AN EFFECTIVE AND EFFICIENT AUTHENTICATION FRAMEWORK FOR MAPREDUCE IN A MULTIPLE PUBLIC CLOUD ENVIRONMENT

A THESIS SUBMITTED TO THE UNIVERSITY OF MANCHESTER
FOR THE DEGREE OF DOCTOR OF PHILOSOPHY
IN THE FACULTY OF SCIENCE AND ENGINEERING

2021

Soontorn Sirapaisan

School of Engineering
Department of Computer Science

LIST OF CONTENTS

LIST OF CONTENTS 2
LIST OF TABLES 6
LIST OF FIGURES 8
ABBREVIATIONS 10
DEFINITIONS 12
ABSTRACT 14
DECLARATION 15
COPYRIGHT STATEMENT 16
ACKNOWLEDGEMENTS 17
Chapter 1 Introduction 18
1.1 Research Context 18
1.2 Research Motivation and Challenges 18
1.3 Research Aim and Objectives 21
1.4 Research Question and Hypothesis 22
1.5 Research Methodology 22
1.5.1 Literature Review and Knowledge Gap Identifications 22
1.5.2 Solution Design 22
1.5.3 Security Analysis and Performance Evaluation.. 23
1.5.4 Experiment Implementation.. 23
1.6 Novel Contributions and Publications 23
1.6.1 Novel Contributions.. 23
1.6.2 Publications.. 25
1.7 Thesis Structure 25
Chapter 2 Big Data Computing: Issues, Challenges, and Solutions 27
2.1 Chapter Introduction 27
2.2 Big Data Computing: Concepts and System Models 27
2.2.1 Big Data Computing and Collaborative Data Analysis 27
2.2.2 System Models.. 28
2.2.3 Cloud Computing 31
2.3 Issues and Challenges 34
2.4 Existing Authentication Solutions and Knowledge Gaps 35
2.4.1 Entity Authentication.. 35
2.4.2 Data Authentication.. 36
2.4.3 Knowledge Gaps.. 37
2.5 A Way Forward 38
2.6 Chapter Summary 38
Chapter 3 Cryptographic Building Blocks 39
3.1 Chapter Introduction 39
3.2 Selections and Justifications 39

3.3 Hash Functions 40
3.4 Hash Trees 41
3.5 Key Derivation Functions 41
3.6 Symmetric-key based Encryption Schemes 41
3.7 Asymmetric-key based Encryption Schemes 42
3.8 Message Authentication Code (MAC) Schemes 42
3.9 Digital Signature Schemes 42
3.10 Chapter Summary 43
Chapter 4 Multi-domain Decentralised Authentication (MDA) Framework 44
4.1 Chapter Introduction 44
4.2 Use Case Description 44
4.3 Generic Use Case Model Construction 46
4.3.1 Choosing a System Architecture 46
4.3.2 Choosing a Big Data Processing Model 51
4.3.3 MapReduce (MR) based Big Data Processing Model 54
4.3.4 Our Collaborative Big Data Computation on a Multiple Public Cloud platform (CBDC-MPC) Model 56
4.4 Threat Analysis 65
4.4.1 Threats and Attacks 65
4.4.2 Threat Model 66
4.5 Requirement Specifications 67
4.5.1 Functional Requirements 67
4.5.2 Security Requirements 67
4.5.3 Performance Requirements 68
4.6 The Running Example 68
4.7 An Overview of the MDA Framework 73
4.8 Chapter Summary 75
Chapter 5 Multi-factor Interaction based Entity Authentication (MIEA) Framework 77
5.1 Chapter Introduction 77
5.2 Existing Entity Authentication Solutions 77
5.2.1 Non-MR Specific Solutions 78
5.2.2 MR Specific Solutions 80
5.2.3 What is Missing 82
5.3 High-level Ideas 83
5.4 Design Assumptions and Notations 87
5.4.1 Design Assumptions 87
5.4.2 Notations 87
5.5 MIEA in Detail 88
5.5.1 An Overview of the MIEA Architecture 88
5.5.2 Credentials 89
5.5.3 Credential Establishment Methods 93
5.5.4 Entity Authentication Protocols 94

5.5.5 Putting Everything Together: MIEA in Action 105
5.6 The Running Example 107
5.7 Security Analysis 110
5.7.1 Informal Analysis 110
5.7.2 Symbolic Analysis 113
5.7.3 Complexity Analysis 118
5.8 Performance Evaluation 120
5.8.1 Notations 120
5.8.2 Computational Overheads 120
5.8.3 Communication Overheads 124
5.9 Experimental Evaluation 127
5.9.1 Methodology and Evaluation Metrics 127
5.9.2 Testbed Setup 127
5.9.3 Parameters and Configurations 129
5.9.4 Experimental Results 129
5.10 Chapter Summary 134
Chapter 6 Communication Pattern based Data Authentication (CPDA) Framework 135
6.1 Chapter Introduction 135
6.2 Existing Data Authentication Solutions 135
6.2.1 Non-MR Specific Solutions 135
6.2.2 MR Specific Solutions 137
6.2.3 What is Missing 139
6.3 High-level Ideas 140
6.3.1 TreeAgg Method 141
6.3.2 FlatAgg Method 144
6.3.3 HybridAgg Method 145
6.4 Design Assumptions and Notations 147
6.4.1 Design Assumptions 147
6.4.2 Notations 147
6.5 CPDA in Detail 147
6.5.1 An Overview of the CPDA Architecture 148
6.5.2 AuthData Aggregation Algorithms 149
6.5.3 Protocol Message Structure and Format 149
6.5.4 O2M Functional Block 150
6.5.5 M2M Functional Block 151
6.5.6 M2O Functional Block 153
6.5.7 Putting Everything Together: CPDA in Action 155
6.6 The Running Example 157
6.7 Security Analysis 160
6.7.1 Informal Analysis 160
6.7.2 Complexity Analysis 161
6.8 Performance Evaluation 163

6.8.1 Notations ... 163
6.8.2 Computational Overheads ... 163
6.8.3 Communication Overheads ... 165
6.9 Experimental Evaluation 168
6.9.1 Methodology and Evaluation Metrics ... 168
6.9.2 Testbed Setup ... 168
6.9.3 Parameters and Configurations ... 170
6.9.4 Experimental Results ... 172
6.10 Chapter Summary 179
Chapter 7 The Detailed Operational Steps for the Running Example 180
7.1 Chapter Introduction 180
7.2 Job Execution Flow 180
7.3 MDA in Action 186
7.4 Chapter Summary 203
Chapter 8 Conclusions and Future Work 204
8.1 Contributions 204
8.2 Conclusions 208
8.3 Future Work 209
References 211
Appendix A Symbolic Analysis Source Codes 225
Appendix B The Execution Flows of the Kerberos and NSLPK Protocols 231
B.1 Kerberos 231
B.2 NSLPK 232
Appendix C Algorithms Implementing the Methods of CPDA 235

Word count: 76,959

LIST OF TABLES

2.1: The comparisons of the grid and cloud models. 29
4.1: The comparisons of SC-SA and MC-SA...48
4.2: The comparisons of MR, Dryad, Hyracks, Nephele, and Apache Spark. 52
4.3: Interaction classifications. 64
4.4: The input (security log files) for the running example. 69
4.5: Notations for JobData objects used in the running example. 71
5.1: Related entity authentication solutions. 83
5.2: Notations used in the description of MIEA. 87
5.3: Ticket fields. 91
5.4: An example CStore of a component. 91
5.5: Keys and the respective establishment methods. 93
5.6: The header format of MIEA protocol messages. 96
5.7: Fields and their possible values. 96
5.8: Credential package format. 97
5.9: The comparisons of security properties achieved by the MIEA protocols, the Kerberos protocol, and the NSLPK protocol. 113
5.10: Notations used in the complexity analysis of MIEA. 118
5.11: The security strength of MIEA. 120
5.12: Notations used in performance evaluation of MIEA. 120
5.13: The comparisons of the computational overheads imposed on individual entities by different entity authentication protocols. 123
5.14: The comparisons of the communication overheads introduced by different entity authentication protocols. 126
5.15: Software specifications. 128
5.16: Hardware specifications. 128
6.1: Related data authentication solutions. 140
6.2: Notations used in the description of CPDA. 147
6.3: Values for PRO and MTYPE of ADD and ACK messages. 150
6.4: The comparisons of security properties achieved by CPDA, the MAC based scheme, and the signature based scheme. 161
6.5: Notations used in the complexity analysis of CPDA. 161
6.6: The comparisons of the security strengths of CPDA, the MAC based scheme, and the signature based scheme. 163
6.7: Notations used in performance evaluation of CPDA. 163
6.8: The comparisons of the computational overheads imposed on individual components by different data authentication solutions. 165
6.9: The comparisons of the communication overheads introduced by different data authentication solutions. 167
6.10: Software specifications. 169
6.11: Hardware specifications. 170

7.1: Output produced by the Mappers. 183
7.2: Input used by the Reducers. 184
7.3: Output produced by the Reducers. 185
7.4: Credentials established prior to the execution of the job. 186
7.5: The summary of entities, credentials, and AuthData involved in authentication when MDA is applied to the job execution of the working example. 198

LIST OF FIGURES

1.1: The architecture of an example CBDC-MPC. 19
1.2: Thesis structure. 26
3.1: Cryptographic building blocks used in our entity authentication service. 40
3.2: Cryptographic building blocks used in our data authentication service. 40
4.1: Use case architecture. 45
4.2: An example of SC-SA. 46
4.3: An example of MC-SA. 47
4.4: MR components. 54
4.5: Data flows during an MR job execution. 55
4.6: An overview of the CBDC-MPC architecture. 56
4.7: A high-level view of a generic job execution flow in the CBDC-MPC model. 57
4.8: Interactions among entities. 59
4.9: The job execution flow of the running example. 72
4.10: MDA framework architecture. 74
4.11: Authentication flow using the MDA framework. 75
5.1: An example showing how the ideas used in designing MIEA are applied. 86
5.2: An overview of the MIEA architecture. 88
5.3: The classifications of credentials used in MIEA. 89
5.4: Ticket structure. 90
5.5: The key hierarchy of non-derived keys. 92
5.6: A generic message transaction flow of the three entity authentication protocols for positive authentication. 95
5.7: The format of MIEA protocol messages. 95
5.8: The format of a credential package. 97
5.9: GP2A preliminary verification (R1). 99
5.10: GP2A RP1 verification (I2). 100
5.11: GP2A RP2 verification (R3). 101
5.12: GE2A preliminary verification (R1). 103
5.13: The job execution flow when MIEA is applied. 107
5.14: The key hierarchy in step 2 through to step 4c. 110
5.15: Symbolic analysis of the three MIEA protocols using Scyther. 118
5.16: Testbed architecture for evaluating the entity authentication services. 128
5.17: The comparisons of the execution times of all the cryptographic algorithms on data objects with the sizes of 16 B and 32 B. 130
5.18: The comparisons of the execution times of AES-128 Encryption and Decryption on objects with varying sizes. 131
5.19: The comparisons of the execution times of all the protocols. 132
5.20: The comparisons of the execution times of GP2A, GE2A, and SOA against the size of credential packages. 133
6.1: An AuthData tree for 8 objects. 142

6.2: The Sibling-AuthData for d_3 in an AuthData tree for 8 objects. 143
6.3: AuthData transmitted among components in the O2M pattern. 144
6.4: AuthData transmitted among components in the M2O pattern. 145
6.5: AuthData exchanged among components in the M2M pattern. 146
6.6: An overview of the CPDA architecture. 148
6.7: A generic message transaction flow used in the AuthData-Delivery protocols of CPDA. 149
6.8: The components involved, and the algorithms and the protocol used, in the O2M functional block. 150
6.9: The components involved, and the algorithms and the protocol used, in the M2M functional block. 152
6.10: The components involved, and the algorithms and the protocol, used in the M2O functional block. 154
6.11: The operations of the CPDA framework during the entire course of a job execution. 156
6.12: The software architecture of our testbed. 169
6.13: Network topology and the deployment of MR components on the testbed. 170
6.14: The comparisons of the execution times of SHA-256, HMAC-128, and RSA-3072 on objects with varying sizes. 172
6.15: The comparisons of the execution times of the data authentication algorithms used in the MAC based, signature (SIG) based and CPDA. 175
6.16: The comparisons of the execution times of MR jobs with and without data authentication. 178

ABBREVIATIONS

ACK	Acknowledgement
ADD	AuthData Delivery
AES	Advanced Encryption Standard
API	Application Programming Interface
APT	Advanced Persistent Threat
CDBC	Collaborative Big Data Computation
CDBC-MPC	Collaborative Big Data Computation being executed on a Multiple Public Cloud platform
CPDA	Communication Pattern based Data Authentication
CPU	Central Processing Unit
DFS	Distributed File System
DPS	Distributed Processing System
ECC	Elliptic-Curve Cryptography
GHCN	Global Historical Climatology Network
HDFS	Hadoop Distributed File System
HKDF	HMAC based Key Derivation Function
HMAC	Hash based Message Authentication Code
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IaaS	Infrastructure-as-a-Service
IBE	Identity-Based Encryption
ID	Identifier
IDS	Intrusion Detection System
IP	Internet Protocol
IPS	Intrusion Prevention System
KDF	Key Derivation Function
LAN	Local Area Network
M2M	Many-to-Many
M2O	Many-to-One
MAC	Message Authentication Code
MC-SA	Multi-Cloud System Architecture
MDA	Multi-domain Decentralised Authentication
MIEA	Multi-factor Interaction based Entity Authentication
MPC	Multiple Public Clouds
MR	MapReduce
NCEI	National Centers for Environmental Information
NIST	National Institute of Standards and Technology
O2M	One-to-Many
PaaS	Platform-as-a-Service
RAM	Random Access Memory

RPC	Remote Procedure Call
SaaS	Software-as-a-Service
SASL	Simple Authentication and Security Layer
SC-SA	Single-Cloud System Architecture
SDK	Software Development Kit
SSO	Single Sign-On
TCB	Trusted Compute Base
TCP/IP	Transmission Control Protocol over Internet Protocol
TPM	Trusted Platform Module
TTP	Trusted Third Party
WAN	Wide Area Network

DEFINITIONS

Cloud A cloud is an infrastructure for Internet-based services. It abstracts a pool of resources of physical machines and provides shared computing and storage resources to service consumers. Groups of machines providing shared computing and storage resources are, respectively, referred to as a computing cluster and a storage cluster.

Cloud domain (CloudDomain) A cloud domain is a domain consisting of computing and storage resources hosted in a cloud.

Collaborators Collaborators are a group of organisations that have established collaborations and agreed to share datasets and resources for data analyses or other collaborative purposes.

Consumer A consumer refers to an entity consuming (using) data.

Container A container is a subset of resources of a machine, which provides an environment for running a piece of software (i.e., a software runtime environment).

Data object (object) A data object refers to a unit of JobData (e.g., a file) used, generated, or processed during the execution of a data processing job.

Distributed Computing Service A distributed computing service is a service that a service consumer uses to process data. It uses computing and storage resources hosted in a distributed cluster of machines (e.g., a cloud).

Distributed File System (DFS) cluster A DFS cluster is a cluster of MR components used to store JobData during a job execution.

Domain A domain is a group of entities that belong to a particular association or have a common purpose or function.

Distributed Processing System (DPS) cluster A DPS cluster is a cluster of MR components used to perform data processing tasks of a job execution.

Entity An entity collectively refers to a person (e.g., a user), an association (e.g., an organisation, or a cloud service provider), or a service component (e.g., a Mapper or a Reducer).

Job data (JobData) JobData are data that are used, generated, or processed during an execution of a data processing job.

Job domain (JobDomain) A JobDomain is a domain containing MR components allocated to a particular job execution.

MR cluster An MR cluster is a set of MR components with a particular function, either processing or storing data.

MR component An MR component is a component of an MR service. It is used to perform a particular task (e.g., data processing, data storage, and task scheduling).

MR domain (MRDomain) An MRDomain is a domain containing all the MR components of an MR service.

MR job An MR job is a data proceesing job submitted by a user to an MR service.

MR service An MR service is an MR framework based application service.

Organisation An organisation refers to a group of people that belong to a particular association, such as a government unit, a private enterprise, or a financial institute. It subscribes to an MR service offered by an MR service provider and may share the MR service with users from other organisations.

Organisation domain (OrgDomain) An OrgDomain is a domain consisting of users and MR clients (ClientApps) of an organisation.

Processing cluster A processing cluster is a set of machines in a cloud that provide computing resources.

Processing service A processing service is a service run on a processing cluster of a cloud. It provides computing resources to service consumers.

Producer A producer refers to an entity producing (generating or supplying) data.

Resources CPUs, RAMs, storage, networks, and other resources used in facilitating data processing are collectively referred to as resources.

Service consumer A service consumer is an entity consuming a service.

Service provider A service provider is an entity providing a service.

Storage cluster A storage cluster is a set of machines in a cloud that provide storage resources.

Storage service A storage resource service is a service run on a storage cluster of a cloud. It provides storage resources to service consumers.

User A user is a member of an organisation. The user uses an MR service of his/her organisation.

ABSTRACT

Increasingly, there is a growing trend for inter-organisational collaborative Big Data sharing and analysis. For efficiency reasons, such Big Data analysis is usually carried out by using distributed computing services deployed in public clouds.

Executing Collaborative Big Data Computation (CBDC) in a Multiple Public Cloud (MPC) environment introduces some open issues. One of these issues is how to maximise security protection level with minimum overhead costs. We set to investigate these issues based on the authentication property as authentication is the first line of defence in any computing systems. The investigation has led to the design, prototype, and evaluation of a novel authentication solution that takes into account of the characteristics of the underlying system. To this end, this thesis has made the following contributions.

Firstly, the thesis has formulated a generic use case model for CBDC-MPC. This model captures an extreme form of distributed computation where multiple collaborators jointly perform CBDC on shared datasets using an example distributed computing framework, MapReduce (MR), deployed in an MPC environment. The model is used to gain a thorough understanding of the threats in relation to impersonation, unauthorised access, and alteration to data in the context and guide the design of an effective, efficient, and scalable authentication solution for distributed systems.

Secondly, the thesis has proposed a novel authentication framework for CBDC-MPC. The framework, called the Multi-domain Decentralised Authentication (MDA) framework, consists of two further novel components, the Multi-factor Interaction based Entity Authentication (MIEA) framework and the Communication Pattern based Data Authentication (CPDA) framework. The MIEA framework provides risk-aware entity authentication to every interaction during the entire execution cycle of a data processing job. The framework has been analysed and evaluated both theoretically and experimentally. The analysis and evaluation results demonstrate that MIEA provides a stronger level of entity authentication but with the same level of overhead cost compared with Kerberos, one of the most used entity authentication protocols in a distributed computing environment.

The CPDA framework provides data authenticity and non-repudiation of origin for every data object processed by the underlying system. To maximise the protection level while minimising the overhead cost, a novel idea of communication pattern based aggregations of authentication data (generation and verification operations) and communication is used in conjunction with multiple cryptographic schemes. The theoretical and experimental evaluation results show that the CPDA approach offers the strongest level of data authenticity protection but reduces the overhead cost by up to 67% in comparison with the most related solution that digitally signs every object individually.

The results demonstrate that the idea of tailoring the design of an authentication solution in line with the characteristics of the underlying system brings much benefit in terms of supporting efficient and scalable authentication in a large-scale distributed system.

DECLARATION

No portion of the work referred to in the thesis has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.

COPYRIGHT STATEMENT

ACKNOWLEDGEMENTS

I wholeheartedly thank my supervisor, Dr Ning Zhang, for her kind guidance throughout the journey of my PhD study. This journey would be impossible without her dedication and constant support. She has taught me how to be a good independent researcher by demonstration. Knowledge I have gained and skills I have learned are from her valuable advice. With these knowledge and skills, I am confident that I can take on any assignments in my future career. I really cannot thank her enough.

I would like to express my deepest gratitude to the University of Manchester for giving me an opportunity to pursue my doctoral degree and for granting me a highly prestigious scholarship, the Dean's Doctoral Scholar Award, to support my research. My study would be impossible without the support given by the university.

I also would like to thank my employer, the National Electronics and Computer Technology Center, Thailand, and my work supervisors for allowing me to continue the PhD study and for giving me valuable advice.

I thank my laboratory colleagues for fruitful discussions. These discussions uncovered my mistakes and also inspired me to come up with brilliant ideas for my research. I also thank my wonderful friends. They made me feel welcome and not alone when I stayed in Manchester. I really enjoyed the times we went out, had meals, and laughed together. I will cherish these memories for many years to come.

Lastly, I would like to thank my beloved parents and girlfriend for being there for me when I was happy and when I was down throughout the whole journey of my PhD. Their emotional support is really important to me. They are the greatest supporters of my life.

Chapter 1
Introduction

1.1 Research Context

Big Data computing is an emerging computing approach to systematically analyse and extract valuable information from an ever-increasing amount of data or data that are too complex to be efficiently handled by traditional data processing systems. Big Data computing has been used in many fields, such as healthcare [1], agriculture [2], and environmental sustainability [3]. As part of this, there is an increase in inter-organisational data sharing and Big Data processing in which collaborative organisations jointly performed analysis on shared datasets [4][5]. For efficiency reasons, Collaborative Big Data Computation (CBDC) is usually carried out by using distributed computing services. In many cases, these services are hosted in public clouds [6][7]. In line with a trend towards computing as a utility [8][9], it is assumed that distributed computing services and cloud services are provided by different third-party service providers. Collaborative Big Data Computation being executed on a Multiple Public Cloud (CDBC-MPC) platform introduces a host of security concerns. The involvement of multiple organisations and service providers with varying levels of trust and the use of datasets from multiple sources with varying levels of sensitivity imply that the strongest security protection is required.

Authentication is an essential security service for any computer systems, including collaborative Big Data processing in this context. It provides authenticity protection, and it is a prerequisite for other security services, such as authorisation and accounting. In this work, authentication is considered in two dimensions, entity authentication and data authentication. Entity authentication provides an assurance of the authenticity of an entity identity. It ensures that the entity is whom it claims to be so that only authorised entities can access the system. Data authentication ensures that data used during a data processing job are authentic, i.e., data are produced by authorised entities and have not been tampered with by any other entities. In addition, data authentication in this context should also provide non-repudiation of origin protection, which is a security property to protect against false denial of data generation.

1.2 Research Motivation and Challenges

There have been many examples of distributed data processing, or CBDC-like, applications reported in literature, ranging from weather data analysis [10][11], biological image processing [12], to collaborative spam detection [13] and mission-critical applications, such as cyberthreat analyses [4][5]. Data sharing among multiple organisations and collaborative (multi-domain) data analyses can lead to more discoveries than single-domain data analyses. Using collaborative cyberthreat analysis as an example, performing analysis on shared datasets, or datasets contributed by multiple organisations, can help to detect threats which may, otherwise, be hard to detect by using a single-domain dataset, or a dataset contributed by a single organisation [14]. Data used in collaborative data analyses are from multiple sources and could be accessed by entities from different administrative domains. The data and data processing components may be hosted in different clouds [13][15][16]. For efficiency reasons, the computing services used to process the data could be deployed in a Multiple Public Cloud (MPC) environment [17][18][19]. Owing to the complexity of CBDC-MPC, the involvement of multiple administrative domains (i.e., collaborative organisations, distributed computing service providers, and infrastructure service providers), and inter-domain data transfer and processing, there is a host of security concerns in this environment [20][21][22][23].

To show potential security issues in the context, a motivating example (which will also be used as a running example) has been developed. The example considers a simplified use case of cyberthreat analysis. Collaborative organisations (e.g., government agencies and private enterprises) subscribe to distributed computing services provided by distributed computing service providers. The distributed computing service providers, in turn, deploy their services on infrastructures managed and provided by third-party infrastructure service providers. The collaborative organisations have established an agreement to share their security logs (containing network activity data) and perform periodical analyses (i.e., network activity tracing) on the shared security logs (by using the distributed computing services). They need to detect if there are any machine break-ins or compromises, and, if so, which other machines have been connected to by the compromised machines, say in the last 30 days. The architecture of the example is shown in Figure 1.1. In the figure, the subscripted numbers are the indices for entities that are of the same classes or in the same domain. The superscripted numbers indicate domains in which the entities belong to.

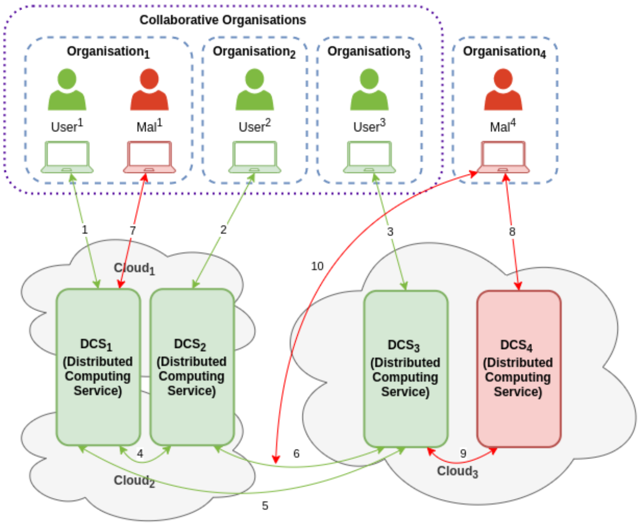

Collaborative Organisations
Organisation$_1$
Organisation$_2$
Organisation$_3$
Organisation$_4$
User^1
Mal^1
User^2
User^3
Mal^4
1
7
2
3
8
10
Cloud$_1$
DCS$_1$ (Distributed Computing Service)
DCS$_2$ (Distributed Computing Service)
DCS$_3$ (Distributed Computing Service)
DCS$_4$ (Distributed Computing Service)
4
6
9
Cloud$_2$
Cloud$_3$
5

Figure 1.1: The architecture of an example CBDC-MPC.

From the figure, it can be seen that the data analysis job in the context is susceptible to threats from multiple sources at multiple points. There are three collaborative organisations, $Organisation_1$, $Organisation_2$, and $Organisation_3$, and one organisation, $Organisation_4$, that does not have any collaboration with the other organisations. $Organisation_1$ has two users, $User^1$ and Mal^1, and subscribes to a distributed computing service, DCS_1, deployed in $Cloud_1$ and $Cloud_2$. $Organisation_2$ has one user, $User^2$, and subscribes to another distributed computing service, DCS_2, also deployed in $Cloud_1$ and

$Cloud_2$. $Organisation_3$ has one user, $User^3$, and subscribes to a distributed computing service, DCS_3, deployed in $Cloud_3$. $Organisation_4$ has one user, Mal^4, and subscribes to a distributed computing service, DCS_4, also deployed in $Cloud_3$. $User^1$, $User^2$, and $User^3$ are allowed to submit and execute collaborative jobs on shared datasets, as indicated by interactions, 1, 2, and 3, respectively. DCS_1, DCS_2, and DCS_3 may exchange data with each other, as indicated by interactions, 4, 5, and 6, respectively. Many of the communications among the components of DCS_1, DCS_2, and DCS_3 (e.g., those indicated by interactions 5 and 6) are of inter-cloud. Mal^1, although not authorised to submit any collaborative jobs, may try to access the data, as indicated by interaction 7. Mal^4 may try to access the data through DCS_4 (which is compromised) as indicated by interactions, 8 and 9, or intercept data-in-transit as indicated by interaction 10. More details regarding the use case and the architecture of the example are explained in Chapter 4 .

Without a proper authentication protection, an unauthorised entity may gain access to the distributed computing services, security logs, and other data generated, processed, and used during the analysis. In addition, if there is no mechanism to hold a participating entity (i.e., an authorised insider) accountable for its actions, the entity may cover up or falsely deny any unintentional or accidental errors that may impact on the data analysis results. An authorised entity (e.g., an unhappy or compromised employee) may even perform malicious actions on the logs such as altering the contents of the logs (e.g., adding, modifying, and deleting log entries). Examples of real-world incidents caused by insider attacks are reported in [24], [25], and [26].

From the perspective of the organisations (as a distributed computing service consumer), violation of data authenticity protection may severely disrupt the analysis. Analysis results that are produced with tampered or incomplete security logs are contaminated. Such contaminated results will lead to wastage of resources and possibly more severe consequences, e.g., some compromised machines may go undetected, which could be used as springboards for launching further attacks.

From the perspective of the distributed computing service providers, such security breaches could tarnish their brands and reputations, leading to the loss of customer loyalty. Existing customers may switch to alternative service providers. The resulting negative publicity may make it harder for the affected service providers to attract new customers. The service providers may have to spend a large sum of money to repair their brands and reputations. They may even be levied heavy fines for failing to comply with data protection regulations (e.g., UK Data Protection Act 2018 [27] and General Data Protection Regulation [28]).

On the other hand, if an authentication service is implemented and applied, an unauthorised entity could not gain access to the distributed computing services and the data used in the analysis, making it more difficult to cause harm to the system and data. Organisations are assured that only authorised entities are allowed to use the distributed computing services and data, and entities can be hold accountable for their actions. Distributed computing service providers can build up their reputation and earn trust of the users (organisations). Hence, organisations need an authentication service to protect their data. Service providers also have incentives to provide such an authentication service to meet the demands of the organisations.

Existing authentication solutions are not specifically designed for distributed Big Data computing services or CDBC in the MPC context. Some of the existing entity authentication solutions [29], [30], [31], [32], [33] only authenticate entities at the gate level, i.e., before the entities are granted with accesses to the services and data. They are designed to thwart threats from outsiders or external entities. They do not address threats from authorised insiders and are vulnerable to session hijacking attacks. Others [34], [35], [36] can support more fine-grained authentication (i.e., interaction-level authentication), but the impact of communication overheads on the authentication performance was not taken as a main design consideration in these solutions. Therefore, their suitability to Big Data computation is not high. With regard to data authentication solutions, there are solutions that are based on cryptographic cryptosystems (symmetric-key based and asymmetric-key based) and other solutions that use a task replication approach. The symmetric-key based solutions [37], [38] cannot protect against insider threats as all the entities in a group would know the same secret, so it is hard to identify who the perpetrator is if there is any tampering with the data. Although the asymmetric-key based solutions [39], [40], [41], [42], [43] can address this issue, i.e., by ensuring data authenticity and originator accountability, they are computationally expensive (the execution time of an asymmetric-key operation could be a thousand times of that of a symmetric-key operation [44][45]). The expensive cost makes this group of solutions unsuited to time-sensitive Big Data applications. The task replication based solutions [46], [47], [48], [49] [50] make use of a task replication approach for output verification. With this approach, a single task is executed by multiple workers, thus multiplying the resources required to execute each task. This could quickly deplete the computation resources and limit the number of data processing jobs that can be run concurrently on the shared resources.

Motivated by these observations, this research aims to investigate how to support cross-domain Big Data sharing and computing more securely and efficiently. Cross-domain computation indicates that security should be provided at the strongest level while the trust on participating entities should be minimal. Big Data computation indicates that security mechanisms provided should be as efficient and as scalable as possible and introduce as less overhead as possible. In other words, in this research, we need to address the following two main challenges:

(CI1) How to provide the strongest authentication protection for inter-organisational Big Data computing using distributed systems deployed on a multi-cloud platform?

(CI2) How to minimise the overhead incurred in achieving such authentication protection and make the rate of increase in overhead in relation to the scale of the distributed computing services as low as possible?

1.3 Research Aim and Objectives

The aim of this research is to investigate how to achieve effective, efficient, and scalable authentication for CBDC-MPC. This aim is supported by the following objectives.

(RO1) To gain a better understanding of the characteristics of CBDC-MPC and how these characteristics correlate to threats and security provisioning in relation to authentication.

(RO2) To investigate how to best support entity authentication in the context in terms of enhancing protection levels while minimising overhead cost.

(RO3) To investigate how to best support data authentication in the context in terms of enhancing protection levels and achieving fine-grained protection while minimising overhead costs.

1.4 Research Question and Hypothesis

The research question that guides this investigation is: how to achieve effective, efficient, and scalable authentication to support multi-domain Big Data processing in the context of CBDC-MPC? To answer this research question, we should answer the following further questions:

(Q1) What are avenues for authentication related threats, or how these threats are mounted, in this environment?

(Q2) How to enhance the protection levels of entity authentication?

(Q3) How to enhance the protection levels of data authentication?

(Q4) How to minimise overhead introduced in providing these authentication services?

The hypothesis of this work is that by taking into account of the characteristics of the underlying distributed computing services, processing and storage infrastructures, and security facilities, we can strengthen the protection level with minimum cost.

1.5 Research Methodology

The research methodology used in this project consists of four components: literature review and knowledge gap identifications, solution design, security analysis and performance evaluation, and experimental implementation.

1.5.1 Literature Review and Knowledge Gap Identifications

The first task carried out in this research was to thoroughly study related work. A survey on trends for distributed computing systems and architectures of commonly used distributed computing frameworks was conducted to investigate their characteristics and how data processing is carried out. We formulated a generic use case model, and based on this model, we analysed authentication related threats and attacks to the system. Based on the identified threats, a set of requirements were specified for an authentication solution. Next, we extensively investigated and critically analysed related existing authentication solutions to identify their strengths and limitations with the aim of building our solution on their strengths but overcoming their limitations. Combined with the characteristics of the underlying systems, the insights gained from related work survey have led us to the design of our effective, efficient, and scalable authentication solution. In addition, we have been regularly reviewing relevant literature throughout this research. The insights gained from the review are used to improve and refine the design of our solution. This task accomplishes the objective (RO1).

1.5.2 Solution Design

The second task carried out in this research was to propose and design an authentication solution for distributed systems in the context of CBDC-MPC. By using the characteristics observed on the generic use case model and the merits of existing authentication solutions, a number of novel ideas and measures were proposed to address the identified knowledge gaps. This has led us to the design of a novel Multi-domain Decentralised Authentication (MDA) framework. It consists of two frameworks, a novel Multi-factor Interaction based Entity Authentication (MIEA) framework and a novel Communication Pattern based Data Authentication (CPDA) framework. We took a modular approach to the designs of these frameworks so that all or part of the frameworks can be applied to other distributed systems in similar contexts. These frameworks have been repeatedly refined and polished by considering new insights gained from regular literature survey and the results of analyses on the frameworks.

1.5.3 Security Analysis and Performance Evaluation

The third task carried out in this research was to analyse the security and evaluate the performance of the MIEA and CPDA frameworks. The security properties of the frameworks were informally analysed against the specified security requirements. In addition, the security properties of MIEA were also formally analysed by using a symbolic analysis method (assisted with a software verification tool). The security strengths of the frameworks were then formally analysed by using complexity analysis. The performances of the frameworks were theoretically evaluated in terms of computational and communication costs introduced. These overhead costs were, respectively, measured as the number of cryptographic operations performed and the number and sizes of protocol messages exchanged.

1.5.4 Experiment Implementation

The performances of the MIEA and CPDA frameworks were further evaluated by experimental evaluations. Two sets of experiments were conducted on real-system testbeds with mock-up and real-world datasets under different parameter settings. For experiment setups, evaluation metrics were defined, evaluation methods were designed, and parameters for cryptographic building blocks were discussed. The components (i.e., authentication methods and protocols) of the MIEA and CPDA frameworks were then implemented by using both C++ and Python programming languages with the Botan cryptographic library. In the first set of experiments, the execution times of MIEA protocols were measured to evaluate the impacts of different parameter settings on MIEA. In the second set of experiments, the times taken to execute data processing jobs when the CPDA framework is applied were measured against sets of parameters. The results were compared with those of the most related entity and data authentication solutions.

Conclusions were drawn from the analyses and evaluations, and recommendations for future work were given. The research findings were documented and published in a high-ranking peer-reviewed journal. The design and evaluation of the MDA framework satisfy the objectives (RO2) and (RO3).

1.6 Novel Contributions and Publications

The research work presented in this thesis has led to the following novel contributions and publications.

1.6.1 Novel Contributions

The main contributions of this work are listed as follows.

(NC1) A generic use case model for CBDC-MPC: The first novel contribution is the formulation of the CBDC-MPC model. CBDC in this model implies that there is a large volume of data that are contributed by more than one organisation and that should be processed in a timely manner. MPC means that the data are processed by a large number of data processing components that are managed or provided by different administrative organisations. CBDC-MPC indicates that security threats in this setting are not only from outsiders but also from authorised insiders and requires that the overhead introduced in protecting the data against these threats should be as low as possible. Owing to the volume of data and the number of processing components involved, a slight increase in the overhead introduced in protecting a single data item (hereafter referred to as a data object) on an individual component could be greatly amplified, the larger the volume of the data and the scale of the service, the larger the amplification effect. The security related processing overhead may cause performance bottlenecks in the system, depleting the benefit of using large-scale distributed components. There are other multi-cloud models that support the storage or transfer of a large volume of data but have a less stringent requirement for timely data processing. Example of such models are distributed data storage models that are based on blockchains [51][52] and data collection models from a large scale (e.g., inter-region or inter-continent) wireless sensor networks or Internet of Things [53][54][55]. Research problems addressed in these existing models are different from ours. For example, the blockchain based model focuses on protecting the integrity of data at rest, whereas our CBDC-MPC model is aimed at protecting data in their entire processing lifecycle, from when the input is being entered into the system to when final computational results are ready to be collected. This lifecycle contains threats to data in-transit and threats to data that are processed by multiple, potentially a large number of, components managed in different administrative domains. Although the security issues in relation to data in-transit are considered in the multi-cloud data collection models for wireless sensor networks and Internet of Things, the design assumptions and requirements for the models are different from those for CBDC-MPC due to the constraint of computation resources and the need for minimising power consumption on data collecting devices. In these models, data usually flow from multiple nodes (devices) to a single sink node and there are minimal computations by, and communications among, the nodes, whereas, in the CBDC-MPC model, individual data processing nodes may execute resource-intensive tasks and interact with many other nodes. The CBDC-MPC model captures the entities involved in the entire cycle of a CBDC job execution and how the entities interconnect and communicate to collaboratively accomplish the execution of a data processing job. This model shows the avenues for authentication related threats and attacks and the characteristics that should be captured in the design of an effective, efficient, and scalable authentication solution for CBDC-MPC. The formulation of the model has laid the groundwork for other contributions made in this thesis. This contribution answers the research question (Q1) and it is fully described in Chapter 4 .

(NC2) A novel approach to entity authentication for CBDC-MPC: The second novel contribution is the proposal and investigation of a novel approach, an interaction based approach, to entity authentication in the context. This approach has been implemented by the design, prototype, and evaluation of a novel entity authentication framework, called the Multi-factor Interaction based Entity Authentication (MIEA) framework. The framework provides entity authentication protection to every interaction taking place during the entire cycle of the execution of a data processing job. The level of protection is adjusted based on the level of risks experienced by the interaction. This contribution answers the research questions (Q2) and (Q4) and it is fully described in Chapter 5 .

(NC3) A novel approach to data authentication for CBDC-MPC: The third novel contribution is the proposal and investigation of a novel approach, a communication pattern based approach, to data authentication in the context. This approach has been implemented by the design, prototype, and evaluation of a novel data authentication framework, called the Communication Pattern based Data Authentication (CPDA) framework. The framework optimises the trade-off between security protection level and computational and communication overhead costs by aggregating the operations of authentication data (data that are used for authentication, e.g., MAC tokens and digital signatures, are collectively referred to as authentication data (AuthData)) generation and verification, and the communications transferring the AuthData, based on the communication patterns between data producers and consumers. The framework provides a data authentication service at the finest granularity level, with the strongest protection level, but with an overhead cost that is lower than the related solutions. This contribution answers the research questions (Q3) and (Q4) and it is fully described in Chapter 6 .

1.6.2 Publications

Parts of the research work presented in this thesis have been reported in the following journals.

- Sirapaisan, S., & Zhang, N. (2021). Multi-factor Interaction Based Entity Authentication (MIEA) Designed for Big Data Processing in a Multiple Public Cloud Environment. (In progress)
- Sirapaisan, S., Zhang, N., & He, Q. (2020). Communication Pattern Based Data Authentication (CPDA) Designed for Big Data Processing in a Multiple Public Cloud Environment. IEEE Access, 8, 107716–107748. https://doi.org/10.1109/ACCESS.2020.3000989

1.7 Thesis Structure

The thesis structure is summarised in Figure 1.2. The remainder of this thesis is structured as follows.

Chapter 2 introduces background for this research in the topics of Big Data computing systems and platforms, and related existing entity and data authentication solutions.

Chapter 3 describes cryptographic building blocks used in the design of our novel authentication solution.

Chapter 4 explains the construction of a generic use case model for CBDC-MPC which is the first novel contribution of this thesis. Based on the model, it gives threat analysis and requirement specifications. It presents an overview of our novel authentication framework, the MDA framework.

Chapter 5 details the design and evaluation of our novel entity authentication framework, the MIEA framework, which is the second novel contribution in this thesis. Parts of this research will be submitted for publication in a journal.

Chapter 6 details the design and evaluation of our novel data authentication framework, the CPDA framework, which is the third novel contribution in this thesis. Parts of this research have been published in a peer-reviewed journal as: "Communication Pattern Based Data Authentication (CPDA) Designed for Big Data Processing in a Multiple Public Cloud Environment" [44].

Chapter 7 gives detailed operational steps for the running example to explain how MapReduce (an example Big Data processing model) executes a job without MDA (our solution) and how it executes the job with MDA.

Chapter 8 concludes this thesis and suggests directions for future work.

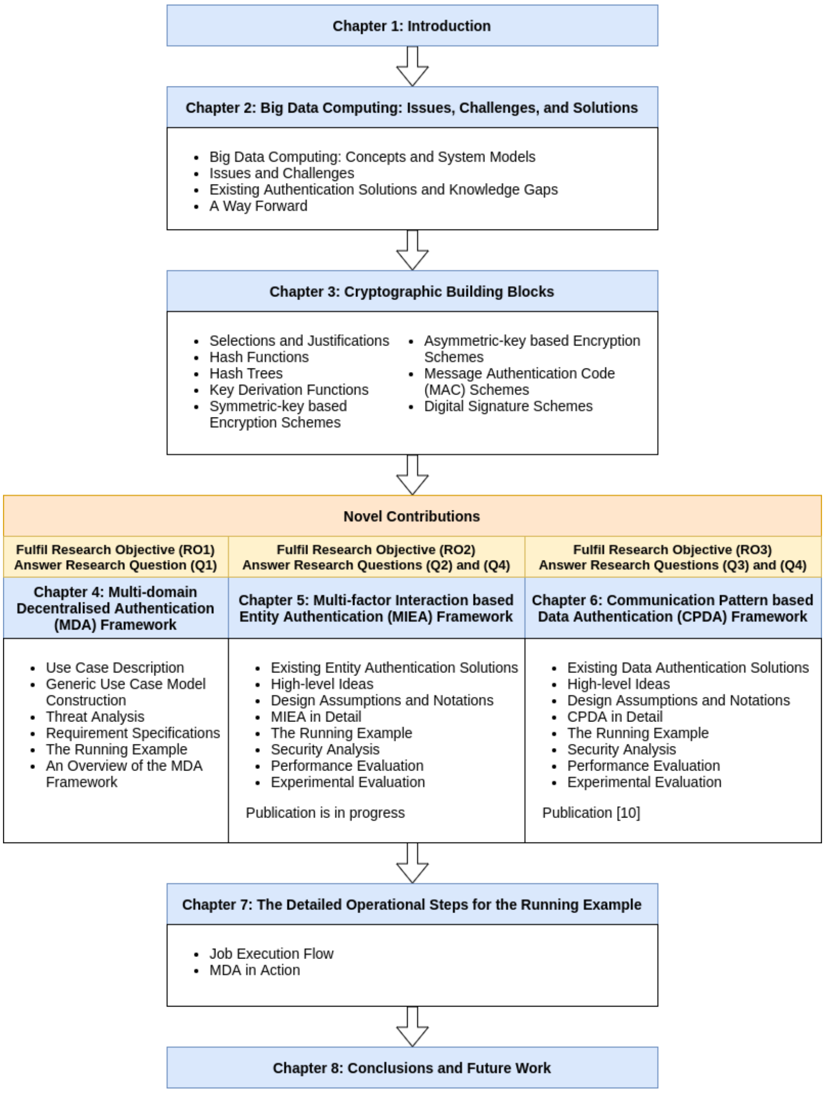

Chapter 1: Introduction
Chapter 2: Big Data Computing: Issues, Challenges, and Solutions
Big Data Computing: Concepts and System Models
Issues and Challenges
Existing Authentication Solutions and Knowledge Gaps
A Way Forward
Chapter 3: Cryptographic Building Blocks
Selections and Justifications
Hash Functions
Hash Trees
Key Derivation Functions
Symmetric-key based Encryption Schemes
Asymmetric-key based Encryption Schemes
Message Authentication Code (MAC) Schemes
Digital Signature Schemes
Novel Contributions
Fulfil Research Objective (RO1) Answer Research Question (Q1)
Fulfil Research Objective (RO2) Answer Research Questions (Q2) and (Q4)
Fulfil Research Objective (RO3) Answer Research Questions (Q3) and (Q4)
Chapter 4: Multi-domain Decentralised Authentication (MDA) Framework
Chapter 5: Multi-factor Interaction based Entity Authentication (MIEA) Framework
Chapter 6: Communication Pattern based Data Authentication (CPDA) Framework
Use Case Description
Generic Use Case Model Construction
Threat Analysis
Requirement Specifications
The Running Example
An Overview of the MDA Framework
Existing Entity Authentication Solutions
High-level Ideas
Design Assumptions and Notations
MIEA in Detail
The Running Example
Security Analysis
Performance Evaluation
Experimental Evaluation
Publication is in progress
Existing Data Authentication Solutions
High-level Ideas
Design Assumptions and Notations
CPDA in Detail
The Running Example
Security Analysis
Performance Evaluation
Experimental Evaluation
Publication [10]
Chapter 7: The Detailed Operational Steps for the Running Example
Job Execution Flow
MDA in Action
Chapter 8: Conclusions and Future Work

Figure 1.2: Thesis structure.

