art provide the answer for this research question. The security and authentication of Hadoop rely heavily on TLS and Kerberos. For the security of data analytic tools, essential components are TLS and Kerberos. Kerberos uses symmetric cryptography. TLS uses a combination of symmetric and public cryptography. TLS prevents eavesdropping and authentication requires that a user confirms its identity. To prevent eavesdropping comes at a price with security overhead costs. In data analytic tools the data can be at rest and is secured by for example the HDFS. Data is going secured over the network by using TLS. Several security layers for Hadoop environment are found. These security layers have their characteristics. The operating layer is for the isolation of processes. An authorisation layer is there to authorise users. Apache Ranger can set different usage policies and fine-grained control for data access. In the authentication layer, Kerberos is mainly responsible for the authentication. Kerberos authenticates users at various services using a single sign-on mechanism. The user can use Kerberos by providing its username and password. Kerberos uses RADIUS, a complex protocol to set up authentication with more than one authentication factor such as a password. RADIUS supports a limited number of authentication modules. Kerberos has more limitations, such as legacy risks, the possibility of a password guessing attack and the golden ticket attack. If an attacker can access the Kerberos domain controller (KDC), obscurity can be achieved, the attacker can create new users and even slowly but persistent get the data out of the system. The limitations and the challenging
multi factor authentication show that Kerberos should be careful setup. There is perimeter security to limit the access to the Hadoop cluster, provided mainly by a service called Apache Knox. For the authentication, several messages come in over different protocols, such as RPC, HTTP, a block access protocol, and third parties that some send messages. An overview is given earlier in Figure 2.9. The data analytic tools can communicate with HDFS storage systems by communicating via Data nodes. In the data analytic tools is a security manager with a keyfactory that makes sure that the connection between the workers and Spark is setup correctly. To setup the connection the Spark master uses the truststore and keystore. The keystore has a certificate given by a CA. The truststore contains the public certificate of this CA. With the truststore and keystore connection can be set up. The certificates in the keystore and truststore form a chain of trust. To conclude, the research question can be answered. The characteristics come mainly from Kerberos and TLS. The characteristics include public and symmetric authentication.

RQ2 Can Kerberos be replaced to improve the security and authentication in data analytic tools?

To see if Kerberos can be replaced, research is performed in data analytic tools and authentication mechanisms. A new authentication mechanism found is called BLESS, the explanation of why the mechanism can replace Kerberos is explained with the State of the Art in Section 2.8, and in the analysis in Section 4.3. A single authentication mechanism is nowadays considered as relatively weak. The current movement to the cloud requires for new authentication requirements. An MFA is stronger and better. In Kerberos, MFA is complicated to implement with Radius. A database can be used as a better alternative in which multiple authentication factors can be stored. The database can then be connected with a script, such as the one implemented in Section 5.5.6.3. Such a script implements a Pluggable Authentication Module that can verify the second factor. Databases allow for access with HTTPS. The new database and authentication solution for Spark make it easier for the user to implement OAuth2.0 and other authentication mechanisms. However, Kerberos has some risks, such as legacy Risks. One of the problems for Kerberos is supplying log-in to more users at the same time, for this replication of the Kerberos database is required. In Kerberos, it is hard to contain consistency and availability when replicating it. Replication is required to handle more concurrent authentication for user accounts. A new method for using certificates called BLESS provides a way to improve the authentication. With BLESS it is possible to achieve single sign-on, this one of the main advantages of using Kerberos for authentication. The authentication happens via a third party that provides certificates. The certificates are valid for a short time to improve security. It allows for using two-factor authentication mechanisms by using PAM. Further, it does careful management of which user has privileges for accessing. BLESS uses certificates to authenticate at different SSH services. The authentication happens in the same way as Kerberos uses its tokens to authenticate at different services. To manage the users in BLESS a database is being used. A database can keep easier track of which user is allowed to access a job compared to Kerberos. The reason for this is that in Kerberos a user can have many tokens. Kerberos does not keep track of the old tokens, and so this data is lost. The new mechanism has these abilities by using a distributed database and CA. The biggest weaknesses Kerberos currently has is the relatively long valid time a ticket is used for authentication. Authenticating users and providing certificates valid for a shorter period than a Kerberos ticket prevents a golden ticket attack. A short valid time for the certificate or Kerberos token requires NTP to synchronise time across machines. The data analytic tools use Kerberos currently only for authentication. To conclude with an answer, Kerberos can be replaced. The security and in special the authentication
can be improved for data analytic tools.

**RQ3** Does the new authentication mechanism implemented in Spark fulfil the authentication requirements?

In the last answer RQ2 there is discovered that the authentication can be improved. The research of replacing Kerberos focuses on the data analytic tools since they are gaining huge popularity in the recent years. In this part, the answer to question RQ3 is given. First, the most important new requirements are given. The information is provided about the implementation, and there is short evaluated if the requirements are fulfilled.

To improve the data analytic tools new requirements are made. The requirements could be made after the analyses in Section 5.1. For the new implementation 18 requirements are proposed. The important requirements are that a way to abuse the tokens for authentication is prevented. The time to use a token should be limited. Further, there is a demand for public key encryption with certificates. On the internet is a rise of the usage of certificates to secure the internet connections. To transport data TLS is a common protocol, and widely used in new solutions, recently implemented in Flink, and active for a longer time in Spark. To prevent brute force attacks on passwords and usernames companies demand MFA.