Chapter 2
Big Data Computing: Issues, Challenges, and Solutions

2.1 Chapter Introduction

This chapter introduces the concept of Big Data computing and compares systems and platforms used to carry out Big Data computing. It then discusses security issues and challenges in addressing such issues in the context of CBDC-MPC. It critically reviews existing entity authentication and data authentication solutions with the aim of identifying knowledge gaps and areas for improvement. In addition, this chapter also outlines a way forward to address these knowledge gaps.

In detail, Section 2.2 introduces Big Data computing and systems supporting Big Data computing. Section 2.3 identifies security issues of Big Data computing using distributed computing systems and challenges in addressing the issues. Sections 2.4 gives critical analysis on related existing entity and data authentication solutions and highlight what is missing. Section 2.5 suggests a way forward to address the identified issues. Lastly, Section 2.6 concludes the chapter.

2.2 Big Data Computing: Concepts and System Models

This section introduces the concept of Big Data computing and a trend for inter-organisational data sharing and analysis. It then compares two prominent system models, grid and cloud models, that can support Big Data computing and select one as a reference distributed computing system model.

2.2.1 Big Data Computing and Collaborative Data Analysis

Big Data computing is an emerging computing approach to systematically extract and analyse valuable insights from data that are large in quantity or too complex to be dealt with efficiently by traditional data processing applications [56][57]. Big Data are usually described by three characteristics: (1) high volume, this refers to the quantity of data generated, stored, and processed, the size of Big Data is usually larger than TBs (terabytes) and PBs (petabytes); (2) high velocity, this refers to the speed at which Big Data are generated and processed, Big Data are usually generated and processed at a high speed, e.g., real-time or near real-time; and (3) high variety, this refers to the types of data, Big Data may contain data that are different in types (e.g., sensor data, texts, images, and videos) and could be structured, semi-structured, or unstructured. Typically, Big Data are collected from multiple distributed and heterogenous sources [56], such as social media [58] and sensors [59].

To further diverse the sources of Big Data thus increasing the likelihood of capturing more valuable insights, many organisations have established collaboration for inter-organisational data sharing and performing analysis on the shared datasets. One example of such collaborations is cyberthreat data sharing and analysis [4][5][60][14][61]. As individual organisations usually have to routinely perform analysis on their datasets, collaborative data analysis can reduce these repetitive tasks; they jointly perform analysis on the shared dataset once, and the analysis result can be shared among the organisations. In addition, collaborative data analysis may also help shorten the time needed to gather insights, allowing the organisations to make use of the insights in a timely manner. This is crucial for many applications, particularly those that are time sensitive. Using cyberthreat analysis and detection as an example, collaborative cyberthreat analysis could lead to earlier detection of threats such as Advance Persistent Threats (APTs) which are difficult for individual organisations to detect, allowing faster application of countermeasures or implementation of mitigation plans.

Owing to large quantity and high complexity, traditional computing systems are not suited to handle Big Data processing with a stringent timeliness requirement. Rather, distributed computing systems are frequently used for this task [62]. Comparisons of distributed computing systems supporting Big Data computing are given in Section 2.2.2.

2.2.2 System Models

Generally, a distributed computing system refers to a system that consists of multiple networked autonomous machines, each of the machines has its own processing and storage components, and the machines communicate with each other through networks [63]. These machines collectively form a pool of shared (processing and storage) resources. An application running on a distributed computing system is referred to as a distributed computing service. At a high level, the execution of a data processing job using a distributed computing service is done by dividing the job into multiple smaller tasks and executing these tasks on distributed machines concurrently, the higher the degree of concurrency, the higher efficiency of the job execution.

Distributed computing systems are used in a wide range of applications with different requirements thus models. Two of the most notable models that can support Big Data computing are grid and cloud models [64][65][66]. From users' perspective, grids and clouds have many similarities as they both share the same goal of providing services to the users through a pool of shared resources. They both support multi-tenancy (i.e., multiple users can access a single grid or cloud service concurrently) and multi-tasking (i.e., each user may use multiple application services hosted in the system to perform different tasks). Both grids and clouds can also support many application services. To contrast the two models, we have drawn the following criteria.

(SMC1) Deployment, management, and business opportunity: The adoption of a system model is influenced by how a distributed computing system can be deployed and managed, what applications can be hosted on the system, and what business models are.

(SMC2) Usability: Usability is an important factor for the selection of a system model by users. It is considered in terms of resource provisioning, infrastructure visibility, performance, and scalability.

(SMC3) Security implications: The two system models are designed based on different design assumptions. They may experience different threats and attacks. We should select a model that presents a broader set of security issues so that a security solution designed for the chosen model should also be able to address some or all of the issues faced by the other model.

The comparisons of the grid and cloud models against the specified criteria are summarised in Table 2.1.

Table 2.1: The comparisons of the grid and cloud models.

		Grid	Cloud
(SMC1)	**Deployment**	Usually, a grid is constructed at a large scale, i.e., a regional, national, or a global scale. The infrastructures supporting the system could be provided by a government or collaborative organisations with a particular interest.	There are many options for cloud deployment. For example, a cloud could be deployed for exclusive use by a single organisation or for shared use by multiple organisations. The infrastructures hosting the cloud could be managed by the organisations using the cloud or a third-party cloud service provider.
	Control and management	The infrastructures hosting the system could be provided by a single organisation (e.g., a government) or multiple collaborative organisations. In the latter case, the control and management of the system are shared among the organisations.	A cloud is usually controlled and managed by a single entity, either an organisation (user) or a cloud service provider. It is also possible to share the control and management of clouds among multiple organisations if the cloud is hosted on infrastructures managed by these organisations.
	Supported applications	A grid is commonly used to host computation-intensive applications, particularly large-scale collaborative scientific research projects.	A cloud is designed to support a wide range of online services and applications, both generic and specific purposes. It is particularly suited to applications with dynamic demands for computation resources.
	Business model	Grids are usually formed to support particular projects and the infrastructure hosting the systems could be sponsored by governments, international organisations, or communities. The members of the projects may use the grids free of charge.	If a cloud is provided and managed by a third-party service provider, users may negotiate the price for using the system with the service provider. Currently, many service providers adopt a pay-as-you-go model [67], i.e., users are only charged for the resources they use.
(SMC2)	**Resource provisioning**	Most grids use a batch-scheduling approach for resource provisioning, i.e., a user submits a request for resources for a period of time, the request is queued, and when the resources are available, the resources will be allocated and assigned to the user.	A cloud aggregates all available distributed resources to create a pool of resources. These resources are shared by all the users. Clouds usually respond to request for resources submitted by users in a relatively shorter time.
	Infrastructure visibility	There is no or little abstraction of resources allocated to a user. The user may be able to access the resources of the underlying infrastructure directly.	Resources are isolated and abstracted by using a virtualisation technology. Users can only access the abstracted resources and these resources can be viewed as users' private resources. The underlying infrastructures are not visible to the users.

		Grid	Cloud
	Performance	Grids can support high performance computing through federated resources. In a grid, resource scheduling is relatively light weight, thus, causing less performance loss.	In a cloud, resources allocated for different users are isolated and abstracted by using a virtualisation technology, causing performance penalties. However, the advancement of hardware virtualisation in recent years has been gradually closing the gaps of performance.
	Scalability	Resource scalability is relatively less flexible. Users typically have to negotiate resources with grids in advance.	Clouds support on-demand resource provisioning, i.e., resources allocated to users are adjusted dynamically upon users' request.
(SMC3)	**Trust**	In a grid, there could be multiple logical groups of users (collaborative organisations), each group is called a virtual organisation. A virtual organisation is formed based on a set of resource-sharing rules and conditions. It is assumed that entities of the same virtual organisation are equally trustworthy.	In a cloud provisioned for exclusive use by a group of organisations, users of such a cloud are assumed to be equally trustworthy. However, if a cloud is provisioned for open use by any organisations, users of such cloud are assumed to be not equally trustworthy.
	Security responsibility	Grids adopt a shared-responsibility model, i.e., collaborative organisations are responsible for implementing and managing security protections for different parts of the system. Security responsibility for each organisation is usually well defined.	Clouds also adopt a shared-responsibility model. However, the division of the responsibility is different from that grids. Security responsibility is divided between a cloud service provider and users (organisations) using the cloud. The users should clearly discuss responsibility division with the cloud service provider when negotiate a service contract.
	Security auditing	It is relatively simpler to facilitate security auditing in grids. Individual organisations may use existing mechanisms to implement local security auditing. All the organisations may discuss how to implement global security auditing.	It is relatively harder to facilitate security auditing in clouds. Users of a cloud can only view a virtualised environment given by the cloud. They cannot see what happen outside the virtualised environment. In most cases, users usually are not allowed to perform security audit by themselves on the cloud.
	Networking	A grid is established on shared resources from different administrative domains. Resources hosted within an administrative domain are usually connected via Local Area Networks (LANs), whereas resources hosted in different domains are connected via Wide Area Networks (WANs) or the Internet.	Cloud resources that are hosted within each particular infrastructure are usually connected via LANs or dedicated private networks. Resources that are hosted in different infrastructures could be connected via WANs or the Internet.
	Security model	Grids are built on an assumption that infrastructures hosting grids are from different geographical locations and managed by different administrative domains. As cross-domain data	The design of clouds mainly focuses on the abstraction of a large pool of shared resources and the support of on-demand resource access by users. Security protections are not

		Grid	Cloud
		transfer and computations are vulnerable to various threats and attacks, many security considerations have been taken into the design of grids, e.g., single sign-on authentication, accounting, and auditing.	incorporated into the design of clouds. In many cloud implementations, third-party security services are used to provide desired security protections.

The summary shown in Table 2.1 suggests that the cloud model is a more suitable system model for this research work. The cloud model provides a higher flexibility for system deployment as it offers multiple deployment options each suited to a different group of users. It also supports a wide range of distributed computing applications. This implies that the cloud model may attract a wider group of users. With a cloud model, a user may choose to subscribe to a cloud service provided by, or delegate the management of the cloud to, a third-party cloud service provider. Many cloud service providers also adopt a pay-as-you-go pricing model, i.e., users only pay for what they use. These could be more cost effective than establishing and maintaining the underlying infrastructure by the users themselves. The cloud model gives each of the users a private set of resources via resource isolation and hides the complex structure of the underlying infrastructures from the users. These are beneficial and more favourable to the users as they can pay more attention to the data analysis process without the troubles of handling low-level resources of the infrastructures. Although, grids generally give higher performance than that of clouds, the performance gaps between them are closing as the virtualisation technology advances. In addition, clouds have higher flexibility in scaling resources based on users' demand. This could also be used to help improve the performance of distributed computing services hosted in clouds. The cloud model experiences a higher level of risk and may present a broader set of security issues than those of the grid model. This is because users from different administrative domains which may not be equally trustworthy have certain access to the shared resources provided by the same cloud. Users of clouds do not have any visibility outside the virtualised environment given by the cloud, so they have to trust the cloud service provider to implement necessary security measures. In addition, as security was not one of the main considerations in the design of clouds and clouds are susceptible to a wide range of threats and attacks due to the nature of cross-geological-location resource sharing, there are rooms for improvement to strengthen security protection for clouds. For these reasons, distributed computing systems discussed in this research are based on the cloud model.

In the next section, we further examine the essential characteristics, the deployment models, and the service models of clouds with the intention of identifying a deployment model and a service model that are suited to collaborative Big Data computing in this context. We also explain how the cloud model follows a trend for utility computing.

2.2.3 Cloud Computing

According to the definition of cloud computing given by National Institute of Standards and Technology (NIST) [68], which is one of the most widely accepted definitions for cloud computing, the cloud model consists of five essential characteristics: on-demand self-service, broad network access, resource pooling, rapid elasticity, and measured service.

(CC1) On-demand self-service: Users can request for and acquire resources (e.g., CPU units and size of storage) from a cloud dynamically as needed without contacting with a human operator of the cloud service provider.

(CC2) Broad network access: Users can use network-enabled devices, such as laptops and smartphones, to access services hosted in a cloud via networks.

(CC3) Resource pooling: The resources of a cloud are aggregated from multiple clusters of machines which could be in different regions or countries. These resources are dynamically assigned and reassigned to different users based on the users' demand. Users generally do not have a fine-grained control and knowledge over the exact location of the provided resources.

(CC4) Rapid elasticity: Resources can be elastically provisioned to and released from users automatically. In other words, resources allocated to each user can be scaled up and down to meet the user's requirement at any time.

(CC5) Measured service: Resource usage by each user is monitored, controlled, and reported to both the user and the cloud service provider so that both parties can see how much of each type of resources has been used.

These characteristics correlate to how security protections should be provided to cloud based distributed computing services. This will be discussed in Section 2.3.

The deployment models for clouds can be largely categorised into four models: private cloud, community cloud, public cloud, and hybrid cloud models.

(CD1) Private cloud model: A private cloud is provisioned for exclusive use by a single organisation (which usually have multiple users, i.e., employees of the organisation). It may be owned, managed, and operated by the organisation using the cloud or by a third-party cloud service provider.

(CD2) Community cloud model: A community cloud is provisioned for exclusive use by a group of organisations sharing the same interest (e.g., collaborative organisations). It could be owned, managed, and operated by one or more members of the group or by a third-party cloud service provider.

(CD3) Public cloud model: Unlike private clouds and community clouds, a public cloud is provisioned for open use by any organisations or individual users. It is usually owned, managed, and operated by a cloud service provider.

(CD4) Hybrid cloud model: A hybrid cloud is a composition of two or more distinct cloud models described above. The clouds forming the hybrid cloud are connected by using proprietary or standardised mechanisms.

In comparison with the private cloud, community cloud, and hybrid cloud models, the public cloud model is exposed to a higher level of security risks as a public cloud is open for public use by any organisations and users which are not equally trustworthy. Sensitive data may be processed by, and stored in, the public cloud. Curious or malicious users may attempt to mount attacks on the cloud to gain unauthorised access to the data. Therefore, the public cloud model requires a more stringent security protection. For this reason, the research work presented in this thesis will be based on the public cloud model.

Regarding service models for clouds, service models can be largely categorised into three groups: Software-as-a-Service (SaaS), Platform-as-a-Service (PaaS), and Infrastructure-as-a-Service (IaaS) models.

(CS1) SaaS model: A cloud service provider provides application services to users. The application services are running on a cloud infrastructure. The users may access the applications by using client applications, such as web browsers, running on network-enabled devices. The users do not control or manage the underlying cloud infrastructure (i.e., hardware components such as storage and network appliances, and software components such as operating systems and databases), but they are usually allowed to change limited application-specific configuration settings.

(CS2) PaaS model: A cloud service provider provides software platform services to users. A software platform service consists of software components, such as databases and Software Development Kits (SDKs), which allow the users to build their cloud based applications on the provided platform and deploy the applications in the cloud. The users do not control or manage the underlying cloud infrastructure and the software platform. However, they have control over the applications they developed and are usually allowed to adjust limited configuration settings for the environment hosting their applications.

(CS3) IaaS model: A cloud service provider provides computation resource services to users. A computation resource service provides fundamental computation resources, such as processing and storage, which can be used to build, deploy, and run any software. The users do not control or manage the hardware components of the underlying cloud infrastructure. However, they have full control over software applications that are running on the provided computation resources and they can also request for more (or less) resource provisioning on demand.

In the IaaS model, users have a higher degree of flexibility in terms of full control over software components run on the cloud compared to the SaaS and PaaS models. In exchange for such control, the users have to manage all the software components and apply security measures to protect these software components by themselves. Applications and services running in the cloud are exposed to a higher level of security risk as there are more avenues for attacks and more opportunities for attackers to mount attacks on the applications and services. Based on this observation, IaaS is chosen as the service model for cloud services used in this research work.

With a wide variety in choices of deployment and service models and the characteristic of metered service, clouds have been moving the ways computation services are provided and consumed towards a utility computing model [8][9]. Cloud service providers can offer different types of services, i.e., SaaS, PaaS, IaaS, or a combination of these. A resource sharing approach improves resource utilisation, which is also beneficial to cloud service providers. As different users may need different resources at different times, cloud service providers do not have to excessively over provision resources for the users, resources released from one user can be assigned to another user that requests for more resources. SaaS cloud service providers can also build their services on top of PaaS or IaaS services offered by other cloud service providers. Examples of such cases includes Netflix, Salesforce, and Snap Inc. who build their applications on Amazon Web Service (AWS) offered by Amazon [69]. AWS provides various computation resources to its customers. In addition, users also have freedom to choose any services offered by any cloud service providers in any combination they like.

2.3 Issues and Challenges

Unlike single-domain Big Data computing where an organisation has full control over data and the systems used for computation, the control and management of data and the systems in this inter-organisational setting is much more complex due to the involvement of multiple entities from different administrative domains. Data with varying levels of sensitivity are shared among collaborative organisations, and within each of the organisations, users with different levels of access permission also have access to storage hosting the data. These data could also be transferred between the respective organisations and external infrastructures for the purposes of data processing and storage. The organisations (data owners) do not have full control over data that are stored in infrastructures managed by other entities (data custodians and users) in other administrative domains. Unauthorised access to these data by unauthorised entities could lead to serious consequences, particularly if the data are mission critical. Using cyber threat analysis and attack detections [70][14] as well as medical condition diagnosis [71] as examples, in the first example, unauthorised access to security logs allows an attacker to learn critical information of the system and delete traces of unauthorised access to the systems, making security breaches go undetected and allowing the attacker to proceed with further attacks, causing more harm to the systems. In the second example, unauthorised access to medical data by unrelated personnel not only violates the privacy of patients, but also gives a malicious attacker an opportunity to tamper with the medical data which could lead to misdiagnoses, causing harm to the patients or even loss of life.

Owing to the large size of data used in CBDC (e.g., hundreds of petabytes of particle collision data for scientific research [72], 24 billion triples (a triple is a set of three elements, i.e., a subject, a predicate, and an object) of Semantic Web data for reasoning [73], tens of terabytes of data per day for social network analytics [74][75]), for efficiency reasons, large-scale distributed computing services deployed in a Multiple Public Cloud (MPC) environment are commonly used. Carrying out CBDC in an MPC environment further complicates the issues as data with varying levels of sensitivity are stored in and processed by clouds which can be accessed by entities with varying levels of trust. The datasets to be processed as well as the computing and storage components used may be physically located in different geographical locations and managed in different administrative domains. In such cases, the datasets, the components, and the underlying infrastructures are likely to be connected via WANs or the Internet, which are vulnerable to a wide range of security threats and attacks. The lack of national boundaries and the anonymous nature of the Internet make the prevention and detection of threats and attacks much more difficult, if not impossible. Furthermore, threats imposed by authorised insiders are also a major concern [76][77][78]. Unlike external entities, insiders usually have certain privileges to access data and the systems used to process the data, so they have more opportunities to tamper with the data and systems. In addition, for Big Data processing, the requirements of efficiency and scalability are more stringent. As a distributed system is usually optimised to support concurrent data processing, a slight increase in one or both of computational and communication overheads may significantly deteriorate the performance of the entire system.

To efficiently support secure Big Data computing in the context of CBDC-MPC, the following challenges should be addressed.

(CH1) How to provide security protection with minimum intervention by users and service providers? Clouds are designed to serve users with minimum intervention from human operators. During a data processing job, there will be many interactions between service components and the user submitting the job may not always be present as the job could last for a long time. Therefore, security protection should be provided with minimum user and service provider intervention.

(CH2) How to achieve the strongest entity authentication throughout the execution of a data processing job? Due to the involvement of multiple entities from different administrative domains with varying levels of trust, the strongest level of entity authentication protection is required to ensure that only authorised entities can gain access to data and systems at any time during a data processing job.

(CH3) How to achieve the strongest data authentication at the finest granularity? Data with varying levels of sensitivity are hosted in clouds managed by external cloud service providers and these data could be accessed by entities from different administrative domain. These data should be protected at the object level with the strongest data authentication protection.

(CH4) How to minimise the overhead costs incurred in achieving such protections? In CBDC-MPC, Big Data are implied. Because of a large volume of data are used during a data processing job and a large number of service components are used to process such data, a slight increase in overhead cost (computational and communication) could considerably degrade the performance of system. Hence, the overhead cost incurred in achieving such protection should be kept minimum.

(CH5) How to balance a trade-off between security protection and overhead costs? Usually, the strength of security protection provided comes with overhead cost imposed on the system, the higher the level of security protection, the higher the overhead cost introduced. In this CBDC-MPC context, balancing a trade-off between protection strength and overhead cost is crucial.

2.4 Existing Authentication Solutions and Knowledge Gaps

In line with our aim of investigating how to support secure CBDC on an MPC platform, we have extensively reviewed existing authentication solutions with focus on entity and data authentication. In the following, we give a high-level summary of these authentication solutions and identify knowledge gaps.

2.4.1 Entity Authentication

Most existing entity authentication solutions are designed based on an assumption that entities that are in the same administrative domain or domains that form a collaboration have the same level of trust, but entities external to the domain are not as trustworthy or are untrustworthy. Threats and attacks are mostly caused by entities external to the domain.

Hence, these solutions are designed to prevent those untrustworthy entities from accessing assets hosted in the domain. In other words, they provide only a gate-level protection. Once an entity is authenticated, it can access any assets within the domain to which it is authorised. For intra-domain authentication, many solutions (e.g., [29], [34], [30], [79], and [80]) adopt a centralised authentication approach, i.e., a trustworthy entity or a group of trustworthy entities are designated for issuing credentials (security data used for identity verification) to other entities in the domain and verifying the identities of the other entities. In a solution report in [31], the verification of the identities of two interacting entities can be done without using the central trustworthy entities at a cost of relatively higher computational overhead cost.

For inter-domain authentication, participating domains typically form a federation and entities within a federated domain are also assumed to have the same level of trust. Some solutions (e.g., [29] and [34]) require that all the participating domains have to use the same authentication solution. In these solutions, the trustworthy entities of all the participating domains form a trust hierarchy and the authentication of two entities from different domains is done through the hierarchy. Some solutions (e.g., [81], [33], and [32]) allow the participating domains to use different authentication solutions and the entities from different domains can authenticate themselves with their home domain. This is done by exchanging standardised security data between the trustworthy entities in the home and foreign domains.

The gate-level entity authentication solutions described above have one major limitation. If the credential of an entity is stolen or a live session is hijacked by an attacker, the attacker can impersonate the entity and gain access to the assets hosted in the domain. A number of solutions (e.g., [82] and [83]) have been proposed to address this issue by using hardware or biometric based credentials as additional factors for authentication, making stolen-credential attacks more difficult. However, these solutions are only suited to the authentication of human users. Other solutions (e.g., [35] and [84]) enhance authentication protection by providing authentication at the interaction level. However, they do not provide a fine-grained accountability. This is because they are designed based on an assumption that a group of entities performing the same function are equally trustworthy, thus, the entities of the same group could share and use the same credential. Without additional measures, it is impossible or extremely difficult to distinguish entities sharing the same credential.

2.4.2 Data Authentication

Data authentication solutions can be largely classified into three groups based on the trust assumption applied. The first group of solutions assume that all the entities involved in a data processing job are equally trustworthy, but entities external to the data processing job are untrustworthy and these entities may mount attacks on data used in the job. Hence, the solutions in this group focus on how to protect the authenticity of the data against external entities. The solutions in this group (e.g., [37] and [38]) provide data origin authentication and data integrity protection but not non-repudiation of origin. These solutions are not suitable for CBDC-MPC due to incompatible assumptions, i.e., in the context of CBDC-MPC, entities involved in the job are from different administrative domains with varying levels of trust.

The second group of solutions assume that some of the entities involved in a data processing job are trustworthy, but the remaining entities are untrustworthy. The solutions in this group (e.g., [46], [47], [48], [49], and [50]) mainly focuses on the correctness (thus integrity protection) of data generated during the execution of the job. Such data correctness is ensured by task replication, i.e., each data processing task is assigned to multiple data processing components and some of the tasks are also assigned to trustworthy data processing components. Owing to redundancy, task replication could significantly add computational and communication overhead cost to the job. These solutions are also not suited to collaborative Big Data processing in this context due to incompatible assumptions and high overhead cost imposed on the system.

The third group of the solutions assume minimal trust among entities and entities within the same domain may not be equally trustworthy. Hence, the solutions in this group (e.g., [39], [40], [41], [42], and [43]) are designed to provide data origin authentication, data integrity protection, as well as non-repudiation of origin. Some solutions can provide such strong data authenticity protection at the object level. However, this is achieved at a cost of high computational and communication overhead cost. Therefore, these solutions are also not suited to CBDC-MPC which involves a large quantity of data and has a stringent requirement for timeliness for data processing.

2.4.3 Knowledge Gaps

Based on observations made on the existing entity and data authentication solutions, we have identified the following knowledge gaps.

(KG1) The gate-level entity authentication solutions do not provide protection against insider threats. These solutions mainly focus on securing domain perimeters to deter outsiders from mounting attacks against systems hosted in the domain. They are not designed to counter insider threats which are a major source of concerns in this context of CBDC-MPC.

(KG2) Some entity authentication solutions can provide a certain level of protection against insider threats; however, the protection provided is coarse-grained. They can protect against insider threats caused by different groups within the domain, but not threats caused by entities within the same group. This could cause issues as, in a data processing job, there could be a large number of entities tasked with the same function thus assigned to the same group.

(KG3) Existing entity solutions supporting cross-domain authentication impose some limitations. Some solutions require that all the participating organisations use the same entity authentication solution, which is typically a centralised solution. Using a centralised trusted entity, or a group of trusted entities, to authenticate a large number of distributed entities is not efficient. The other solutions allow the use of different entity authentication solutions among different organisations, but they add another layer of authentication, introducing additional overhead cost thus inefficient.

(KG4) In this CBDC-MPC context, the issue of trade-offs among protection granularity, protection strength, and efficiency has not been yet addressed in the existing data authentication solutions. To provide a fine-grained and strong level of protection (i.e., protecting against both outside and inside threats), each data objects should be individually signed. This will introduce an excessive level of computational overhead, particularly when being applied to a large volume of data. However, alternative existing solutions do not protect against insider threats, although they are efficient.

2.5 A Way Forward

To address the knowledge gaps identified above, thus supporting secure Big Data computation as efficient and scalable as possible in the context of CBDC-MPC, we propose to apply the following ideas:

(W1) To enhance the protection of the system in this inter-organisational setting, the protection should be provided at the finest granularity and should be against both outside and inside threats. In other words, the protection should be applied at the object level, and in addition to authenticity, accountability (in terms of non-repudiation) should also be provided.

(W2) Entity identity and data authenticity should be verified at every interaction, or every point where data change hands. Threats of impersonation, unauthorised data access, unauthorised data modification, and non-repudiation of origin may be realised at any interaction between a pair of entities or any point where there is a data transmission and reception, so applying authentication at every interaction can maximise the strength of protection.

(W3) Protection should be built on any already-established security infrastructure in participating organisations. Participating organisations typically have already got security infrastructures established in their respective domains. By making use of these security infrastructures (such as secure channels for credential distribution) already established, we can avoid duplicating efforts, thus reducing unnecessary overheads required, in establishing such infrastructures, while allowing organisations to streamline their security managements.

(W4) The aggregations of operations for generating and verifying AuthData and communication messages can reduce computational as well as communication cost. The cost incurred in providing strong protection at the finest granularity (as explained in (W1)) could be high as individual components have to generate, verify, and transmit AuthData for multiple data objects. The idea of aggregations, along with minimising the use of computationally expensive cryptographic primitives, may help us to minimise overhead costs incurred in enhancing the protection.

2.6 Chapter Summary

This chapter has presented the concept of Big Data computing and a trend for inter-organisational Big Data computation. It has described and compared two prominent distributed computing system models and then selected one for this research work. Based on the selected system model, it has analysed and identified issues related to authentication and challenges in addressing such issues. It has presented a critical analysis on the related existing authentication solutions from a high-level perspective, highlighting knowledge gaps and areas for improvements. Finally, it has outlined ideas for the design of an authentication solution that can address the identified knowledge gaps to best support secure CBDC-MPC efficiently.

The next chapter presents the cryptographic building blocks used to design our solution.

Chapter 3
Cryptographic Building Blocks

3.1 Chapter Introduction

This chapter introduces cryptographic schemes that are used as building blocks for the designs of our authentication solution. These building blocks provide the required security protections and functions. This chapter describes the building blocks before listing the algorithms of the building blocks and the interfaces (specifying the input and output) of the algorithms.

In detail, Section 3.2 explains the selections of the cryptographic building blocks and gives justifications for such selections. Sections 3.3, 3.4, 3.5, 3.6, 3.7, 3.8, and 3.9, respectively, describe hash functions, hash trees, key derivation functions, symmetric-key based encryption schemes, asymmetric-key based encryption schemes, Message Authentication Code (MAC) schemes, and digital signature schemes. Section 3.10 concludes the chapter.

3.2 Selections and Justifications

In accomplishing entity and data authentication protections, our authentication solution performs a number of tasks, including the generation, verification, and secure distribution of credentials, AuthData, and other secrets used in facilitating the authentication process. For these tasks, the following functions should be fulfilled: (1) to generate a digest for a set of data objects; (2) to generate new keys from a given set of secrets; (3) to protect data confidentiality; (4) and to ensure the authenticity of data objects.

A hash function and a hash tree are, respectively, used to generate a digest for a data object and a set of data objects, respectively. A digest is a fixed-length token that is used to represent the object, or the whole set of the objects. Hence, a hash function and a hash tree can accomplish function (1). A key derivation function is used to derive new keys from a given set of secrets (e.g., a master key and a nonce). It accomplishes function (2). A symmetric-key based encryption scheme is used to obfuscate the content of a data object (and also to reverse the process). It accomplishes function (3). A MAC scheme and a digital signature scheme are used to generate and verify an authentication data token ensuring the authenticity of an object. They accomplish function (4). Unlike MAC, a digital signature scheme also provides protection of non-repudiation of origin to the object. It is worth noting that, although asymmetric-key based encryption schemes are not used in the design of our solution, they are used by an entity authentication solution to be compared with our solution. Therefore, these schemes are also described in this chapter. Diagrams showing how these cryptographic building blocks are used in achieving entity authentication and data authentication are, respectively, depicted in Figure 3.1 and Figure 3.2.

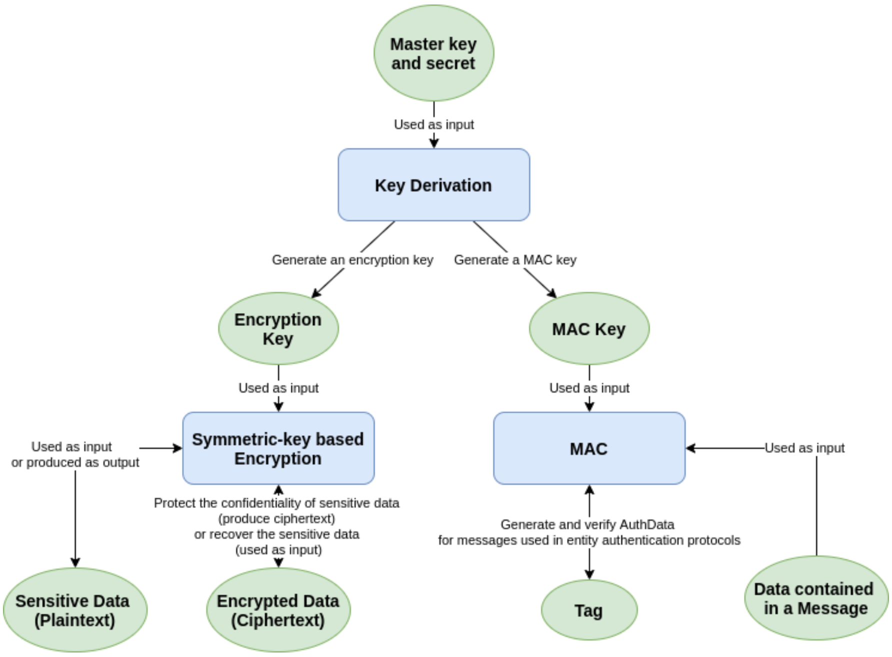

Master key and secret
Used as input
Key Derivation
Generate an encryption key
Generate a MAC key
Encryption Key
MAC Key
Used as input
Used as input
Used as input or produced as output
Symmetric-key based Encryption
MAC
Used as input
Protect the confidentiality of sensitive data (produce ciphertext) or recover the sensitive data (used as input)
Generate and verify AuthData for messages used in entity authentication protocols
Sensitive Data (Plaintext)
Encrypted Data (Ciphertext)
Tag
Data contained in a Message

Figure 3.1: Cryptographic building blocks used in our entity authentication service.

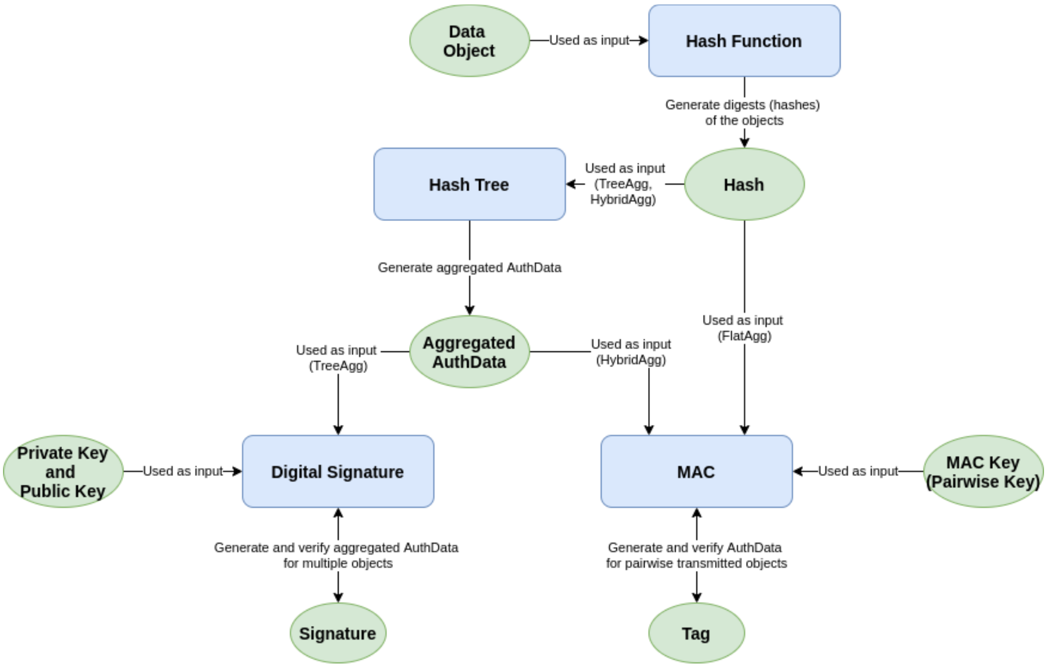

Data Object
Used as input
Hash Function
Generate digests (hashes) of the objects
Hash Tree
Used as input (TreeAgg, HybridAgg)
Hash
Generate aggregated AuthData
Used as input (FlatAgg)
Used as input (TreeAgg)
Aggregated AuthData
Used as input (HybridAgg)
Private Key and Public Key
Used as input
Digital Signature
MAC
Used as input
MAC Key (Pairwise Key)
Generate and verify aggregated AuthData for multiple objects
Generate and verify AuthData for pairwise transmitted objects
Signature
Tag

Figure 3.2: Cryptographic building blocks used in our data authentication service.

Any implementations of these cryptographic building blocks can be used to implement our authentication solution interchangeably as long as they support the defined interfaces.

3.3 Hash Functions

A hash function (also known as a cryptographic hash function) is used to generate a digest of an object, and the digest is called a hash. Some examples are SHA-2 [85], SHA-3 [86], and BLAKE2 [87]. A hash function contains a hash generation algorithm. This algorithm takes a variable-length object d as input and returns a fixed-length hash h as output, denoted as $h = H(d)$.

3.4 Hash Trees

A hash tree, also called a Merkle tree [88][89][90], is a tree containing aggregated hashes for a set of objects. In a hash tree, each leaf node is the hash of a respective object and each internal node is the hash of the concatenation of its child nodes. The root node (also called the root hash) is the aggregated hash of all the objects. Sibling nodes along the path from a given leaf node to the root node are collectively referred to as Sibling-AuthData. The hash tree used in our solution contains three algorithms, Hash Tree Construction (HT-Construction), Sibling-AuthData Extraction (SA-Extraction), and Root-AuthData Recovery (RA-Recovery).

(HTA1) HT-Construction: HT-Construction takes a set of hashes $h_1, h_2, \dots, h_N$ as input and returns a hash tree ht as output, denoted as $ht = HTC(h_1, h_2, \dots, h_N)$.

(HTA2) SA-Extraction: SA-Extraction takes a hash tree ht and a hash h_i (of an object d_i) as input and returns a Sibling-AuthData token containing a set of hashes and their positions (i.e., left or right) along the path from h_i to the root node as output, denoted as $sa_i = SAE(ht, h_i)$. If a binary hash tree is used, the number of the hashes contained in each sa_i is at most the height of the tree, i.e., $\lceil \log N \rceil$.

(HTA3) RA-Recovery: RA-Recovery takes a hash h_i and a Sibling-AuthData token sa_i as input and returns a root hash rh as output, denoted as $rh = RAR(h_i, sa_i)$.

3.5 Key Derivation Functions

A key derivation function is a cryptographic hash function specifically designed for generating symmetric keys from secret values (e.g., a master secret key) and, optionally, other values (e.g., a salt value). An example of commonly used key derivation functions is HMAC (Hash-based Message Authentication Code)-based Key Derivation Function (HKDF) [91]. A key derivation function contains one key derivation algorithm, referred to as Key-Derivation. The algorithm takes a length l (set for the derived key), an input key ik, and a salt value s as input and returns a derived key dk as output, denoted as $dk = HKDF(l,\ ik,\ s)$.

3.6 Symmetric-key based Encryption Schemes

A symmetric-key based encryption scheme is commonly used to protect the confidentiality of data. It provides two functions: encryption and decryption. Encryption is a function that hides the content of data by transforming the data (also called plaintext) into encrypted data (also called ciphertext) with a secret key. Decryption, the reverse function of encryption, transforms ciphertext back to plaintext with the same key. Examples of symmetric-key based encryption schemes include AES [92], Blowfish [93], and RC6 [94]. A scheme used in our design should contain an encryption (Sym-Encryption) algorithm and a decryption (Sym-Decryption) algorithm.

(SEA1) Sym-Encryption: Sym-Encryption is an algorithm that accepts a secret key *k* and a plaintext *d* as input and returns a ciphertext ψ as output, denoted as $\psi = SE(k, d)$.

(SEA2) Sym-Decryption: Sym-Decryption is an algorithm that accepts a secret key *k* and a ciphertext ψ as input and returns a plaintext d as output, denoted as $d = SD(k, \psi)$.

3.7 Asymmetric-key based Encryption Schemes

An asymmetric-key based encryption scheme protects the confidentiality of data by using two asymmetric keys, a public key and a private key of a receiver (an entity receiving the data). The public key is used for encryption and the private key for decryption. In other words, any entities knowing the public key can encrypt data (plaintext) to generate ciphertext, but only the receiver can decrypt the ciphertext to recover the plaintext. Examples of asymmetric-key based encryption schemes include RSA [95] and ElGamal [96]. An asymmetric-key based encryption scheme should contain an encryption (Asym-Encryption) algorithm and a decryption (Asym-Decryption) algorithm.

(AEA1) Asym-Encryption: Asym-Encryption is an algorithm that accepts a public key pk and a plaintext *d* as input and returns a ciphertext ψ as output, denoted as $\psi = AE(pk, d)$.

(AEA2) Asym-Decryption: Asym-Decryption is an algorithm that accepts a private key sk and a ciphertext ψ as input and returns a plaintext d as output, denoted as $d = AD(sk, \psi)$.

3.8 Message Authentication Code (MAC) Schemes

A MAC scheme is a symmetric-key based data authentication scheme. It provides two security properties: data origin authentication and data integrity protection. To protect the authenticity of a data object, a MAC scheme along with a secret key are used to generate an AuthData token, called a tag, for the object. The authenticity of the object can then be verified against the tag by using the same key. Some examples of MAC schemes are HMAC [37], OMAC [38], and UMAC [97]. A MAC scheme should contain two algorithms, signing (MAC-Signing) and verification (MAC-Verification).

(MA1) MAC-Signing: MAC-Signing is an algorithm that accepts a secret key *k* and a data object *d* as input and returns a tag τ as output, denoted as $\tau = MS(k, d)$.

(MA2) MAC-Verification: MAC-Verification is an algorithm that accepts a secret key *k*, an object *d*, and a tag τ as input and returns a verification result mv as output, denoted as $mv = MV(k, d, \tau)$. The output is either positive or negative.

3.9 Digital Signature Schemes

A digital signature scheme is an asymmetric-key based data authentication scheme which provides three security properties: data origin authentication, data integrity protection, and non-repudiation of origin. Unlike MAC, a private key and a public key of a sender (an entity sending a data object) are, respectively, used for signing to generate an AuthData token, called a signature, and for verifying the signature. In this way, only the sender can sign the object whereas any entities knowing the public key of the sender can verify the authenticity of the object. Some examples are RSA [95], DSA [98], and ECDSA [98]. It contains two algorithms, signing (SIG-Signing) and verification (SIG-Verification).

(SA1) SIG-Signing: SIG-Signing is an algorithm that accepts a private key *sk* and an object *d* as input and returns a signature *σ* as output, denoted as $\sigma = SS(sk, d)$.

(SA2) SIG-Verification: SIG-Verification is an algorithm that accepts a public key *pk*, an object *d*, and a signature *σ* as input and returns a verification result sv as output, denoted as $sv = SV(pk, d, \sigma)$. The output is either positive or negative.

3.10 Chapter Summary

This chapter has presented the cryptographic building blocks used in the design of our authentication solution. The next chapter presents the construction of a generic use case model for CBDC-MPC. It also describes the architecture and the components of our novel authentication solution, the Multi-domain Decentralised Authentication (MDA) framework.

Chapter 4
Multi-domain Decentralised Authentication (MDA) Framework

4.1 Chapter Introduction

This chapter presents the formulation of a generic use case model for CBDC-MPC, which forms the foundation for this research work. The CBDC-MPC model has been thoroughly investigated and analysed to gain a better understanding of the characteristics of CBDC-MPC, how these characteristics correlate to threats and attacks, and how they may influence the provision of authentication protection. This chapter also presents the architecture of our novel authentication framework, the Multi-domain Decentralised Authentication (MDA) framework. It gives an overview of the components of the MDA framework and explain an authentication flow when MDA is applied.

In detail, Section 4.2 describes the use case for this work. Section 4.3 examines potential system architectures and Big Data processing models and constructs the CBDC-MPC model based on the chosen architecture and models. Section 4.4 gives a threat analysis based on the CBDC-MPC model. Section 4.5 gives a set of requirements to counter the identified threats. Section 4.6 explains the architecture of the MDA framework. Lastly, Section 4.8 concludes the chapter.

4.2 Use Case Description

This section formulates a generic use case model which is based on the running example discussed in Section 1.2. The use case model is an extreme form of distributed computing in which multiple collaborators from different administrative domains jointly perform a collaborative data analysis on shared datasets using shared resources. The distributed computing and infrastructure services used are provided by external providers. Minimal trust among the organisations is assumed. The use case is chosen based on two main considerations. Firstly, this use case captures the characteristics of Big Data processing using distributed computing systems as described in Section 2.2. Secondly, it presents a broader set of challenges encompassing those presented in other use cases (e.g., single-domain Big Data computation). This means that a security solution design for this use case should also be applicable to the other use cases as well. The architecture of the use case is depicted in Figure 4.1. It is worth noting that the architecture shows only the entities involved in the collaborative job execution.

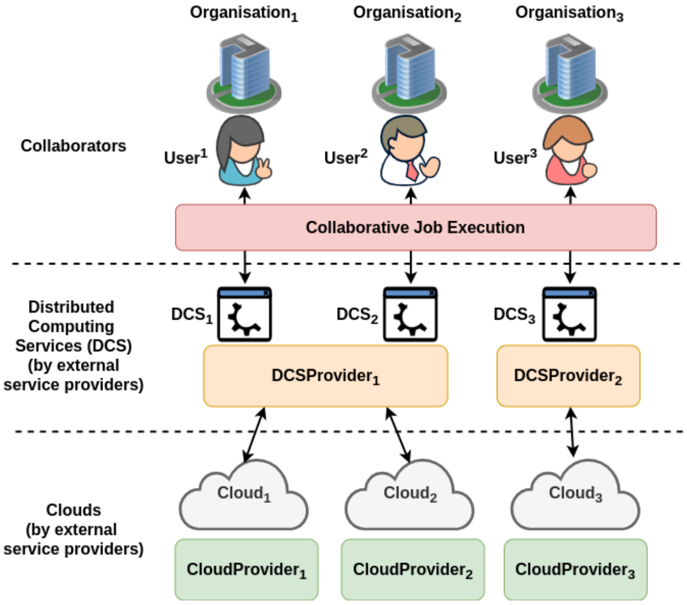


Figure 4.1: Use case architecture.

The figure shows the relations among entities involved in a collaborative job execution. Based on their roles, the entities are classified into three groups, cloud service providers, distributed computing service providers, and collaborators (organisations). Without losing generality, it is assumed that there are three cloud service providers ($CloudProvider_1$, $CloudProvider_2$, and $CloudProvider_3$), two distributed computing service providers ($DCSProvider_1$ and $DCSProvider_2$), and three organisations ($Organisation_1$, $Organisation_2$, and $Organisation_3$). $CloudProvider_1$, $CloudProvider_2$, and $CloudProvider_3$, respectively, manage clouds $Cloud_1$, $Cloud_2$, and $Cloud_3$. Each of the clouds provides one or both of processing and storage services. A set of machines hosting a processing service is referred to as a processing cluster, and a storage service as a storage cluster. $DCSProvider_1$ and $DCSProvider_2$ use processing and storage services provided by cloud service providers to build distributed computing services. $DCSProvider_1$ subscribes to processing and storage services hosted in $Cloud_1$ and $Cloud_2$ (more details with regard to system architectures for cluster deployment are given in Section 4.3.1), whereas $DCSProvider_2$ subscribes to both processing and storage services hosted in $Cloud_3$. $Organization_1$ and $Organization_2$ each subscribe to a dedicated distributed computing service (DCS_1 and DCS_2, respectively) provided by $DCSProvider_1$, whereas $Organisation_3$ subscribes to a distributed computing service (DCS_3) provided by a different provider, $DCSProvider_2$. Users ($User^1$, $User^2$, and $User^3$) can use the distributed computing services subscribed by their respective organisations, may request shared datasets as well as computation resources, and carry out collaborative job executions on behalf of their respective organisations.

Depending on the system architecture and Big Data processing model used, there are multiple ways of carrying out this collaborative data analysis. These will be discussed in the next section.

4.3 Generic Use Case Model Construction

In the following, we examine potential system architectures and Big Data processing models and select ones to construct our CBDC-MPC model. We also report the characteristics observed on the model.

4.3.1 Choosing a System Architecture

This section discusses two different system architectures, a Single-Cloud System Architecture (SC-SA), and a Multi-Cloud System Architecture (MC-SA), for the deployment of distributed computing service components in clouds and selects one for constructing the CBDC-MPC model. The two system architectures are devised based on an observation that, at a high level, a distributed computing service should consist of two clusters of components, processing and storage components, and each of such clusters can be hosted in a different cloud. In the following, we discuss the system architectures for the deployment of DCS_1 and DCS_2 (shown in Figure 4.1).

4.3.1.1 Single-Cloud System Architecture (SC-SA)

In the SC-SA architecture, both processing and storage components of a distributed computing service are, respectively, hosted on processing and storage clusters of the same cloud. An example of SC-SA where DCS_1 and DCS_2 are deployed in a single cloud, $Cloud_1$, is shown in Figure 4.2. From the figure, it can be seen that the underlying infrastructures for both DCS_1 and DCS_2 are managed by a single cloud service provider, $CloudProvider_1$. Usually, after a user is authenticated to the cloud, the user can access any resources allocated to the user.

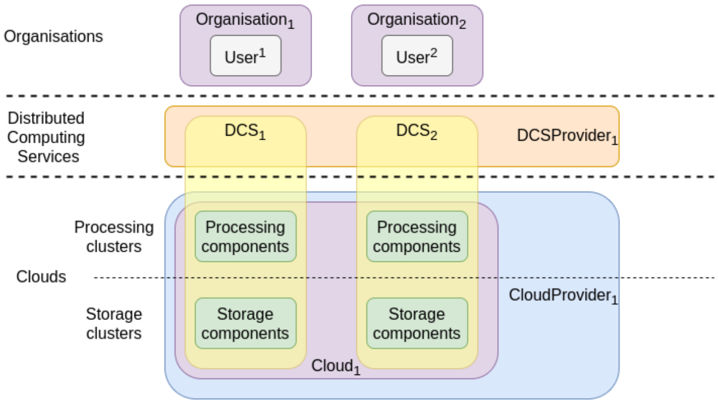


Figure 4.2: An example of SC-SA.

SC-SA is adopted by many cloud service providers, such as Amazon Web Services [99] and DigitalOcean [100]. Many providers choose to provide both of the processing and storage services (e.g., Amazon EC2 [101] and Amazon S3 [102] are, respectively, examples of the processing and storage services provided by Amazon Web Services) to reach a wider group of users with different demands and to build their own ecosystems. This architecture has also been used in many applications, including Amazon EMR [103], AzureMapReduce [104], and Twister4Azure [105].

SC-SA brings a number of advantages. From a functional aspect, users are assured that the connection between the clusters should be well established and maintained. They should receive the best support the cloud service provider can offer. From a security aspect, SC-SA should be less susceptible to threats caused by external entities owing to security measures enforced by the cloud service provider. However, SC-SA also has a number of limitations. If the cloud service provider uses a proprietary system, it may limit users' freedom to choose different services offered by different cloud service providers due to incompatibility issues. In addition, the reliance on the ecosystem provided by a particular provider may cause an issue of vendor lock-in. In other words, it might be technically difficult or cost-prohibitive for users to migrate their services and data from one provider to another provider.

4.3.1.2 Multi-Cloud System Architecture

Contrarily, in the MC-SA architecture, the processing and storage components of the same distributed computing service are, respectively, hosted in different clouds each managed by a different cloud service provider. An example of MC-SA where the processing and storage components of DCS_1 and DCS_2 are, respectively, deployed in two clouds, $Cloud_1$ and $Cloud_2$, is shown in Figure 4.3.

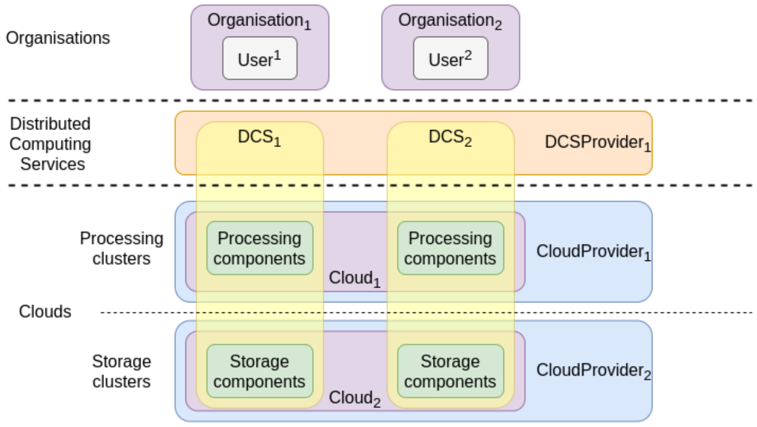


Figure 4.3: An example of MC-SA.

As shown in the figure, unlike Figure 4.2, the processing components of DCS_1 and DCS_2 are hosted in the processing clusters of $Cloud_1$, whereas the storage components of the two distributed computing services are hosted in the storage clusters of $Cloud_2$. In other words, the underlying infrastructures supporting a distributed computing service are managed by multiple cloud service providers. Due to this difference, there are three implications for using MC-SA. The first is that a user would need to be authenticated by two different authentication services each hosted on a different cloud prior to accessing the services hosted in both of the clouds, adding overhead costs. The second is that there may be interactions among components hosted in different clouds during a data processing job, and such interactions would also require authentication. These authentication requirements are of inter-cloud nature, which is different from the SC-SA case where authentication requirements are of intra-cloud nature. The third is that data involved in the job would need to be transferred between clouds managed by different cloud service providers, i.e., there will be inter-cloud data transfers. The networks connecting these clouds are likely to be Wide Area Networks (WANs) or the Internet, which are open to a wide range of threats and attacks.

There are some existing works in prototyping the MC-SA architecture. Resilin [19] is one of such works that develops a prototype using MC-SA. In this prototype, the processing and storage components are hosted in different clouds provided by different cloud service providers.

MC-SA gives a user greater flexibility in the selection of services as the user has a higher degree of freedom to subscribes to services provided by different cloud service providers. However, in comparison with SC-SA, there are additional security complications introduced by MC-SA because of inter-domain communication. As multiple clouds are involved during a job and different clouds may have varying levels of trust, the authentication services for inter-cloud interactions, inter-cloud resource access, and inter-cloud data transfer should be considered at multiple levels, which are more levels than the case for SC-SA. For example, the authentication services may be required at job-level, component-level, user-level, distributed computing service-level, and cloud-level. Also, inter-cloud data transfer may be through open and insecure channels, increasing the risk of the authenticity of the data being compromised.

4.3.1.3 Making the Selection

With regard to architecture selection, we should select one that presents a greater set of research issues and problems so that a solution, designed to address these issues and problems, could be applied to both architectures. To this end, we have identified the following four criteria to guide the selection.

(SAC1) Usability and flexibility from distributed computing service providers' perspective: The benefits and disadvantages are considered in terms of scalability, vendor dependency, interoperability, and freedom of processing and storage service subscriptions.

(SAC2) Manageability and expandability from cloud service providers' perspective: A system architecture chosen by a cloud service provider may influence the provider's business potential. For example, it may restrict the number of services the provider could provide, how they may expand their service in the future, or both. The chosen architecture should best support the freedom in service offering, control, and management, and provide flexibility in service capacity expansion.

(SAC3) Service deployment trend: The chosen system architecture should follow the trend for CBDC-MPC so that an authentication solution designed based on the use case model will be applicable to current and future applications.

(SAC4) Security complications: Clouds are vulnerable to various threats and attacks. Due to different characteristics of different system architectures, distributed computing services deployed on the clouds may encounter different sets of threats and attacks.

The analysis of SC-SA and MC-SA against the specified criteria is summarised in Table 4.1.

Table 4.1: The comparisons of SC-SA and MC-SA.

		SC-SA	**MC-SA**
(SAC1)	**Scalability**	The resources accessible to a distributed computing service provider are limited by the resource capacity of a single cloud. This also limits the number of users the provider can serve.	The resources accessible to a distributed computing service provider are not limited by the resource capacity of a single cloud. The distributed computing service provider may subscribe to services provided by more than one cloud.

		SC-SA	MC-SA
			Therefore, MC-SA may potentially have more resources to serve a higher number of users.
	Vendor dependency	A distributed computing service provider may experience the issues of vendor lock-in as well as service and data migration.	The vendor lock-in issue is unlikely to happen as there is incentive for cloud service providers to use standardised software and hardware. Data and service migration should be possible and with relatively less difficulty.
	Interoperability	Components hosted in different clouds might not be able to communicate with each other as different implementations of cloud services may not be compatible with each other.	There is incentive for cloud service providers to improve interoperability [106]. Components hosted in different clouds are more likely to be able to communicate with each other as these components are more likely to conform to standard Application Programming Interface (API) specifications.
	Freedom of service subscription	A distributed computing service provider has to subscribe to both processing and storage services provided by the same cloud service provider.	A distributed computing service provider may subscribe to processing and storage services offered by different cloud service providers.
(SAC2)	**Freedom of service offering**	A cloud service provider should offer both processing and storage services.	A cloud service provider may offer one or both of processing and storage services.
	Control and management	A cloud service provider is responsible for managing and maintaining all the services hosted on its cloud, and it has complete control over the services.	A cloud service provider only has control over the services hosted in its cloud. It does not have to manage and maintain the services that interoperate with its services but are offered by other cloud service providers.
	Resource expansion	To increase the resource capacity to meet the growth of distributed computing service providers' demands, a cloud service provider has to add more hardware and software components to its cloud.	The resource capacity can be increased either by adding more hardware and software components or by establishing collaborations with business partners. An example of such partnership is reported in [107]. In this way, multiple cloud service providers can offer a combined pool of resources or a bundled service to distributed computing service providers.
(SAC3)	**Proposals in literature**	A similar system architecture is used in AzureMapReduce [104] and Twister4Azure [105].	A similar system architecture is used in Resilin [19].
	Commercial services in use	A similar system architecture is used in Amazon Web Services [99].	We have not found any commercial services using the MC-SA architecture or anything similar at the time of writing.
	Market considerations	To gain a bigger market share, a cloud service provider may increase service variety and resource capacity. As a result, the management tasks may become more challenging or difficult,	As multiple cloud service providers are involved in this architecture, service variety and capacity can be increased without overwhelming a particular cloud service provider. The service

		SC-SA	MC-SA
		and the quality of service provided to users may deteriorate.	capacity and variety can be increased by pulling together the services provided by multiple cloud service providers.
(SAC4)	**Trust**	Usually, the components hosted in the same cloud are equally trustworthy, as they are managed by the same administrative entity.	There are two possibilities: (a) multiple cloud service providers form a federation to agree on how and to what level they form their mutual trust, and in this case, they will follow their agreement, and (b) different cloud service providers handle their trust of other cloud service providers differently and individually, and in this case, there will be different levels of trust among components in different clouds.
	Identity management	A cloud service provider usually uses a private identity management scheme to manage the identities of its users.	A distributed or federated identity management scheme may be deployed.
	Data authenticity	Usually, the authenticity of data stored in storage (data-at-rest) and data transmitted via networks (data-in-transit) can be ensured by mechanisms such as Message Authentication Codes (MACs) and digital signatures. Cloud service providers may not enforce the use of such methods but provide them as an optional service.	Authenticity of data-at-rest and data-in-transit should be protected as an essential service and by using more stringent mechanisms than those used in the SC-SA case.
	Data confidentiality	A data confidentiality service should also be provided to data-at-rest and data-in-transit. As data are only transmitted among components hosted in the same cloud and there are many mature security solutions, such as firewall, Intrusion Detection System (IDS) and Intrusion Prevention System (IPS), that provide protection against outsider attacks, therefore, the risk of data exposure caused by a malicious outsider is relatively lower.	In this architecture, the confidentiality service is provided to both data-at-rest and data-in-transit as an essential service, especially when the data are transmitted among components hosted in different clouds.
	Component status monitoring	In most cases, it is assumed that the components in the same cloud work correctly, as usually the components on the same cloud are connected via Local Area Networks (LANs) or dedicated networks.	There might be a lack of trust among the cloud service providers hosting the components. The components hosted in different clouds are usually connected via public Wide Area Networks (WANs), typically the Internet, and security threats on WANs are much higher than on LANs. Hence, component status monitoring and threats monitoring are required.
	Attacks and threats	In most cases, it is assumed that components hosted in the same cloud and data sent between the	Due to the same reason as mentioned above, the level of threats in this architecture is much higher. In addition to attacks on data sent

		SC-SA	MC-SA
		components are safe from outside attacks.	between components hosted in different clouds, there are also threats caused by authorised insiders, such as compromised or malicious entities.

As summarised in Table 4.1, MC-SA is a more preferable choice as the system architecture for our use case because of the following reasons. MC-SA provides a higher level of usability and flexibility to distributed computing service providers. A distributed computing service provider can subscribe to services offered by any cloud service providers without worrying about the vendor lock-in and migration issue. MC-SA provides a higher level of manageability and expandability to cloud service providers. A cloud service provider may provide one or both of processing and storage services to service consumers. The capacity and quality of a service provided by a cloud service provider can be enhanced by establishing collaborations with other cloud service providers. MC-SA better resembles a growing trend of virtual operators, i.e., entities that provide services built on top of services provided by other service providers to service consumers. In addition, MC-SA captures a broader set of security issues and requirements, so solutions designed based on MC-SA will also be applicable to SC-SA. For these reasons, we have chosen MC-SA as the system architecture for our use case.

4.3.2 Choosing a Big Data Processing Model

As the popularity in Big Data analysis increases, there exist many distributed computing models proposed for different applications. Based on applications, they can be classified into application-specific models and non-application-specific models. Application-specific models are formulated and optimised for some particular applications. Examples of such models include GraphLab [108] which is designed for machine learning and data mining, Dremel [109] which are designed for table based data query, Pregel [110] which is suitable for implementing large-scale graph algorithms in distributed settings, and Apache Storm [111] which is designed to process unbounded streams of data. Non-application-specific models are usually constructed to serve a wide range of applications. Among these non-application-specific models, the most notable models are MapReduce (MR) [112], Dryad [113], Hyracks [114], Nephele [115], and Apache Spark [116].

All the Big Data processing models mentioned above capture essential characteristics of distributed computing, i.e., data are transferred among, processed by, and stored on, distributed and networked components, allowing tasks of a data processing job to be executed concurrently. In line with our aim to design an authentication solution that can make a greater impact on distributed computing, we should select a Big Data processing model that is most used and is likely to be used in the future. For this reason, we exclude application-specific models from the selection and consider only non-application-specific models. To contrast the non-application-specific models, we have identified the following five criteria to guide the selection.

(BMC1) Study and adoption: Big Data processing models that are well studied and widely adopted by industry and academic are more likely to be considered and used by users.

(BMC2) Implementations and support: A Big Data processing model could be implemented by using different tools. Different implementations of the model could be conformed to different standards and governed by different licences (proprietary or opensource). Documentation and support for a model implementation are also important for users, particularly when the Big Data processing service is used for production. In addition, extensions that extend the features and functions of model implementations also help to support a broader set of users' demands.

(BMC3) Infrastructure requirements: Some Big Data processing models are designed based on some specific hardware setups. They may only achieve their full potential on those hardware setups. This may limit the selection of infrastructure service providers.

(BMC4) Security implications: Different Big Data processing models may be designed for different environments with different security requirements. As discussed earlier, we should choose a model with a broader set of security issues as an authentication solution designed for this model could be applied to other models.

The analysis of MR, Dryad, Hyracks, Nephele, and Apache Spark against the criteria specified above is summarised in Table 4.2.

Table 4.2: The comparisons of MR, Dryad, Hyracks, Nephele, and Apache Spark.

(BMC1)	**MR**	MR is well studied and widely adopted by both academia and industry. For example, according to the Web of Science Core Collection database, there are 7,022 indexed papers regarding MR (by using the keyword "MapReduce" for Topic search) between years 2000 and 2019. Examples where MR is used for research and production purposes include High Energy Physics group at Caltech [117], Facebook [75], and Twitter [118].
	Dryad	There are a limited number of papers published in literature. For example, using the same database and time range as above, there are only 156 papers containing the keyword "Dryad". Dryad has been used in many applications, including relational queries, large-scale matrix computations, and many text-processing tasks [113].
	Hyracks	There are a limited number of papers published in literature. For example, there are only 7 indexed papers shown when searched with the keyword "Hyracks" using the same database and time range as above.
	Nephele	There are a limited number of papers published in literature, e.g., there are 33 indexed papers containing the keyword "Nephele" using the same database and time range as above.
	Apache Spark	There is an increasing trend of study on, and usage of, Apache Spark reported in literature. For example, there are 1,366 papers containing the keyword "Apache Spark" using the same database and time range as above. Apache Spark has been used in a number of applications, including SQL, streaming, machine learning, and graph processing [119].
(BMC2)	**MR**	There have been many MR implementations developed by different organisations. One of the most notable opensource Big Data system supporting MR is Apache Hadoop [120]. MR is also commonly integrated as part of commercial Big Data solutions, such as Hortonworks Data Platform [121], and MapR Converged Data Platform [122]. Hence, there are extensive documentation and support available. In addition, although MR was originally designed for batch data processing, there are many extensions developed to extend the capabilities of MR, such as the supports for stream [123] and iterative [124] data processing .
	Dryad	There exist a limited number of systems using the Dryad model for data processing, e.g., DryadLINQ [125] and Comet [126], thus limited documentation and support.

	Hyracks	There is an opensource implementation of Hyracks by its authors [127]. Hyracks is also used as an execution engine by AsterixDB [128]. There are limited documentation and support available.
	Nephele	Nephele has been used in a number of Big Data processing systems, including Nephele/PACT [129] and Apache Flink [130]. Apache Flink has been used by many enterprises, including Amazon, Ebay, and Uber [131]. Documentation and support are provided by its active community.
	Apache Spark	The most prominent implementation is an Apache project with the same name, Apache Spark [119]. It is used by various enterprises, such as Alibaba Taobao, Amazon, and Ebay [116]. Documentation and support are provided by its active community. In addition, a number of extensions have been developed to extends the functions of Apache Spark, i.e., SQL and DataFrames, Spark Streaming, MLib, and GraphX [116].
(BMC3)	**MR**	MR is designed for the deployment on generic hardware and there are no specific hardware requirements. It can be deployed on a physical platform such as clusters of physical machines and on a virtualised platform such as clusters of virtual machines hosted in clouds.
	Dryad	There are no specific hardware requirements.
	Hyracks	There are no specific hardware requirements.
	Nephele	Nephele uses in-memory channels for data transfer to optimise performance. To benefit from this, Nephele based systems should be deployed on hardware with high memory capacity.
	Apache Spark	Apache Spark maximises the use of in-memory channels to reduce overhead cost incurred in disk reading and writing operations. It works best on hardware with high memory capacity.
(BMC4)	**MR**	All the Big Data processing models do not incorporate any security measures in the design of the models. It is assumed that security protections are provided by third-party security services. Owing to similar characteristics of these models, the Big Data processing services implementing the models should experience the same level of threats when they are deployed in the same environment.
	Dryad	
	Hyracks	
	Nephele	
	Apache Spark	

As summarised in the table, MR is a more preferrable Big Data processing model due to the following reasons. Among the non-application-specific models, MR is the most well studied and widely adopted Big Data processing model at the moment as indicated by the number of papers published in literature. In addition, owing to its popularity, there are extensive documentation and support available. For opensource MR implementations, users can find support they need from opensource communities. They can also choose to get support offered by commercial enterprises. MR is designed for deployment on generic hardware, so it is compatible with a wide range of hardware configurations or setups. With this model, users would have more options to choose from with regard to infrastructure service providers. Although on some specific hardware setups, the performance of MR may be lower than those of Nephele and Apache Spark. With regard to security implications, the level of threats experienced by MR is the same as those of the other models. In other words, all the models present a similar set of issues and challenges. An authentication solution designed for MR should also be applicable to the other models as well. For these reasons, we have chosen MR as the Big Data processing model for our use case. We will explain the MR based Big Data processing model in Section 4.3.3 before describing our use case model in Section 4.3.4.

4.3.3 MapReduce (MR) based Big Data Processing Model

MR [112] has been adopted in many applications thus architectures. In this report, we have chosen the YARN [132] based architecture. YARN is one of the most used architectures for resource scheduling, including MR. It is more scalable than other architectures for MR such as the one reported in [112]. It can support large scale job executions involving tens of thousands of tasks running on thousands of machines [133]. In addition, it is used in Apache Hadoop [120], one of the most prominent Big Data systems.

To describe the MR based Big Data processing model, we first explain the components of an MR service involved in the execution of a data processing job, before describing an MR based job execution flow.

4.3.3.1 MR Components

The components used by an MR service are machines and containers (e.g., application process) of resources that are hosted on the machines. Based on their functions, the MR components can be largely classified into three groups: client components, processing components, and storage components. Processing components and storage components can be separately hosted in clusters of different clouds. For generality, each group is assumed to be hosted in a separate cluster, thus leading to three clusters, i.e., a Client cluster, a Distributed Processing System (DPS) cluster, and a Distributed File System (DFS) cluster. An overview of the MR components of an MR service is depicted in Figure 4.4.

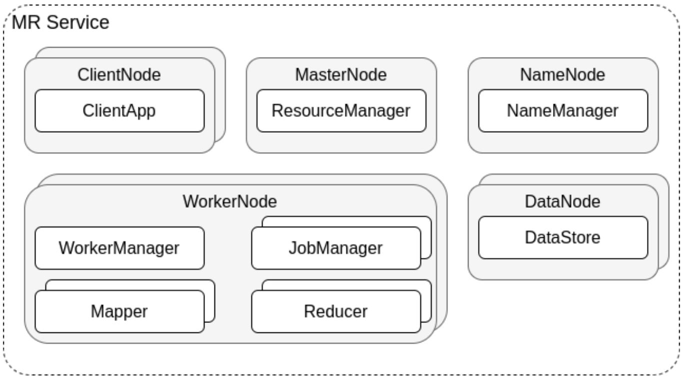


Figure 4.4: MR components.

The Client cluster hosts multiple ClientNodes. Each ClientNode hosts one ClientApp container. ClientApp allows a user to submit jobs and input of the jobs, and to retrieve the output of the jobs.

The DPS cluster consists of one MasterNode and multiple WorkerNodes. MasterNode hosts one ResourceManager container. ResourceManager manages the resources of the MR service and schedules job executions. Each WorkerNode hosts one WorkerManager container and multiple sets of JobManager, Mapper, and Reducer containers; each such set serves a particular job execution. WorkerManager manages the resources of the WorkerNode. JobManager schedules and manages the execution of the tasks of a job carried out by Mappers and Reducers assigned to the job. Each Mapper and Reducer, respectively, carry out a map task and a reduce task. Mappers and Reducers are also collectively referred to as Workers.

The DFS cluster consists of one NameNode and multiple DataNodes. NameNode hosts one NameManager container. NameManager maintains a file system, the metadata of all the files (the input and output of the jobs), and directories storing the files. Each DataNode hosts one DataStore container. DataStore keeps portions of the files, called data blocks.

4.3.3.2 Job Execution Flow

The process of a job execution consists of three phases: the job submission phase, the map phase, and the reduce phase. In the job submission phase, a user uses ClientApp to submit a job execution request to ResourceManager. If the request is accepted, the user uploads the input data and the job configuration file onto the DFS cluster. The MR service will divide the input data into multiple items, called InputSplits. The number of the InputSplits is set by the user in the job configuration file, and this is determined by the size of each InputSplit and how the InputSplits should be divided. The number of the InputSplits dictates the number of Mappers assigned to the job, as the number of InputSplits should be equal to the number of Mappers assigned to the job. In other words, each InputSplit will be processed (i.e., consumed) by a different Mapper. After the user finishes uploading the InputSplits and notifies ResourceManager, ResourceManager launches JobManager to manage and orchestrate the execution of this job. JobManager starts Workers (Mappers and Reducers) and monitors the progress of the execution of the tasks on the Workers.

In the map phase, each Mapper retrieves the assigned InputSplit from the DFS cluster, executes a map task, and produces an output file, called IntermediateResult. Each IntermediateResult contains multiple data items, called PartitionSegments. The maximum number of PartitionSegments contained in an IntermediateResult is equal to the number of Reducers assigned to the job. Each PartitionSegment will be retrieved and consumed by a different Reducer. The PartitionSegments produced by a Mapper are stored in the local storage of the WorkerNode hosting the Mapper.

In the reduce phase, each Reducer retrieves the assigned PartitionSegments (one from a different Mapper) from the corresponding WorkerNodes, executes a reduce task, and produces an output file, called FinalResult. The FinalResults produced by all the Reducers are uploaded onto the DFS cluster. When the job execution finishes, ClientApp retrieves the FinalResults from the DFS cluster and notifies the user. Data flows during an MR job execution is shown in Figure 4.5.

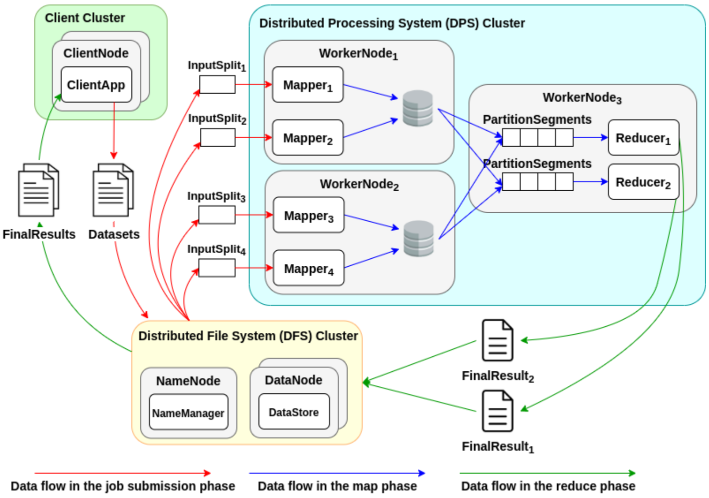


Figure 4.5: Data flows during an MR job execution.

4.3.4 Our Collaborative Big Data Computation on a Multiple Public Cloud platform (CBDC-MPC) Model

In this section, we describe the CBDC-MPC model, which is constructed based on the chosen MC-SA architecture and MR model. We give classifications of components and data used, interactions taking place, and communication patterns exhibited by the model.

4.3.4.1 Model Description

In this CBDC-MPC model, the distributed computing services used are MR services, MR_1 and MR_2 managed by $MRProvider_1$. MC-SA is applied in the deployment of the processing and storage components of the MR services MR_1 and MR_2 in $Cloud_1$ (managed by $CloudProvider_1$) and $Cloud_2$ (managed by $CloudProvider_2$), respectively. An overview of the CBDC-MPC architecture is depicted in Figure 4.6.

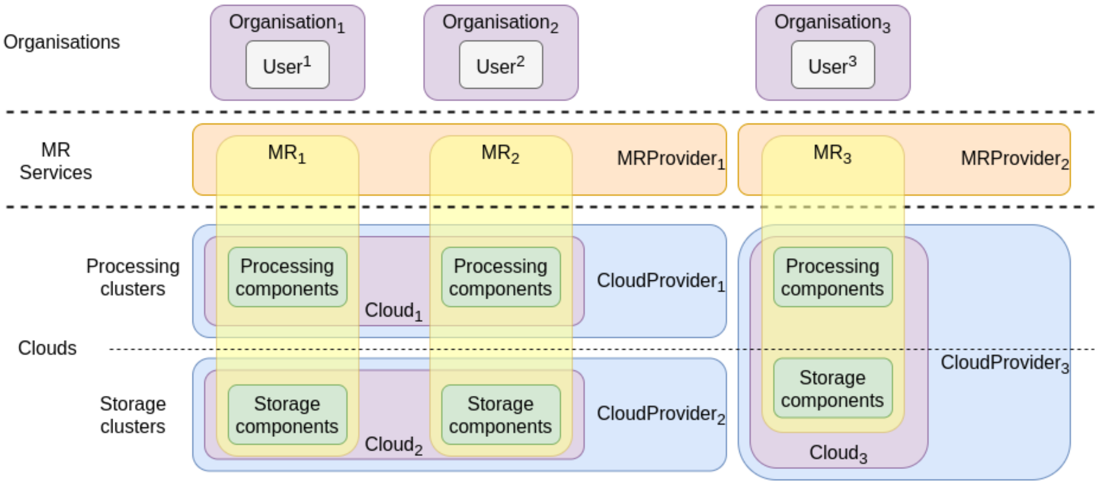


Figure 4.6: An overview of the CBDC-MPC architecture.

The process of a job execution in this CBDC-MPC model is complex as there are multiple entities from different domains involved in the job execution. To describe this process, we have formulated a generic job execution flow. Figure 4.7 shows a high-level view of the generic job execution flow, highlighting entities (including MR components) involved and the interactions among the components. The MR components involved in the job execution are organised by using a multi-layer structure similar to the MR Layered Authentication Model (MR-LAM) [84]. This multi-layer structure helps us to identify avenues for attacks at different layers and what can we used in the design of our authentication solution to protect against such attacks. In this structure, for each layer, entities are grouped into domains based on their functions or their associations. Users and ClientApps of an organisation form a domain, called an OrgDomain. Processing and storage resources hosted in a cloud form the second type of domains, called a CloudDomain. Components allocated to an MR service form the third type of domains, called an MRDomain. Components serving a particular job form the fourth type of domains, called a JobDomain [84].

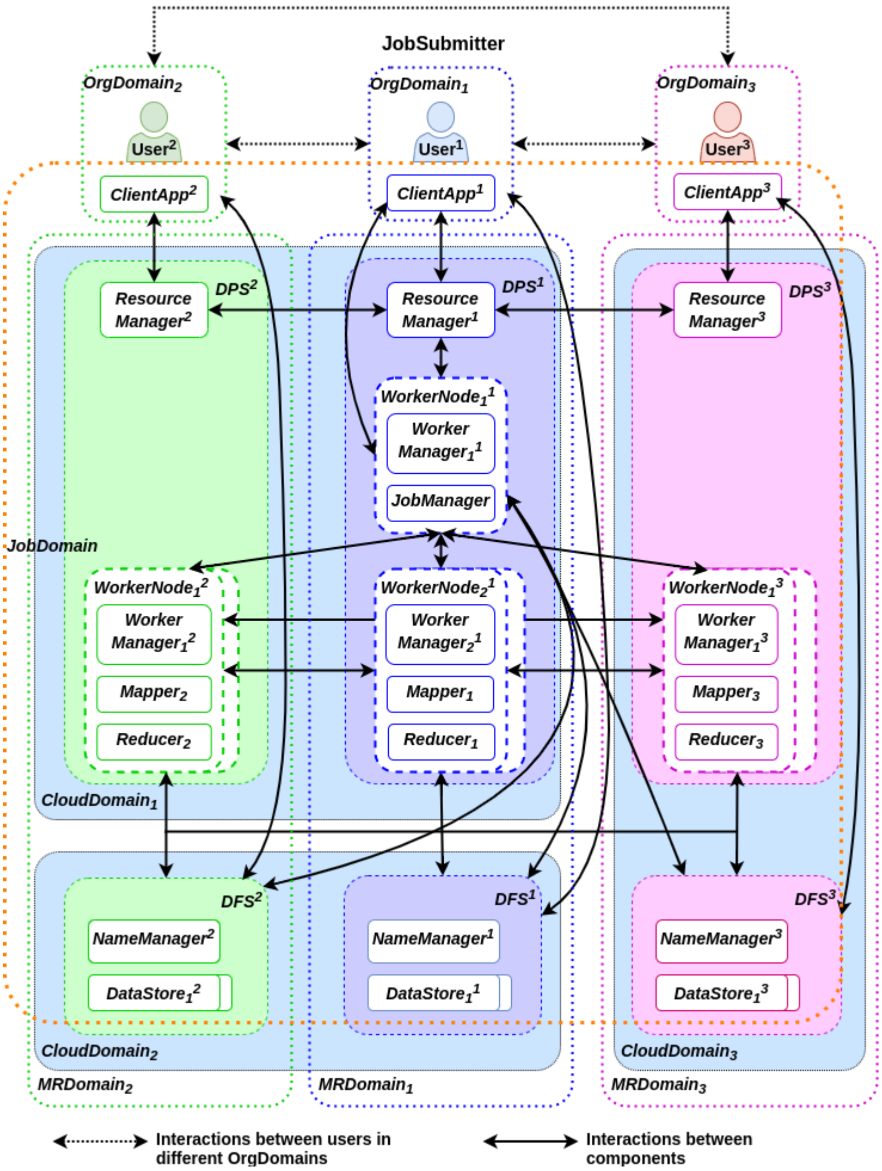
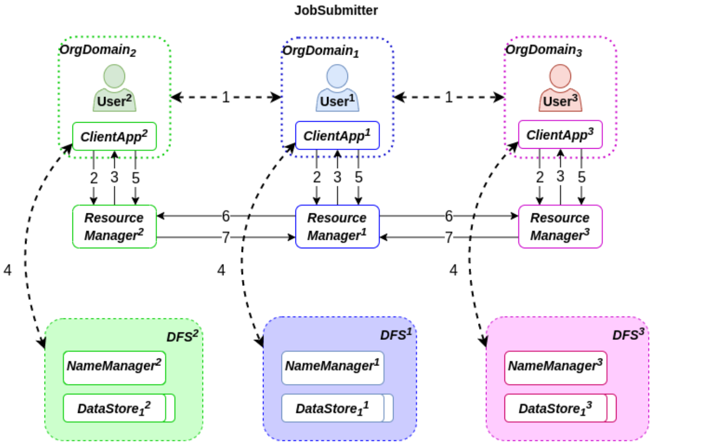


Figure 4.7: A high-level view of a generic job execution flow in the CBDC-MPC model.

The execution of a job starts from when $User^1$ (referred to as JobSubmitter) sends a request for shared datasets to users in other OrgDomains to when the results of the job execution are ready for collection. As explained in Section 4.3.3, the job execution is divided into three phases, the job submission phase, the map phase, and the reduce phase. As there are many interactions among components in the job submission phase, for ease of discussion, we further divide the job submission phase into two steps, the execution request step and the worker allocation step. The execution request step starts from when JobSubmitter sends a request for shared datasets to the other users to when all the users finish uploading the shared datasets onto the respective DFS clusters. The worker allocation step starts from when $ResourceManager^1$ launches a $JobManager$ to oversee the execution of the job to when all the Workers are launched.

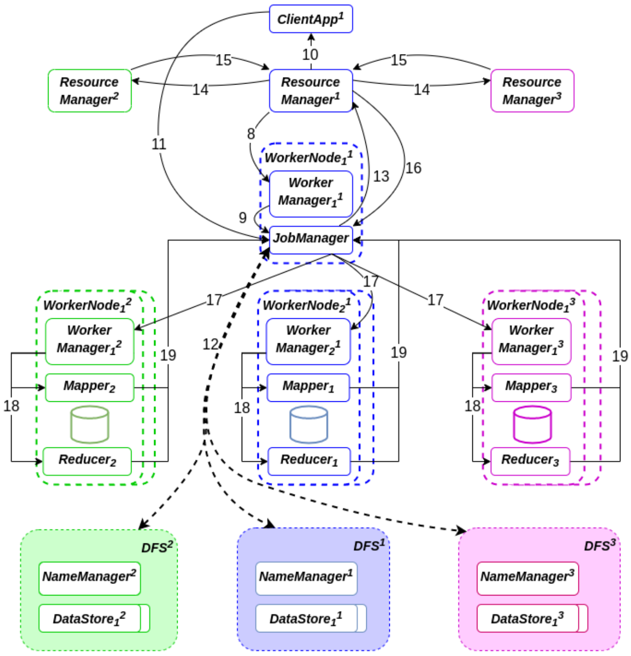
The interaction flows among entities in different job execution phases are shown in Figure 4.8. An interaction between a pair of components is shown as a unidirectional solid arrowed line. For ease of presentation, a set of interactions among users from different organisations (interaction 1) and a set of interactions between an entity and a DFS cluster (interactions 4, 12, 21, 26, and 29) are depicted as a bidirectional dashed arrowed line. It is also worth noting that this figure omits interactions among components that are job-independent (i.e., those that are not specifically created to serve a particular job) and are in the same MRDomain as these interactions can be protected by using existing security solutions (e.g., an MR service level authentication service). Examples of such interactions are data block duplication between DataStores (of the same DFS cluster) and storage capacity report between NameManager and ResourceManager (of the same MRDomain).



1: Request shared datasets
2: Request a new job ID and a path to upload the data
3: Reply the job ID and the path
4: Upload the data onto the DFS cluster
   4a: Request a list of DataStores (ClientApp to NameManager)
   4b: Reply the list (NameManager to ClientApp)
   4c: Write the data to the DataStores (ClientApp to DataStores)

5: Notify the completion of data uploading and submit the job
6: Request data uploading status
7: Notify the completion of data uploading

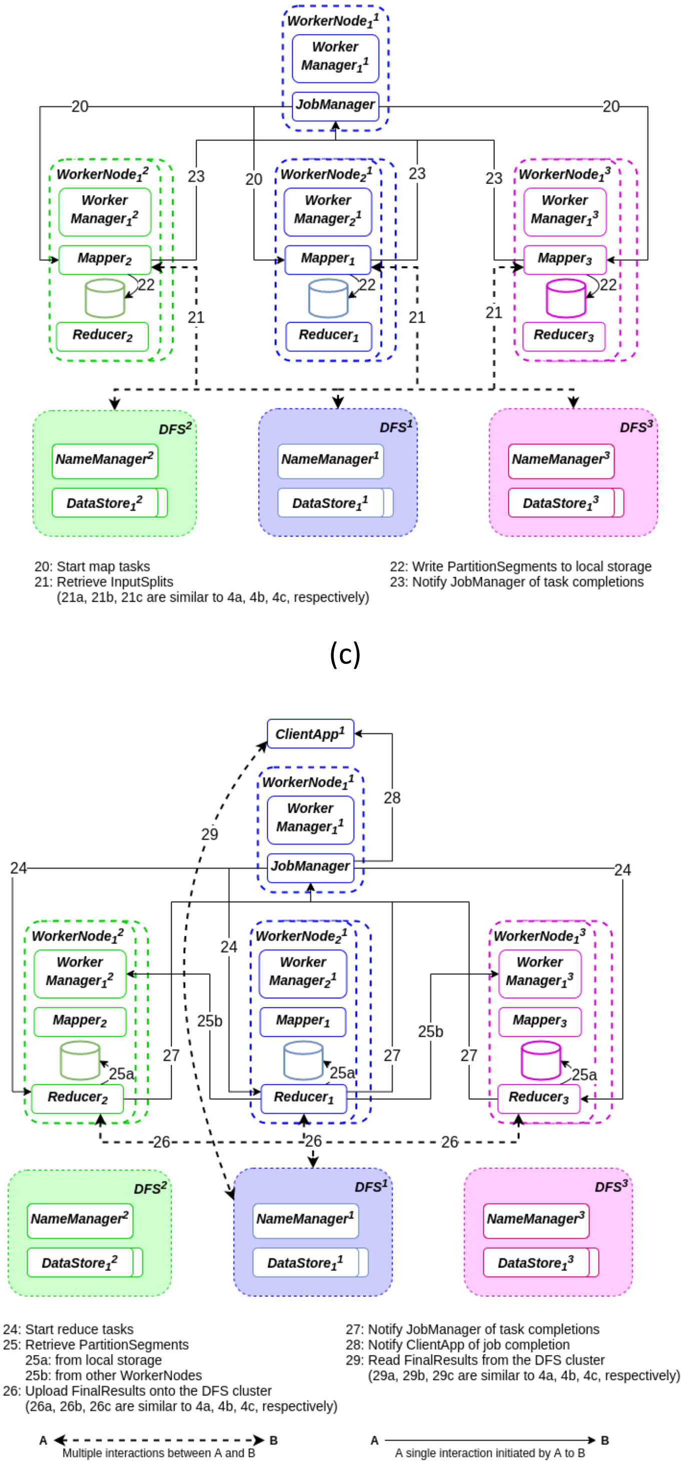
(a)



8: Request to launch a JobManager
9: Launch the JobManager
10: Notify ClientApp of the launch of JobManager
11: Query job execution progress
12: Read job configuration data
   (12a, 12b, 12c are similar to 4a, 4b, 4c, respectively)
13: Request computing resources

14: Request computing resources from the other MR services
15: Reply lists of WorkerNodes
16: Reply the lists of WorkerNodes
17: Request to launch Mappers and Reducers
18: Launch the Mappers and the Reducers
19: Report status to JobManager

(b)



20: Start map tasks
21: Retrieve InputSplits
(21a, 21b, 21c are similar to 4a, 4b, 4c, respectively)

22: Write PartitionSegments to local storage
23: Notify JobManager of task completions

(c)

24: Start reduce tasks
25: Retrieve PartitionSegments
25a: from local storage
25b: from other WorkerNodes
26: Upload FinalResults onto the DFS cluster
(26a, 26b, 26c are similar to 4a, 4b, 4c, respectively)

27: Notify JobManager of task completions
28: Notify ClientApp of job completion
29: Read FinalResults from the DFS cluster
(29a, 29b, 29c are similar to 4a, 4b, 4c, respectively)

Multiple interactions between A and B

A single interaction initiated by A to B

(d)

Figure 4.8: Interactions among entities.
(a) Job submission phase: execution request step.
(b) Job submission phase: worker allocation step. (c) Map phase. (d) Reduce phase.

The generic operational steps of a job execution are described in the following.

(GM1) $User^1$ sends a reference identifier (ID) and a request for shared datasets and resources to $User^2$ and $User^3$. $User^2$ and $User^3$ receive and approve the request, then reply the confirmation back to $User^1$.

(GM2) Each user sends the reference ID and a request for a new job ID and a path to write the dataset and the job configuration file to the ResourceManager of his/her MR service via ClientApp.

(GM3) Each ResourceManager receives the request. If the job is accepted, it generates a job ID based on the received reference ID (hence, all ResourceManagers generate the same job ID) and replies the job ID and the path back to the respective user.

(GM4) Each ClientApp (on behalf of its user) receives the reply and writes the dataset and the job configuration file onto the respective DFS cluster. The writing process consists of three interactions.

a. Each ClientApp sends a request for data writing to the respective NameManager.
b. Each NameManager receives the request and replies the respective ClientApp with a list of DataStores.
c. Each ClientApp receives the list. It contacts and writes the dataset and the job configuration file to DataStores. The dataset is divided into multiple InputSplits.

(GM5) After each ClientApp finishes writing its data, it notifies the respective ResourceManager.

(GM6) $ResourceManager^1$ contacts $ResourceManager^2$ and $ResourceManager^3$ to inquire the status of data writing.

(GM7) After $ClientApp^2$ and $ClientApp^3$ finish writing their data and notify their respective ResourceManagers, $ResourceManager^2$ and $ResourceManager^3$ notify $ResourceManager^1$ of the completion of data writing.

(GM8) $ResourceManager^1$ sends a request for launching $JobManager$ for the job to $WorkerManager_1^1$.

(GM9) $WorkerManager_1^1$ allocates resources and starts $JobManager$.

(GM10) After $JobManager$ is successfully launched, $ResourceManager^1$ notifies $ClientApp^1$.

(GM11) $ClientApp^1$ contacts $JobManager$ to inquire the progress of job execution. *

(GM12) $JobManager$ reads the job configuration files from the DFS clusters. The reading process consists of three interactions.

a. $JobManager$ sends a request for data reading to all NameManagers.
b. Each NameManager receives the request and replies $JobManager$ with a list of DataStores.
c. $JobManager$ receives the list. It contacts the DataStores and read the job configuration files.

(GM13) $JobManager$ sends a request for worker allocation to $ResourceManager^1$.

(GM14) $ResourceManager^1$ receives the request and sends a request for worker allocation to $ResourceManager^2$ and $ResourceManager^3$.

(GM15) $ResourceManager^2$ and $ResourceManager^3$ receive the request and reply $ResourceManager^1$ with lists of WorkerNodes with available resources.