The research implements the solution to verify if the requirements can be fulfilled. The solution implements a password and TOTP for authentication, herewith providing multi-factor authentication. A database stores the password and TOTP. TOTP is a time based six digit secret. A successful authentication generates a request for a time-based certificate at the Certificate Authority. The signed certificate starts the job in the data analytic tool. The authentication process and the job executed by Spark is logged. In this way auditing is achieved and attacks happening on the services can be discovered. The analysis of big data requires the overhead for security to be minimised because petabytes of data have to be analysed. To encrypt the data TLS is used, for the authentication there is not much extra overhead caused by the new certificates, obtaining such a certificate can happen within 0.3 seconds. In Kerberos, the authentication also happens within seconds [113]. Authenticating 100 users takes on average 0.7 seconds with the new solution. These certificates can be used to set up the TLS connection.
Chapter 8

Conclusion and Future Work

8.1 Conclusion

Companies analyse large growing amounts of sensitive data. To understand the data requires multiple machines to do the analysis. The data is being analysed using for example frameworks like Hadoop and data analytic tools such as Apache Spark and Flink [10][13][38]. In the introduction, it turned out that three most important aspects of choosing a particular data analytic tool are security, performance, and reliability [18]. Tests on Spark and Flink are focused on performance and reliability [15][16]. However, security and moreover authentication is not yet sufficiently considered [17]. This thesis develops a new authentication mechanism that improves the access control for Apache Spark.

The original Hadoop environment uses Kerberos for authentication and transport layer security (TLS) for the security of data transportation. The State of the Art reveals that Kerberos has several weaknesses [23]. Kerberos uses passwords vulnerable to brute force attacks [51]. The configuration in Kerberos is challenging to setup [48][23]. For example, it’s challenging to set up a more sophisticated multi factor authentication (MFA) [52]. The ticket used to authenticate users has several times been exposed to manipulation, rendering Kerberos useless [50][53]. To resolve the problems with Kerberos and other authentication mechanisms Netflix developed a new certificate authentication tool for their SSH services [7]. To comply with the new cloud authentication requirements the tool allows thousands of developers to access at the same time. Consequently, this thesis proposes a concept to resign Kerberos and investigate the use case of certificates in TLS for both authentication and secure data transportation in big data analytic tools.

The in this thesis proposed concept for using TLS as an authentication mechanism demands new requirements. The new authentication has to be scalable. It requires MFA to make it harder for intruders to access the system. Further, the authentication has to be understandable to setup in order to prevent security leaks. The implementation provides MFA and uses certificates together with a user database. The certificates generated by a certificate authority (CA) are short lived and thereby harder to manipulate compared to Kerberos tickets. There can be multiple CAs and a distributed database to provide scalability.

The concept of this thesis implemented in Apache Spark uses a username together with a password and time-based authentication. In the evaluation, performance test showed that authentication of a user happens in 0.4 seconds. In Kerberos, the authentication also happens within seconds [113]. However, Kerberos has difficulties with scalability MFA. In Section 6.6 there is an evaluation of which requirements are achieved by the new solution. The new authentication implemented in Apache Spark satisfies new requirements for cloud authentication, providing scalability and multi factor authentication.
8.2 Future Work

For the future work, four recommendations are given.

1. The replacement of the authentication in Apache Flink. Apache Flink is the second favourite big data analytic tool. In Section 4.4 of the analysis is realised that Apache Flink is a little less popular than Apache Spark, and the documentation is less thorough. Apache Flink would benefit from an improved security and is a promising data analytic tool for the future. Future research can reveal if this implementation is suitable for Apache Flink.

2. The distribution of certificates over YARN is still facing some challenges. Workers are added and dropped dynamically when dynamic allocation is turned on. These workers require certificates in order to setup TLS. Currently this works for the valid time of the certificate. With an invalid certificate it is not possible, so the certificates should be renewed once in a while in order to support long running jobs with YARN. A promising solution is proposed in Section 5.5.5.1, the feasibility requires more research with a proof of concept.

3. In Section 6.4 the licensed certificates are evaluated. To automate the retrieval of licensed certificates to authenticate and setup a connection is challenging. A company supplying the licensed certificates with a certificate authority needs to have a possibility to receive the request, and send the certificate within 30 seconds. Otherwise, the certificate request takes to long, and thus the user has to wait to start the job. Further, it has to become clear over which protocol can be communicated. This script is not yet implemented because it is not clear which company can provide such a solution, and more information is required to overcome these obstacles.

4. Throughout the Hadoop environment authentication happens with Kerberos. The concept proposed in this thesis replaces Kerberos successfully for Apache Spark. However, the replacement of Kerberos is not achieved across all Hadoop services. Some services still require Kerberos tokens for authentication. In Section 2.7.2 there is observed that Hadoop HDFS heavily relies on Kerberos for its authentication. These tokens can be replaced such as in Hadoop open Platform-as-a-Service (Hops), this creates a new Hadoop distribution. Building new Hadoop distribution destroys the working of other services and the users have to decide if they want to join this distribution. It was out of scope for this project to see if this problem should be solved. Replacing Kerberos across the environment is demanded and has a high chance of solving current authentication challenges in the Hadoop environment. As a future work, there could be researched if Kerberos could be replaced at other services. In order to replace Kerberos across Hadoop a front end bastion such as in Netflix Bless is the case would be beneficial. This bastion is shown in Section 2.8.2. The authentication of the user moves more to the front and the certificates would be sent to the Spark master by the bastion. Such a bastion is also already the case in Hops, explained in Section 2.1.6.
Bibliography