(GM16) $ResourceManager^1$ receives the lists and forwards them to $JobManager$.

(GM17) $JobManager$ sends a request to each of the WorkerManagers to launch Mappers and Reducers.

(GM18) Each of the WorkerManagers receives the request and starts Mappers and Reducers on its node.

(GM19) Each of the Mappers and Reducers contacts and reports their status to $JobManager$. *

(GM20) $JobManager$ issues a command to all of Mappers to start map tasks.

(GM21) Each Mapper receives the command and reads the assigned InputSplit from the respective DFS cluster. The reading process consists of three interactions.

a. Each Mapper sends a request for data reading to the respective NameManager.
b. Each NameManager receives the request and replies the respective Mapper with a list of DataStores.
c. Each Mapper receives the list. It contacts and read the assigned InputSplit from the DataStores.

(GM22) Each Mapper performs the map task on the assigned InputSplit. After the task finishes, it writes an IntermediateResult (containing PartitionSegments) to the local storage of the machine.

(GM23) Each Mapper notifies $JobManager$ when the map task and the writing of the IntermediateResult are finished.

(GM24) After all map tasks finishes, $JobManager$ issues a command to all of the Reducers to start reduce tasks.

(GM25) Each Reducer receives the command and reads the assigned PartitionSegments from two sources: the local storage on the machine; and remote WorkerNodes (via WorkerManagers)

(GM26) Each Reducer performs the reduce task on the assigned PartitionSegments. After the task finishes, it writes a FinalResult to DFS^1. The writing process consists of three interactions.

a. Each Reducer sends a request for data writing to $NameManager^1$.
b. $NameManager^1$ receives the request and replies the respective Reducer with a list of DataStores.
c. Each Reducer receives the list. It contacts and writes the FinalResult to DataStores.

(GM27) Each Reducer notifies $JobManager$ when the reduce task and the writing of the FinalResult finish.

(GM28) $JobManager$ notifies $ClientApp^1$ when the FinalResults are ready for retrieval.

(GM29) $ClientApp^1$ receives the notification and retrieves the FinalResults for $User^1$. The reading process consists of three interactions.

a. $ClientApp^1$ sends a request for data reading to $NameManager^1$.
b. $NameManager^1$ receives the request and replies $ClientApp^1$ with a list of DataStores.
c. $ClientApp^1$ receives the list. It contacts and reads the FinalResults from DataStores.

Note: While most interactions take place only once during the job execution, there are interactions that will be repeated periodically. These interactions are referred to as recurrent interactions. They are marked with an asterisk (*).

4.3.4.2 Component Classifications

This section presents the classifications of the MR components involved in a job execution. Based on observation made on the CBDC-MPC model, we have identified two criteria, job dependency and functions, for component classifications. Job dependency dictates the selection authentication methods and credentials used, whereas functions indicate the levels of risks experienced by the components.

Job dependency determines the lifetime of a component, i.e., when the component is created and when it is terminated. With this criterion, components can be classified into job-dependent and job-independent components. Job-dependent components are components that are created to serve a particular job. They are created when the job is being executed and they are terminated when the execution of the job is completed. The components in this group are JobManager, Mapper, and Reducer. Job-independent components are components that are created and destroyed independent of the execution of a job. These components may serve multiple jobs, thus, their lifetimes are usually longer than those of job-dependent components. The components in this group are ClientApp, ResourceManager, NameManager, WorkerManager, and DataStore. Applying an authentication scheme with a high credential generation cost to job-dependent components is not efficient as the lifetimes of these components are relatively short. In addition, the number of these components could be potentially large, particularly in a large-scale job execution. The costs incurred in generating credentials for these components could be too high.

Based on functions, MR components can be largely classified into two groups: management components and data-handling components. Management components are components whose functions are to manage resources or supervise task executions. These components are ResourceManager, NameManager, WorkerManagers, and JobManager. Data-handling components are components whose functions are to produce, consume, or store JobData. These components are ClientApps, Mappers, Reducers, and DataStores. As the functions of management components are important to job executions and to the system, there usually are security measures put in place to protect these components. There exists a number of solutions that can accomplish this task and an example of such solutions is Trusted Computing [134]. Data-handling components, on the other hand, may not have the same (or adequate) level of protections or security assurance. This is because, firstly, these components are in large quantities, so providing strong security protections to all of them may not be practical due to efficiency reasons. Secondly, unlike management components which interact with only other software components, data-handling components also interact with users at large or execute user-supplied codes (i.e., map and reduce functions) which may contain vulnerable or malicious codes. Thirdly, data are assets, hence, data-handling components may be more attractive to attackers. Owing to these reasons, data-handling components are more vulnerable to threats and attacks. They could be compromised for attacking the data or be used as a springboard for further attacks against the system.

4.3.4.3 Data Classifications

For ease of discussion, data that are used, processed, and generated during a job execution are collectively referred to as JobData. An entity producing (generating) JobData is called a producer and an entity consuming (using) JobData is called a consumer. There are three groups of JobData, InputSplits for the job submission phase, IntermediateResults and PartitionSegments for the map phase, and FinalResults for the reduce phase.

An InputSplit is a portion of the input data (shared datasets) of a job. The InputSplits of an organisation are generated (supplied) by the ClientApp of the user representing the organisation and stored in the DFS cluster of the MR service subscribed by the organisation. Each of the InputSplits is assigned to, and used (consumed) by, a different Mapper which could be from a different MRDomain.

An IntermediateResult is an output data file produced by a Mapper. It contains multiple data items, called PartitionSegments, one for a different Reducer. The PartitionSegments produced by the same Mapper are stored in the local storage of the WorkerNode hosting the Mapper. Each of the Reducers retrieves the assigned PartitionSegments from the WorkerManagers of the WorkerNodes hosting the respective Mappers.

FinalResults are the output of the job execution, each of which is produced by a different Reducer. All the FinalResults are uploaded onto the DFS cluster of the MR service of JobSubmitter. When all the FinalResults are ready for collection, the ClientApp of JobSubmitter retrieves the FinalResults from the DFS cluster.

Based on the data classifications described above, we can make the following observations. As components hosted in different domains may not be equally trustworthy and JobData objects could be from an attacker or a compromised component, a consumer needs an assurance that the JobData it consumes are indeed produced by the claimed producers and have not been tampered with. However, ensuring the authenticity (origin and integrity) of the JobData is difficult in this context as JobData are stored and managed by components other than the producers of the JobData. Producers do not have a complete control over the JobData they produce after the generation of the JobData and consumers cannot directly contact the respective producers to get the assigned JobData. The JobData may be accessed and tampered with by an attacker at any point of data processing. In addition, a producer may produce multiple JobData objects each for a different consumer, and a consumer may also consume multiple JobData objects each produced by a different producer. This indicates that data authenticity protection should be provided at the object level and should be as efficient as possible due to the large quantity of JobData.

4.3.4.4 Interaction Classifications

Interactions highlighted in Figure 4.8, can be classified into two groups, initial and subsequent interactions. An initial interaction refers to the first interaction between a pair of components, such as an interaction between ClientApp and NameManager when the ClientApp inquires a list of DataStores for writing InputSplits (in the job submission phase). A subsequent interaction refers to an interaction between two components that have prior interactions, such as an interaction between NameManager and ClientApp when the NameManager replies a list of DataStores to the ClientApp. The classifications of these interactions are summarised in Table 4.3.

Owing to the impact of allowing remote (and potentially untrustworthy) entities to access local resources which may contain sensitive and high-value data, initial interactions introduce a higher level of risks, particularly when data providing and consuming components are from different organisations or domains (e.g., hosted in different clouds). If two entities of an interaction are from different clouds, they are more likely being connected via public networks, such as the Internet, which are vulnerable to a broader range of threats than private networks. In addition, initial interactions typically involve entities that have yet established any trust or shared secrets and these interactions are usually used to establish such secrets. If initial interactions, or the secrets being established during the initial interactions, are compromised, the security of subsequent interactions will also be put at risk. Subsequent interactions, on the other hand, use temporary secrets established in the authentication of preceding interactions. They may impact on a limited set of interactions should the temporary secrets be compromised. Thus, they experience a lower level of risks.

Table 4.3: Interaction classifications.

Group	**Step numbers (as shown in** Figure 4.8**)**
Initial interactions	2, 4a, 4c, 6, 8, 11, 12a, 12c, 13, 17, 19, 21a, 21c, 25, 26a, 26c
Subsequent interactions	3, 4b, 5, 7, 10, 12b, 14, 15, 16, 20, 21b, 23, 24, 26b, 27, 28

Notes: Steps 1, 9, 18, and 22 are excluded as these interactions can be authenticated by using existing authentication mechanisms.

4.3.4.5 Communication Pattern Classifications

A different phase of the execution is characterised by a different communication pattern, i.e., the job submission phase is characterised by the one-to-many (O2M) pattern, the map phase by the many-to-many (M2M) pattern, and the reduce phase by the many-to-one (M2O) pattern.

In the O2M pattern, there is one producer (ClientApp) but multiple consumers (Mappers). The producer produces multiple objects (InputSplits), one for each consumer. In the job submission phase, although the input datasets for the job are from multiple users, each of the users (through his/her ClientApp) provides a different set of InputSplits and each of the InputSplits is assigned to a different Mapper, thus, characterised by the O2M pattern.

In the M2M pattern, there are multiple (P) producers (Mappers) and multiple (Q) consumers (Reducers). Each producer produces up to Q objects (PartitionSegments, each of which contains a different set of key-value pairs), one for a different consumer. Each consumer consumes up to P objects, one from a different producer. The communication between Mappers and Reducers in the map phase is characterised by the M2M pattern.

In the M2O pattern, there are multiple producers (Reducers) but one consumer (ClientApp of $User^1$). Objects (FinalResults) produced by different producers are typically different (each of the FinalResults contains a different set of key-value pairs). This M2O pattern captures the characteristics of the communication between Reducers and ClienApp in the reduce phase.

By taking into account the characteristics of these communication patterns, we may be able to improve the efficiency of our authentication solution. The communication patterns help identify steps where components have to produce or consume a potentially large number of JobData objects as the costs incurred in processing and transmitting these items could be large, increasing risks of creating performance bottlenecks. These steps are ClientApps producing InputSplits in the job submission phase, Mappers producing

PartitionSegments and Reducers consuming the PartitionSegments in the map phase, and ClientApp (of JobSubmitter) consuming FinalResults in the reduce phase.

The job execution flow used in the CBDC-MPC model is based on the one reported in [112] and [132]. The CBDC-MPC model, in its current form, does not support iterative job execution (i.e., by chaining the output of Reducers in one iteration to the input of Mappers in the next iteration). However, the communication patterns of an iterative job execution can still be captured by the three patterns (O2M, M2M, and M2O). This means that an authentication solution designed based on our model should also be applicable to other applications that support iterative job execution.

4.4 Threat Analysis

Based on the CBDC-MPC model (described in Section 4.3.4), it can be seen that threats and attacks can be mounted on the system at multiple points during a job execution. This section gives a critical threat analysis, identifying threats with regard to violation of entity identity and data authenticity protections. It presents threat classifications before describing a threat model used for the design of our authentication solution.

4.4.1 Threats and Attacks

During a job execution, threats and attacks could be mounted at any of the job level, the MR service level, and the cloud (or infrastructure) level.

At the job level, an unauthorised entity may impersonate, or gain interactions with, any of the authorised entities. Such threats may happen at any interactions from when ClientApp submits a job to ResourceManager to when it finishes reading the result of the job from the DFS cluster. These threats may be mounted via Man-in-the-Middle (MITM) attacks, or theft of an authorised entity's authentication credential. If the unauthorised entity is successful in mounting such an attack, it could gain access to MR services, users' data, or both. This could cause severe consequences, including, damages to the underlying systems and other systems connected to these systems, users' privacy being compromised, and contamination of the job execution result.

At the MR service level, an MR service may serve multiple jobs submitted by different users concurrently. In other words, users with different access rights (including those that are not authorised to carry out a particular MR job) may use the same MR service. Curious or malicious users may attempt to gain unauthorised access to data used in a particular job. They may do so directly or indirectly via compromising service components.

At the cloud level, messages exchanged between MR components hosted in different clouds may be transmitted through insecure communication channels connecting the components; these channels are usually WAN (e.g., the Internet) based and are vulnerable to a wide range of threats and attacks. Such attacks include intercepting, altering, and replaying messages exchanged among the components. For example, an attacker may intercept a data reading request sent by ClientApp to DFS and replay the request at a later time to gain access to the data that the attacker is not authorised to access. In addition, resource sharing in clouds allows multiple tenants to access shared resources. By exploiting vulnerabilities or misconfiguration, tenants of the same cloud but external to the MR service may also gain access to JobData used by the MR service.

In summary, the threats and attacks discussed above can be classified into the following 7 categories:

(T1) Impersonation attacks: Impersonation attacks refer to attempts to assume the identity of an authorised entity. These attacks may be mounted via intercepting the identity credential of an authorised entity or by guessing the secret related to the credential. MITM attacks are one of such attacks. MITM attacks are performed by relaying (and possibly altering) intercepted messages exchanged between authorised entities or by hijacking a live session.

(T2) Confidential data exposure threats: Sensitive data (e.g., identity credentials) may be exposed if not protected properly, particularly when data are transmitted over public networks.

(T3) Replay attacks: These attacks refer to attempts to capture messages and repeat the messages to entities. These attacks are commonly used to orchestrate impersonation attacks, which allow an attacker to assume the identity of an authorised entity without the knowledge of identity credential. Addressing these attacks in the CBDC-MPC context is particularly important, as a job execution may last for a long period of time, giving attackers many opportunities to launch such attacks.

(T4) Message tampering attacks: These attacks refer to alteration of intercepted messages before sending the modified messages to targets. Data sent by an authorised entity may be replaced with fraudulent data, e.g., a fraudulent session key planted by an attacker. An attacker may use such fraudulent data to launch further attacks on the job and the systems.

(T5) Data injection attacks: These attacks refer to unauthorised attempts to inject new instances of fraudulent JobData at any points of the data flow.

(T6) Data tampering attacks: These attacks refer to unauthorised alterations to JobData, such as adding, modifying, deleting some portions of JobData. (T5) and (T6) are external attacks. These attacks can lead to the contamination of the results of a job execution.

(T7) Repudiation attacks: These attacks refer to any false denials of the generation of JobData. Repudiation is commonly used to evade responsibility or accountability. (T7) are insider attacks. Addressing these attacks is necessary in a collaborative environment, such as the CBDC-MPC context, where multiple organisations are involved and datasets from multiple organisations are used.

Threats(T1) through to (T4) are entity identity related threats, whereas threats (T5) through to (T7) are data authenticity related threats.

4.4.2 Threat Model

A threat model defines the trust boundary of MR components. The threat model for our solution should take into account of the characteristics of the underlying Big Data computing platform, which is multi-cloud MR in this case. Existing threat models (or standard threat models) do not capture the characteristics of MR based CDBC-MPC; they do not consider the functions of the components and the characteristics of inter-domain communication, therefore not suited to our problem context. As discussed in Section 4.3.4.2, based on functions, MR components can be classified into management and data-handling components. The level of risks experienced by data-handling components is higher than that of management components. Based on these considerations, our threat model is defined as follows:

(TM1) The management components are trustworthy; they will perform their functions faithfully.

(TM2) The data-handling components are untrustworthy; they may be malicious and actively use any of the attack methods highlighted in Section 4.4.1.

Entities that are external to an MR job or the MR service, including those on the Internet, are untrustworthy; they may gain access to the shared resources and mount attacks on the job and the systems.

4.5 Requirement Specifications

To counter the threats and attacks identified in Section 4.4.1, we here specify a set of requirements for an effective, efficient, and scalable authentication solution for CBDC-MPC. The requirements will be used to guide the design of our solution. The requirements consist of functional, security, and performance requirements.

4.5.1 Functional Requirements

(FR1) Full-cycle protection: Every interaction taking place, and every JobData object transmitted, during a job execution should be authenticated, from when a user submits a job execution request to when the user retrieves the result of the job execution.

(FR2) Cross-domain authentication: Entities involved in a job execution, including those that are from different administrative domains should be able to mutually authenticate each other.

(FR3) Automated authentication: After a user has submitted a job to the MR service, the authentication of any interacting entities (i.e., component-to-component authentication) should be accomplished without the intervention or involvement of the user.

(FR4) Fine-grained verifiability: JobData objects should be individually verifiable. This is necessary as objects produced by a producer may be consumed by multiple different consumers, objects assigned to a consumer are produced by different producers, and different consumers may consume the assigned objects at different times.

(FR5) Limited JobData exposure: In providing data authentication, the exposure of JobData should not increase. In other words, JobData should not be revealed to any other components than those that are involved in the processing of the JobData.

4.5.2 Security Requirements

(SR1) Mutual authentication: Interacting entities should be able to verifies the identities of each other before the interaction can be proceeded. Mutual authentication ensures that entities are interacting with the intended entities. This requirement is used to counter impersonation attacks (T1).

(SR2) Sensitive data confidentiality: The confidentiality of sensitive data (i.e., secrets) exchanged during an authentication process should be preserved; they should not be revealed to any other entities than the claimant and verifier of the authentication process. This requirement is used to counter confidential data exposure threats (T2).

(SR3) Replay attack protection: Messages used in every authentication instance should be fresh. Replayed messages should be detected by the receiving entities. This requirement is used to counter replay attacks (T3).

(SR4) Message authenticity protection: The authenticity of messages exchanged in achieving entity authentication should be protected. The authenticity protection encompasses origin authentication (messages are generated by the claimed source) and integrity protection (messages are not tampered with since their origination). This requirement is used to counter message tampering attacks (T4).

(SR5) Data origin authentication: The origin of each JobData object should be verifiable to ensure that the object is indeed produced by the claimed producer. This requirement is used to counter data injection attacks (T5).

(SR6) Data integrity protection: The integrity of each JobData object should be verifiable to ensure that the object has not been tampered with since its generation. This requirement is used to counter data tampering attacks (T6).

(SR7) Non-repudiation of origin: The generation of each JobData object should be bound to its producer so that any false denial of its generation can be detected. This requirement is used to counter repudiation attacks (T7).

4.5.3 Performance Requirements

(PR1) Low overheads: The overheads imposed on a job execution as a result of achieving authentication should be as low as possible. The overheads are considered in two aspects: (1) computational overhead, i.e., computational cost of generating and verifying AuthData; and (2) communication overhead, i.e., the amount of AuthData transmitted over networks.

(PR2) High scalability: When the number of components and the volume of JobData increase, the rate of increase in the overheads should be no more than linear.

4.6 The Running Example

This section further develops the running example discussed in Section 1.2, giving more details about the input datasets for the job and how the job is executed by using MR in a multi-cloud setting. The example is also used to explain the design decisions made in the above sections and motivations for the decisions.

As described in Section 1.2, the three collaborative organisations ($Organisation_1$, $Organisation_2$, and $Organisation_3$) perform a data analysis job to identify any (potentially compromised) machines in the collaborative organisations in the past 30 days (say May 2020). Potentially compromised machines are machines that have been connected to by compromised machines. The IP address blocks of the three organisations are, respectively, 10.1.0.0/16, 10.2.0.0/16, and 10.3.0.0/16. It is assumed that there are four compromised machines and their IP addresses are 10.1.0.101, 10.1.0.102, 10.2.0.101, and 10.3.0.101.

The input of the job comprises three input datasets, one from each of the participating organisations. Each dataset is a security log file containing network activities (connection details) in the respective organisation. The file contains tabular data in which each entry (row) contains date, time, source IP address, source port number, destination IP address, and destination port number. It is assumed that the attacks are mounted only on port 22. The contents of the files ($File_1$, $File_2$, and $File_3$) are shown in Table 4.4. The entries showing connections from the compromised machines to the potentially compromised machines are highlighted in grey.

Table 4.4: The input (security log files) for the running example.

Entry No.	Date (DD/MM/YYYY)	Time	Source IP	Source Port	Destination IP	Destination Port
$File_1$						
1	03/05/2020	01:00:00	10.1.0.101	61001	10.2.0.201	22
2	03/05/2020	01:30:00	10.2.0.201	62001	10.1.0.202	80
3	03/05/2020	02:00:00	10.1.0.101	61002	10.3.0.201	22
4	03/05/2020	03:00:00	10.1.0.102	61001	10.3.0.202	22
5	04/05/2020	10:30:00	10.3.0.201	63001	10.1.0.201	80
$File_2$						
1	03/05/2020	01:30:00	10.2.0.201	22	10.1.0.101	61001
2	04/05/2020	01:00:00	10.2.0.101	62001	10.1.0.201	22
3	04/05/2020	01:30:00	10.1.0.201	61001	10.2.0.202	22
4	04/05/2020	02:00:00	10.2.0.101	62002	10.2.0.203	22
$File_3$						
1	03/05/2020	09:30:00	10.1.0.202	61001	10.3.0.201	80
2	04/05/2020	09:30:00	10.3.0.201	63001	10.1.0.202	80
3	05/05/2020	01:00:00	10.3.0.101	63001	10.2.0.201	22
4	05/05/2020	02:00:00	10.3.0.101	63001	10.3.0.202	22

With regard to the system architecture for the deployment of distributed computing service components in clouds, the MC-SA (multi-cloud) architecture is applied in the deployment of the processing and storage components of DCS_1 and DCS_2, i.e., the processing components are hosted in $Cloud_1$ and the storage components in $Cloud_2$. Compared with the SC-SA (single-cloud) architecture, MC-SA is more flexible and presents a broader set of security challenges. The distributed computing service providers of DCS_1 and DCS_2 have more options of infrastructure services to choose from. They may choose to subscribe to processing and storage services provided by the same or different cloud service providers. In this case, they subscribe to a processing service provided by $Cloud_1$ and a storage service provided by $Cloud_2$. This also reduces the risk of vendor lock-in. As the networks connecting $Cloud_1$ and $Cloud_2$ are likely to be WANs or the Internet, the communication channels connecting the processing components hosted in $Cloud_1$ and the storage components hosted in $Cloud_2$ may not be secure and susceptible to threats and attacks caused by malicious attackers (e.g., Mal^4). An authentication solution designed for MC-SA based applications should be applicable to SC-SA based applications, but the reverse is not true. Therefore, MC-SA is chosen for this work.

The MR framework is chosen, as it is one of the most used Big Data processing models. It is highly versatile and can support a wide range of applications, including cyberthreat analysis jobs (such as the one addressed in this example and other jobs [135][136][137]). The map and reduce computations of MR can be tailored by end users to process on unstructured (e.g., raw sensor data), semi-structured (e.g., images with metadata tags), and structured data (e.g., text data in a tabular format). Although, in many cases, SQL-based applications can answer queries that can be answered by MR-based applications, SQL-based applications cannot process unstructured data like MR. In this example, the map tasks are used to filter out irrelevant network activities and to identify entries related to attacks on potentially compromised machines, and the reduce tasks are used to merge and sort the entries and to generate the reports. As MR is designed for deployment on generic hardware, it should be compatible with processing and storage services provided by $Cloud_1$, $Cloud_2$, and $Cloud_3$. In addition, MR shares many characteristics and has security implications with other Big Data processing models. An authentication solution designed for MR should also work on the other models as well.

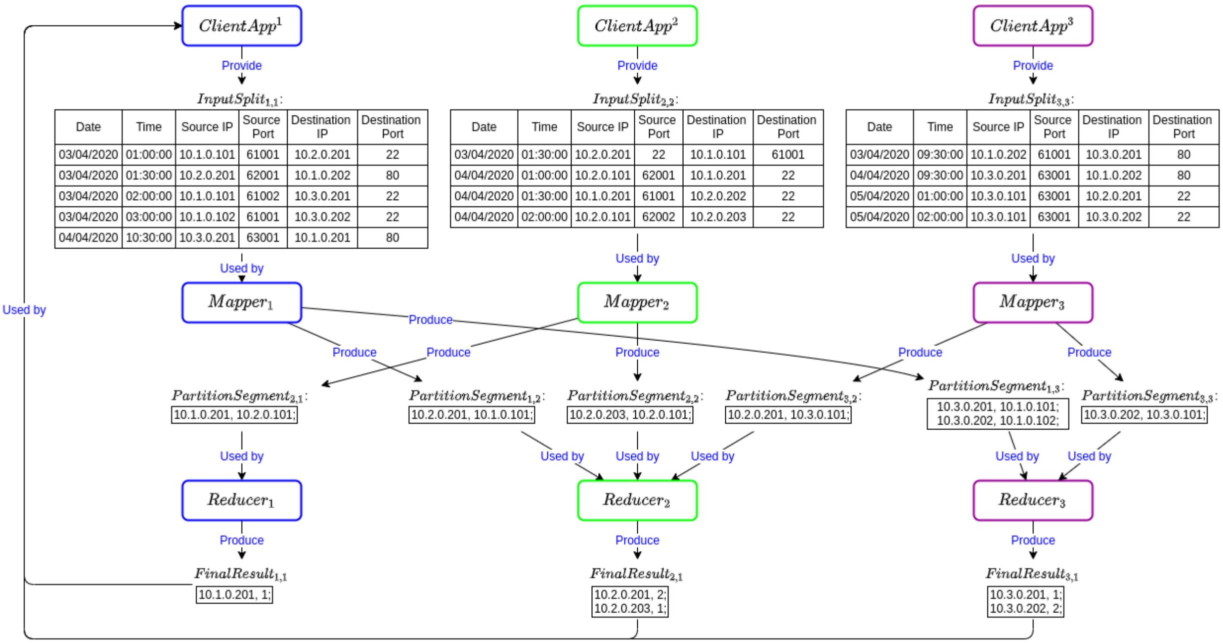
As discussed above, MR is chosen as the Big Data processing model for the example, DCS_1, DCS_2, and DCS_3 are MR services (hereafter referred to as MR_1, MR_2, and MR_3, respectively). For the execution of a job, the users ($User^1$, $User^2$, and $User^3$), from three respective collaborative organisations ($Organisation_1$, $Organisation_2$, and $Organisation_3$), use their ClientApps ($ClientApp^1$, $ClientApp^2$, and $ClientApp^3$, respectively) to communicate with their respective MR services. Three ResourceManagers ($ResourceManager^1$, $ResourceManager^2$, and $ResourceManager^3$), three NameManagers ($NameManager^1$, $NameManager^2$, and $NameManager^3$), three DataStores ($DataStore_1^1$, $DataStore_1^2$, and $DataStore_1^3$), four WorkerManagers ($WorkerManager_1^1$, $WorkerManager_2^1$, $WorkerManager_1^2$, and $WorkerManager_1^3$), one JobManager ($JobManager$), three Mappers ($Mapper_1$, $Mapper_2$, and $Mapper_3$), and three Reducers ($Reducer_1$, $Reducer_2$, and $Reducer_3$), are involved in the job execution. $WorkerManager_1^1$ and $JobManager$ are hosted on $WorkerNode_1^1$. $WorkerManager_2^1$, $Mapper_1$, and $Reducer_1$ are hosted on $WorkerNode_2^1$. $WorkerManager_1^2$, $Mapper_2$, and $Reducer_2$ are hosted on $WorkerNode_1^2$. $WorkerManager_1^3$, $Mapper_3$, and $Reducer_3$ are hosted on $WorkerNode_1^3$. This setting is similar to that shown in Figure 4.7 but without Mal^1, $Organisation^4$, and Mal^4 (as these entities are not involved in the job execution).

In this example, it is assumed that $User^1$ initiates the execution of the job. The query of the job is to identify any potentially compromised machines and how many times these machines have been connected to by the compromised machines in the past 30 days. This can be translated into the map tasks, i.e., to find the entries in the log files showing a connection from a compromised machine to a potentially compromised machine on port 22, and the reduce tasks, i.e., to count how many connections by the compromised machines have been made to each of the compromised machines. The job execution flow follows the steps outlined in Section 4.3.4.1. $User^1$ (as JobSubmitter) contacts and sends a request for the security log files ($File_2$ and $File_3$) to $User^2$ and $User^3$, respectively. $User^2$ and $User^3$ accept the request and upload their security log files onto the DFS cluster of their MR services, respectively. The security log files ($File_1$, $File_2$, and $File_3$) are the input of the job. The input will be divided into multiple items each assigned to a different Worker. The processing of the data is carried out in two phases, the map phase and the reduce phase. The output of the job is produced and ready for retrieval by the user at the end of the reduce phase. For ease of discussion, the notations for JobData objects (InputSplits, IntermediateResults, PartitionSegments, and FinalResults) used in this working example are described in Table 4.5.

Table 4.5: Notations for JobData objects used in the running example.

Symbol	Meaning
$InputSplit_{i,j}$	An InputSplit containing data supplied by $ClientApp^i$ and used by $Mapper_j$
$IntermediateResult_i$	An IntermediateResult produced by $Mapper_i$
$PartitionSegment_{i,j}$	A PartitionSegment produced by $Mapper_i$ and used by $Reducer_j$
$FinalResult_{i,j}$	A FinalResult produced by $Reducer_i$ and used by $ClientApp^j$

The input data are divided into 3 InputSplits ($InputSplit_{1,1}$, $InputSplit_{2,2}$, and $InputSplit_{3,3}$). Each of the InputSplits corresponds to a respective security log file ($File_1$, $File_2$, and $File_3$), and is assigned to a different Mapper ($Mapper_1$, $Mapper_2$, and $Mapper_3$, respectively). In the map phase, each of the Mappers identifies all the potentially compromised machines, the machines that have ever come into contact with any of the compromised machines (via port number 22), using the data in the given security log. Upon the completion of the map tasks, each of the Mappers produces an output file ($IntermediateResult_1$, $IntermediateResult_2$, and $IntermediateResult_3$, respectively). Each of these files contains multiple entries. Each entry is a pair of values, a destination IP address (pointing to the potentially compromised machine) and a source IP address (pointing to the compromised machine). For each $IntermediateResult_i$, where $i \in \{1, 2, 3\}$, the entries are partitioned into segments (PartitionSegments) based on the destination IP addresses (a segment for an IP address block). As a result, $IntermediateResult_1$ is partitioned into two PartitionSegments ($PartitionSegment_{1,2}$ and $PartitionSegment_{1,3}$), $IntermediateResult_2$ is partitioned into two PartitionSegments ($PartitionSegment_{2,1}$ and $PartitionSegment_{2,2}$), and $IntermediateResult_3$ is partitioned into two PartitionSegments ($PartitionSegment_{3,2}$ and $PartitionSegment_{3,3}$). Each of the PartitionSegments is assigned to a respective Reducer ($Reducer_1$, $Reducer_2$, and $Reducer_3$, respectively). In the reduce phase, each of the Reducers combines the PartitionSegments from the Mappers that are assigned to it and outputs a list of potentially compromised machines along with the number of connections (i.e., connection count) each potentially compromised machine has with the compromised machines. Upon the completion of the reduce tasks, each of the Reducers produces an output file ($FinalResult_{1,1}$, $FinalResult_{2,1}$, and $FinalResult_{3,1}$, respectively). The FinalResults are the output of the job execution. They are stored in DFS^1 and are ready for retrieval by $User^1$. The flow of the job execution is summarised in Figure 4.9. The operational steps are detailed in Section 7.2.


$ClientApp^1$
$ClientApp^2$
$ClientApp^3$
Provide
$InputSplit_{1,1}$:
$InputSplit_{2,2}$:
$InputSplit_{3,3}$:
Used by
$Mapper_1$
$Mapper_2$
$Mapper_3$
Produce
$PartitionSegment_{2,1}$: 10.1.0.201, 10.2.0.101;
$PartitionSegment_{1,2}$: 10.2.0.201, 10.1.0.101;
$PartitionSegment_{2,2}$: 10.2.0.203, 10.2.0.101;
$PartitionSegment_{3,2}$: 10.2.0.201, 10.3.0.101;
$PartitionSegment_{1,3}$: 10.3.0.201, 10.1.0.101; 10.3.0.202, 10.1.0.102;
$PartitionSegment_{3,3}$: 10.3.0.202, 10.3.0.101;
$Reducer_1$
$Reducer_2$
$Reducer_3$
$FinalResult_{1,1}$ 10.1.0.201, 1;
$FinalResult_{2,1}$ 10.2.0.201, 2; 10.2.0.203, 1;
$FinalResult_{3,1}$ 10.3.0.201, 1; 10.3.0.202, 2;

$InputSplit_{1,1}$:

Date	Time	Source IP	Source Port	Destination IP	Destination Port
03/04/2020	01:00:00	10.1.0.101	61001	10.2.0.201	22
03/04/2020	01:30:00	10.2.0.201	62001	10.1.0.202	80
03/04/2020	02:00:00	10.1.0.101	61002	10.3.0.201	22
03/04/2020	03:00:00	10.1.0.102	61001	10.3.0.202	22
04/04/2020	10:30:00	10.3.0.201	63001	10.1.0.201	80

$InputSplit_{2,2}$:

Date	Time	Source IP	Source Port	Destination IP	Destination Port
03/04/2020	01:30:00	10.2.0.201	22	10.1.0.101	61001
04/04/2020	01:00:00	10.2.0.101	62001	10.1.0.201	22
04/04/2020	01:30:00	10.1.0.201	61001	10.2.0.202	22
04/04/2020	02:00:00	10.2.0.101	62002	10.2.0.203	22

$InputSplit_{3,3}$:

Date	Time	Source IP	Source Port	Destination IP	Destination Port
03/04/2020	09:30:00	10.1.0.202	61001	10.3.0.201	80
04/04/2020	09:30:00	10.3.0.201	63001	10.1.0.202	80
05/04/2020	01:00:00	10.3.0.101	63001	10.2.0.201	22
05/04/2020	02:00:00	10.3.0.101	63001	10.3.0.202	22

Note:
- Each entry of a PartitionSegment contains a pair of Destination IP and Source IP.
- Each entry of a FinalResult contains a pair of Destination IP and a connection count.

Figure 4.9: The job execution flow of the running example.

The job execution is susceptible to threats and attacks from multiple sources at multiple points. Using Figure 1.1 and Figure 4.8 as a reference, in step 1, Mal^1, which is not authorised to initiate a collaborative job and access the shared datasets, may try to impersonate $User^1$ and submit a request for shared dataset and starting a job execution to $User^2$ and $User^3$ (T1). In step 2, when $User^1$ authenticates to $ResourceManager^1$ to submit a job execution request to MR_1, Mal^1 may intercept authentication credentials contained in messages transmitted between $Organisation_1$ and $Cloud_1$ or hijack a session of $User^1$ (T2). In step 3, Mal^1 may replace a session key in the intercepted message with a fraudulent key to mount MITM attacks (T4). During the job execution, in step 4, Mal^1 may write fraudulent data onto DFS^1 (T5) or tamper with JobData stored in DFS^1 (T6). Mal^1 may then falsely deny performing such actions (T7). When the FinalResults are uploaded to DFS^1 and ready for retrieval by $User^1$, in step 29, Mal^4 may intercept the FinalResult reading request by $User^1$ and replay the request to DFS^1 at a later time (T3).

Owing to the involvement of entities from different administrative domains, accomplishing authentication in this example is a challenging task. As shown in this example, collaborative organisations, MR service providers, and cloud service providers are from different administrative domains which may have varying levels of trust. The identities of entities in each of the domains are managed by the respective domains. Verifying the identity of an entity in another domain requires exchanging additional messages, adding overhead cost thus a delay to the job execution. For example, when $Reducer_2$ authenticates to $NameManager^1$ to write $FinalResult_{2,1}$, $NameManager^1$ has to contact the identity provider or the authentication service of MR_2 to verify the identity of $Reducer_2$ and $Reducer_2$ may use the same method to verify the identity of $NameManager^1$, further adding communication overhead. $JobManager$ and data processing components (i.e., $Mapper_1$, $Mapper_2$, $Mapper_3$, $Reducer_1$, $Reducer_2$, and $Reducer_3$) are job-dependent;

they are created when the job starts and terminated when the job finishes. This means that the identities of these components and credentials for verifying the identities should be established during the job execution process. As discussed earlier, attacks could be mounted on the job and the system at multiple points during the job execution. Thus, strong authentication protection should be provided to every interaction from when $User^1$ submits a request to start the job to when $User^1$ retrieves the FinalResults. As outlined in Section 4.3.4.1, the job execution consists of 29 operational steps and the number of interactions in each of the steps is dependent on the number of MR components involved in the step. In this example, with just only three MR services (each with one ClientApp, one ResourceManager, three Mappers, three Reducers, one NameManager, and one DataStore), there are a total of about 100 entity and data authentication instances taking place during the job execution. Therefore, an authentication solution designed for CBDC-MPC should be as efficient and scalable as possible to minimise negative impacts on the performance of Big Data computation.

4.7 An Overview of the MDA Framework

To support secure inter-organisational Big Data sharing and processing, we have designed a novel authentication framework, called the Multi-domain Decentralised Authentication (MDA) framework. The MDA framework provides two authentication services, entity and data authentication services, which satisfies all the specified requirements. The entity authentication service verifies the identities of entities involved in a job execution, ensuring that entities are whom they claim to be. It is also used for establishing credentials that are bound to the identities of the authorised entities. These credentials are necessary for other security services, including a data authentication service. The data authentication service ensures that JobData generated, processed, and used during the job execution can be traced back to their origins and are free from unauthorised modification, ensuring the authenticity of JobData and accountability of entities producing the JobData.

The MDA framework consists of one novel entity authentication framework and one novel data authentication framework. The novel entity authentication framework, called the Multi-factor Interaction based Entity Authentication (MIEA) framework implements the entity authentication service. The novel data authentication framework, called the Communication Pattern based Data Authentication (CPDA) framework, implements the data authentication service. Figure 4.10 shows the architecture of the MDA framework, highlighting the components of the MIEA and CPDA frameworks and how these components are collectively used to support authentication for MR based distributed computing system in the context of CBDC-MPC.

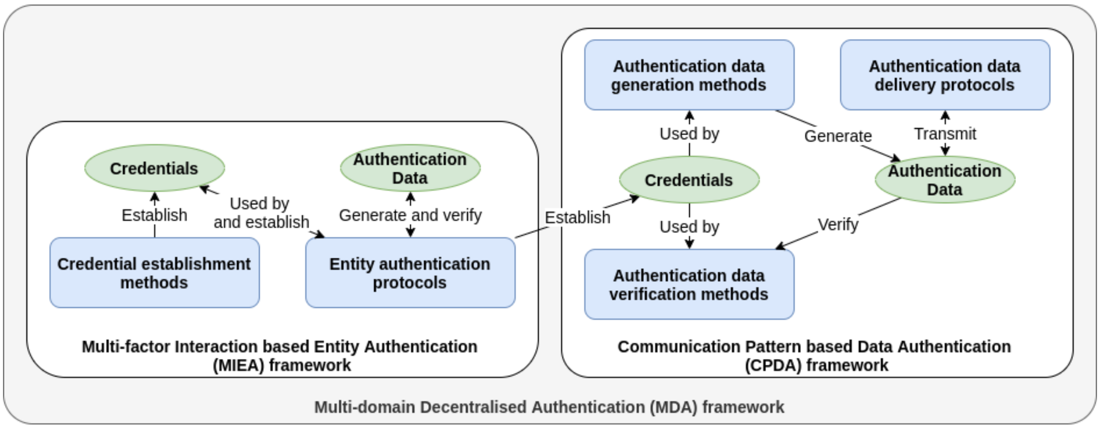

Authentication data generation methods
Authentication data delivery protocols
Used by
Generate
Transmit
Credentials
Authentication Data
Establish
Used by
Verify
Authentication data verification methods
Communication Pattern based Data Authentication (CPDA) framework
Credentials
Authentication Data
Establish
Used by and establish
Generate and verify
Credential establishment methods
Entity authentication protocols
Multi-factor Interaction based Entity Authentication (MIEA) framework
Multi-domain Decentralised Authentication (MDA) framework

Figure 4.10: MDA framework architecture.

As shown in the figure, the MIEA framework contains credential establishment methods and entity authentication protocols. The credential establishment methods are used to establish credentials on MR components. These credentials are then used by the entity authentication protocols to verify the identities of the two interacting entities. The entity authentication protocols verify the identities of the interacting entities by verifying the AuthData exchanged between the entities. Once the identities of the entities are verified, these protocols also facilitate the transmission of new credentials for subsequent authentication. The CPDA framework contains AuthData generation methods, AuthData verification methods, and AuthData delivery protocols. The AuthData generation methods use the credentials established during entity authentication to generate AuthData for JobData objects. The AuthData verification methods use the credentials to verify the authenticity of the objects against the generated AuthData. The AuthData delivery protocols are used to deliver these AuthData from one component to another component.

Figure 4.11 demonstrates an authentication flow when the MDA framework is applied. The authentication flow starts from when credentials for entity authentication are established on entities (i.e., MR components) to when a component consuming JobData objects verifies the authenticity of the objects assigned to it. The MIEA framework is used for entity authentication from Step 1 through to Step 5 and the CPDA framework is used for data authentication from Step 6 through to Step 8. Without losing generality, it is assumed that A and B are the interacting components, A is a data producer, and B is a data consumer. In Step 1, the credential establishment methods of MIEA are used to establish credentials needed for entity authentication. From Step 2 through to Step 5, the entity authentication protocols of MIEA are used to facilitate the mutual authentication (i.e., verification of the identities) of A and B. For the verification of the identity of A, in Step 2, A generates AuthData with its credentials. In Step 3, A then sends the AuthData to B. In Step 4, B verifies the received AuthData. If the verification result is positive, the identity of A is positively verified. The verification of the identity of B is carried out using the same steps as outlined in Step 2, Step 3, and Step 4. In Step 5, if the identities of both of A and B are positively verified, new credentials are established. With regard to data authentication, in Step 6, A generates AuthData for JobData objects it produces by using the AuthData generation methods of CPDA. In Step 7, the AuthData delivery protocols are used to transmit the AuthData from A to B. In

Step 8, B verifies the authenticity of the objects assigned to it against the received AuthData. If the verification result is positive, B is assured that the objects it consumes are authentic.

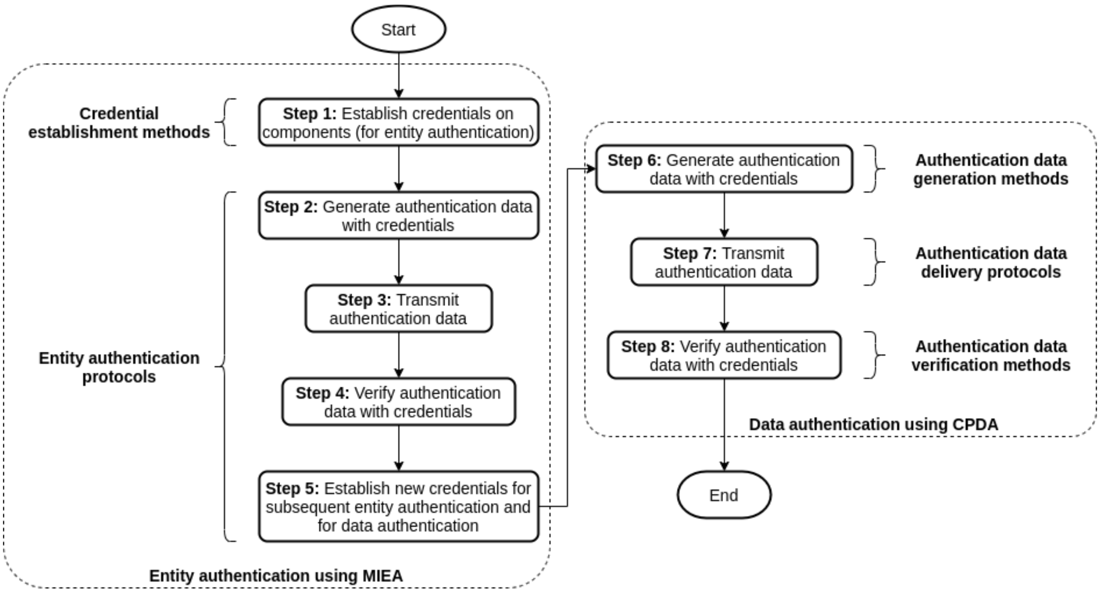

Start
Credential establishment methods
Step 1: Establish credentials on components (for entity authentication)
Step 2: Generate authentication data with credentials
Step 3: Transmit authentication data
Entity authentication protocols
Step 4: Verify authentication data with credentials
Step 5: Establish new credentials for subsequent entity authentication and for data authentication
Entity authentication using MIEA
Step 6: Generate authentication data with credentials
Authentication data generation methods
Step 7: Transmit authentication data
Authentication data delivery protocols
Step 8: Verify authentication data with credentials
Authentication data verification methods
Data authentication using CPDA
End

Figure 4.11: Authentication flow using the MDA framework.

Detailed literature reviews with regard to entity authentication and data authentication, the detailed descriptions of the MIEA and the CPDA frameworks, and their theoretical analyses and experimental evaluations are given in Chapter 5 and Chapter 6 , respectively.

4.8 Chapter Summary

This chapter has described a reference use case model used by this research. In formulating the use case model, it has examined possible system architectures and Big Data processing models, and selected suitable ones for constructing our use case model. As a result, the MC-SA architecture and the MR model are chosen for the use case model. Compared with SC-SA, MC-SA is more flexible and presents a broader sets of security requirements. Big Data processing models share many common characteristics. This means that an authentication solution designed for one model should also be applicable to other models. Although some models (e.g., Apache Spark) may perform better (in terms of job execution time) than MR under certain conditions, considering that MR is designed for deployment on generic hardware and that MR is one of the most widely used Big Data processing models and there are extensive supports and documents available to MR users, we have chosen MR to construct our generic use case model. Using the use case model, our threat analysis indicates that MR based data processing in this context is susceptible to threats caused by not only outsiders but also insiders. To counter the threats, this chapter has also specified a set of requirements to guide the design of our authentication solution. The formulation of this use case model is the first novel contribution (NC1) of this research work. The threat analysis on the model answers the research question (Q1). In addition, the observations made on the use case model show that some characteristics (e.g., communication patterns) can be used to improve the efficiency of the authentication solution. This has led to the design of our authentication solution, the MDA framework. An overview of the MDA framework has been presented in this chapter. The detailed descriptions and the evaluations of the components of MDA are to be presented in the following chapters, i.e., the MIEA framework for entity authentication in Chapter 5 and the CPDA framework for data authentication in Chapter 6.

The next chapter presents in detail a novel approach, an interaction based approach, to entity authentication which provides strong security protections against entity identity related threats efficiently to every interaction during the execution of a job.

Chapter 5
Multi-factor Interaction based Entity Authentication (MIEA) Framework

5.1 Chapter Introduction

This chapter presents a novel entity authentication framework, called the Multi-factor Interaction based Entity Authentication (MIEA) framework, which is part of the MDA framework. The MIEA framework is designed to provide entity authentication protection for every interaction taking place during a data processing job but with minimal overhead cost. The design of MIEA makes use of three main ideas. The first is Multi-factor Interaction based Authentication (MIA) in which credentials and an authentication method are determined based on the risk level associated with an interaction. The second is a Decentralised approach with Combined use of group-and-entity-dependent Symmetric keys (DCS) in which the distribution and the verification of credentials are carried out without the assistance of a central server and symmetric-key cryptosystems are used to generate, verify, and securely transmit credentials. The third is a Hierarchical Key Structure (HKS) where a limited set of credentials (keys) is used to securely distribute additional keys for subsequent authentication instances and to derive new keys for tasks (e.g., message authentication) facilitating the authentication. The MIEA framework has been extensively evaluated by using both theoretical analysis and experimental evaluation to demonstrate the effectiveness (the strength of protections), efficiency (the cost incurred in providing the protections), and scalability (the increase in cost in relation to the number of entities involved in a job execution). The theoretical analysis is conducted by using both informal and formal methods. The experiments are conducted by implementing authentication protocols and executing the implemented protocols on a testbed under different parameter settings.

In detail, Section 5.2 critically reviews existing entity authentication solutions at a technical level, highlighting knowledge gaps and areas for improvements. Sections 5.3, 5.4, and 5.5 respectively, describe high-level ideas, notations and design assumptions, and detailed description of MIEA. Sections 5.7, 5.8, and 5.9, respectively, report security analysis, theoretical, and experimental performance evaluations. The results are compared with those of the most related solutions. Lastly, Section 5.10 concludes the chapter.

5.2 Existing Entity Authentication Solutions

Existing entity authentication solutions designed for networked and distributed systems can be largely classified into two groups: those that are designed for non-MR based services (referred to as non-MR specific solutions) and those that are specifically designed for MR services (referred to as MR specific solutions).

5.2.1 Non-MR Specific Solutions

Remote Authentication Dial-In User Service (RADIUS) [29], Kerberos [34][138][139], and Security Assertion Markup Language (SAML) [140] are among the most commonly used authentication solution families, which are designed for networked systems. In addition, this section also describes the Needham-Schroeder-Lowe Public Key (NSLPK) protocol [31], one of the most studied and referred asymmetric-key based authentication protocols proposed in literature.

RADIUS [29] is a client-server networking protocol commonly used to implement an entity authentication service to support secure access to resources hosted in the network of an organisation. It supports both intra-domain and inter-domain (also called inter-realm or roaming) authentication. To authenticate a user requesting to access a service located in the same domain, the user sends a request to a gateway, called a RADIUS client. The client then passes the request to an authentication server, called a RADIUS server. Then the user, via the RADIUS client, communicates with the server to accomplish the authentication process. For inter-domain authentication where a user and the requested service are in different domains and each independently managed by their respective domains, a local authentication server (also called a proxy server) of the user's domain acts as a proxy between the user and a remote authentication server (also called a master server) in the domain of the requested service. RADIUS has a number of limitations [141][142]. One of the limitations is that it does not have provisions for congestion control, so it has scalability and reliability issues. Eduroam [143][144], a secure world-wide roaming access service developed for the international research and education community, improves on the scalability problem by allowing the use of multiple RADIUS servers and constructing the servers into a hierarchical network. Another peer-to-peer based solution, called Diameter [30][145], was proposed to address some of the limitations of RADIUS. Diameter uses a number of measures, namely the support of a reliable transport mechanism (e.g., the Stream Control Transmission Protocol (SCTP) [146]), a fail-over procedure, and a capability negotiation facility, to improve on the reliability, scalability, and compatibility issues of RADIUS. It also provides a higher level of security protection (e.g., hop-to-hop and end-to-end secure communication channel) than RADIUS. This is done by the use of transport layer security solutions, such as the Internet Protocol Security (IPsec) protocol suite [147] and the Transport Layer Security (TLS) protocol [148]. However, as the above solutions are only a transport facility and does not have a built-in confidential channel, they need external solutions to protect the confidentiality of any authentication credentials and data sent over the channel. The overhead cost introduced by these solutions could be excessive, particularly when multiple interactions (each with a different pair of entities) need to be authenticated and each authentication is only for transmitting a small amount of data (e.g., service status report), as in the case of our CBDC-MPC context. In addition, there is also a usability issue, as each authentication instance may require the user to input his/her credential manually (e.g., if password based credentials are used), and this is impractical for Big Data processing where there are many component-to-component interactions and the process of executing a job may last for a long time.

Kerberos [34][138][139], a symmetric-key based entity authentication solution, is particularly suited for Single Sign-on (SSO) in an organisational environment. With Kerberos, a user only uses one password to gain access to multiple service servers, with minimal exposure of the password. It achieves this by introducing ideas of temporary secrets and a hierarchical secret structure. In this structure, a user's password is used to derive a master key. The master key is only used for secure distribution of temporary secrets (keys). Access to a service server is granted if the user (via the user's client) demonstrates the knowledge of the corresponding temporary secret. This approach makes the authentication service more secure as users' passwords are never sent over networks. In addition, the number of interactions required from a user is reduced. This is because, once a ticket granting ticket (an encrypted secret in the mid-level of the key hierarchy) is acquired, the user's client can acquire service tickets (an encrypted secret in the bottom-level of the key hierarchy) on behalf of the user, making the system more user friendly than the RADIUS based solutions. However, owing to the distribution of the temporary secrets, each new service access (or each interaction with a service entity) requires 3 to 5 messages to complete a mutual authentication process. This is excessive both in terms of computational and communication costs, particularly if a user needs to access (or interact with) multiple service entities, and each such interaction is just for transmitting small amount of data. For inter-domain authentication, the authentication servers of participating domains are structured using a hierarchical tree similar to that used in the Domain Name System (DNS) [149]. For a user to gain access to a service server in another domain, the authentication process requires traversing the tree of the authentication servers. This introduces a number of additional messages, further increasing computational and communication costs.

SAML [140] is particularly suited for inter-domain authentication in a collaborative environment where each domain represents a different organisation. With the use of SAML, participating organisations do not have to use the same authentication solution and users from different organisations can authenticate themselves with their home organisations while accessing resources provided by external organisations. This is done by using a standard for exchanging security data (e.g., authentication and authorisation data) among different domains, in particular, between a service user and a service provider. Examples of SAML based solutions are Active Directory Federation Services [81], Access Policy Manager [33], and Shibboleth [32]. However, to authenticate a particular user, another layer of authentication is required to authenticate the user to the user's organisation before the user's organisation asserts the user's identity and the associated attributes to a resource providing organisation. For these reasons, SAML is not suited to our CBDC-MPC context.

NSLPK [31] is an authentication protocol based on asymmetric-key cryptography. Unlike symmetric-key based solutions such as Kerberos, with NSLPK, a user and a service server do not have to establish a share secret prior to authentication. However, a trusted third party (e.g., a certificate authority) is required for certifying and distributing the public keys of the user and the service server. Mutual authentication of the user and the service server is done by demonstrating the knowledge of the corresponding private keys. The user and the service server exchange a series of messages containing challenges and responses which are protected (encrypted) with the public key of the other entity. Only the intended user or service server can read the public-key-protected challenge with the respective private key and be able to generate the corresponding response for the challenge. NSLPK introduces a high level of computational cost due to the cost of asymmetric-key operations, which is more expensive than that of symmetric-key operations [45]. This is not suitable for Big Data processing in this context which has a stringent requirement for timeliness.

5.2.2 MR Specific Solutions

Based on the approaches used, MR specific authentication solutions can be largely classified into three groups, password based, symmetric-key based, and asymmetric-key based.

Password based solutions are the most used for gate-level authentication. Gate-level authentication authenticates a user when the user makes a request to access, or to interact with, a service. As passwords are vulnerable to theft, some solutions, such as [79] and [80], have been proposed to make password-stolen attacks more difficult. The approach used in these solutions is to divide an authentication credential into multiple pieces and store each of the pieces on a different server. In [79], a user's credential is transformed and divided into multiple pieces by using a mathematical method that is based on the properties of triangles. These pieces are separately stored on three different servers, one authentication server and two backend servers. The solution proposed in [80] generates multiple authentication tokens from a single password. To generate the tokens, a user is issued with a new one-time secret key (for a user's password) when the user is registered to the system and when the user logs out of the system each time. The password and the secret key are used to encrypt each other, respectively, generating two tokens. The tokens are then stored on two different servers, an authentication server and a backend server. Password based solutions usually require users to manually enter their credentials into the system at the time of authentication. They are not readily applicable to component-to-component authentication.

A symmetric key, similar to a password but with a higher entropy, is also commonly used to achieve authentication. A symmetric key (either a group key if it is shared among a group of entities or a pairwise key if it is shared between two entities) is used to generate and verify AuthData. Only the entities knowing the key can generate AuthData that can be verified with the same key, ensuring the authenticity of the entity. Apache Hadoop [120], one of the most prominent opensource Big Data solutions, employs a symmetric-key based authentication solution, a Kerberos based system. Kerberos is used for authentication between a user (via ClientApp) and the MR service (i.e., between ClientApp and ResourceManager), ClientApp and NameManager, ClientApp and JobManager, and JobManager and WorkerManagers. Additional authentication tokens are introduced to complement Kerberos and reduce the use of Kerberos credentials (i.e., service tickets). Three sets of tokens are introduced, namely delegation tokens, block access tokens, and job tokens. A delegation token is used by components (e.g., ClientApp and Workers) to request services (e.g., file listing) from NameManager. Similarly, a block access token is used by components to request access to data blocks from DataStore. A job token is used by Mappers and Reducers to authenticate themselves to WorkerManagers. The Apache Hadoop solution only support the deployment of an MR service in a single domain.

To support the deployment of an MR service in a multi-cloud environment, an entity authentication solution, called a Virtual Domain based Authentication Framework (VDAF) [84][36], was proposed. In VDAF, a novel MR Layered Authentication Model (MR-LAM) is proposed. The model consists of two layers of authentication. One is for authenticating components serving multiple jobs. The other is for authenticating components serving a particular job. The components serving a job form a virtual domain, called JobDomain. VDAF enforces authentication at every interaction during a job execution. This is achieved by using a Password and Token based Multi-point Multi-factor Authentication (PT2M-AuthN) method. It implements two main ideas, one is the principle of the separation of duty-and-credential, and the other is a key wrap-and-swap operation to support mutual authentication.

Symmetric-key based solutions have limitations. One is that establishing a pairwise credential (e.g., a key) between every pair of interacting entities is resource-consuming, especially when entities are from different domains with varying levels of trust. The overhead cost of establishing a pairwise symmetric key increases at a polynomial rate as the number of entities increases. The other is that the sharing of a symmetric key among a group of entities limits the accountability of the entities. In such cases, it is difficult, if not impossible, to hold any of the entities accountable for their actions.

Asymmetric-key based solutions are often used for cross-domain authentication, or in a multi-domain (e.g., multi-cloud) environment, where there is a lack of trust among the entities from different domains. In an asymmetric-key based solution, two keys (a private key and a public key) are used to accomplish an authentication process. As long as the private key is kept secret and the public key is certified, the identity of the entity owning the keys can be verified thus authenticated. Although asymmetric-key based solutions are computationally more expensive, they can reduce the number of keys an entity has to store and manage, and relieve the key establishment issue existed with symmetric-key based methods. These benefits are particularly significant when the number of entities one needs to interact with is large. In [150], an asymmetric-key based solution was proposed for a multi-cloud based MR architecture called G-Hadoop [151]. This solution supports authentication between users and a management component, called a master node, and between the master node and data processing components, called slave nodes. A central certificate authority is introduced to issue and certify public-key certificates for all the components. The authentication between a user and the master node is done by using a password based method, and the authentication between the master node and each of the slave nodes is done by using an asymmetric-key based method similar to the handshaking process of Secure Sockets Layer (SSL) [152]. Trusted Scheme for Hadoop Cluster (TSHC) [153] introduces a Trusted Compute Base (TCB) component to facilitate asymmetric-key based authentication in MR. TCB is used to generate asymmetric keys for MR components by using an Identity-Based Encryption (IBE) method. In [154], an Efficient Authentication Protocol for Hadoop (HEAP) was proposed. In HEAP, two authentication servers are used to authenticate a user to the system. The user is issued a set of credentials. A subset of the credentials is stored on each of the servers. During the authentication process, AuthData exchanged between the user and the two servers are generated by using an Elliptic-Curve Cryptography (ECC) based asymmetric system (for signature generation) and an Advanced Encryption Standard (AES) based symmetric system (for credential encryption). All the above solutions are vulnerable to software based attacks, e.g., by exploiting the vulnerabilities of unpatched applications or operating systems to steal authentication credentials from the memories or storage of the machines hosting the entities.

To enhance protection against software based attacks, a special hardware module, called Trusted Platform Module (TPM), is used. TPM provides a number of cryptographic functions (e.g., key generation, random number generation, and signature generation and verification) as well as persistent and volatile memory to store both long-term (e.g., root keys) and short-term (e.g., temporary secret keys) security data. In such a solution, a baseline state (i.e., an uncompromised state) of a component is established by measuring the software installed and the configurations used. The digest of the baseline state is then signed and stored in the volatile memory of the TPM of the component, ensuring the integrity of the component. The state of the component may be attested by another component (e.g., a management component such as JobManager) periodically or before an interaction is started. Some examples of such TPM based solutions include [155] and [156]. Owing to their limited computational capabilities, the use of TPMs in performing a large number of cryptographic operations (e.g., attesting the integrity of a large number of components) may increase the execution time and lowering the performance of a job execution.

5.2.3 What is Missing

We have critically analysed the existing work presented above against the requirements with regard to identity protection, i.e., (FR1), (FR2), (FR3), (SR1), (SR2), (SR3), (SR4), (PR1), and (PR2), as specified in Section 4.5. The result of the analysis is summarised in Table 5.1. Based on the analysis result, we can make the following remarks.

- None of the existing entity authentication solutions discussed above provides all the specified functional requirements, i.e., full-cycle (FR1), cross-domain (FR2), and automated (FR3) authentication protection. The non-MR specific solutions are mainly designed for gate-level authentication to deter unauthorised entities from accessing the service. They provide authentication services to authenticating users to the service, not mutual authentication among service components. One important characteristic of our use case model is that data processing components are allocated and assigned to a job when the job is accepted by the MR service and these components are terminated and released when the job finishes. This implies that the identities of the data processing components are not known before the job is executed. This characteristic was not specifically considered in the design of the existing solutions, such as Kerberos and NSLPK, which support component-to-component authentication. Therefore, when being applied to the CBDC-MPC context, the execution time of the job may suffer due to the costs incurred in generating, transmitting, and verifying AuthData tokens. None of the MR specific solutions (with the exception of VDAF) supports interaction-level authentication for entities throughout the whole cycle of the job execution. Although VDAF provides full-cycle protection, it is not designed for cross-MR service authentication, thus, not readily applicable to our use case.
- Password-based solutions are not suited to our use case as they require user intervention which lower usability in this CBDC-MPC context where users may not always be present. These password-based solutions are typically more suitable for applications that require only gate-level authentication.

- Symmetric-key based solutions, as discussed earlier, have two main limitations. The first is that establishing pairwise keys for every pair of components is resource-consuming and the cost incurred increases at a polynomial rate as the number of the components increase. The second is that using group keys cannot protect against insider threats as any members of the group can use the keys to gain access to protected resources. This also nullifies the protection of accountability.
- The overhead costs introduced by asymmetric-key based solutions are too high for large-scale collaborative Big Data processing which has a stringent requirement for timeliness. Although TPMs can enhance the security protection and make software-based attacks much more difficult, the number of public clouds supporting such specialised modules is limited. Furthermore, TPMs may increase the risk of creating performance bottleneck and further lower the performance of the job owing to its limited computation power and resources.

Table 5.1: Related entity authentication solutions.

	Requirements								
Approaches	**(FR1)**	**(FR2)**	**(FR3)**	**(SR1)**	**(SR2)**	**(SR3)**	**(SR4)**	**(PR1)**	**(PR2)**
Non-MR specific									
RADIUS [29], Eduroam [143][144], Diameter [30][145]	×	√	∗	∗	∗	∗	∗	×	∗
Kerberos [34][138][139]	√	√	∗	√	√	√	×	∗	∗
SAML based [81][33][32]	×	√	∗	∗	√	∗	∗	×	√
NSLPK [31]	√	√	√	√	√	√	×	×	√
MR specific									
Password based [79][80]	×	×	×	×	√	×	×	×	∗
Symmetric-key based [84][120] [36]	√	×	∗	√	√	√	×	∗	∗
Asymmetric-key based [150][153][154][155][156]	×	√	∗	√	√	∗	∗	×	∗

Notes:

√: Requirement is addressed.

∗: Requirement can be addressed with additional plug-in modules or minor modifications, or there is room for improvement.

×: Requirement is not addressed.

To improve on the existing solutions and satisfy all the specified requirement, we have proposed a novel entity authentication framework, the MIEA framework. The remaining of this chapter explains high-level ideas, the detailed description, and the evaluation of the framework.

5.3 High-level Ideas

MIEA is designed for CBDC-MPC, which is characterised by the following characteristics: (1) each job execution potentially involves a large number data processing components; (2) these components may be from different administrative domains and could be governed by different policies; and (3) the data processing components are ephemeral (job-dependent), they are dynamically created and deployed in the systems when a job is submitted and terminated when the execution of the job is completed. In addition, it should also be emphasised that, as such systems are intended for Big Data computation, there is a stringent requirement for timeliness of job executions.

The design of MIEA makes use of three main ideas. The first is Multi-factor Interaction-based Authentication (MIA), i.e., the number of factors (keys) used to authenticate two entities involved in an interaction are determined based on the risk level associated with, and the purpose of, the interaction. This idea is applied to every interaction taking place during a job execution. In the context of CBDC-MPC, two risk levels, a high-risk level and a low-risk level, have been identified. The high-risk level is tagged to initial interactions and the low-risk level to non-initial (subsequent) interactions. Owing to the impact of allowing remote (and potentially untrustworthy) entities to access local resources which may contain sensitive and high-value data, initial interactions, such as an initial attempt to access data stored in DFS clusters by a data processing component, introduce a higher level of risks, particularly when data providing and consuming components are from different organisations or domains (e.g., hosted in different clouds). If two entities of an interaction are from different clouds, they are more likely being connected via public networks, such as the Internet, which are vulnerable to a broader range of threats than private networks. In addition, initial interactions typically involve entities that have yet established any trust or shared secrets and these interactions are usually used to establish such secrets. If initial interactions, or the secrets being established during the initial interactions, are compromised, the security of subsequent interactions will also be put at risk. For these reasons, a stronger level of protection, or a higher level of assurance, should be obtained during authentication, thus, applying two-factor authentication. Using two factors doubles the effort needed by an attacker to successfully mount an attack on an authentication process, as the attacker has to compromise both factors. Subsequent interactions, on the other hand, use temporary secrets established in the authentication of preceding interactions. They may impact on a limited set of interactions should the temporary secrets be compromised. Thus, they experience a lower level of risks, so one-factor authentication can be applied.

The two-factor authentication is accomplished based on our observation that, usually, a secret could be established among a group of entities sharing a common interest. This group secret, along with a unique (pairwise) secret shared between two interacting entities can facilitate two-factor authentication. In other words, in each two-factor authentication instance, two credentials are used, one is a group key and the other is a pairwise key. A group key is shared among the members of a particular group (e.g., a domain or a cluster). This group key is used to deter outsider threats, i.e., threats caused by entities external to the group. A pairwise key is shared between two interacting entities. It is used to counter insider threats, i.e., threats caused by entities in the same group. This pairwise key also narrows the accountability to the two entities sharing the key. In each one-factor authentication instance, a pairwise key established during the preceding two-factor authentication is used. In MIA, the credentials used for each authentication instance are selected from a pool of credentials already being established at multiple points as the job execution progresses. Furthermore, compared with the solutions that support only gate-level authentication, applying authentication at every interaction, i.e., at the interaction level, can lower the risk of systems being compromised. This approach shortens the time window during which an attacker can launch an attack against a system, making it harder to compromise the system.

The second idea is a Decentralised approach with Combined use of group-and-entity-dependent Symmetric keys (DCS). The 'decentralised approach' is implemented in two dimensions. The first is the decentralised distribution of keys. The distribution of group keys is handled by a trustworthy entity in each respective group. For two entities involving in an initial interaction, their pairwise key is distributed by a trusted third party via the use of an existing mechanism or via prior authentication (e.g., during initial interactions) with other entities. For a non-initial interaction, a pairwise key sharing between two interacting entities is established during the initial interaction. The second dimension of the 'decentralised approach' lies in authentication verifications. All such verifications are performed by the interacting entities. Owing to the characteristics of the distributed setting and the hybrid use of group and pairwise keys, this approach has an inherent feature of the separation of duties, or entity segregation, i.e., entities are compartmentalised into groups and a different group key is dispatched to each group. The more distributed the setting is, or the more domains or clusters the setting has, the more groups there are, thus the more separated the duties. This feature can limit the impact should an entity be compromised or should a key be exposed. In addition, the least privilege is applied in the usage of keys. Each entity obtains only a minimal set of keys needed to perform its function.

The number of interactions among a large number of entities will be large, the cost of the interaction-level authentication could thus be high. To reduce the cost, the computational complexity of authentication verification should be as low as possible. For this reason, we have chosen to use a symmetric-key based authentication method. In the MR context, most entities are job-dependent (e.g., Mappers are created before the map phase and destroyed once they finish their map tasks). In contrast, with an asymmetric-key based method, the costs of credential establishment and authentication verification are computationally expensive and inefficient. The delay introduced by the authentication operations may hinder the performance of the underlying system, particularly for a time-sensitive data processing system such as the one addressed in this paper.

The third idea is the use of a Hierarchical Key Structure (HKS). This idea is implemented in two dimensions. The first is the distribution of keys for different interactions. Keys in a higher level of the key hierarchy are used to securely distribute keys in a lower level of the key hierarchy. This is based on an observation that there are multiple tasks (e.g., data access and resource allocation) during a job execution, each task consists of multiple interactions, and these interactions are taking place in order. For example, a data processing component has to communicate with NameManager before sending a request to a DataStore to access job data. In this case, the keys for the component to authenticate itself to the DataStore can be established during the authentication of the component and the NameManager. The second dimension of HKS is key diversification. Key diversification refers to the generation (derivation) of new keys (i.e., keys in a lower level of the key hierarchy) from a limited set of master keys (i.e., keys in a higher level of the key hierarchy). These new keys are used to accomplish the specified security requirements (e.g., to ensure message authenticity). This is done by using a key derivation function, such as a HMAC-based Key Derivation Function (HKDF) [91]. This idea of HKS makes mounting an attack on an authentication process harder as different sets of keys are used for different interactions, and for each interaction, different keys are used for different authentication tasks. In addition, the scope of impact should keys be compromised can be narrowed down. This is because theft of keys in the lower level of the key hierarchy should not reveal keys in the higher level of the key hierarchy, and the exposure of keys used in one interaction group does not affect the keys used in another interaction group.

In the following, we use an example, as shown in Figure 5.1, to explain, at a high level, how the three ideas described above are implemented. From the figure, there are six components serving two different jobs, $JobDomain_1$ and $JobDomain_2$. $JobDomain_1$ consists of a DPS cluster and a DFS cluster. The DPS cluster contains one $JobManager$ and two Mappers, $Mapper_1$ and $Mapper_2$. The DFS cluster contains one $NameManager$ and one $DataStore$. $JobDomain_2$ contains one Mapper, $Mapper_3$. $Mapper_2$ initially holds two sets of secret keys established for authentication to $JobManager$ and $NameManager$ (the least privilege). The first set consists of gk_1 and k_{jm,m_2} and the second consists of gk_2 and k_{nm,m_2}. gk_1 and gk_2 are group keys whereas k_{jm,m_2} and k_{nm,m_2} are pairwise keys. The two sets of keys are, respectively, issued by $ResourceManager$ and $NameManager$ (DCS: decentralised distribution of keys). They are, respectively, used for the authentications between $Mapper_2$ and $JobManager$ and between $Mapper_2$ and $NameManager$ (DCS: decentralised authentication verifications). The idea of MIA is applied in the authentication of $Mapper_2$ to each of $JobManager$, $NameManager$, and $DataStore$. When $Mapper_2$, respectively, initiates the first interactions with $JobManager$, $NameManager$, and $DataStore$ (interactions 1, 2, and 4), two-factor authentication is applied (i.e., both a group key and a pairwise key are used). For the subsequent interaction between $Mapper_2$ and $NameManager$ (interaction 3), only one pairwise key $tk_{nm,m2}$ is used. The first dimension of HKS is applied in the distribution of tk_{nm,m_2} and $k_{m_2,ds}$. During the interaction 2, $Mapper_2$ is issued with two new pairwise keys, tk_{nm,m_2} and $k_{m_2,ds}$, by $NameManager$ for authentication of succeeding interactions. $Mapper_1$ (assuming it is compromised) cannot successfully impersonate $Mapper_2$ when it sends a service request to $JobManager$ (interaction 5) as it does not know k_{jm,m_2}. Similarly, $Mapper_3$ (assuming it is also compromised) cannot successfully impersonate $Mapper_2$ when it sends a service request to $NameManager$ (interaction 6) as it knows neither gk_2 nor k_{nm,m_2}. The use of key diversification (the second dimension of HKS) will be explained later on in Section 5.5.

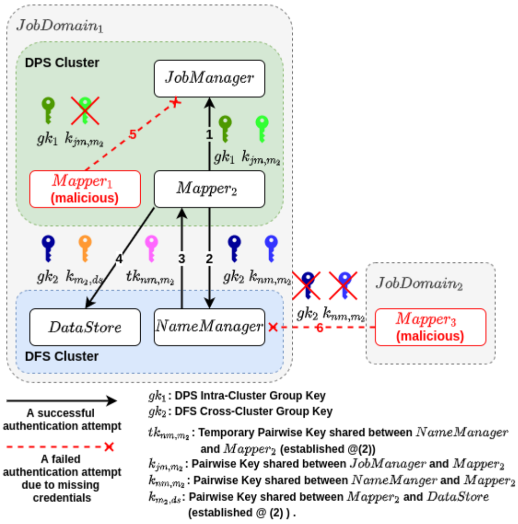

JobDomain1
DPS Cluster
JobManager
gk1
kjm,m2
5
1
gk1
kjm,m2
Mapper1 (malicious)
Mapper2
4
3
2
gk2
km2,ds
tknm,m2
gk2
knm,m2
gk2
knm,m2
JobDomain2
DataStore
NameManager
6
Mapper3 (malicious)
DFS Cluster
A successful authentication attempt
A failed authentication attempt due to missing credentials
gk1 : DPS Intra-Cluster Group Key
gk2 : DFS Cross-Cluster Group Key
tknm,m2 : Temporary Pairwise Key shared between NameManager and Mapper2 (established @(2))
kjm,m2 : Pairwise Key shared between JobManager and Mapper2
knm,m2 : Pairwise Key shared between NameManger and Mapper2
km2,ds : Pairwise Key shared between Mapper2 and DataStore (established @ (2)) .

Figure 5.1: An example showing how the ideas used in designing MIEA are applied.

5.4 Design Assumptions and Notations

This section details the design assumptions and the notations used in the design of MIEA.

5.4.1 Design Assumptions

The following assumptions are used in the design of MIEA.

(EAS1) Users are already authenticated prior to accessing the MR service. Collaborative organisations have established trust and secure communication channels with each other.

(EAS2) In each MRDomain, the MR components that are job-independent, i.e., ClientApps, ResourceManager, NameManager, WorkerManagers, and DataStores, are already registered and authenticated to the MR service. ClientApps, NameManager, and WorkerMangers have established a pairwise key with ResourceManager. DataStores have also established pairwise keys with NameManager.

(EAS3) The ResourceManagers of the collaborative MR services have established trust and pairwise keys with each other.

(EAS4) The time of all components are synchronised to the same time source.

(EAS5) Protection against message loss is provided by a lower layer of the network protocol stack (e.g., the Transmission Control Protocol (TCP)).

5.4.2 Notations

The notations used in the description of MIEA are shown in Table 5.2.

Table 5.2: Notations used in the description of MIEA.

Symbols	Meanings
C	The numbers of collaborative organisations and MRDomains
W, D	The numbers of WorkerNodes and DataNodes in each MRDomain
M, E	The numbers of Mappers, Reducers
I, R	Initiator, Respondent
c^i, rm^i, nm^i	ClientApp, ResourceManager, NameManager of the ith MRDomain
wm_u^i, ds_v^i	The uth WorkerManager, vth DataStore of the ith MRDomain
jm	JobManager serving the current JobDomain
m_a, r_b	The ath Mapper, bth Reducer serving the current JobDomain
ok^i	An OrgDomain key of the ith OrgDomain
jk	A JobDomain key of the current JobDomain
pik^i	A DPS Intra-cluster key of the ith MRDomain
pck^i	A DPS Cross-cluster key of the ith MRDomain
dfk^i	A DFS Cross-cluster key of the ith MRDomain
$pmk_{x,y}$	A primary key shared between x and y
$tkt_{x,y}^z$	A ticket containing $pmk_{x,y}$ issued by z
$sck_{x,y}$	A secondary key shared between x and y
$slk_{y,z}$	A sealing key shared between y and z
$ssk_{x,y}$	A session key shared between x and y
$mk_{x,y}$	MAC key
$ck_{x,y}$	Credential encryption key used by x and y
$@n$	Operational step number n as labelled in Figure 4.8
$S(x)$	The size of x in bytes (B).

Notes:
- $i, j \in \{1,\ 2, \dots, C\}$, $i \neq j$, $u \in \{1,\ 2, \dots\ ,\ W\}$, $v \in \{1,\ 2, \dots, D\}$, $a \in \{1,\ \ 2, \dots, M\}$, $b \in \{1,\ 2, \dots, E\}$
- Without losing generality, it is assumed that JobSubmitter is in $OrgDomain_1$ and the job is submitted via c^1 in $MRDomain_1$.

5.5 MIEA in Detail

This section describes in detail the MIEA framework. It gives an overview of MIEA, and then explains the components of MIEA. Subsequently, it puts together all of the components and shows how they are used to facilitate the authentication of every interaction during a job execution.

5.5.1 An Overview of the MIEA Architecture

The MIEA framework consists of three building blocks, i.e., credentials, credential establishment methods, and entity authentication protocols. An overview of the MIEA architecture is depicted in Figure 5.2.

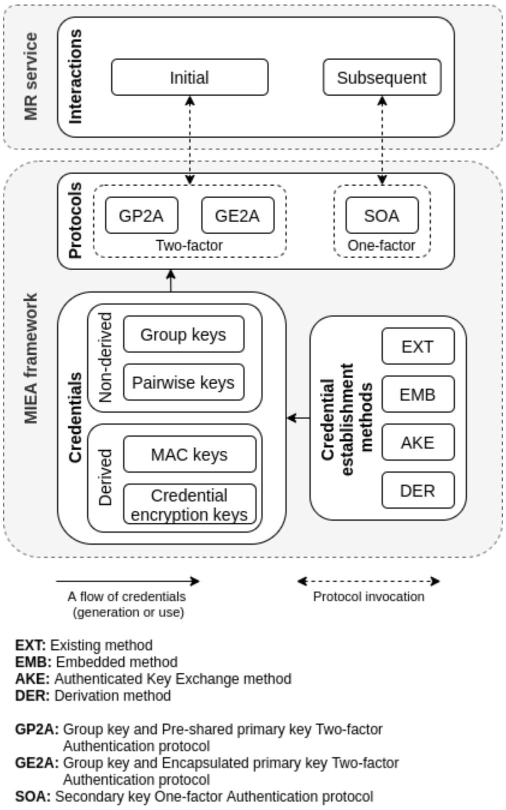

MR service
Interactions
Initial
Subsequent
MIEA framework
Protocols
GP2A
GE2A
Two-factor
SOA
One-factor
Credentials
Non-derived
Group keys
Pairwise keys
Derived
MAC keys
Credential encryption keys
Credential establishment methods
EXT
EMB
AKE
DER
A flow of credentials (generation or use)
Protocol invocation
EXT: Existing method
EMB: Embedded method
AKE: Authenticated Key Exchange method
DER: Derivation method
GP2A: Group key and Pre-shared primary key Two-factor Authentication protocol
GE2A: Group key and Encapsulated primary key Two-factor Authentication protocol
SOA: Secondary key One-factor Authentication protocol

Figure 5.2: An overview of the MIEA architecture.

As shown in the figure, credentials used in authentication are classified into two groups, non-derived keys (group keys and pairwise keys) and derived keys (MAC keys and credential encryption keys). The credentials (keys) are established on each of the MR components by using one of the four methods: the existing method, the embedded method, the authenticated key exchange method, and the derivation method. The methods are selected based on the characteristics and functions of the components. To support two classes of interactions (initial and subsequent interactions), we have designed three entity authentication protocols, namely the Group key and Pre-shared primary key Two-factor Authentication (GP2A) protocol, the Group key and Encapsulated primary key Two-factor Authentication (GE2A) protocol, and the Secondary key One-factor Authentication (SOA) protocol. As indicated by the names, GP2A and GE2A are designed for initial interactions whereas SOA is designed for subsequent interactions. The credentials, credential establishment methods, and entity authentication protocols will be described in detail in the following subsections. For ease of discussion, an entity initiating an interaction is referred to as an initiator, and the other interacting entity is referred to as a respondent.

5.5.2 Credentials

To support authentication of complex interactions during the execution of a job, multiple classes of credentials are introduced. The classifications of the credentials are summarised in Figure 5.3.

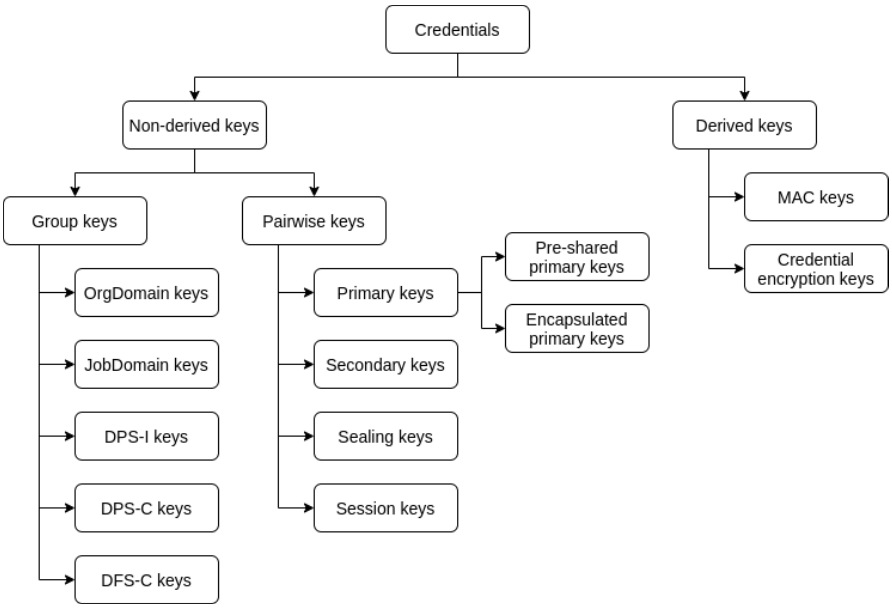
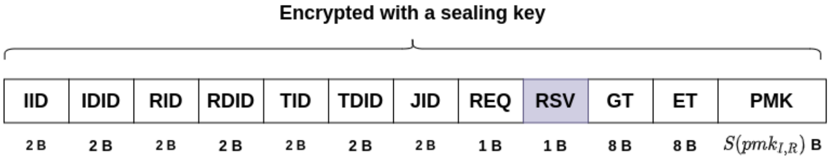


Figure 5.3: The classifications of credentials used in MIEA.

As shown in the figure, at the highest level, there are two groups of credentials, non-derived and derived keys. Non-derived keys are freshly generated independent of other keys, whereas derived keys are derived from non-derived keys and other data (e.g., nonces). Non-derived keys are classified into group keys and pairwise keys. Derived keys are classified into MAC keys and credential encryption keys.

Based on the organisation of entities involved in a job execution, group keys are further classified into five classes, namely OrgDomain keys, JobDomain keys, DPS Intra-cluster (DPS-I) keys, DPS Cross-cluster (DPS-C) keys, and DFS Cross-cluster (DFS-C) keys. An OrgDomain key is shared between ClientApps and ResourceManager of the same MR service. It is used to authenticate a ClientApp to the ResourceManager when a job is submitted. A JobDomain key is shared and used for authentication among ResourceManagers serving a particular job but are in different MR services. It is established by the users (through their respective ClientApps) prior to the submission of a collaborative job and dispatched to the ResourceManagers by the respective ClientApps along with job submission requests. A DPS-I key is shared and used for authentication among ResourceManager, WorkerManagers, and JobManagers hosted in the same DPS cluster. ResourceManager creates and dispatches the key to the WorkerManagers when they are registered to the MR service. The key is embedded into the JobManagers by their WorkerManagers when they are created and initialised. A DPS-C key is issued by ResourceManager of a DPS cluster, shared among the local components of the DPS cluster, and dispatched to components external to the DPS cluster for access to this DPS cluster. The local components are WorkerManagers, JobManagers, Mappers, and Reducers, and the external components are ClientApps of the same MR service as well as JobManagers, Mappers, and Reducers hosted by the other MR services. Similarly, a DFS-C key is issued by NameManager of a DFS cluster, shared among the local components of the DFS cluster, and dispatched to components external to the DFS cluster for access to this DFS cluster. The local components are DataStores and the external components are ClientApps of the same MR service, as well as JobManagers, Mappers, and Reducers hosted by the same and the other MR services.

Pairwise keys are classified into four classes: primary keys, secondary keys, sealing keys, and session keys. A primary key is a long-term secret key established on two interacting components. By long-term, we mean the key is not job dependent and its lifetime is at least as long as the lifecycle of a job execution. A primary key is used in the mutual authentication of two entities starting an initial interaction to establish short-term credentials (which may expire before the end of a job execution) for subsequent interactions. There are two groups of primary keys, pre-shared and encapsulated. A pre-shared key is established on the two interacting entities when they are initialised or when they are registered to their domains. These keys are pmk_{rm^i,rm^j}, pmk_{c^i,rm^i}, $pmk_{wm_u^i,rm^i}$, pmk_{jm,rm^1}, pmk_{jm,m_a}, and pmk_{jm,r_b}. An encapsulated key is a key issued by a trusted third party (referred to as an issuer) which has established a shared key with each of the two interacting entities. These keys are pmk_{c^i,nm^i}, pmk_{c^i,ds_v^i}, $pmk_{c^1,jm}$, pmk_{jm,nm^i}, pmk_{jm,ds_v^i}, pmk_{jm,wm_u^i}, pmk_{m_a,nm^i}, pmk_{m_a,ds_v^i}, pmk_{r_b,wm_u^i}, pmk_{r_b,nm^i}, and pmk_{r_b,ds_v^i}. Encapsulated keys are dispatched from an initiator to a respondent at the start of the authentication process (to be explained later on in the description of the GE2A protocol). To ensure that the key is not revealed to other components than the respondent, the issuer encapsulate the key in a container encrypted with a secret key (i.e., a sealing key) shared between the respondent and the issuer. The encrypted container is called a ticket. The ticket structure is shown in Figure 5.4. A ticket contains seven items. The descriptions of these items are summarised in Table 5.3. Possible values for an action request (REQ) are shown in Table 5.7. Assuming that $slk_{R,z}$ is the sealing key used to encrypt a ticket $tkt_{I,R}^z$; id_I, id_R, id_z are the IDs of the initiator, respondent, and issuer, respectively; did_I, did_R, did_z are the domain IDs (DIDs) of the initiator, respondent, and issuer, respectively; jid is the job ID (JID); req is the action request; gt, et are the ticket generation and expiration times, respectively; and $pmk_{I,R}$ is the key to be encapsulated, the ticket can be expressed as $tkt_{I,R}^z = E(slk_{R,z},\ id_I \mid\mid did_I \mid\mid id_R \mid\mid did_R \mid\mid id_z \mid\mid did_z \mid\mid jid \mid\mid req \mid\mid gt \mid\mid et \mid\mid pmk_{I,R})$ (for ease of presentation, the RSV field is omitted).



IID	IDID	RID	RDID	TID	TDID	JID	REQ	RSV	GT	ET	PMK
2 B	2 B	2 B	2 B	2 B	2 B	2 B	1 B	1 B	8 B	8 B	$S(pmk_{I,R})$ B

Figure 5.4: Ticket structure.

Table 5.3: Ticket fields.

Item	Size (bytes)	Description
IID	2	Initiator ID
IDID	2	Initiator domain ID
RID	2	Respondent ID
RDID	2	Respondent domain ID
TID	2	Ticket issuer ID
TDID	2	Ticket domain ID
JID	2	Job ID
REQ	1	Action request
RSV	1	Reserved for future use
GT	8	Ticket generation time
ET	8	Ticket expiration time
PMK	$S(pmk_{I,R})$	Primary key
Total	32+$S(pmk_{I,R})$	

A secondary key is a short-term secret key established on two interacting components during the preceding (initial) interaction. It is used to mutually authenticate the entities that have previously interacted with each other. The use of secondary keys reduces the exposure of primary keys, thus, lowering the risk of the primary keys being compromised. These keys are sck_{rm^1,rm^j}, sck_{c^i,rm^i}, $sck_{c^1,jm}$, sck_{c^i,nm^i}, sck_{jm,rm^1}, sck_{jm,nm^i}, $sck_{m_a,jm}$, $sck_{r_b,jm}$, sck_{m_a,nm^i}, and sck_{r_b,nm^i}. A sealing key is a long-term secret key shared between a respondent and an issuer. It is used to encrypt a ticket by the issuer and to decrypt the ticket by the respondent. These keys are slk_{nm^i,rm^i}, $slk_{ds_v^i,nm^i}$, $slk_{wm_u^i,rm^i}$, $slk_{nm^i,jm}$, and $slk_{wm_u^i,jm}$. A session key is a short-term secret key established on two interacting components during the authentication of each interaction. A session key is used to protect the confidentiality of sensitive data exchanged after the interacting entities are positively authenticated. The total number of session keys is equal to the number of interactions taking place during a job execution. The key hierarchy of non-derived keys are depicted in Figure 5.5.

Non-derived keys, tickets, and their associated data (metadata) should be stored on secure storage, referred to as Credential Store (CStore). CStore could be backed by protected volatile memory or encrypted persistent storage. As a reference example, we use a table based data structure to implement CStore. With this structure, a metadata entry consists of credential ID; class (e.g., OrgDomain key and Primary key); component ID (only for a pairwise key); domain ID; component class (e.g., Mapper); expiry; and the path of the credential. An example CStore of a component is shown in Table 5.4.

Table 5.4: An example CStore of a component.

Cred. ID	Class	Component ID	Domain ID	Component Class	Expiry (Unix time)	Path
00001	DPS-C	-	0001	-	1588118400	/path/key1
00002	Primary key	0002	0001	Mapper	1588118400	/path/key2
…						

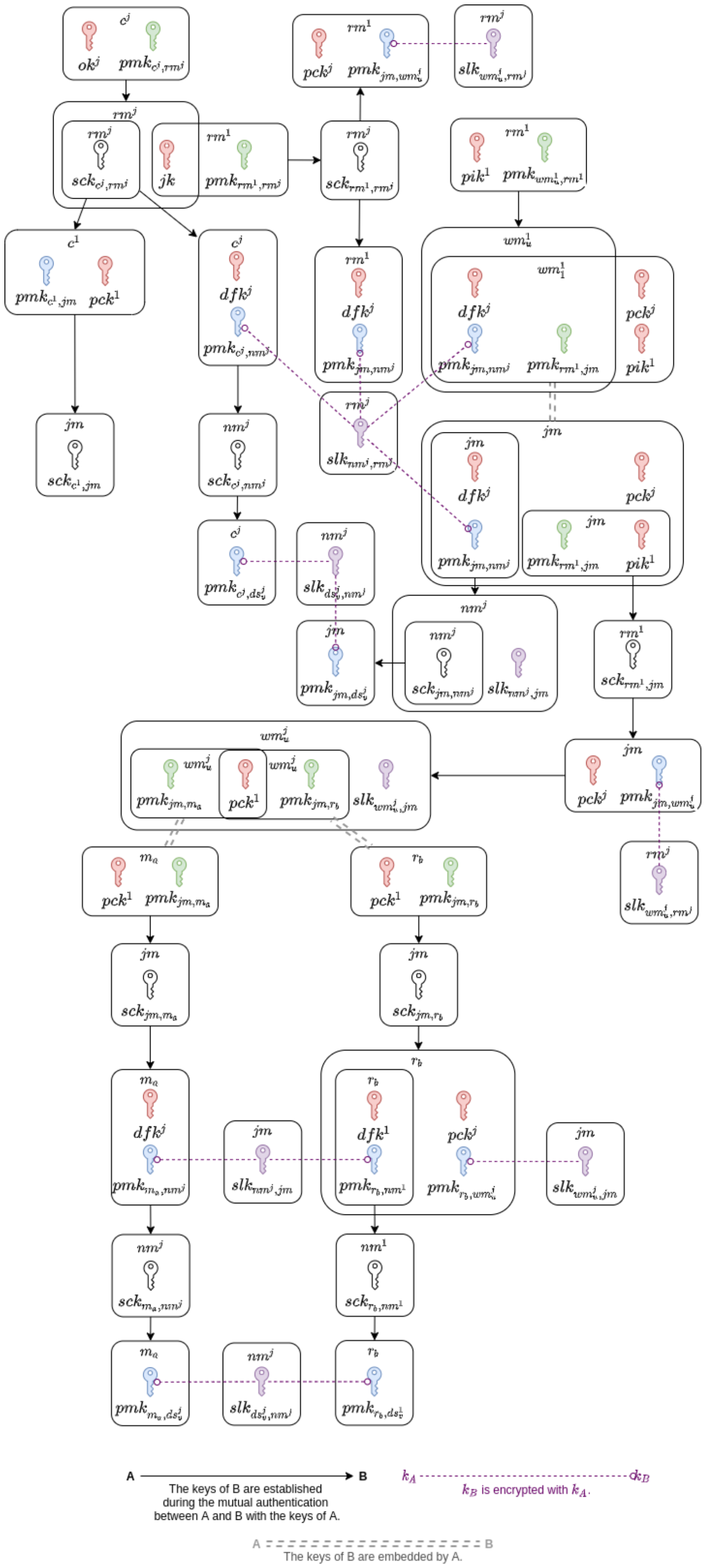


Figure 5.5: The key hierarchy of non-derived keys.

Derived keys are short-term secret keys that are generated during the process of entity authentication and deleted at the end of the process. Based on their purposes, derived keys are classified into MAC keys and credential encryption keys. A MAC key is used for message authentication. In other words, it is used to generate and verify MAC tags for messages used in entity authentication protocols. A credential encryption key is used to encrypt and decrypt credentials for subsequent authentication that are dispatched from an initiator to a respondent. All of the keys (non-derived and derived) used in MIEA have the same length.

5.5.3 Credential Establishment Methods

As shown in Figure 5.2, four credential establishment methods, the existing method, the embedded method, the authenticated key exchange method, and the derivation method (respectively referred to as EXT, EMB, AKE, and DER) are used to establish credentials on components; EXT, EMB, and AKE are for non-derived keys, and DER is for derived keys.

As indicated by the name, the existing method uses existing mechanisms, such as the registration and authentication services of the underlying system, to establish keys on components. This method is used to establish OrgDomain keys, DPS-I keys, DPS-C keys, DFS-C keys, pre-shared primary keys, and sealing keys.

The embedded method is used by WorkerManagers to embed keys into newly created containers, i.e., JobManagers, Mappers, and Reducers. The embedding can be done by using methods such as memory sharing and configuration templates. DPS-I keys, DPS-C keys, DFS-C keys, as well as pre-shared and encapsulated primary keys are established by using this method.

The authenticated key exchange method establishes keys on a respondent by using the entity authentication protocols of MIEA. This method is used to establish JobDomain keys, DPS-C keys, DFS-C keys, encapsulated primary keys, all the secondary keys, sealing keys, and all the session keys.

The derivation method uses a key derivation function to generate derived keys (MAC keys and credential encryption keys) from non-derived keys and nonces known to interacting components during an authentication instance.

The descriptions of keys and their establishment methods are summarised in Table 5.5.

Table 5.5: Keys and the respective establishment methods.

Method	Keys	When the keys are established
EXT	$ok^i(rm^i), pmk_{c^i,rm^i}$	c^i is registered to rm^i
EXT	$pik^i, pck^i\ (wm_u^i),$ $pmk_{wm_u^i,rm^i}, slk_{wm_u^i,rm^i}$	wm_u^i is registered to rm^i
EXT	$dfk^i\ (rm^i), slk_{nm^i,rm^i}$	nm^i is registered to rm^i
EXT	$dfk^i\ (ds_v^i), slk_{ds_v^i,nm^i}$	ds_v^i is registered to nm^i
EXT	pmk_{rm^i,rm^j}	Collaboration is established
EMB	$pik^1, pck^1, dfk^i\ (jm),$ $pmk_{jm,rm^1}, pmk_{jm,nm^i}$	jm is launched by wm_u^1 @9
EMB	$pck^1\ (m_a), pmk_{jm,m_a}$	m_a is launched by wm_u^i @18
EMB	$pck^1\ (r_b), pmk_{jm,r_b}$	r_b is launched by wm_u^i @18
AKE	$jk\ (rm^i), sck_{c^i,rm^i}$	c^i authenticates to rm^i @2
AKE	$pck^1\ (c^1)$	rm^1 authenticates to c^1 @3
AKE	$dfk^i\ (c^i),$ pmk_{c^i,nm^i}	rm^i authenticates to c^i @3
AKE	$pmk_{c^1,jm}$	rm^1 authenticates to c^1 @10
AKE	$pck^j\ (rm^1)$	rm^j authenticates to rm^1 @15
AKE	$pck^j\ (jm), pmk_{jm,wm_u^i}$	rm^1 authenticates to jm @16
AKE	$pck^j, dfk^i\ (r_b),$	jm authenticates to r_b @24

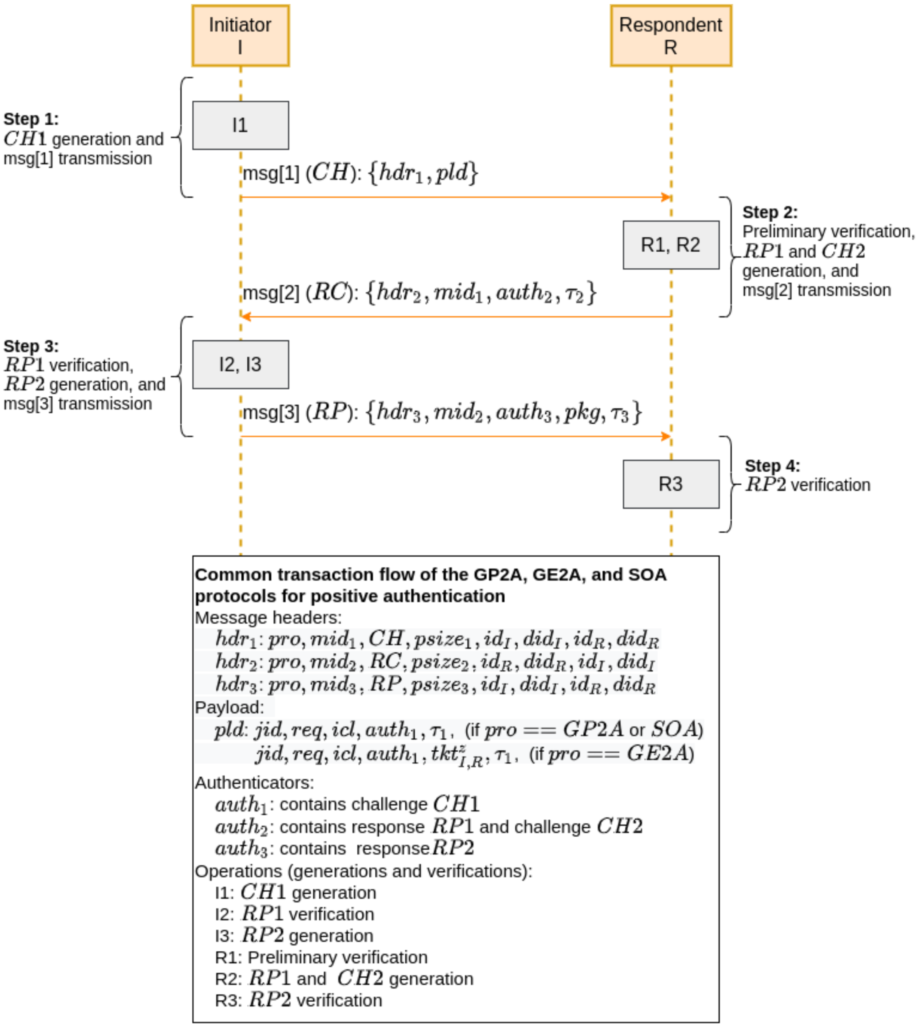
Method	Keys	When the keys are established
	$pmk_{r_b,wm_u^i}, pmk_{r_b,nm^i}$	
AKE	$dfk^j\ (rm^1)$	rm^j authenticates to rm^1 @7
AKE	$dfk^i\ (m_a), pmk_{m_a,nm^i}$	jm authenticates to m_a @20
AKE	pmk_{c^i,ds_v^i}	nm^i authenticates to c^i @4b
AKE	pmk_{jm,ds_v^i}	nm^i authenticates to jm @12b
AKE	pmk_{m_a,ds_v^i}	nm^i authenticates to m_a @21b
AKE	pmk_{r_b,ds_v^i}	nm^i authenticates to r_b @26b
AKE	sck_{rm^1,rm^j}	rm^1 authenticates to rm^j @6
AKE	$sck_{c^1,jm}$	c^1 authenticates to jm @11
AKE	sck_{c^i,nm^i}	c^i authenticates to nm^i @4a
AKE	sck_{jm,rm^1}	jm authenticates to rm^1 @13
AKE	$sck_{jm,nm^i}, slk_{nm^i,jm}$	jm authenticates to nm^i @12a
AKE	$sck_{m_a,jm}$	m_a authenticates to jm @19
AKE	$sck_{r_b,jm}$	r_b authenticates to jm @19
AKE	sck_{m_a,nm^i}	m_a authenticates to nm^i @21a
AKE	sck_{r_a,nm^i}	r_a authenticates to nm^i @26a
AKE	$slk_{wm_u^i,jm}$	jm authenticates to wm_u^i @17
AKE	$ssk_{x,y}$	x authenticates to y
DER	$mk_{x,y}$	x authenticates to y
DER	$ck_{x,y}$	x authenticates to y

Note: “$gk_1, gk_2, \ldots\ (x)$” means group keys $gk_1, gk_2, \ldots$ are established on x.

5.5.4 Entity Authentication Protocols

The GP2A, GE2A, and SOA protocols implement the ideas highlighted in Section 5.3. They support the distribution of session keys and keys for authenticating subsequent interactions. With each of the protocols, two interacting components (an initiator I and a respondent R) perform authentication by generating and verifying a series of challenges and responses (i.e., nonces). Such generation and verification are done using symmetric-key cryptosystems. Depending on the number of factors used, the challenges and responses are encrypted with a pairwise key or both of a group key and a pairwise key. The encrypted challenges and responses are collectively referred to as authenticators. After the responses are positively verified, the components are mutually authenticated, and the interaction can be proceeded.

The three protocols share a common transaction flow and message structure. There are four classes of messages, Challenge (CH) messages, Response-and-Challenge (RC) messages, Response (RP) messages, and Reject (RJ) messages. CH, RC, and RP messages are collectively used to transmit challenges and responses between I and R, whereas an RJ message is used to inform the other component of negative authentication. For each positive authentication instance, four operational steps are performed: **Step 1**, I generates a challenge $CH1$ ($I1$) and sends it to R; **Step 2**, R performs preliminary verification ($R1$), generates a response $RP1$ (for $CH1$) and a challenge $CH2$ ($R2$), and sends them back to I; **Step 3**, I verifies $RP1$ ($I2$), generates a response $RP2$ (for $CH2$) ($I3$), and send $RP2$ back to R; and **Step 4**, R verifies $RP2$ ($R3$). During these steps, three messages are exchanged: a CH message is sent at **Step 1**, an RC message at **Step 2**, and an RP message at **Step 3**. The exchange of these messages is depicted in Figure 5.6.


Initiator
I
Respondent
R
Step 1:
$CH1$ generation and msg[1] transmission
I1
msg[1] (CH): $\{hdr_1, pld\}$
Step 2:
Preliminary verification, $RP1$ and $CH2$ generation, and msg[2] transmission
R1, R2
msg[2] (RC): $\{hdr_2, mid_1, auth_2, \tau_2\}$
Step 3:
$RP1$ verification, $RP2$ generation, and msg[3] transmission
I2, I3
msg[3] (RP): $\{hdr_3, mid_2, auth_3, pkg, \tau_3\}$
Step 4:
$RP2$ verification
R3

Common transaction flow of the GP2A, GE2A, and SOA protocols for positive authentication

Message headers:

hdr_1: $pro, mid_1, CH, psize_1, id_I, did_I, id_R, did_R$
hdr_2: $pro, mid_2, RC, psize_2, id_R, did_R, id_I, did_I$
hdr_3: $pro, mid_3, RP, psize_3, id_I, did_I, id_R, did_R$

Payload:

pld: $jid, req, icl, auth_1, \tau_1$, (if $pro == GP2A$ or SOA)
$jid, req, icl, auth_1, tkt^z_{I,R}, \tau_1$, (if $pro == GE2A$)

Authenticators:

$auth_1$: contains challenge $CH1$
$auth_2$: contains response $RP1$ and challenge $CH2$
$auth_3$: contains response $RP2$

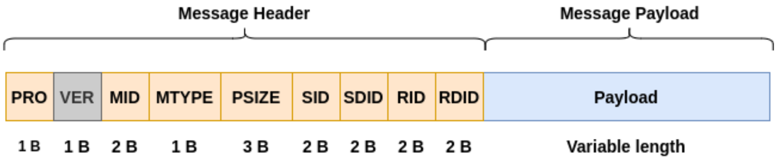
Operations (generations and verifications):

I1: $CH1$ generation
I2: $RP1$ verification
I3: $RP2$ generation
R1: Preliminary verification
R2: $RP1$ and $CH2$ generation
R3: $RP2$ verification

Figure 5.6: A generic message transaction flow of the three entity authentication protocols for positive authentication.

In a negative authentication instance, an RJ message is sent from one component to the other to terminate the protocol. For example, if keys used to generate an authenticator $auth_1$ (at **Step 1**) are expired or invalid, R will send an RJ message (instead of an RC message) to I and abort the protocol (at **Step 2**).

The common message structure is shown in Figure 5.7. It consists of a header and a payload. The header further consists of nine fields, and these fields are described in Table 5.6. As the current version of all the protocols is set to 1, for ease of presentation, the VER field will be omitted in the following message description.



PRO	VER	MID	MTYPE	PSIZE	SID	SDID	RID	RDID	Payload
1 B	1 B	2 B	1 B	3 B	2 B	2 B	2 B	2 B	Variable length

Figure 5.7: The format of MIEA protocol messages.

Table 5.6: The header format of MIEA protocol messages.

Field	Size (bytes)	Description
PRO	1	Protocol
VER	1	Protocol version (the current version is 1)
MID	2	Message ID
MTYPE	1	Message type
PSIZE	3	The size of the payload in bytes (B)
SID	2	Sender ID
SDID	2	Sender Domain ID
RID	2	Receiver ID
RDID	2	Receiver Domain ID
Total	16	

Note: Possible values for PRO and MTYPE are shown in Table 5.7.

Table 5.7: Fields and their possible values.

Field	Numerical Value	Notation	Description
Protocol (PRO)	1	$GP2A$	GP2A protocol
	2	$GE2A$	GE2A protocol
	3	SOA	SOA protocol
Message Type (MTYPE)	1	CH	Challenge messages
	2	RC	Response-and-Challenge messages
	3	RP	Response messages
	4	RJ	Reject messages
Action Request (REQ)	1	$JSUB$	Job submission
	2	NFY	Notification
	3	INQ	Inquiry
	4	$DSLST$	Get a list of DataStores
	5	$WRITE$	Write data
	6	$READ$	Read data
	7	$LNCH$	Launch a container
Initiator and Respondent Classes (ICL and RCL)	1	RM	ResourceManager
	2	NM	NameManager
	3	WM	WorkerManager
	4	DS	DataStore
	5	C	ClientApp
	6	JM	JobManager
	7	M	Mapper
	8	R	Reducer
Credential Class (CCL)	1	OK	OrgDomain key
	2	JK	JobDomain key
	3	PIK	DPS-I key
	4	PCK	DPS-C key
	5	DFK	DFS-C key
	6	PMK	Primary key
	7	SCK	Secondary key
	8	SLK	Sealing key
	9	SSK	Session key
	10	TKT	Ticket
Error Code (ERR)	1	$NEGTAG$	Negative tag verification
	2	$INVJID$	R is not allocated for the job with jid
	3	$ACTUN$	The requested action in unavailable on R
	4	$ACTDE$	I does not have a permission to perform the action
	5	$INVCD$	Invalid credentials (e.g., wrong keys and expired keys)
	6	$NEGRP$	Negative response verification

For each of the four message classes (CH, RC, RP, and RJ), the payloads of messages used in different protocols share a common structure (except for CH messages used in G2EA which have one more item than those used in GP2A and SOA). For CH messages used in G2PA and SOA, the payload consists of five items: Job ID (JID); action Request (REQ), indicating the purpose of the interaction; Initiator Class (ICL), indicating the component class of I; an authenticator containing a challenge ($CH1$) generated by I; and a MAC tag for the message. For CH messages used in GE2A, the payload also contains a ticket containing a primary key used for authenticating I to R, which is positioned before the tag.

For RC messages, the payload consists of three items: the MID of the preceding CH message; an authenticator containing a response ($RP1$) and a challenge ($CH2$) generated by R; and a tag.

For RP messages, the payload consists of four items: the MID of the preceding RC message; an authenticator containing a response ($RP2$) generated by I; an encrypted credential package (containing keys, tickets, and their metadata); and a tag. The credentials contained in the package are dependent on interactions. In other words, the credentials contained in different RP messages are different. The package structure and description of metadata contained in the package are shown in Figure 5.8 and Table 5.8, respectively. Unlike CH and RC messages whose payload lengths are fixed, the payloads of RP messages have variable lengths.

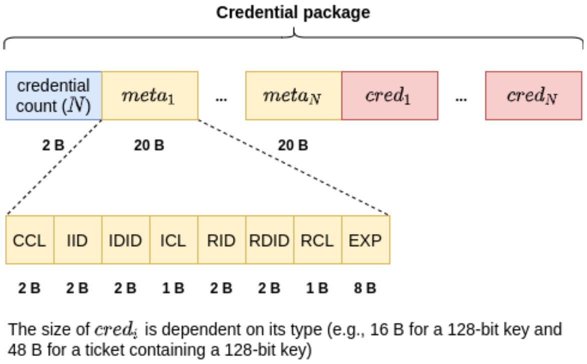


Figure 5.8: The format of a credential package.

Table 5.8: Credential package format.

Field	Size (bytes)	Description
CCL	2	Credential class
IID	2	Initiator ID
IDID	2	Initiator Domain ID
ICL	1	Initiator class
RID	2	Receiver ID
RDID	2	Receiver Domain ID
RCL	1	Respondent class
EXP	8	Expiry
Total	20	

Note: Possible values for CCL, ICL and RCL are shown in Table 5.7.

For RJ messages, the payload consists of two items: the MID of the preceding (CH or RC) message and an error code (which is dependent on the cause of negative verification). A MAC tag is not included in each RJ message as the interacting components may not have established shared secret keys that can be used to generate and verify the tag. Assuming pro is the current protocol used; $mid_1$and mid_2 are, respectively, the MIDs of the preceding and this RJ messages; x is the component sending the preceding message (mid_1); y is the component sending this RJ message (mid_2); and err is an error code, the RJ message msg-RJ can be expressed as msg-RJ: $\{pro,\ mid_2,\ RJ,\ S(MID)+S(ERR),\ id_y,\ did_y,\ id_x,\ did_x,\ mid_1, err\}$. Like CH and RC messages, the payloads of RJ messages also have fixed lengths.

In the following, we describe the operational steps of each protocol in detail.

5.5.4.1 GP2A Protocol

The GP2A protocol is a two-factor (group key and pre-shared primary key) entity authentication protocol for initial interactions between two components (I and R). It makes use of: (1) a symmetric-key based encryption scheme to generate and verify authenticators; (2) a MAC scheme to generate and verify tags contained in messages; and (3) a key derivation function to generate a MAC key and a credential encryption key. The group and pre-shared keys used are established on both components prior to the execution of the protocol. The protocol consists of four operational steps, as illustrated in Figure 5.6.

Step 1: In $I1$, I performs $CH1$ generation, generates a CH message (msg-GP2A1), and sends the message to R. For the generation of $CH1$, I generates a nonce n_1 and uses it as a challenge ($CH1$) for R. An authenticator $auth_1$ is then generated with n_1 using nested encryption, i.e., n_1 is encrypted with a pre-shared primary key $pmk_{I,R}$ and then is encrypted again with a group key gk. The generation of $auth_1$ is expressed as $auth_1 = SE(gk, SE(pmk_{I,R}, n_1))$.

I generates a MAC key mk with a length l by invoking a key derivation algorithm with $pmk_{I,R}$ and gk. The generation of mk is expressed as $mk_{I,R} = HKDF(l,\ pmk_{I,R},\ gk)$.

Next, I generates a MAC tag τ_1 using MAC-Signing with mk and message data which consist of message ID mid_1 for this CH message, job ID jid, action request req, initiator class icl, and the authenticator $auth_1$. The generation of τ_1 is expressed as $\tau_1 = MS(mk_{I,R}, mid_1||jid||req||icl||auth_1)$. The other message fields are not signed because the alteration of these fields can be easily detected, leading to negative verification (thus, termination of the protocol). For example, the value of MTYPE of the message can only be CH; and fraudulent values of SID and SDID would lead to incorrect or unsuccessful key lookup. After generating τ_1, I then generates msg-GP2A1 and sends the message to R. msg-GP2A1 is expressed as msg-GP2A1: $\{GP2A,\ mid_1,\ CH,\ S(JID)+S(REQ)+S(ICL)+S(auth_1)+S(\tau_1),\ id_I,\ did_I,\ id_R\ ,\ did_R,\ jid,\ req,\ icl,\ auth_1,\ \tau_1\}$.

Step 2: Upon receiving msg-GP2A1, R performs preliminary verification ($R1$), then generates $RP1$ and $CH2$ ($R2$), before generating an RC message (msg-GP2A2). In $R1$, R generates a MAC key mk used for verifying the authenticity of msg-GP2A1 against τ_1. The generation of mk is the same as that of **Step 1**. The verification of the authenticity of the message using MAC-Verify is expressed as $mv_1 = MV(mk_{I,R},\ mid_1||jid||req||icl||auth_1, \tau_1)$.

Following a positive verification of τ_1, R checks the validity of the interaction and keys used for authentication. This is done by ensuring that: (1) R is allocated for the job with a job ID of jid; (2) I can interact with R by checking icl (e.g., Reducers may interact with WorkerManager, but Mappers may not); (3) R supports the requested action req (e.g., WorkerManagers support $READ$ requested by Reducers but not $WRITE$); (4) I can perform the requested action by checking req (e.g., Mappers may read data from, but not write data to, the DFS clusters); and (5) R has established gk and $pmk_{I,R}$ with I by looking up id_I, did_I and icl in its CStore, the key entries should exist and not expire. If any of these conditions is not met, R sends an RJ message back to I with a corresponding error code and aborts the protocol. The verification steps of $R1$ are shown in Figure 5.9.

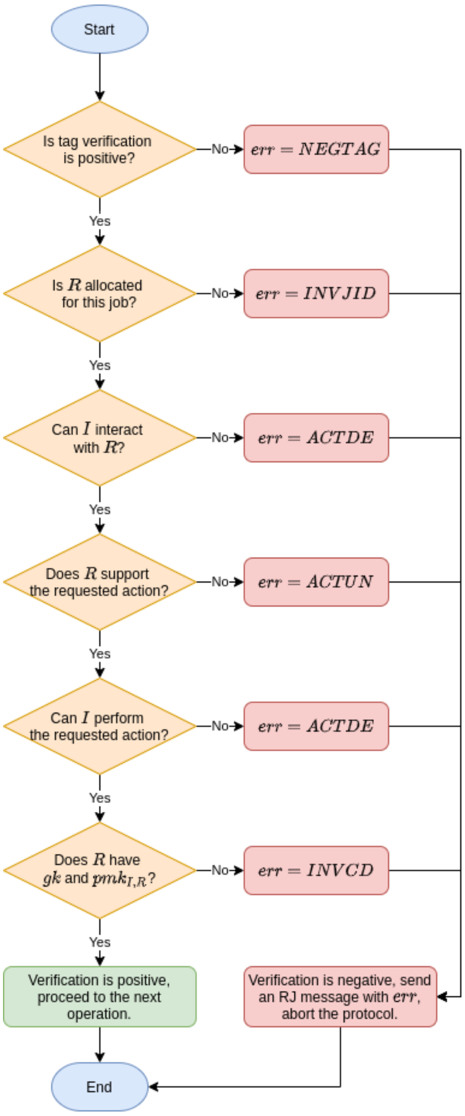

Start
Is tag verification is positive?
No
$err = NEGTAG$
Yes
Is R allocated for this job?
No
$err = INVJID$
Yes
Can I interact with R?
No
$err = ACTDE$
Yes
Does R support the requested action?
No
$err = ACTUN$
Yes
Can I perform the requested action?
No
$err = ACTDE$
Yes
Does R have gk and $pmk_{I,R}$?
No
$err = INVCD$
Yes
Verification is positive, proceed to the next operation.
Verification is negative, send an RJ message with err, abort the protocol.
End

Figure 5.9: GP2A preliminary verification (R1).

In $R2$, R generates $RP1$ for $CH1$, generates $CH2$, and generates msg-GP2A2. To generate $RP1$, R first decrypts $auth_1$ with gk and $pmk_{I,R}$ to obtain n_1 (used as $RP1$). The process is expressed as $n_1 = SD(pmk_{I,R},\ SD(gk,\ auth_1))$.

R generates a nonce n_2 as a challenge ($CH2$) for I. It then generates an authenticator $auth_2$ containing a concatenation of $RP1$ and $CH2$ using nested encryption. The generation of $auth_2$ is expressed as $auth_2 = SE(gk, SE(pmk_{I,R}, n_1||n_2))$.

Next R generates a tag τ_2 using MAC-Signing with mk and message data which consist of the message ID mid_2 for this RC message, the message ID mid_1 of the preceding CH message, and the authenticator $auth_2$. The generation of τ_2 is expressed as $\tau_2 = MS(mk_{I,R}, mid_2||mid_1||auth_2)$.

R generates msg-GP2A2 and sends the message back to I. msg-GP2A2 is expressed as msg-GP2A2: $\{GP2A,\ mid_2,\ RC,\ S(MID) + S(auth_2) + S(\tau_2),\ id_R,\ did_R,\ id_I\ ,\ did_I,\ mid_1,\ auth_2,\ \tau_2\}$.

Step 3: I receives msg-GP2A2, performs $RP1$ verification ($I2$) and $RP2$ generation ($I3$), before generating an RP message (msg-GP2A3). In $I2$, I verifies the authenticity of msg-GP2A2 against τ_2. The verification of the authenticity of the message using MAC-Verify is expressed as $mv_2 = MV(mk_{I,R},\ mid_2||mid_1||auth_2, \tau_2)$.

If the verification is positive, I continues the protocol; otherwise, I sends an RJ message (with an error code $NEGTAG$) back to R and terminates the protocol. Following a positive verification of τ_2, I decrypts $auth_2$ with gk and $pmk_{I,R}$ to obtain $RP1$ and $CH2$, which are, respectively, assigned to $n_1{}'$ and n_2. This operation is expressed as $(n_1', n_2) = SD(pmk_{I,R}, SD(gk,\ auth_2))$.

I verifies $RP1$ by comparing n_1' with n_1 (the nonce it sent in **Step 1**). If $n_1'\ == n_1$ then the verification is positive, I will proceed to $I3$; otherwise, the verification is negative, I will send an RJ message back to R and terminate the protocol. The verification steps of $I2$ are shown in Figure 5.10.

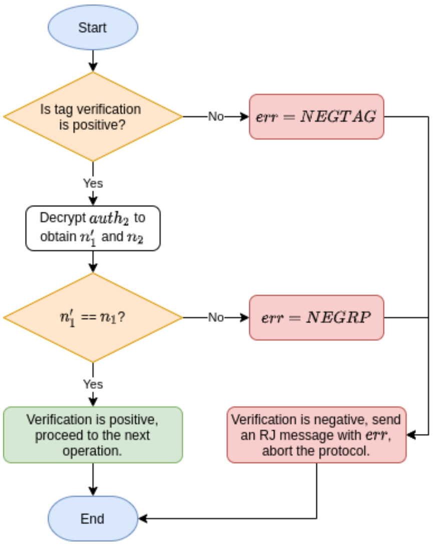

Start
Is tag verification is positive?
No
$err = NEGTAG$
Yes
Decrypt $auth_2$ to obtain n_1' and n_2
$n_1' == n_1$?
No
$err = NEGRP$
Yes
Verification is positive, proceed to the next operation.
Verification is negative, send an RJ message with err, abort the protocol.
End

Figure 5.10: GP2A RP1 verification (I2).

In $I3$, I generates $RP2$ for $CH2$ using n_2 obtained in $I2$, and prepares an RP message. To generate $RP2$, an authenticator $auth_3$ containing $RP2$ (n_2) is generated using nested encryption. The generation of $auth_3$ is expressed as $auth_3 = SE(gk, SE(pmk_{I,R}, n_2))$.

I generates a credential encryption key ck with a length l using a key derivation algorithm with $pmk_{I,R}$ and n_2. The generation of ck is expressed as $ck_{I,R} = HKDF(l,\ pmk_{I,R},\ n_2)$.

I generates a credential package containing credentials (keys, tickets, and their metadata) to be dispatched and encrypts the package with ck. The encrypted package pkg is expressed as $pkg = SE(ck_{I,R}, credential\ package)$.

I generates a tag τ_3 using MAC-Signing with mk and message data consisting of the message ID mid_3 for this RP message, the message ID mid_2 of the preceding RC message, the authenticator $auth_3$, and the encrypted credential package pkg. The generation of τ_3 is expressed as $\tau_3 = MS(mk_{I,R},\ mid_3||mid_2||auth_3||pkg)$.

I generates msg-GP2A3 and sends the message back to R. The RP message is expressed as msg-GP2A3: $\{GP2A,\ mid_3,\ RP,\ S(MID) + S(auth_3) +\ S(pkg) + S(\tau_3),\ id_I,\ did_I,\ id_R\ ,\ did_R,\ mid_2,\ auth_3, pkg,\ \tau_3\}$.

Step 4: Upon receiving msg-GP2A3, R performs $RP2$ verification ($R3$). In $R3$, similar to $I2$, R verifies the authenticity of msg-GP2A3 against τ_3. The verification of the authenticity of the message using MAC-Verify is expressed as $mv_3 =\ MV(mk_{I,R}, mid_3||mid_2||auth_3||pkg, \tau_3)$.

Following a positive verification, R decrypts $auth_3$ with gk and $pmk_{I,R}$ to obtain $RP2$ and then assign the obtained value to n_2'. This operation is expressed as $n_2' = SD(pmk_{I,R},\ SD(gk,\ auth_3))$.

R verifies $RP2$ by comparing n_2' with n_2 (the nonce it sent in **Step 2**). If $n_2' == n_2$ then the verification is positive, I and R are mutually authenticated, R then proceeds to the next operation; otherwise, the verification is negative and R will terminate the protocol. If the verification is positive, R generates a credential encryption key ck (used for decryption). The generation of ck is the same as that of **Step 3**. R then decrypts pkg with ck. The decryption of pkg is expressed as $credential\ package = SD(ck_{I,R},\ pkg)$.

Lastly, the decrypted credentials are stored in CStore of R. The verification steps of $R3$ are shown in Figure 5.11.

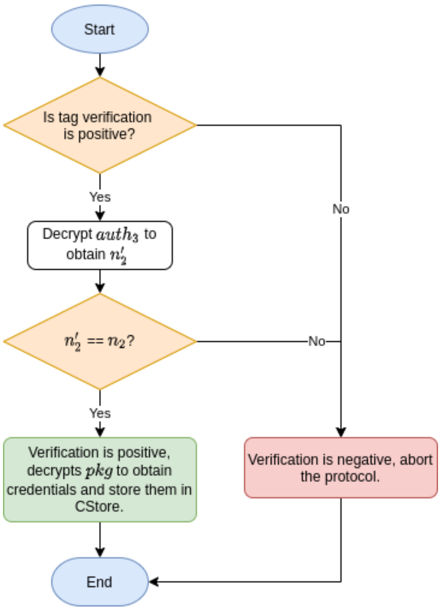

Start
Is tag verification is positive?
Yes
No
Decrypt $auth_3$ to obtain n_2'
$n_2' == n_2$?
No
Yes
Verification is positive, decrypts pkg to obtain credentials and store them in CStore.
Verification is negative, abort the protocol.
End

Figure 5.11: GP2A RP2 verification (R3).

5.5.4.2 GE2A Protocol

The GE2A protocol is also a two-factor (group key and encapsulated primary key) entity authentication protocol. Like GP2A, it is used for authentication of initial interactions between two components (I and R). Unlike GP2A, the encapsulated primary key is issued by a trusted third party (z). I obtains the key and a ticket containing the key from z, and then sends the ticket to R with the first message (a CH message) of the protocol. Upon receiving the message,

R decrypts the ticket with the sealing key shared with z to obtain the encapsulated key and use it for authentication. The protocol also consists of four operational steps as shown in Figure 5.6.

Step 1: In $I1$, I performs $CH1$ generation and generates a CH message (msg-GE2A1). For these tasks, I generates a nonce n_1 and an authenticator $auth_1$, and then derives a MAC key $mk_{I,R}$ in the same manner as **Step 1** of GP2A. However, the generation of a MAC tag τ_1 takes one additional item, i.e., a ticket $tkt_{I,R}^{z}$. The generation of τ_1 is expressed as $\tau_1 = MS(mk_{I,R}, mid_1 \,||\, jid \,||\, req || \, icl \,||\, auth_1 \,||\, tkt_{I,R}^{z})$.

I then generates msg-GE2A1 and sends the message to R. msg-GE2A1 is expressed as msg-GE2A1: $\{GE2A,\ mid_1,\ CH,\ S(JID) + S(REQ) + S(ICL) + S(auth_1) + S(\tau_1),\ id_I,\ did_I,\ id_R\ ,\ did_R,\ jid,\ req,\ icl,\ auth_1,\ tkt_{I,R}^{z},\ \tau_1\}$.

Step 2: Upon receiving msg-GE2A1, R performs preliminary verification ($R1$), generates $RP1$ and $CH2$ ($R2$), before generating an RC message (msg-GE2A2). In $R1$, R decrypts $tkt_{I,R}^{z}$ with $slk_{R,z}$ to obtain $pmk_{I,R}$ and the associated data. This operation is expressed as $\left(id_I, did_I, id_R, did_R, id_z, did_z, jid, req, gt, et,\ pmk_{I,R}\right) = SD(slk_{R,z}, tkt_{I,R}^{z})$.

R checks the validity of $pmk_{I,R}$ and performs preliminary verification. In addition to conditions (1) to (4) listed in $R1$ of **Step 2** of GP2A, the checking conditions here further include: (1) R has established gk with I; (2) the IDs and DIDs of the initiator and respondent match the ones of I and R, respectively; (3) req contained in the ticket matches req of the message; and (4) $pmk_{I,R}$ is not expired, i.e., the current time (observed by R) is not earlier than the key generation time (gt) and not later than the key expiration time (et). If any of these conditions is not met, R sends an RJ message back to I with a corresponding error code and aborts the protocol. The verification steps of $R1$ are shown in Figure 5.12.

Next, R generates a MAC key $mk_{I,R}$ using the same method as **Step 1**. After that, it verifies the authenticity of msg-GE2A1 against τ_1. The verification of τ_1 is expressed as $mv_1 = MV(mk_{I,R},\ mid_1||jid||req||icl||auth_1||tkt_{I,R}^{z},\ \tau_1)$.

In $R2$, R generates $RP1$ for $CH1$, generates $CH2$, and generates msg-GE2A2. The operations are the same as those of $R2$ in **Step 2** of GP2A. Therefore, msg-GE2A2 sent from R back to I is expressed as msg-GE2A2: $\{GE2A,\ mid_2,\ RC,\ S(MID) + S(auth_2) + S(\tau_2),\ id_R,\ did_R,\ id_I\ ,\ did_I,\ mid_1,\ auth_2,\ \tau_2\}$.

Step 3: I receives msg-GE2A2, performs $RP1$ verification ($I2$) and $RP2$ generation ($I3$), before generating an RP message msg-GE2A3. This step is the same as that of GP2A. Hence, msg-GE2A3 sent from I to R is expressed as msg-GE2A3: $\{GE2A,\ mid_3,\ RP,\ S(MID) + S(auth_3) +\ S(pkg) + S(\tau_3),\ id_I,\ did_I,\ id_R,\ did_R,\ mid_2,\ auth_3, pkg,\ \tau_3\}$.

Step 4: Upon receiving msg-GE2A3, R performs $RP2$ verification ($R3$). This step is the same as that of GP2A described earlier.

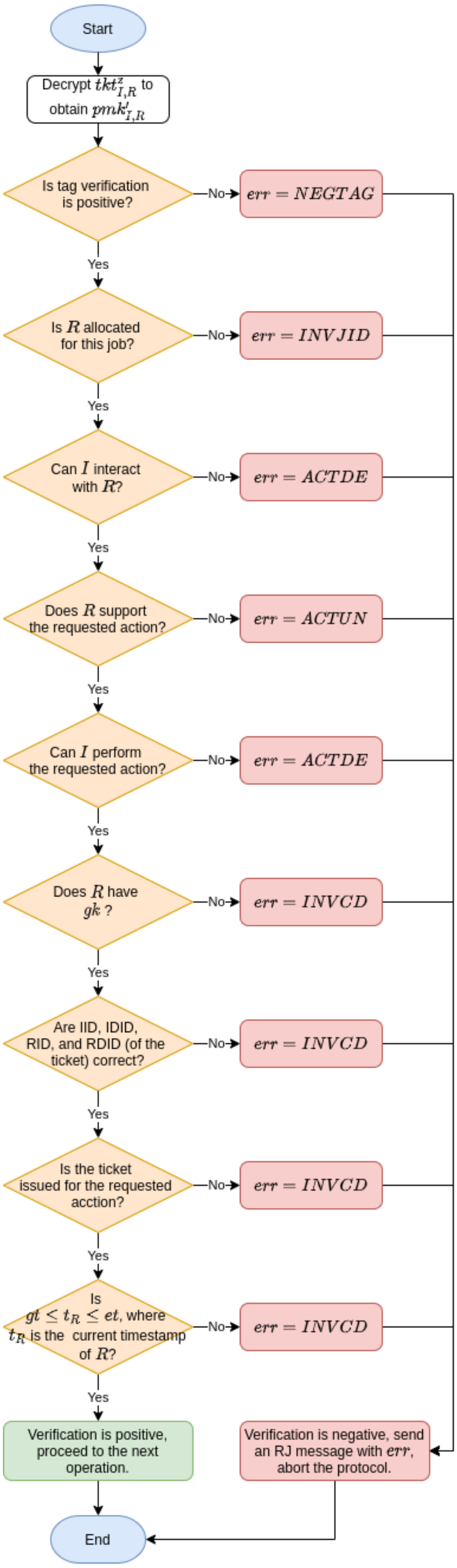

Start
Decrypt $tkt^{z}_{I,R}$ to obtain $pmk'_{I,R}$
Is tag verification is positive?
No
$err = NEGTAG$
Yes
Is R allocated for this job?
No
$err = INVJID$
Yes
Can I interact with R?
No
$err = ACTDE$
Yes
Does R support the requested action?
No
$err = ACTUN$
Yes
Can I perform the requested action?
No
$err = ACTDE$
Yes
Does R have gk ?
No
$err = INVCD$
Yes
Are IID, IDID, RID, and RDID (of the ticket) correct?
No
$err = INVCD$
Yes
Is the ticket issued for the requested acction?
No
$err = INVCD$
Yes
Is $gt \leq t_R \leq et$, where t_R is the current timestamp of R?
No
$err = INVCD$
Yes
Verification is positive, proceed to the next operation.
Verification is negative, send an RJ message with err, abort the protocol.
End

Figure 5.12: GE2A preliminary verification (R1).

5.5.4.3 SOA Protocol

The SOA protocol is a one-factor (secondary key) entity authentication protocol used for authentication of subsequent interactions. SOA is different from GP2A in that only one secondary (pairwise) key $sck_{I,R}$ is used for generating and verifying authenticators; a MAC key $mk_{I,R}$ is generated with only $sck_{I,R}$; and a credential encryption key $ck_{I,R}$ is generated with $sck_{I,R}$ instead of $pmk_{I,R}$. Again, the protocol also consists of four operational steps as shown in Figure 5.6. As these steps are similar to those of GP2A, in the following, we will highlight only the differences.

Step 1: I generates an authenticator $auth_1$ with $sck_{I,R}$. The generation of $auth_1$ is expressed as $auth_1 = SE\left(sck_{I,R}, n_1\right)$.

I generates a MAC key $mk_{I,R}$ with a length l by invoking a key derivation algorithm with $sck_{I,R}$. The generation of $mk_{I,R}$ is expressed as $mk_{I,R} = HKDF(l,\ sck_{I,R},\ NULL)$.

A CH message msg-SOA1 generated by I is expressed as msg-SOA1: $\{SOA,\ mid_1,\ CH,$ $S(JID) + S(REQ) + S(ICL) + S(auth_1) + S(\tau_1),\ id_I,\ did_I,\ id_R\ ,\ did_R,\ jid,\ req,\ icl,$ $auth_1,\ \tau_1\}$.

Step 2: Upon receiving msg-SOA1, in $R1$, R generates a MAC key $mk_{I,R}$ using the same method as **Step 1** and verifies the authenticity of msg-SOA1 against τ_1. Following a positive verification, R checks the validity of the interaction and key used for authentication. The checking conditions are the same as those in GP2A with the exception of the lack of a group key gk in condition (5).

In $R2$, $sck_{I,R}$ is used to decrypt $auth_1$ and encrypt $auth_2$. The decryption of $auth_1$ to obtain n_1 is expressed as $n_1 = SD(sck_{I,R},\ auth_1)$.

With n_1 along with n_2 (generated by R), the generation of $auth_2$ is expressed as $auth_2 = SE(sck_{I,R}, n_1||n_2)$.

An RC message (msg-SOA2) generated by R and to be sent to I is expressed as msg-SOA2: $\{SOA,\ mid_2,\ RC,\ S(MID) + S(auth_2) + S(\tau_2),\ id_R,\ did_R,\ id_I\ ,\ did_I,\ mid_1,\ auth_2,$ $\tau_2\}$.

Step 3: Upon receiving msg-SOA2, in $I2$, the extraction of n_1' and n_2 from $auth_2$ is done using $sck_{I,R}$, expressed as $(n_1', n_2) = SD(pmk_{I,R},\ auth_2)$. The steps for verifying n_1' are the same as those shown in Figure 5.10.

In $I3$, the generation of an authenticator $auth_3$ containing $RP2$ (n_2) is expressed as $auth_3 = SE(sck_{I,R}, n_2)$.

I generates a credential encryption key $ck_{I,R}$ with a length l using a key derivation algorithm with $sck_{I,R}$ and n_2. The generation of $ck_{I,R}$ is expressed as $ck_{I,R} = HKDF(l,$ $sck_{I,R},\ n_2)$.

An RP message msg-SOA3 generated by I for R is expressed as msg-SOA3: $\{SOA,\ mid_3,$ $RP,\ S(MID) + S(auth_3) +\ S(pkg) + S(\tau_3),\ id_I,\ did_I, id_R, did_R,\ mid_2,\ auth_3, pkg, \tau_3\}$.

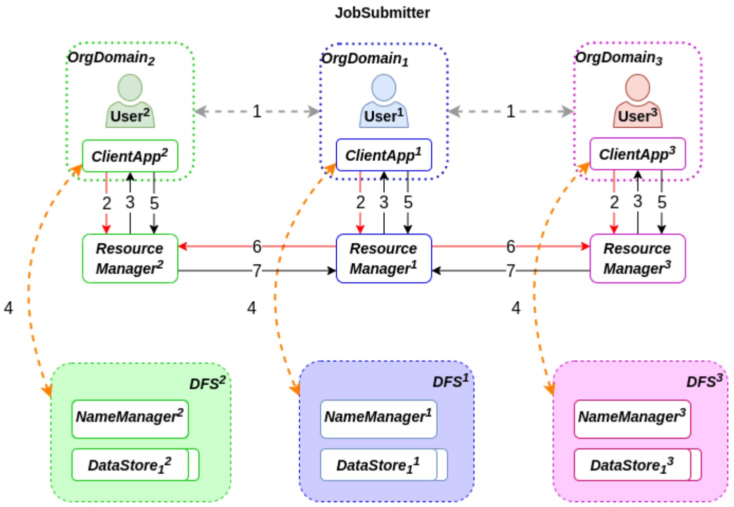
Step 4: Upon receiving msg-SOA3, in $R3$, R extracts n_2' from $auth_3$ with $sck_{I,R}$. The extraction of n_2' is expressed as $n_2' = SD(sck_{I,R},\ auth_3)$.

R then generates a credential encryption key $ck_{I,R}$ using the same method as **Step 3**. The remaining verification steps are the same as those of GP2A.

Upon a successful completion of the execution of any of the GP2A, GE2A, and SOA protocols (thus, positive authentication), each of the interacting components has an assurance that (1) it is interacting with the claimed component; (2) the authentication messages are authentic and freshly generated by the claimed component; and (3) session keys and keys for authenticating subsequent interactions are securely transmitted.

5.5.5 Putting Everything Together: MIEA in Action

The job execution flow when MIEA is applied is depicted in Figure 5.13. The figure highlights the protocol and the keys used to authenticate each of the interactions, the establishment methods for such keys, and credentials exchanged during each authentication instance. It is worth noting that, in the reduce phase @ 29 (shown in Figure 5.13 (d)), SOA is not used for the authentication of c^1 to nm^1. This is because there could be a long gap of time (e.g., hours) since the last interaction between c^1 and nm^1; the longer the gap, the higher risk of credentials being compromised. Hence, GE2A is applied to achieve a higher level of protection.



1: EXT ← none : jk

2: GP2A ← $ok^i, pmk_{c^i,rm^i} : jk, sck_{c^i,rm^i}, ssk_{c^i,rm^i}$

3: SOA ← $sck_{c^i,rm^i} : pck^1$ (only c^1), $dfk^i, pmk_{c^i,nm^i}, tkt^{rm^i}_{c^i,nm^i}, ssk_{rm^i,c^i}$

4a: GE2A ← $dfk^i, pmk_{c^i,nm^i} : sck_{c^i,nm^i}, ssk_{c^i,nm^i}$

4b: SOA ← $sck_{c^i,nm^i} : \{pmk_{c^i,ds^i_v}\}, \{tkt^{nm^i}_{c^i,ds^i_v}\}, ssk_{nm^i,c^i}$

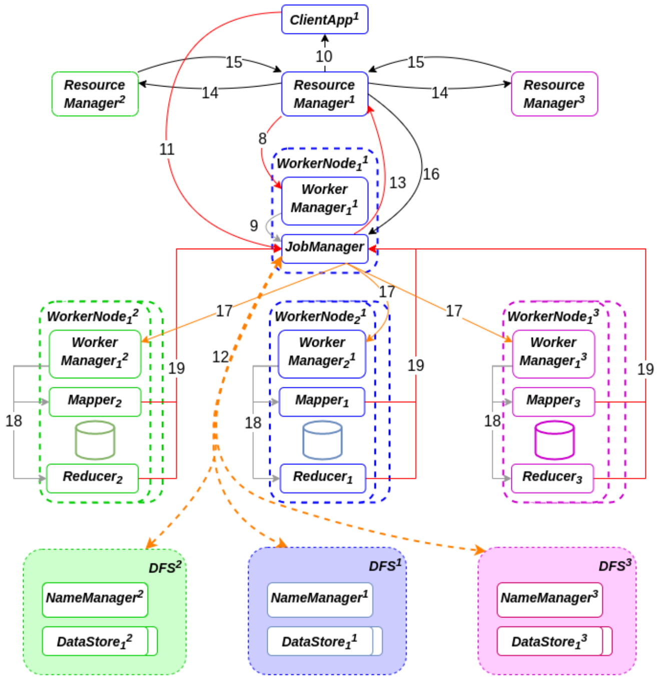
4c: GE2A ← $dfk^i, pmk_{c^i,ds^i_v} : ssk_{c^i,ds^i_v}$

5: SOA ← $sck_{c^i,rm^i} : ssk_{c^i,rm^i}$

6: GP2A ← $jk, pmk_{rm^1,rm^j} : sck_{rm^1,rm^j}, ssk_{rm^1,rm^j}$

7: SOA ← $sck_{rm^1,rm^j} : dfk^j, pmk_{jm,nm^j}, tkt^{rm^j}_{jm,nm^j}, ssk_{rm^j,rm^1}$

(a)



8: GP2A $\leftarrow pik^1, pmk_{wm_1^1, rm^1} : \{dfk^j\}, pmk_{rm^1, jm},$
$\{pmk_{jm,nm^j}\}, \{tkt^{rm^j}_{jm,nm^j}\}, ssk_{rm^1, wm_1^1}$

9: EXT $\leftarrow$ none : $pik^1, pck^1, \{dfk^j\}, pmk_{rm^1, jm},$
$\{pmk_{jm,nm^j}\},\ \{tkt^{rm^j}_{jm,nm^j}\}$

10: SOA $\leftarrow sck_{c^1, rm^1} : pmk_{c^1, jm}, tkt^{rm^1}_{c^1, jm}, ssk_{rm^1, c^1}$

11: GE2A $\leftarrow pck^1, pmk_{c^1, jm} : sck_{c^1, jm}, ssk_{c^1, jm}$

12a: GE2A $\leftarrow dfk^i, pmk_{jm,nm^i} : sck_{jm,nm^i}, tsk_{nm^i, jm},$
ssk_{jm,nm^i}

12b: SOA $\leftarrow sck_{jm,nm^i} : \{pmk_{jm,ds_v^i}\}, \{tkt^{nm^i}_{jm,ds_v^i}\},$
$ssk_{nm^i, jm}$

12c: GE2A $\leftarrow dfk^i, pmk_{jm,ds_v^i} : ssk_{jm,ds_v^i}$

13: GP2A $\leftarrow pik^1, pmk_{rm^1, jm} : sck_{rm^1, jm}, ssk_{jm, rm^1}$

14: SOA $\leftarrow sck_{rm^1, rm^i} : ssk_{rm^1, rm^j}$

15: SOA $\leftarrow sck_{rm^1, rm^i} : pck^i, \{pmk_{jm, wm_u^i}\},$
$\{tkt^{rm^i}_{jm, wm_u^i}\}, ssk_{rm^j, rm^1}$

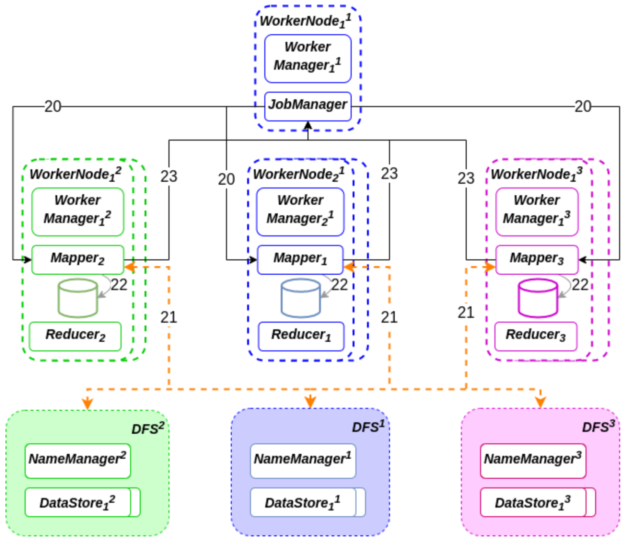
16: SOA $\leftarrow sck_{rm^1, jm} : \{pck^j\}, \{pmk_{jm, wm_u^j}\},$
$\{tkt^{rm^i}_{jm, wm_u^i}\}, ssk_{rm^1, jm}$

17: GE2A $\leftarrow pik^1/pck^i, pmk_{jm, wm_u^i} : pck^1,$
$\{pmk_{jm, m_a}\}, \{pmk_{jm, r_b}\}, tsk_{wm_u^i, jm}, ssk_{jm, wm_u^i}$

18: EXT $\leftarrow$ none :
$pik^1, pck^1, \{pmk_{jm, m_a}\}, \{pmk_{jm, r_b}\}$

19: GP2A $\leftarrow pik^1/pck^1, pmk_{jm, m_a}/pmk_{jm, r_b}$:
$sck_{jm, m_a}/sck_{jm, r_b},\ ssk_{m_a, jm}/ssk_{r_b, jm}$

(b)



20: SOA $\leftarrow sck_{jm, m_a} : \{dfk^j\}, \{pmk_{m_a, nm^j}\}, \{tkt^{jm}_{m_a, nm^j}\},$
ssk_{jm, m_a}

21a: GE2A $\leftarrow dfk^j, pmk^{jm}_{m_a, nm^j} : sck_{m_a, nm^j}, ssk_{m_a, nm^j}$

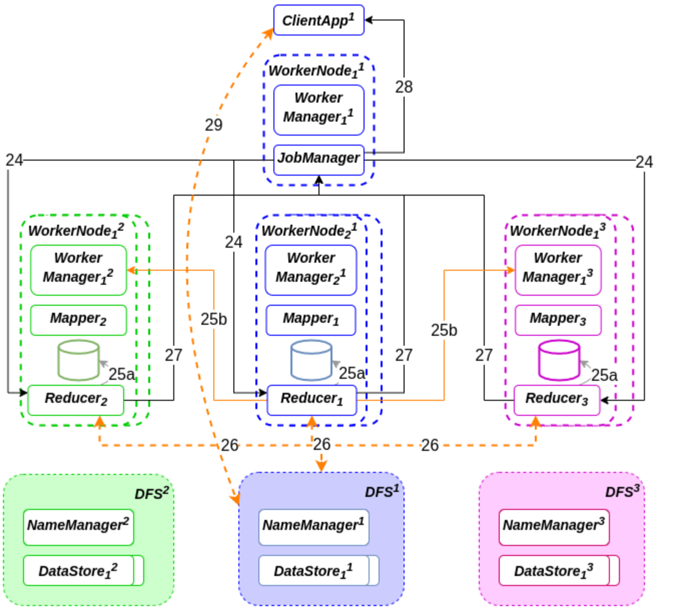
21b: SOA $\leftarrow sck_{m_a, nm^j} : \{pmk_{m_a, ds_v^j}\}, \{tkt^{nm^j}_{m_a, ds_v^j}\}, ssk_{nm^j, m_a}$

21c: GE2A $\leftarrow dfk^j, pmk_{m_a, ds_v^j} : ssk_{m_a, ds_v^j}$

22: EXT $\leftarrow$ none : none

23: SOA $\leftarrow sck_{jm, m_a} : ssk_{m_a, jm}$

(c)



24: SOA ← sck_{jm,r_b} : $\{pck^j\}, dfk^1, \{pmk_{r_b,wm_u^j}\}, \{tkt^{jm}_{r_b,wm_u^j}\}, pmk_{r_b,nm^1}, tkt^{jm}_{r_b,nm^1}, ssk_{jm,r_b}$
25a: EXT ← none : none
25b: GE2A ← pck^j, pmk_{r_b,wm_u^j} : ssk_{r_b,wm_u^j}
26a: GE2A ← dfk^1, pmk_{r_b,nm^1} : $sck_{r_b,nm^1}, ssk_{r_b,nm^1}$
26b: SOA ← sck_{r_b,nm^1} : $\{pmk_{r_b,ds_v^1}\}, \{tkt^{nm^1}_{r_b,ds_v^1}\}, ssk_{nm^1,r_b}$
26c: GE2A ← dfk^1, pmk_{r_b,ds_v^1} : ssk_{r_b,ds_v^1}

27: SOA ← sck_{jm,r_b} : $ssk_{r_b,jm}$
28: SOA ← $sck_{c^1,jm}$: ssk_{jm,c^1}
29a: GE2A ← dfk^1, pmk_{c^1,nm^1} : $sck_{c^1,nm^1}, ssk_{c^1,nm^1}$
29b: SOA ← sck_{c^1,nm^1} : $\{pmk_{c^1,ds_v^1}\}, \{tkt^{nm^1}_{c^1,ds_v^1}\}, ssk_{nm^1,c^1}$
29c: GE2A ← dfk^1, pmk_{c^1,ds_v^1} : ssk_{c^1,ds_v^1}

Conventions and notes:
- The authentication protocol (pro), the authentication keys used ($authkey_1[, authkey_2]$) , and keys ($key_1, key_2, \ldots$) exchanged during authentication are denoted as
  $pro \leftarrow authkey_1[, authkey_2] : key_1, key_2, \ldots ;$
- 'none' indicates no MIEA keys are used or exchanged;
- $\{item_1, item_2, ...\}$ indicates a set of items $item_1, item_2, ...$;
- The following keys are established in each $MRDomain_i$ prior to the job execution: $ok^i, pik^i, pck^i, dfk^i$, $\{pmk_{rm^i,rm^j}\}, pmk_{c^i,rm^i}, \{pmk_{wm_u^i,rm^i}\}, tsk_{nm^i,rm^i}, \{tsk_{ds_v^i,nm^i}\}, \{tsk_{wm_u^i,rm^i}\}$;

A ←- - - - - - - - → B Multiple interactions between A and B

A ——→ B A single interaction initiated by A to B

EXT / EMB

GP2A

GE2A

SOA

(d)

Figure 5.13: The job execution flow when MIEA is applied.
(a) Job submission phase: execution request step.
(b) Job submission phase: worker allocation step. (c) The map phase. (d) The reduce phase.

5.6 The Running Example

We here use the running example described in Section 4.6 to explain how the components of MIEA work when MIEA is applied to the cyberthreat analysis job in the example. The explanation also covers how different classes of keys are used to accomplish entity authentication and how the keys in a higher level of the key hierarchy are used to distribute the keys in a lower level of the key hierarchy.

The execution of the job in the example consists of 29 operational steps (as detailed in Section 7.2). In each of the steps, there can be interactions taking place between multiple pairs of MR components involved. For example, in step 2 shown in Figure 4.8, there are three interactions, one for each of the pairs, $ClientApp^1$ and $ResourceManager^1$, $ClientApp^2$ and $ResourceManager^2$, and $ClientApp^3$ and $ResourceManager^3$. Entity authentication is enforced before any interaction is taking place to verify and establish the identities of the interacting components. Each such authentication instance is carried out by using one of the

MIEA protocols, i.e., GP2A, GE2A, and SOA. GP2A and GE2A are two-factor (two keys) entity authentication protocols designed for initial interactions. SOA is a one-factor (one key) entity authentication protocol designed for subsequent interactions. The classifications of the keys used in the authentication are summarised in Figure 5.3 and the notations used to refer to the keys are given in Table 5.2. For GP2A and GE2A, a group key and a primary key (pre-shared and encapsulated, respectively) are used to generate and verify challenges and responses exchanged between the interacting entities. The group key is used to deter attacks caused by outsiders (e.g., components that do not have a DFS-C key dfk^1 cannot access DFS^1). The primary key (pairwise key) makes it more difficulty for insiders to mount impersonation attacks on the system (e.g., $Mapper^2$ cannot impersonate $Mapper^1$ when it contacts $JobManager$ as it does not have pmk_{jm,m_1}). The hybrid use of a group key and a primary key also prepares the ground for future work or other researchers to incorporate other security services, e.g., fault tolerance. For SOA, only a secondary key is used to accomplish the authentication task. A sealing key is shared between two components for the distribution of an encapsulated primary key (the container for an encapsulated primary key is called a ticket). A session key is used to secure sensitive data transmitted between the components after they are mutually authenticated, and it is only used for the session. A MAC key is used to ensure the authenticity of the protocol messages. A credential encryption key is used to securely distribute new keys for subsequent authentications. For each authentication instance, a MAC key and a credential encryption key are derived locally on the components and with a combined use of a group key, a primary key, a secondary key, and a nonce. This is described in the descriptions of each of the GP2A, GE2A, and SOA protocols in Section 5.5.4.

To demonstrate how different classes of keys are used, we here explain how entity authentication is carried out in step 2 through to step 4c (shown in Figure 5.13). In this demonstration, we shall describe only the actions performed by $ClientApp^1$, $ResourceManager^1$, $NameManager^1$, and $DataStore_1^1$ and the keys needed for the authentication of these components. It is assumed that, prior to the execution of the job, an OrgDomain key (a group key) ok^1 and a pre-shared primary key pmk_{c^1,rm^1} are established on $ClientApp^1$ and $ResourceManager^1$; a DFS-C key (a group key) dfk^1 is established on $ResourceManager^1$, $NameManager^1$, and $DataStore_1^1$; a sealing key slk_{nm^1,rm^1} is established on $NameManager^1$ and $ResourceManager^1$; and a sealing key $slk_{ds_1^1,nm^1}$ is established on $DataStore_1^1$ and $NameManager^1$.

In step 2, $User^1$ authenticates to $ClientApp^1$ by using an existing authentication service (e.g., a password-based authentication for Linux systems). $User^1$ then uses $ClientApp^1$ to authenticate to $ResourceManager^1$ before submitting a request to start the new job. The authentication between $ClientApp^1$ and $ResourceManager^1$ is done by using the GP2A protocol with the OrgDomain key (group key) ok^1 and the primary key (pairwise key) pmk_{c^1,rm^1}. The detailed descriptions of the authentication flow and protocol messages of GP2A have been given in Section 5.5.4.1. In the last protocol message, $User^1$ sends the secondary key sck_{c^1,rm^1} and a session key ssk_{c^1,rm^1} to $ResourceManager^1$. sck_{c^1,rm^1} will be used to authenticate $ResourceManager^1$ to $ClientApp^1$ in step 3 and ssk_{c^1,rm^1} will be used to secure data transmitted between $ResourceManager^1$ to $ClientApp^1$ in this session (all the remaining session keys in a similar fashion).

In step 3, $ResourceManager^1$ accepts the request sent by $ClientApp^1$. It authenticates to $ClientApp^1$ before sending a reply. The authentication is done by using the SOA protocol with the secondary key sck_{c^1,rm^1} established in step 2. The detailed descriptions of the authentication flow and protocol messages of SOA have been given in Section 5.5.4.3. In the last protocol message, $ResourceManager^1$ sends the DFS-C key dfk^1, the primary key pmk_{c^1,nm^1}, the ticket $tkt_{c^1,nm^1}^{rm^1}$ (containing pmk_{c^1,nm^1} which is encrypted with slk_{nm^1,rm^1}), and a new session key ssk_{rm^1,c^1} to $ClientApp^1$. dfk^1 and pmk_{c^1,nm^1} will be used to authenticate $ClientApp^1$ to $NameManager^1$ in step 4a and $tkt_{c^1,nm^1}^{rm^1}$ will be used to distribute pmk_{c^1,nm^1} to $NameManager^1$ in step 4a.

In step 4, after $ClientApp^1$ receives the reply containing the keys to authenticate to $NameManager^1$ from $ResourceManager^1$, $ClientApp^1$ authenticates to DFS^1 to write the input dataset ($File_1$) and the job configuration file onto DFS^1. This step consists of three further steps.

- In step 4a, $ClientApp^1$ authenticates to $NameManager^1$ before sending a request for a list of DataStores to write the data. The authentication is done by using the GE2A protocol with the DFS-C key (group key) dfk^1 and the primary key (pairwise key) pmk_{c^1,nm^1} established in step 3. The authentication flow and protocol messages of GE2A have been described in Section 5.5.4.2. It is worth noting that $NameManager^1$ obtains pmk_{c^1,nm^1} from $tkt_{c^1,nm^1}^{rm^1}$ contained in the first protocol message sent by $ClientApp^1$. In the last protocol message, $ClientApp^1$ sends the secondary key sck_{c^1,nm^1} and a session key ssk_{c^1,nm^1} to $NameManager^1$. sck_{c^1,nm^1} will be used to authenticate $NameManager^1$ to $ClientApp^1$ in step 4b.
- In step 4b, $NameManager^1$ accepts the request. It authenticates to $ClientApp^1$ before sending the list of DataStores (here, the list contains only $DataStore_1^1$). The authentication is done by using the SOA protocol with the secondary key sck_{c^1,nm^1} established in step 4a. In the last protocol message, $NameManager^1$ sends the primary key pmk_{c^1,ds_1^1}, the ticket $tkt_{c_1,ds_1^1}^{nm^1}$ (containing pmk_{c^1,ds_1^1} which is encrypted with $slk_{ds_1^1,nm^1}$), and a new session key ssk_{nm^1,c^1} to $ClientApp^1$. pmk_{c^1,ds_1^1} will be used to authenticate $ClientApp^1$ to $DataStore_1^1$ and $tkt_{c_1,ds_1^1}^{nm^1}$ will be used to distribute pmk_{c^1,ds_1^1} to $DataStore_1^1$ in step 4c.
- In step 4c, $ClientApp^1$ receives the reply. It authenticates to $DataStore_1^1$ before writing the data to $DataStore_1^1$. The authentication is done by using the GE2A protocol with the DFS-C key dfk^1 established in step 3 and the primary key pmk_{c^1,ds_1^1} established in step 4b. $DataStore_1^1$ obtains pmk_{c^1,ds_1^1} from $tkt_{c_1,ds_1^1}^{nm^1}$ contained in the first protocol message sent by $ClientApp^1$. In the last protocol message, $ClientApp^1$ sends a session key ssk_{c^1,ds_1^1} to $DataStore_1^1$.

The key hierarchy showing how keys are distributed during step 2 through to step 4c is shown in Figure 5.14. The detailed descriptions of all the operational steps are given in Section 7.3.

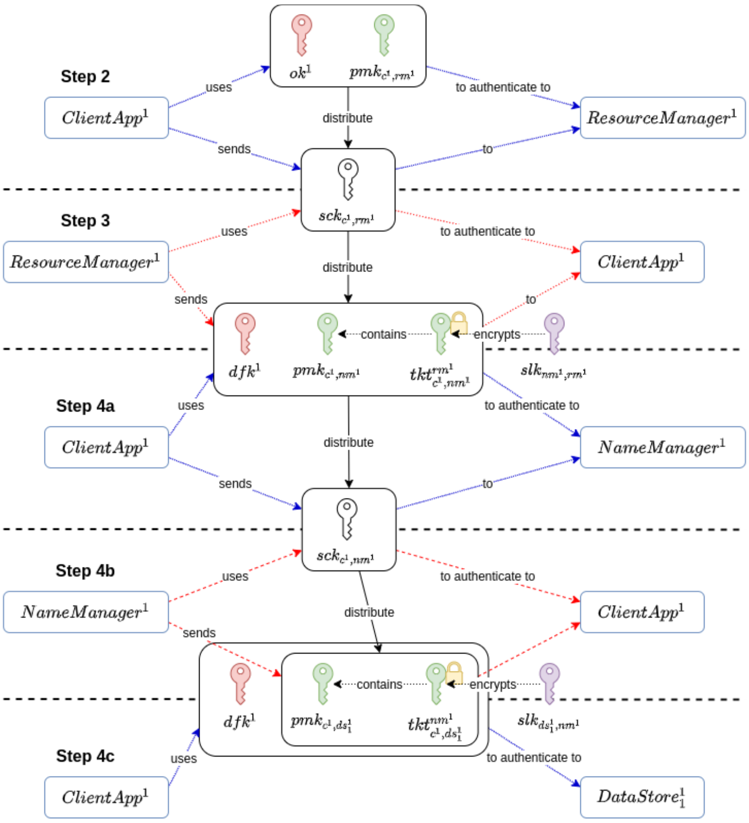


Figure 5.14: The key hierarchy in step 2 through to step 4c.

5.7 Security Analysis

The security of MIEA is analysed using informal and formal analysis methods. With the informal analysis method, we analyse the security properties of MIEA against the security requirements specified in Section 4.5.2. Next, MIEA is formally analysed by using symbolic analysis and complexity analysis. The symbolic analysis validates the security properties of each of the three protocols by finding traces of states that lead to the violation of the properties. The complexity analysis is done on the weakest link of MIEA, showing how much effort (in terms of computation) is required to mount any of the attacks identified in Section 4.4.1 against the system.

5.7.1 Informal Analysis

MIEA provides entity authentication to MR services throughout the course of a job execution. Each of the interactions between MR components is protected by a corresponding protocol (GP2A, GE2A, or SOA). As the three protocols have a common authentication flow, they provide security protections in a similar manner. In the last subsection, we compare the security properties of the MIEA protocols with those of the most related entity authentication protocols, i.e., Kerberos and NSLPK.

5.7.1.1 Mutual Authentication

Two components, an initiator I and a respondent R, of an interaction should be able to verify the identity of each other. With GP2A, the identities of I and R are assured by demonstrating the knowledge of a group key gk and a pairwise key (primary key) $pmk_{I,R}$ which should be known to only I and R. As explained in Section 5.5.4.1, in **Step 3**, I verifies the identity of R by checking whether the response (n_1') generated by R equals the challenge (n_1) generated by I in **Step 1**. If $n_1 == n_1'$, then R is positively authenticated to I. Similarly, in **Step 4**, R verifies the identity of I by checking whether the response (n_2') generated by I equals the challenge (n_2) generated by R in **Step 2**. If $n_2 == n_2'$, then I is positively authenticated to R. Without the knowledge of both group key and pre-shared primary key, it is difficult to learn challenges (which are protected by nested encryption). In other words, it is hard for components other than I and R to generate messages containing the correct responses to the given challenges. Therefore, at the end of the execution of GP2A, both I and R are mutually authenticated.

GE2A differs from GP2A in how I and R establish a primary key $pmk_{I,R}$ used for mutual authentication. In GE2A, I obtains $pmk_{I,R}$ from a trusted third party (a ticket issuer) z whereas R obtains $pmk_{I,R}$ by decrypting $tkt_{I,R}^{z}$ sent from I with a sealing key $slk_{R,z}$ shared with z. It is difficult for components other than R to obtain $pmk_{I,R}$ from $tkt_{I,R}^{z}$. After R obtains $pmk_{I,R}$, the authentication flows are the same as those of GP2A. Therefore, using the same reasoning, at the end of the execution of GE2A, both I and R are mutually authenticated.

With SOA, I and R verify the identities of each other in the same manner as that of GP2A and GE2A. The only difference is that, instead of using two keys, a group key gk and a primary key $pmk_{I,R}$, only one secondary key $sck_{I,R}$ is used to protect challenges and responses exchanged between I and R. Therefore, at the end of the execution of SOA, I and R are mutually authenticated.

As a result of the above discussion, MIEA satisfies the requirement of mutual authentication (SR1).

5.7.1.2 Sensitive Data Confidentiality

Entities other than I and R should not be able to learn sensitive data transmitted between I and R; these data are authentication keys, nonces (challenges and responses), MAC keys, credential encryption keys, and credential packages. In GP2A, authentication keys (gk and $pmk_{I,R}$) are established on I and R prior to the current authentication instance through a secure channel and they are known to only I and R. Nonces are protected using a nested encryption method with both the authentication keys before transmitting with protocol messages. A MAC key and a credential encryption key used are generated locally by each of I and R. These keys are not transmitted over networks, thus, cannot be intercepted by any other entities. A credential package is protected using encryption with a corresponding credential encryption key. Although an attacker may intercept messages transmitted over networks, without knowing the authentication keys, it is computationally difficult for the attacker to decrypt the protected data and learn the nonces and credentials contained in credential packages. In addition, in an event that an attacker was able to learn the MAC key or the credential encryption key (e.g., by mounting brute-force attacks on tags or encrypted credential package), the attacker should not be able to learn the authentication keys used to derive the MAC and credential encryption keys owing to the security property provided by key derivation functions. Therefore, the confidentiality of the sensitive data is protected.

In GE2A, a ticket $tkt_{I,R}^{z}$ containing $pmk_{I,R}$ is protected by a sealing key $slk_{R,z}$ shared between R and a trusted third party z. It is computationally difficult for entities other than R to learn $pmk_{I,R}$ from $tkt_{I,R}^{z}$. As the authentication flow of GE2A is the same as that of GP2A, using the same method with GP2A, it is computationally difficult to reveal the sensitive data exchanged between I and R without knowing the authentication keys used, thus, the confidentiality of the data is protected.

With SOA, a secondary key $sck_{I,R}$, rather than gk and $pmk_{I,R}$, is used to protect sensitive data exchanged between I and R. $sck_{I,R}$ is securely distributed from I to R during the preceding initial interaction. Thus, $sck_{I,R}$ should not be revealed to any entities other than I and R. Again, using the same method with GP2A, the confidentiality of other sensitive data is also protected.

Based on the above discussion, MIEA meets the requirement of sensitive data confidentiality (SR2).

5.7.1.3 Replay Attack Protection

Protocol messages used for each authentication instance should be freshly generated. Any messages captured and replayed should be detected. MIEA uses freshly generated nonces (encrypted with authentication keys) to ensure the freshness of the messages. For a CH message (msg-GP2A1 for GP2A, msg-GE2A1 for GE2A, and msg-SOA1 for SOA) used in each of the protocols, even if an attacker can capture and replay the CH message, the attacker cannot impersonate I as it cannot read the challenge sent from R and generate the respective response in **Step 3**. In **Step 3** of each of the protocols where I receives an RC message (msg-GP2A2 for GP2A, msg-GE2A2 for GE2A, and msg-SOA2 for SOA), if the nonce n_1' contained in the message matches the nonce n_1 generated by I, I is assured that the RC message is not replayed and the component sending the message is indeed R. Similarly, in **Step 4** of each of the protocols where R receives an RP message (msg-GP2A3 for GP2A, msg-GE2A3 for GE2A, and msg-SOA3 for SOA), if the nonce n_2' contained in the message matches the nonce n_2 generated by R, R is assured that the RP message is not replayed and the component sending the message is indeed I. Therefore, MIEA provides a protection against replay attacks, thus, satisfying the requirement of replay attack protection (SR3).

5.7.1.4 Message Authenticity Protection

Entities other than the component generating a message should not be able to tamper with the message. Any tampered messages should be detected and discarded. This is achieved by using MACs. Each protocol message used in MIEA contains a MAC tag that protects the content of the message. The MAC key used to sign and verify the message is derived locally from a group key and a primary key (with the exception of SOA where the MAC key is derived from one secondary key) and the keys are kept secret by each of I and R. It is computationally difficult to forge a MAC tag for a new (or modified) message without knowing the MAC key shared between I and R. Any modifications made to a MAC-protected message will result in a negative verification, thus, the fraudulent attempt will be detected. Therefore, MIEA provides an assurance of message authenticity, meeting the requirement of message authenticity protection (SR4).

5.7.1.5 The Comparisons of Security Properties

Kerberos provides security properties of mutual authentication and replay attack protection, but not sensitive data confidentiality and message authenticity protection. With Kerberos, I and R mutually authenticate each other by demonstrating the knowledge of a pairwise secret key ($k_{I,R}$ issued by T) shared between the two entities, thus, achieving mutual authentication. Encrypted nonces (generated by K and T) and timestamps (generated by I) are used to ensure that messages are freshly (or recently) generated as only the entities knowing the secret key can generate such messages. However, nonces sent by I to K and T are not encrypted, thus, could be intercepted by an attacker. There is no data authentication facility in Kerberos. Therefore, it does not provide a protection of message authenticity.

NSLPK achieves mutual authentication, sensitive data confidentiality, and replay attack protection, but not message authenticity protection. These security properties are achieved by using nonces in conjunction with an asymmetric-key cryptosystem. With NSLPK, I and R exchange a number of messages containing challenges and responses encrypted with the public keys of the receiving entities. As private keys are kept secret to the respective owners, no other entities can decrypt a challenge message to obtain a nonce so that it can generate a response message. By demonstrating the knowledge of the private keys, I and R are mutually authenticated. As nonces and private keys are not revealed to other unrelated entities, the confidentiality of sensitive data is preserved. As long as nonces used are freshly generated and not repeated, a protection against replay attacks is achieved. Like Kerberos, there is no data authentication facility in NSLPK, therefore, it does not provide a message authenticity protection.

The comparisons of the security properties achieved by MIEA, Kerberos, and NSLPK are summarised in Table 5.9.

Table 5.9: The comparisons of security properties achieved by the MIEA protocols, the Kerberos protocol, and the NSLPK protocol.

Security Requirement	Kerberos	NSLPK	MIEA
(SR1) Mutual authentication	√	√	√
(SR2) Sensitive data confidentiality	×	√	√
(SR3) Replay attack protection	√	√	√
(SR4) Message authenticity protection	×	×	√

5.7.2 Symbolic Analysis

Symbolic analysis validates the security properties provided by protocols and helps identify security weaknesses that are subtle and could be missed by informal analysis [157][158]. An example is a discovery of an MITM attack on the Needham-Schroeder Public Key (NSPK) authentication protocol [159] by Lowe [31] using Casper [160] and Failures-Divergence Refinement (FDR) checker [161] (also collectively referred to as Casper/FDR).

Compared with a computational approach which provides a strong security verification proof, a symbolic approach is less complicated and can be used by inexperienced users. In additions, a number of symbolic based software tools have been developed to automate the verification of the security properties, and examples of such tools include Naval Research

Laboratory (NRL) protocol analyzer [162], Automated Validation of Internet Security Protocols and Applications (AVISPA) [163], and FDR [161], all of which are based on a state-exploration technique. This lowers the risk of errors made by the user [164][165].

With a symbolic approach, a security analysis is performed on an abstract view of a protocol [164][165]. In this abstract view, data contained in protocol messages are expressed using symbolic terms and cryptographic schemes are expressed as functions operated on the terms. Generally, symbolic analysis consists of three operational steps: (1) formally modelling a protocol; (2) specifying security properties to be verified; and (3) verifying the protocol model against the specified security properties. Steps (1) and (2) are accomplished using a high-level specification language. The selection of such a language is determined based on the symbolic analysis tool used. In this way, a user may gain the benefits of software-assisted verification. In Step (3), the tool is used to verify the protocol model formulated in Step (1) against the security properties specified in Step (2). If no attacks are found under a certain set of conditions (e.g., bounded or unbounded number of sessions), the tool returns a positive verification; otherwise, the tool returns traces of possible attacks (i.e., steps to mount such attacks), showing the flaws of the protocol.

In the following, we compare different verification tools reported in literature and select one for our work. We explain the attacker model used. Next, we describe the security properties that are supported by Scyther. We then describe how the protocols are formally modelled, before presenting verification results.

5.7.2.1 Verification Tool Comparisons and Selection

There are a number of symbolic analysis tools reported in literature which have been successfully used to verify the correctness of, or identify attacks on, security protocols. Examples of such tools include NRL [162], Murφ [166], Athena [167], Casper/FDR [160][161], AVISPA [163], Scyther [168], ProVerif [165][169], and TAMARIN [170]. As NRL [162], Murφ [166], and Athena [167] are not publicly available at the time of this writing, they are excluded from this work. In the following, we contrast the remaining tools before selecting one for our work.

FDR [161] is a refinement checker designed to analyse formal models of protocols or applications. The models are expressed in the Communicating Sequential Processes (CSP) language [171]. Casper [160] was created to simplify the process of expressing a security protocol with CSP. With Casper, a user can express a model using more abstract notations. A file containing such notations is called a Casper script. Casper translates the Casper script into CSP which can be processed by FDR. According to [172], the performance (in terms of execution time) of Casper/FDR is typically lower than other tools.

AVISPA [163] integrates four different backends, namely On-the-Fly Model-Checker (OFMC), CL-based Attack Searcher (CL-AtSe), SAT-based Model Checker (SATMC), and Tree-Automata-based Protocol Analyzer (TA4SP), for the verification of protocols. The specifications of a protocol and security properties are written in High-Level Protocol Specification Language (HLPSL). The HLPSL file is then translated into an intermediate format that is supported by the backend used. For demonstration, a number of security protocols have been modelled in HLPSL and a collection of these protocol specifications are stored in the AVISPA Library [173]. Despite its popularity and ease of use, the performance of AVISPA is lower than those of other tools [172].

Scyther [168] extends the ideas used in Athena [167] to verify the security properties (including authentication and secrecy) of a protocol. It uses a symbolic backward search based on patterns. It supports bounded and unbounded verifications with guaranteed termination. A protocol model and security properties can be expressed by using Security Protocol Description Language (SPDL). With SPDL, the protocol model can be expressed using notations similar to those used in literature for describing security protocols. Scyther has been used to model protocols collected in the Security Protocols Open Repository (SPORE) library [174]. Based on the experimental evaluation reported in [172], Scyther is the second fastest tool after ProVerif.

ProVerif [165][169] abstracts a representation of a protocol by using a set of Horn clauses. It can analyse the protocol for an unbounded number of sessions. It supports a wide range of cryptographic primitives and can verify secrecy, correspondence, and a number of equivalence properties. In comparison with other tools, the performance of ProVerif is the highest [172]. However, in ProVerif, modelling a protocol is more difficult than other tools. In addition, it may find false attacks and it also does not always terminate.

TAMARIN [170] generalises a backward search approach used by Scyther [168]. The specifications of a protocol and security properties are done, respectively, by multiset rewriting rules and in a guarded fragment of first-order logic. It supports complex control flows (e.g., loops), complex security properties (e.g., eCK model [175] for key exchange protocols), and equational theories (e.g., Diffie-Hellman and bilinear pairings). These features are achieved at a cost of more complicated protocol modelling in comparison with other tools such as Scyther. Another limitation of TAMARIN is that it does not always terminate.

Based on the above discussions, we have chosen Scyther as the symbolic analysis tool for our work. The selection of Scyther is made based on the following reasons: (1) Scyther has been successfully used to verify many security protocols, including those collected in the SPORE library [174], this has demonstrated its capabilities and effectiveness; (2) it supports the verification of authentication and secrecy which can be used to, respectively, verify the security properties of mutual authentication (SR1) and sensitive data confidentiality (SR2); (3) the notations used in SPDL are widely used in describing protocols published in literature, thus, improving readability and understandability; (4) the verification of security properties can be done automatically, eliminating errors that could be introduced by a manual verification method; and (5) the performance of Scyther is much higher than other tools with similar features [172].

5.7.2.2 Attacker Model

The following attacker model is used in the verification of the MIEA protocols using Scyther.

(EAM1) Perfect cryptography is assumed, cryptographic schemes are secure, and tokens generated by the schemes cannot be reverse without corresponding keys.

(EAM2) An attacker can intercept and modify any messages, inject new messages, and send them to any entities in the network.

(EAM3) The attacker can perform any cryptographic operations as long as it knows the corresponding keys.

(EAM4) The attacker cannot mount cryptanalytical attacks and cannot guess secrets (e.g., nonces and keys).

5.7.2.3 Security Properties

Scyther supports the verifications of a number of security properties. For this work, we consider two properties, non-injective synchronisation (authentication) and secrecy (confidentiality) [176].

Non-injective synchronisation ensures that messages exchanged between two entities are indeed sent and received by the claimed sender and receiver, the messages have not been tampered with, and they are exchanged in the correct order. In other words, the message exchange has occurred exactly as specified by the protocol description. We use this property to verify the security requirement of mutual authentication (SR1). It is worth noting that non-injective synchronisation only considers the contents and the ordering, but not the freshness, of the messages. Hence, it does not provide a protection against replay attacks.

Secrecy ensures that data transmitted between two honest and uncompromised entities are not revealed to an attacker, particularly when the data are transmitted over an insecure network where the attacker may intercept any transmitted messages. We use this property to verify the security requirement of sensitive data confidentiality (SR2).

5.7.2.4 Protocol Modelling

As mentioned earlier, with Scyther, the specifications of a protocol and security properties are expressed in SPDL. Each SPDL file contains the descriptions of protocols and the associated data (e.g., static constants, user-defined functions, and keys) for a single analysis. Each protocol contains the descriptions of entity roles. Each role further consists of three sections: constants and variables, messages, and security claims (security properties to be verified). The detail SPDL specifications and how to use Scyther are explained in the manual of Scyther (https://github.com/cascremers/scyther/blob/master/gui/scyther-manual.pdf).

In this work, each of the three MIEA protocols along with the corresponding security properties is written in a separate SPDL file. The analyses of the protocols are conducted independently. We do not consider multi-protocol or cross-protocol verification as the three protocols are designed to run independently and the messages used in each of the protocols are different (i.e., different PRO values).

For GP2A, two entity roles are defined, I for initiators and R for respondents. The contents of the messages and the message flows are the same as those described in Section 5.5.4.1. In each of I and R, there is one non-injective synchronisation claim and seven secrecy claims (for gk, $pmk_{I,R}$, $mk_{I,R}$, $ck_{I,R}$, n_1, n_2, and $credential\ package$).

For GE2A, in addition to I and R, an additional role Z for a trusted third party is introduced. This is for the establishment of a primary key $pmk_{I,R}$ and a ticket $tkt_{I,R}^{Z}$ on an initiator of role I. At the beginning of the protocol description, two messages are exchanged between I and Z to, respectively, request and dispatch $pmk_{I,R}$ and $tkt_{I,R}^{z}$. The remaining messages and authentication flows are the same as those described in Section 5.5.4.2. Like GP2A, in each of I and R, there is one non-injective synchronisation claim and seven secrecy claims (for gk,

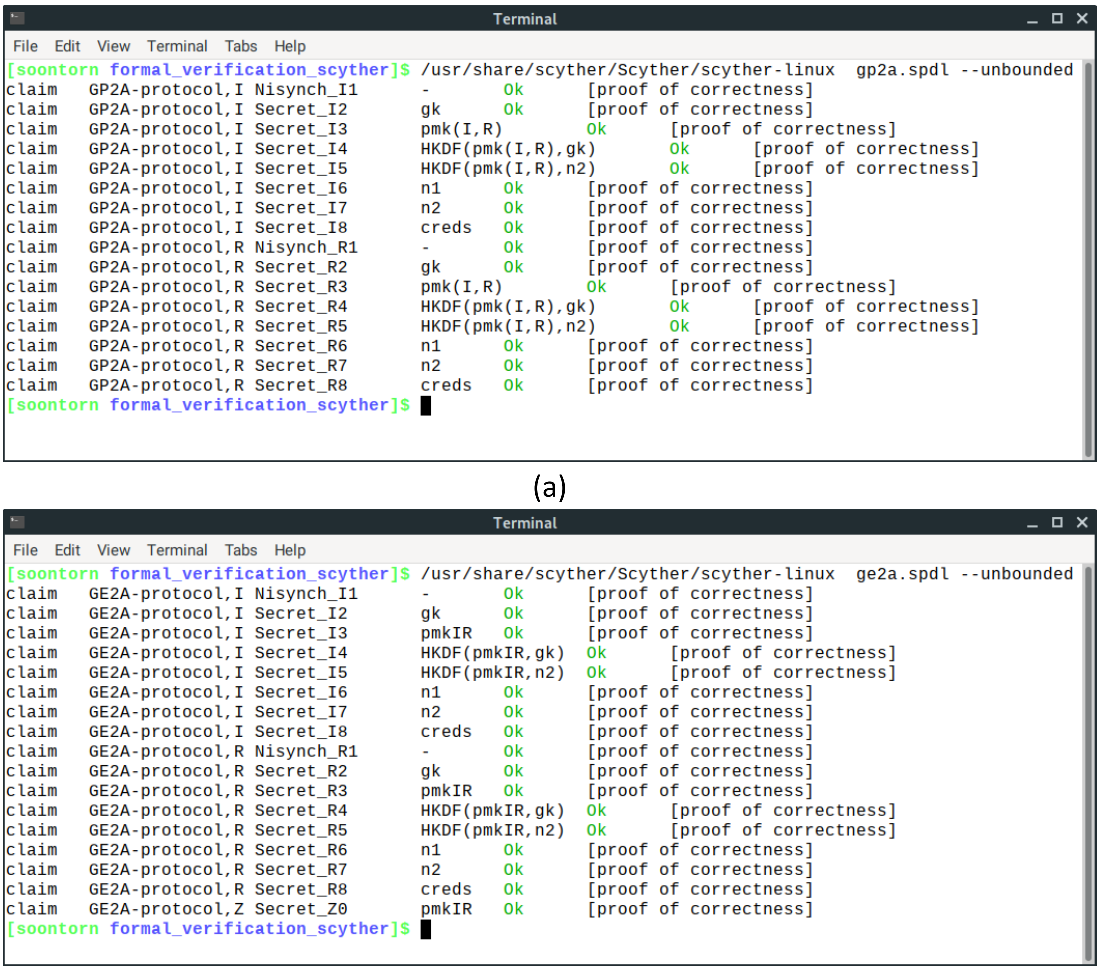
$pmk_{I,R}$, $mk_{I,R}$, $ck_{I,R}$, n_1, n_2, and $credential\ package$). There is one additional secrecy claim for $pmk_{I,R}$ in role Z.

The specifications of SOA and the corresponding security properties are similar to those of GP2A. The only difference is that a group key gk is not used in SOA. Therefore, the secrecy claim for gk is excluded from I and R.

The contents of the SPDL files for the three protocols (i.e., gp2a.spdl for GP2A, ge2a.spdl for GE2A, and soa.spdl for SOA) are shown in Appendix A.

5.7.2.5 Verification Results

The verification results of the three protocols using Scyther under an unbounded number of sessions are presented in Figure 5.15. The results show that each of the three protocol passed the verifications against the specified security claims. Therefore, the three protocols satisfy the security requirements of mutual authentication (SR1) and sensitive data confidentiality (SR2).

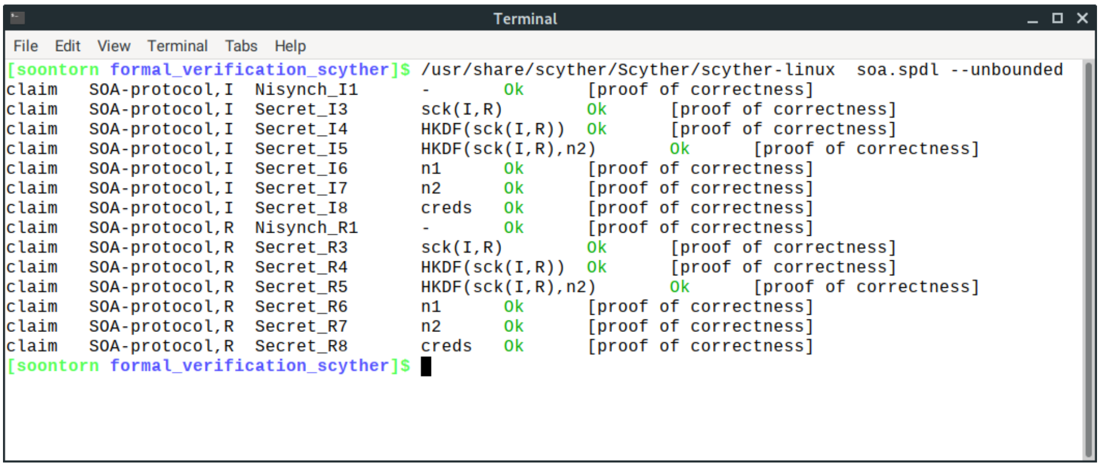


```
[soontorn formal_verification_scyther]$ /usr/share/scyther/Scyther/scyther-linux  gp2a.spdl --unbounded
claim   GP2A-protocol,I Nisynch_I1      -       Ok      [proof of correctness]
claim   GP2A-protocol,I Secret_I2       gk      Ok      [proof of correctness]
claim   GP2A-protocol,I Secret_I3       pmk(I,R)        Ok      [proof of correctness]
claim   GP2A-protocol,I Secret_I4       HKDF(pmk(I,R),gk)       Ok      [proof of correctness]
claim   GP2A-protocol,I Secret_I5       HKDF(pmk(I,R),n2)       Ok      [proof of correctness]
claim   GP2A-protocol,I Secret_I6       n1      Ok      [proof of correctness]
claim   GP2A-protocol,I Secret_I7       n2      Ok      [proof of correctness]
claim   GP2A-protocol,I Secret_I8       creds   Ok      [proof of correctness]
claim   GP2A-protocol,R Nisynch_R1      -       Ok      [proof of correctness]
claim   GP2A-protocol,R Secret_R2       gk      Ok      [proof of correctness]
claim   GP2A-protocol,R Secret_R3       pmk(I,R)        Ok      [proof of correctness]
claim   GP2A-protocol,R Secret_R4       HKDF(pmk(I,R),gk)       Ok      [proof of correctness]
claim   GP2A-protocol,R Secret_R5       HKDF(pmk(I,R),n2)       Ok      [proof of correctness]
claim   GP2A-protocol,R Secret_R6       n1      Ok      [proof of correctness]
claim   GP2A-protocol,R Secret_R7       n2      Ok      [proof of correctness]
claim   GP2A-protocol,R Secret_R8       creds   Ok      [proof of correctness]
[soontorn formal_verification_scyther]$
```

(a)

```
[soontorn formal_verification_scyther]$ /usr/share/scyther/Scyther/scyther-linux  ge2a.spdl --unbounded
claim   GE2A-protocol,I Nisynch_I1      -       Ok      [proof of correctness]
claim   GE2A-protocol,I Secret_I2       gk      Ok      [proof of correctness]
claim   GE2A-protocol,I Secret_I3       pmkIR   Ok      [proof of correctness]
claim   GE2A-protocol,I Secret_I4       HKDF(pmkIR,gk)  Ok      [proof of correctness]
claim   GE2A-protocol,I Secret_I5       HKDF(pmkIR,n2)  Ok      [proof of correctness]
claim   GE2A-protocol,I Secret_I6       n1      Ok      [proof of correctness]
claim   GE2A-protocol,I Secret_I7       n2      Ok      [proof of correctness]
claim   GE2A-protocol,I Secret_I8       creds   Ok      [proof of correctness]
claim   GE2A-protocol,R Nisynch_R1      -       Ok      [proof of correctness]
claim   GE2A-protocol,R Secret_R2       gk      Ok      [proof of correctness]
claim   GE2A-protocol,R Secret_R3       pmkIR   Ok      [proof of correctness]
claim   GE2A-protocol,R Secret_R4       HKDF(pmkIR,gk)  Ok      [proof of correctness]
claim   GE2A-protocol,R Secret_R5       HKDF(pmkIR,n2)  Ok      [proof of correctness]
claim   GE2A-protocol,R Secret_R6       n1      Ok      [proof of correctness]
claim   GE2A-protocol,R Secret_R7       n2      Ok      [proof of correctness]
claim   GE2A-protocol,R Secret_R8       creds   Ok      [proof of correctness]
claim   GE2A-protocol,Z Secret_Z0       pmkIR   Ok      [proof of correctness]
[soontorn formal_verification_scyther]$
```

(b)



```
[soontorn formal_verification_scyther]$ /usr/share/scyther/Scyther/scyther-linux  soa.spdl --unbounded
claim   SOA-protocol,I  Nisynch_I1      -       Ok      [proof of correctness]
claim   SOA-protocol,I  Secret_I3       sck(I,R)        Ok      [proof of correctness]
claim   SOA-protocol,I  Secret_I4       HKDF(sck(I,R))  Ok      [proof of correctness]
claim   SOA-protocol,I  Secret_I5       HKDF(sck(I,R),n2)       Ok      [proof of correctness]
claim   SOA-protocol,I  Secret_I6       n1      Ok      [proof of correctness]
claim   SOA-protocol,I  Secret_I7       n2      Ok      [proof of correctness]
claim   SOA-protocol,I  Secret_I8       creds   Ok      [proof of correctness]
claim   SOA-protocol,R  Nisynch_R1      -       Ok      [proof of correctness]
claim   SOA-protocol,R  Secret_R3       sck(I,R)        Ok      [proof of correctness]
claim   SOA-protocol,R  Secret_R4       HKDF(sck(I,R))  Ok      [proof of correctness]
claim   SOA-protocol,R  Secret_R5       HKDF(sck(I,R),n2)       Ok      [proof of correctness]
claim   SOA-protocol,R  Secret_R6       n1      Ok      [proof of correctness]
claim   SOA-protocol,R  Secret_R7       n2      Ok      [proof of correctness]
claim   SOA-protocol,R  Secret_R8       creds   Ok      [proof of correctness]
[soontorn formal_verification_scyther]$ 
```

(c)

Figure 5.15: Symbolic analysis of the three MIEA protocols using Scyther. (a) The GP2A protocol. (b) The GE2A protocol. (c) The SOA protocol.

5.7.3 Complexity Analysis

The strengths of the security protections offered by MIEA rely on the strengths of the underlying cryptographic schemes. In the following, we first give a list of notations used in this analysis, then the security strengths of the cryptographic schemes (i.e., a symmetric-key based encryption scheme and a MAC scheme), before analysing the strength of MIEA.

5.7.3.1 Notations

The notations used in this analysis are shown in Table 5.10. The lengths are expressed in bits.

Table 5.10: Notations used in the complexity analysis of MIEA.

Symbol	Meaning
L_k	Key length
L_d	Plaintext length
L_m	MAC input data length
L_τ	MAC tag length

Notes: - All group keys and pairwise keys have the same lengths (L_k).
- MAC input data refer to data to be signed with MAC.

5.7.3.2 The Strengths of Cryptographic Schemes

The strengths of cryptographic schemes are measured as the upper bound of computational complexity needed to compromise an authentication token. Such complexity is usually expressed as 2^n where the value of n is dependent on the scheme and parameters used.

Attacks on cryptographic schemes can be largely classified into two groups, cryptanalytical attacks and brute-force attacks [177]. Cryptanalytical attacks on symmetric-key cryptosystems (encryption and MAC schemes) can be mitigated by using schemes that have been well studied and have no known vulnerabilities. Hence, these attacks are not considered in this work.

Attacks on encryption schemes can be classified into two groups, encryption key attacks (guessing the keys used for encryption) and plaintext attacks (guessing the plaintexts of given encrypted data). Mounting such attacks requires complexities of 2^{L_k} and 2^{L_d}, respectively [178]. Therefore, the complexity of successfully mounting an attack on an encryption scheme is $2^{\min(L_k, L_d)}$.

Attacks on MACs are tag forgery. This is done by (1) finding a new data object that produces the same tag, (2) guessing the key used to sign (and verify) the tag, or (3) guessing the tag for a new data object. The complexities of these actions are 2^{L_m}, 2^{L_k}, and 2^{L_τ}, respectively [178]. Therefore, the complexity of successfully mounting a tag forgery attack is $2^{\min (L_m, L_k, L_\tau)}$.

5.7.3.3 *Impersonation Attacks*

To mount an impersonation attack, an attacker has to generate a response in correspondence to a given challenge (a nonce) which is protected by encryption. To obtain such a challenge, the adversary may either (1) guess the keys used for decryption or (2) guess the nonce. For (1), two keys (a group key and a primary key) are used for encryption when GP2A or GE2A is applied, thus, guessing these two keys requires a complexity of $2^{L_k} + 2^{L_k} = 2^{L_k+1}$. When SOA is applied, one key (a secondary key) is used for encryption, thus, the complexity of guessing the key is 2^{L_k}. For (2), in all the cases, guessing the challenge requires a complexity of 2^{L_d}. Therefore, the complexity of mounting a successful impersonation attack against MIEA is $2^{\min (L_k, L_d)}$.

5.7.3.4 *Confidential Data Exposure Attacks*

Confidential data used in MIEA are the nonces (used as challenges and responses), group keys, and pairwise keys. The confidentiality is breached if any of these items are revealed. To expose one of these items, the adversary should guess any of the keys (keys for encryption and keys to be encrypted) and nonces, which requires a complexity of 2^{L_k} and 2^{L_d}, respectively. Therefore, the complexity of mounting a successful confidential data exposure attack against MIEA is $2^{\min (L_k, L_d)}$.

5.7.3.5 *Replay Attacks*

As long as a challenge (nonce) is freshly generated and is not repeated, an adversary cannot mount a replay attack due to the lack of a message containing a corresponding response. Assuming that a challenge is repeated and the adversary has captured a message containing the response to the challenge, to mount a replay attack, the adversary has to learn the challenge (to find the corresponding response) which is protected with encryption. This is similar to mounting an impersonation attack. As explained earlier, the complexities of guessing the encryption keys are 2^{L_k+1} when GP2A or GE2A is applied and 2^{L_k} when SOA is applied, whereas the complexity of guessing the challenge is 2^{L_d}. Therefore, the complexity of mounting a successful replay attack against MIEA is also $2^{\min (L_k, L_d)}$.

5.7.3.6 *Message Tampering Attacks*

To tamper with a message without being detected, an adversary has to mount a tag forgery on the message. As explained earlier, this requires a complexity of $2^{\min (L_m, L_k, L_\tau)}$. Therefore, the complexity of mounting a successful message tampering attack against MIEA is $2^{\min (L_m, L_k, L_\tau)}$.

The security strength of MIEA is summarised in Table 5.11.

Table 5.11: The security strength of MIEA.

Attacks	Complexity
(T1) Impersonation attacks	$2^{\min(L_k, L_d)}$
(T2) Confidential data exposure attacks	$2^{\min(L_k, L_d)}$
(T3) Replay attacks	$2^{\min(L_k, L_d)}$
(T4) Message tampering attacks	$2^{\min(L_m, L_k, L_\tau)}$

5.8 Performance Evaluation

The performance of MIEA is theoretically evaluated in two aspects, computational and communication overheads. For benchmarking, the results are compared with the most related solutions, Kerberos [34][138][139] and NSLPK [31]. Kerberos is chosen because it is an efficient symmetric-key based entity authentication protocol that provides strong security protections and it is commonly used to provide secure access to many applications, such as Apache Hadoop [179]. NSLPK is chosen because it is an asymmetric-key based entity authentication protocol that have been well-studied and frequently discussed in literature. Although asymmetric-key based entity authentication protocols can be used in a context compatible with CBDC-MPC, they usually introduce a high-level of overhead costs. By evaluating the performance of NSLPK, we could learn how much overhead costs an asymmetric-key based entity authentication protocol introduces in comparison with symmetric-key based ones such as ours and Kerberos. The message transaction flows and operational steps of Kerberos and NSLPK are detailed in Appendix B.

5.8.1 Notations

The notations used in this performance evaluation are shown in Table 5.12.

Table 5.12: Notations used in performance evaluation of MIEA.

Symbols	Meanings
O_{se}, O_{sd}	Sym-Encryption, Sym-Decryption operation
O_{ae}, O_{ad}	Asym-Encryption, Asym-Decryption operation
O_{ss}, O_{sv}	SIG-Signing, SIG-Verification operation
O_{ms}, O_{mv}	MAC-Signing, MAC-Verification operation
O_{kd}	Key derivation operation
L_{hd}	The length of a message header
$L_{jid}, L_{req}, L_{icl}$ L_{mid}, L_{eid}	The lengths of a JID field, a REQ field, an ICL field, an MID field, an entity ID (EID) field
$L_n, L_t, L_\sigma, L_\tau, L_{tkt}$	The lengths of a nonce, a timestamp, a signature, a tag, a ticket
L_k, L_{pk}	The lengths of a symmetric key, a public key
L_{pkg}	The length of an encrypted credential package

5.8.2 Computational Overheads

The computational overheads are evaluated in terms of the number of cryptographic operations performed by each of the entities involved in an authentication instance. Non-cryptographic operations, such as equality check, are omitted as their costs (in terms of execution times) are negligible in comparison with those of cryptographic operations. Cryptographic operations are classified into five groups: Sym-Encryption and Sym-Decryption (O_{se}, O_{sd}), Asym-Encryption and Asym-Decryption (O_{ae}, O_{ad}), MAC-Signing and MAC-

Verification (O_{ms}, O_{mv}), SIG-Signing and SIG-Verification (O_{ss}, O_{sv}), and key derivation (O_{kd}). As the costs of operations are dependent on the sizes of data objects, we mark operations on potentially large objects with a superscripted asterisk (*).

5.8.2.1 GP2A Protocol

As shown in Figure 5.6, there are two entities, an initiator I and a respondent R, involved in an authentication instance. I performs cryptographic operations in **Step 1** and **Step 3**. In **Step 1**, I performs three sets of operations: the first is for generating an authenticator $auth_1$, i.e., encrypting a nonce n_1 ($2 * O_{se}$); the second is for generating a MAC key (O_{kd}); and the third is for generating a tag for a message msg-GP2A1 (O_{ms}). In **Step 3**, I performs six sets of operations: the first is for verifying a tag of a message msg-GP2A2 (O_{mv}); the second is for verifying an authenticator $auth_2$, i.e., decrypting a concatenation of two nonces ($2 * O_{sd}$); the third is for generating an authenticator $auth_3$, i.e., encrypting n_2 ($2 * O_{se}$); the fourth is for generating a credential encryption key (O_{kd}); the fifth is for encrypting a credential package (O_{se}^*); and the sixth is for generating a tag for a message msg-GP2A3 (O_{ms}^*). The total number of operations performed by I is $4 * O_{se} + 2 * O_{sd} + O_{ms} + O_{mv} + 2 * O_{kd} + O_{se}^* + O_{ms}^*$.

R performs cryptographic operations in **Step 2** and **Step 4**. In **Step 2**, R performs five sets of operations: the first is for generating a MAC key (O_{kd}); the second is for verifying a tag of a message msg-GP2A1 (O_{mv}); the third is for verifying an authenticator $auth_1$, i.e., decrypting a nonce n_1 ($2 * O_{sd}$); the fourth is for generating an authenticator $auth_2$, i.e., encrypting a concatenation of two nonces ($2 * O_{se}$); and the fifth is for generating a tag for a message msg-GP2A2 (O_{ms}). In **Step 4**, R performs four sets of operations: the first is for verifying a tag for a message msg-GP2A3 (O_{mv}^*); the second is for verifying an authenticator $auth_3$, i.e., decrypting n_2 ($2 * O_{sd}$); the third is for generating a credential decryption key (O_{kd}); and the fourth is for decrypting a credential package (O_{sd}^*). The total number of operations performed by R is $2 * O_{se} + 4 * O_{sd} + O_{ms} + O_{mv} + 2 * O_{kd} + O_{sd}^* + O_{mv}^*$.

5.8.2.2 GE2A Protocol

With the GE2A protocol, cryptographic operations performed by I and R are similar to those when GP2A is applied, with an exception that here, in **Step 2**, R has to decrypt a ticket $tkt_{I,R}^z$ to obtain a primary key $pmk_{I,R}$, thus, having one additional decryption operation (O_{sd}). Therefore, the numbers of operations performed by I and R are, respectively, $4 * O_{se} + 2 * O_{sd} + O_{ms} + O_{mv} + 2 * O_{kd} + O_{se}^* + O_{ms}^*$ and $2 * O_{se} + 5 * O_{sd} + O_{ms} + O_{mv} + 2 * O_{kd} + O_{sd}^* + O_{mv}^*$.

5.8.2.3 SOA Protocol

The authentication flow of SOA is similar to that of GP2A. The only difference is that, when SOA is applied, only one secondary key is used for the generation and verification of authenticators, cutting the total number of O_{se} by 3 (2 for I and 1 for R) and O_{sd} by 3 (1 for I and 2 for R). Therefore, the numbers of operations performed by I and R are, respectively, $2 * O_{se} + O_{sd} + O_{ms} + O_{mv} + 2 * O_{kd} + O_{se}^* + O_{ms}^*$ and $O_{se} + 2 * O_{sd} + O_{ms} + O_{mv} + 2 * O_{kd} + O_{sd}^* + O_{mv}^*$.

5.8.2.4 Kerberos Protocol

As shown in Figure B.1 in Appendix B, there are four entities, an initiator I, a KDC server K, a TGS server T, and a respondent R, involved in an authentication instance. I performs cryptographic operations in **Step 3** and **Step 5**. In each of these steps, I performs two sets of operations: the first is for decrypting the encrypted concatenation of a key and a nonce (O_{sd}); and the second is for generating an authenticator, i.e., encrypting a timestamp (O_{se}). Hence, the total number of operations performed by I is $2 * O_{se} + 2 * O_{sd}$.

K performs two sets of operations in **Step 2**: the first is for encrypting a concatenation of a pairwise key and a nonce (O_{se}); and the second is for generating a ticket $tkt_{I,T}^{K}$ (O_{se}). Hence, the total number of operations is $2 * O_{se}$.

T performs four sets of operations in **Step 4**: the first is for decrypting a ticket $tkt_{I,T}^{K}$ to obtain $k_{I,T}$ (O_{sd}); the second is for verifying an authenticator $auth_1$, i.e., decrypting $auth_1$ (O_{sd}); the third is for encrypting a concatenation of a pairwise key and a nonce (O_{se}); and the fourth is for generating a ticket $tkt_{I,R}^{T}$ (O_{se}). Hence, the total number of operations is $2 * O_{se} + 2 * O_{sd}$.

R performs two sets of operations in **Step 6**: the first is for decrypting a ticket $tkt_{I,R}^{T}$ to obtain $k_{I,R}$ (O_{sd}); and the second is for verifying an authenticator $auth_2$, i.e., decrypting $auth_2$ (O_{sd}). Hence, the total number of operations is $2 * O_{sd}$.

In a case that I have already obtained $k_{I,T}$ and $tkt_{I,T}^{K}$ from K, I can reuse the key and the ticket, thus, performing one less decryption operation (O_{sd}), K needs not perform any operation, and each of T and R performs the same number of operations.

5.8.2.5 NSLPK Protocol

As shown in Figure B.2 in Appendix B, there are three entities, an initiator I, a key server Z, and a respondent R, involved in an authentication instance. I performs cryptographic operations in **Step 3** and **Step 7**. In **Step 3**, I performs two sets of operations: the first is for verifying the public key pk_R (O_{sv}); and the second is for encrypting a concatenation of a nonce and an EID (O_{ae}). In **Step 7**, I performs two sets of operations: the first is for decrypting the encrypted concatenation of two nonces and an EID (O_{ad}); and the second is for encrypting n_2 (O_{ae}). Hence, the total number of operations is $2 * O_{ae} + O_{ad} + O_{sv}$.

Z performs cryptographic operations in **Step 2** and **Step 5**. In each of these steps, Z performs one operation, i.e., signing a concatenation of a public key and an entity ID (O_{ss}). Hence, the total number of operations is $2 * O_{ss}$.

R performs cryptographic operations in **Step 4**, **Step 6**, and **Step 8**. In **Step 4**, R performs one operation, i.e., decrypting the encrypted concatenation of a nonce and an EID (O_{ad}). In **Step 6**, R performs two sets of operations: the first is for verifying the public key pk_I (O_{sv}); and the second is for encrypting a concatenation of two nonces and an EID (O_{ae}). In **Step 8**, R performs one operation, i.e., decrypting the encrypted n_2 (O_{ad}). Hence, the total number of operations is $O_{ae} + 2 * O_{ad} + O_{sv}$.

In a case that both I and R already know the public key of each other, I performs one less SIG-Verification operation (O_{sv}), K needs not perform any operation, and R performs one less SIG-Verification operation (O_{sv}).

5.8.2.6 The Comparisons of Computational Overheads

The computational overheads imposed on individual entities when different entity authentication protocols are applied are summarised in Table 5.13. The result shows that, among the protocols, the three MIEA protocols (GP2A, GE2A, and SOA) introduce the largest number of cryptographic operations and the NSLPK protocol introduces the lowest number of operations. However, when the protocols are deployed on real systems, the computational overhead cost (in terms of execution times) introduced by the NSLPK protocol is likely to be the highest and the cost introduced by Kerberos should be the lowest. This is because the operations of the NSLPK protocol are asymmetric-key based, which is much more computationally expensive (a few magnitudes [45]) than symmetric-key based [177]. In contrast, the MIEA protocols and Kerberos, which are symmetric-key based, should be more efficient. In the CBDC-MPC context, NSLPK and Kerberos may increase the risk of creating performance bottlenecks on centralised credential servers (Z for NSLPK and K and T for Kerberos) when being applied to large-scale job executions involving a large number of Workers (particularly in the map phase). This is due to a large number of authentication requests by the Workers when they read data from and write data to the DFS clusters (each of a reading or writing request requires three interactions thus three authentication instances).

Table 5.13: The comparisons of the computational overheads imposed on individual entities by different entity authentication protocols.

Kerberos		
	Without $k_{I,T}$ and $tkt_{I,T}^{K}$ caching	With $k_{I,T}$ and $tkt_{I,T}^{K}$ caching
I	$2 * O_{se} + 2 * O_{sd}$	$2 * O_{se} + O_{sd}$
K	$2 * O_{se}$	-
T	$2 * O_{se} + 2 * O_{sd}$	$2 * O_{se} + 2 * O_{sd}$
R	$2 * O_{sd}$	$2 * O_{sd}$
Total	$6 * O_{se} + 6 * O_{sd}$	$4 * O_{se} + 5 * O_{sd}$
NSLPK		
	Without pk_I and pk_R caching	With pk_I and pk_R caching
I	$2 * O_{ae} + O_{ad} + O_{sv}$	$2 * O_{ae} + O_{ad}$
Z	$2 * O_{ss}.$	-
R	$O_{ae} + 2 * O_{ad} + O_{sv}$	$O_{ae} + 2 * O_{ad}$
Total	$3 * O_{ae} + 3 * O_{ad} + 2 * O_{ss} + 2 * O_{sv}$	$3 * O_{ae} + 3 * O_{ad}$
GP2A		
I	$4 * O_{se} + 2 * O_{sd} + O_{ms} + O_{mv} + 2 * O_{kd} + O_{se}^{*} + O_{ms}^{*}$	
R	$2 * O_{se} + 4 * O_{sd} + O_{ms} + O_{mv} + 2 * O_{kd} + O_{sd}^{*} + O_{mv}^{*}$	
Total	$6 * O_{se} + 6 * O_{sd} + 2 * O_{ms} + 2 * O_{mv} + 4 * O_{kd} + O_{se}^{*} + O_{sd}^{*} + O_{ms}^{*} + O_{mv}^{*}$	
GE2A		
I	$4 * O_{se} + 2 * O_{sd} + O_{ms} + O_{mv} + 2 * O_{kd} + O_{se}^{*} + O_{ms}^{*}$	
R	$2 * O_{se} + 5 * O_{sd} + O_{ms} + O_{mv} + 2 * O_{kd} + O_{sd}^{*} + O_{mv}^{*}$	
Total	$6 * O_{se} + 7 * O_{sd} + 2 * O_{ms} + 2 * O_{mv} + 4 * O_{kd} + O_{se}^{*} + O_{sd}^{*} + O_{ms}^{*} + O_{mv}^{*}$	
SOA		
I	$2 * O_{se} + O_{sd} + O_{ms} + O_{mv} + 2 * O_{kd} + O_{se}^{*} + O_{ms}^{*}$	
R	$O_{se} + 2 * O_{sd} + O_{ms} + O_{mv} + 2 * O_{kd} + O_{sd}^{*} + O_{mv}^{*}$	
Total	$3 * O_{se} + 3 * O_{sd} + 2 * O_{ms} + 2 * O_{mv} + 4 * O_{kd} + O_{se}^{*} + O_{sd}^{*} + O_{ms}^{*} + O_{mv}^{*}$	

5.8.3 Communication Overheads

The communication overheads are evaluated in terms of the number and sizes of messages exchanged among entities involved in an authentication instance. The total size of a message equals the sum of the size of the header and the size of the payload. For all messages, the size of the headers is fixed (L_{hd}). The size of the payload is dependent on the number and sizes of data items (e.g., authenticators, tickets, and tags) contained in the payload. For comparison, it is assumed that the messages used in Kerberos and NSLPK have the same header as shown in Figure 5.7 (thus, the header size is L_{hd}) and the tickets used in Kerberos share the same ticket structure as shown in Figure 5.4 (thus, the ticket size is L_{tkt}).

5.8.3.1 GP2A Protocol

There are three messages exchanged between I and R. In **Step 1**, I sends msg-GP2A1 to R. The message contains one JID (L_{jid}), one REQ (L_{req}), one ICL L_{icl}, one authenticator[1] (containing one nonce) (L_n), and one tag (L_τ). Hence, the size of the message is $L_{hd} + L_{jid} + L_{req} + L_{icl} + L_n + L_\tau$. In **Step 2**, R sends msg-GP2A2 to I. The message contains one MID (L_{mid}), one authenticator (containing two nonces) ($2 * L_n$), and one tag (L_τ). Hence, the size of the message is $L_{hd} + L_{mid} + 2 * L_n + L_\tau$. In **Step 3**, I sends msg-GP2A3 to R. The message contains one MID (L_{mid}), one authenticator (containing one nonce) (L_n), one credential package, and one tag (L_τ). The size of the credential package is interaction dependent. For ease of discussion and without losing generality, here the size of the credential package is denoted as L_{pkg}. Hence, the size of the message is $L_{hd} + L_{mid} + L_n + L_{pkg} + L_\tau$.

5.8.3.2 GE2A Protocol

Similar to GP2A, there are also three messages exchanged between I and R. The only difference is that, in **Step 1**, the first message (msg-GE2A1) sent by I to R contains one additional item, i.e., a ticket. Hence, the size of the message is $L_{hd} + L_{jid} + L_{req} + L_{icl} + L_n + L_{tkt} + L_\tau$.

5.8.3.3 SOA Protocol

With SOA, the number and the sizes of messages exchanged between I and R are the same as those of GP2A[2].

5.8.3.4 Kerberos Protocol

Without $k_{I,T}$ and $tkt_{I,T}^{K}$ caching, there are a total of five messages exchanged among entities: two messages exchanged between I and K; two messages exchanged between I and T; and one message sent from I to R. For messages exchanged between I and K, in **Step 1**, I sends msg-K1 to K. The message contains two EIDs ($2 * L_{eid}$) and one nonce (L_n). Hence, the size of the message is $L_{hd} + 2 * L_{eid} + L_n$. In **Step 2**, K sends msg-K2 to I. The message contains one encrypted concatenation of a pairwise key and a nonce ($L_k + L_n$) and one ticket (L_{tkt}). Hence, the size of the message is $L_{hd} + L_k + L_n + L_{tkt}$. For messages exchanged between I

[1] The size of an encrypted data token, such as an authenticator, is dependent on the size of the plaintext (input data) to be encrypted as well as the encryption scheme used. Some block cipher-based encryption schemes add padding to the input data and produce an encrypted data token whose size is multiple of block sizes (specific to a particular scheme) or key sizes.

[2] Depending on an encryption scheme and a padding scheme used, nested encryption may produce larger encrypted tokens. In other words, SOA may produce smaller messages than those of GP2A.

and T, in **Step 3**, I sends msg-K3 to T. The message contains one authenticator (containing one timestamp) (L_t), one ticket (L_{tkt}), one EID (L_{eid}), and one nonce (L_n). Hence, the size of the message is $L_{hd} + L_t + L_{tkt} + L_{eid} + L_n$. In **Step 4**, T sends msg-K4 to I. The message contains one encrypted concatenation of a pairwise key and a nonce ($L_k + L_n$) and one ticket (L_{tkt}). Hence, the size of the message is $L_{hd} + L_k + L_n + L_{tkt}$. For the message (msg-K5) sent from I to R, the message contains one authenticator (containing one timestamp) (L_t) and one ticket (L_{tkt}). Hence, the size of the message is $L_{hd} + L_t + L_{tkt}$.

With $k_{I,T}$ and $tkt_{I,T}^{K}$ caching, there are three messages (msg-K3, msg-K4, and msg-K5) exchanged among I, T, and R. The sizes of these messages are the same as analysed above.

5.8.3.5 NSLPK Protocol

Without pk_I and pk_R caching, there are a total of seven messages exchanged among entities: two messages exchanged between I and Z; two messages exchanged between Z and R; and three messages between I and R. For the messages exchanged between I and Z, in **Step 1**, I sends msg-N1 to Z. The message contains two EIDs ($2 * L_{eid}$). Hence, the size of the message is $L_{hd} + 2 * L_{eid}$. In **Step 2**, Z sends msg-N2 to I. The message contains a concatenation of a public key and an EID ($L_{pk} + L_{eid}$) and a corresponding signature (L_{σ}). Hence, the size of the message is $L_{hd} + L_{pk} + L_{eid} + L_{\sigma}$. The messages (msg-N4 and msg-N5) exchanged between Z and R are similar to those (msg-N1 and msg-N2) exchanged between I and Z. Hence, the sizes of the messages are, respectively, $L_{hd} + 2 * L_{eid}$ and $L_{hd} + L_{pk} + L_{eid} + L_{\sigma}$. For the messages exchanged between I and R, in **Step 3**, I sends msg-N3 to R. The message contains one encrypted concatenation of a nonce and an EID ($L_n + L_{eid}$). Hence, the size of the message is $L_{hd} + L_n + L_{eid}$. In **Step 6**, R sends msg-N6 to I. The message contains one encrypted concatenation of two nonces and an EID ($2 * L_n + L_{eid}$). Hence, the size of the message is $L_{hd} + 2 * L_n + L_{eid}$. In **Step 7**, I sends msg-N7 to R. The message contains one encrypted nonce (L_n). Hence, the size of the message is $L_{hd} + L_n$.

With pk_I and pk_R caching, there are only three messages (msg-N3, msg-N6, and msg-N7) exchanged between I and R. The sizes of these messages are the same as analysed above.

5.8.3.6 The Comparisons of Communication Overheads

The communication overheads when different entity authentication protocols are applied are shown in Table 5.14. The result shows that GP2A, GE2A, and SOA introduce the same number and sizes of messages with the exception of msg-GE2A1 which has one additional L_{tkt}. In comparison with Kerberos without $k_{I,T}$ and $tkt_{I,T}^{K}$ caching and NSLPK without pk_I and pk_R caching, the MIEA protocols introduce fewer number of messages, reducing message transmission overhead (e.g., network-level packet headers). When key caching is applied, the MIEA protocols introduce the same number of messages, i.e., 3 messages, as those of Kerberos and NSLPK. However, each of the messages used in each of the MIEA protocols contains more data items (i.e., tags and credentials for subsequent authentication) than those in Kerberos or NSLPK.

Table 5.14: The comparisons of the communication overheads introduced by different entity authentication protocols.

Kerberos		
	Without $k_{I,T}$ and $tkt_{I,T}^{K}$ caching	With $k_{I,T}$ and $tkt_{I,T}^{K}$ caching
Between I and K	2 messages: msg-K1: $L_{hd} + 2 * L_{eid} + L_n$ msg-K2: $L_{hd} + L_k + L_n + L_{tkt}$	-
Between I and T	2 messages: msg-K3: $L_{hd} + L_t + L_{tkt} + L_{eid} + L_n$ msg-K4: $L_{hd} + L_k + L_n + L_{tkt}$	2 messages: msg-K3: $L_{hd} + L_t + L_{tkt} + L_{eid} + L_n$ msg-K4: $L_{hd} + L_k + L_n + L_{tkt}$
Between I and R	1 message: msg-K5: $L_{hd} + L_t + L_{tkt}$	1 message: msg-K5: $L_{hd} + L_t + L_{tkt}$
Total	5 messages	3 messages
NSLPK		
	Without pk_I and pk_R caching	With pk_I and pk_R caching
Between I and Z	2 messages: msg-N1: $L_{hd} + 2 * L_{eid}$ msg-N2: $L_{hd} + L_{pk} + L_{eid} + L_{\sigma}$	-
Between Z and R	2 messages: msg-N4: $L_{hd} + 2 * L_{eid}$ msg-N5: $L_{hd} + L_{pk} + L_{eid} + L_{\sigma}$	-
Between I and R	3 messages: msg-N3: $L_{hd} + L_n + L_{eid}$ msg-N6: $L_{hd} + 2 * L_n + L_{eid}$ msg-N7: $L_{hd} + L_n$	3 messages: msg-N3: $L_{hd} + L_n + L_{eid}$ msg-N6: $L_{hd} + 2 * L_n + L_{eid}$ msg-N7: $L_{hd} + L_n$
Total	7 messages	3 messages
GP2A		
Between I and R	3 messages: msg-GP2A1: $L_{hd} + L_{jid} + L_{req} + L_{icl} + L_n + L_{\tau}$ msg-GP2A2: $L_{hd} + L_{mid} + 2 * L_n + L_{\tau}$ msg-GP2A3: $L_{hd} + L_{mid} + L_n + L_{pkg} + L_{\tau}$	
Total	3 messages	
GE2A		
Between I and R	3 messages: msg-GE2A1: $L_{hd} + L_{jid} + L_{req} + L_{icl} + L_n + L_{tkt} + L_{\tau}$ msg-GE2A2: $L_{hd} + L_{mid} + 2 * L_n + L_{\tau}$ msg-GE2A3: $L_{hd} + L_{mid} + L_n + L_{pkg} + L_{\tau}$	
Total	3 messages	
SOA		
Between I and R	3 messages: msg-SOA1: $L_{hd} + L_{jid} + L_{req} + L_{icl} + L_n + L_{\tau}$ msg-SOA2: $L_{hd} + L_{mid} + 2 * L_n + L_{\tau}$ msg-SOA3: $L_{hd} + L_{mid} + L_n + L_{pkg} + L_{\tau}$	
Total	3 messages	

5.9 Experimental Evaluation

To evaluate performance of each of the three MIEA protocols (GP2A, GE2A, and SOA) when deployed on a real-world system, we have implemented the protocols and conducted experiments to measure protocol execution times under different sets of parameter values. For benchmarking, the results are compared with those of the Kerberos protocol and the NSLPK protocol. In this section, we first explain methodology and evaluation metrics, then describe testbed setup and parameters used, before reporting our experimental results.

5.9.1 Methodology and Evaluation Metrics

The performances of the MIEA protocols are dependent on computational (operational costs generating and verifying AuthData) and communication (volume of traffics transmitted over networks for the exchange of protocol messages) overheads introduced by the protocols. To evaluate such overheads, each of the MIEA protocols is implemented and executed on a testbed. The evaluation consists of two experiments, Exp1 and Exp2. Exp1 evaluates the costs of cryptographic algorithms used in the MIEA protocols, i.e., Sym-Encryption, Sym-Decryption, MAC-Signing, MAC-Verification, and key derivation. For comparison, it also evaluates the costs of Asym-Encryption, Asym-Decryption, SIG-Signing, and SIG-Verification which are used in NSLPK. Exp2 evaluates the performance of each of the MIEA protocols, Kerberos, and NSLPK when executed on the testbed.

The costs of the cryptographic algorithms and the performance of the protocols are, respectively, measured in terms of the execution times of the algorithms and the protocols. Multiple samples of execution times are collected for each particular set of parameter values. Statistical values are calculated from the collected samples. These values are mean values for showing the costs and standard error of the mean for estimating measurement errors (i.e., showing how dispersed sample means are in relation to the population mean).

5.9.2 Testbed Setup

The testbed consists of five entity authentication services, respectively, implementing GP2A, GE2A, SOA, Kerberos, and NSLPK. These services are deployed on a single machine. Only one machine is used due to accessibility to equipment. The software and hardware used are described in detail in the following.

5.9.2.1 Software

The architecture of our testbed is shown in Figure 5.16. In this figure, an executable file (red rectangle) implementing all the protocols (GP2A, GE2A, SOA, Kerberos, and NSLPK), called ProtocolServices, is hosted on a machine (green rectangle). Each dotted rectangle is an application process implementing an entity instance (e.g., an initiator or a respondent). The number of the processes and their tasks are dependent on the protocol used. For example, three processes (Initiator, Respondent, and Key server) are executed when NSLPK is applied, whereas two processes (Initiator and Respondent) are executed when each of GP2A, GE2A, and SOA is applied. The initiation of the processes (i.e., entity instances) is shown as dotted unidirectional arrowed lines. The communication among the processes is shown as solid bidirectional arrowed lines. It is implemented by using a TCP connection.

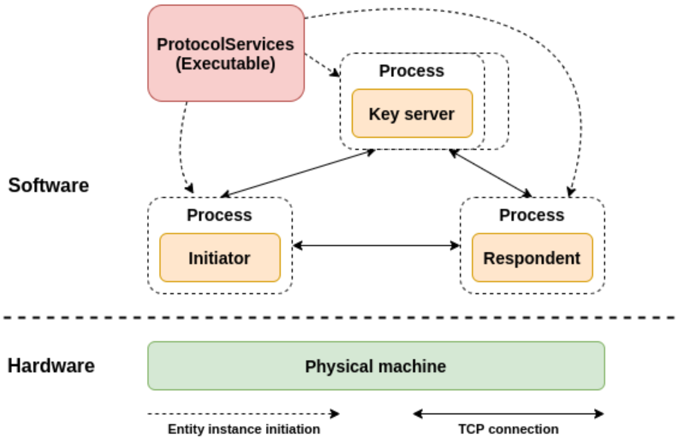


Figure 5.16: Testbed architecture for evaluating the entity authentication services.

ProtocolServices is written in C++. The cryptographic functions used are provided by the Botan cryptographic library [180]. Botan is selected as it has been used in a wide range of projects and supported by many organisations (including the German government, opensource communities, and commercial enterprises) [181]. To implement the required cryptographic algorithms, we have chosen the following schemes: (1) AES with the CBC mode, PKCS#7 padding, and 128-bit keys (referred to as AES-128) for the symmetric-key based encryption scheme; (2) RSA with SHA-256, OAEP padding, and 3072-bit keys (referred to as RSAEnc-3072) for the asymmetric-key based encryption scheme; (3) RSA with SHA-256, PSS padding, and 3072-bit keys (referred to as RSASig-3072) for the digital signature scheme; (4) HMAC with SHA-256 and 128-bit keys (referred to as HMAC-128) for the MAC scheme; and (5) HKDF with HMAC and SHA-256 (outputting 128-bit keys) for the key derivation scheme. The sizes of keys, tags, signatures, and nonces are set to achieve a sufficient level of security protection; at the time of this writing, NIST [182] has recommended a security level of 128 bits.

The specifications of the underlying operating system, the C/C++ compiler, and the cryptographic library used are given in Table 5.15.

Table 5.15: Software specifications.

Component	Specification
Operating system	Linux Manjaro 20.1 Mikah Kernel: 4.14.193-1-MANJARO x86_64
C/C++ compiler	gcc 10.1.0
Cryptographic library	botan 2.15.0

5.9.2.2 Hardware

The testbed consists of one machine. The machine hosts the ProtocolServices executable file. All the inter-process communications are TCP connections over the loopback (with the IP address 127.0.0.1) interface of the machine. The specifications of the machine are summarised in Table 5.16.

Table 5.16: Hardware specifications.

Component	Specification
CPU	Quad Core Intel Core i7-6700, 64-bit, max 4.0 GHz
RAM	DDR4, 2133 MT/s, 16 GB
Storage	HDD 1 TB

5.9.3 Parameters and Configurations

In both Exp1 and Exp2, the sizes of data objects are expressed in bytes (B). In Exp1, we measure the execution times of all the cryptographic algorithms performed on data objects with the sizes of 16 B (the size of one nonce) and 32 B (the total size of two nonces)[3]. In addition, we also measure the execution times of symmetric-key based encryption algorithms (AES encryption and decryption) performed on objects of different sizes. The sizes range from 16 B to 16,384 B[4] with an increment of twofold. The objects used are randomly generated binary data[5]. The sample size for each measurement is 4,000.

In Exp2, we measure the execution times of all the protocols (GP2A, GE2A, SOA, Kerberos, and NSLPK) when credentials for subsequent authentication are not transmitted. As the MIEA protocols are also used to transmit credentials for subsequent authentication, we also measure the execution times of each of the MIEA protocols against the size of credential packages. The package size ranges from 16 B to 16,384 B (for justification, please see footnote 5). As the content of the package should not affect the evaluations, the data used are randomly generated binary data. Each measurement for a specific set of parameter values is collected from 1,000 samples.

The accuracy of the measurements of the execution times is statistically evaluated by using standard error of the mean. By choosing the sample sizes of 4,000 for Exp1 and 1,000 for Exp2, the uncertainties of the mean execution times in terms of relative standard error of the mean (standard error of the mean divided by the mean execution times) are lower than 1%. Although using a larger sample size should result in more accurate results, a slight increase in the sample sizes would greatly increase the time needed for conducting the experiments. This does not justify a marginal gain of accuracy.

5.9.4 Experimental Results

In this section, we report the experimental results and discuss our findings.

5.9.4.1 Exp1: Costs of Cryptographic Algorithms

The execution times of all the cryptographic algorithms on 16-B and 32-B data objects, and AES-128 Encryption and Decryption on objects with varying sizes, are depicted in Figure 5.17 and Figure 5.18, respectively.

[3] In actual protocol executions, most operations in a symmetric-key based protocol are performed on small objects and the sizes of these objects are mainly dependent on the sizes of nonces (here, the size of a nonce is 16 B). In contrast, when NSLPK, an asymmetric-key based protocol, is applied, operations are performed on much larger objects (e.g., signature signing on an asymmetric key with the size of 384 B). For comparison, we only measure the execution times of the operations performed on objects with the sizes of 16 B and 32 B.

[4] In each of the MIEA protocols, the AES algorithms are performed on objects with different sizes, i.e., packages of credentials and associated metadata. The largest package is transmitted when $ResourceManager^1$ sends credentials to $JobManager$ (@16) and when $JobManager$ sends credentials to each $Reducer_b$ (@24); these credentials are for authentication to each $WorkerManager_u^j$. For an MR service with 1,000 Workers, the number of WorkerNodes (thus WorkerManagers) is approximately 100. The size of the package is approximately the number of WorkerManagers times the sum of the sizes of metadata, a primary key, and a ticket = 100 * (20 + 16 + 48) B = 8,400 B.

[5] The cryptographic algorithms used in the experiments operate on binary data.

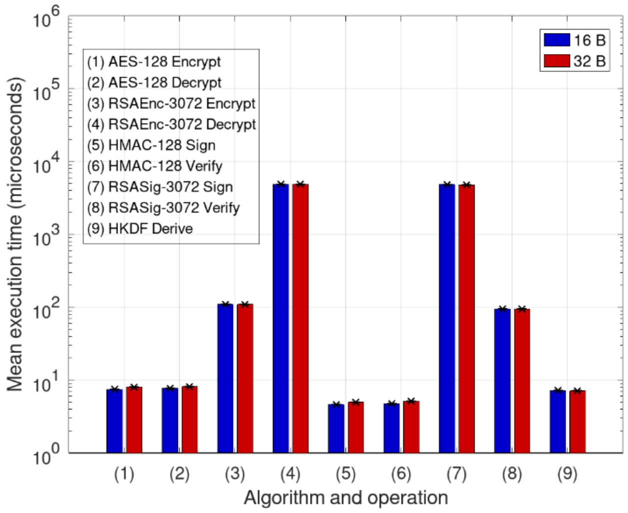


Figure 5.17: The comparisons of the execution times of all the cryptographic algorithms on data objects with the sizes of 16 B and 32 B.

From Figure 5.17, we can see that, the execution times of the symmetric-key based algorithms are much lower than those of the asymmetric-key based algorithms. Among the symmetric-key based algorithms, HMAC-128 cost the smallest (in terms of the execution times), and AES-128 and HKDF introduce the same level of costs. For example, when the object size is 16 B, the values of HMAC-128 Sign and Verify are approximately 5 microseconds, the values of AES-128 Encrypt and Decrypt and HKDF Derive are 7 microseconds, the values of RSAEnc-3072 Encrypt and RSASig-3072 Verify are 100 microseconds, and the values of RSAEnc-3072 Decrypt and RSASig-3072 Sign are 4,800 microseconds. The values of RSAEnc-3072 Encrypt and RSASig-3072 Verify are one magnitude higher than those of the symmetric-key based algorithms, and RSAEnc-3072 Decrypt and RSASig-3072 are two magnitudes higher. Such large differences are because of the complexity of computation in asymmetric-key based algorithms. In addition, in RSA based algorithms, there are a large difference in computational costs for operations with different keys; operations with private keys are more computationally expensive than those with public keys.

When AES-128 Encrypt and Decrypt and HMAC-128 Sign and Verify are applied on objects with the sizes of 16 B and 32 B, respectively, there are slight increases in the execution times. However, when RSA based algorithms are applied, the values are at the same level. For example, AES-128 Encrypt takes 7.5 microseconds and 8 microseconds to execute when performed on 16 B and 32 B, respectively, resulting in an increase of 0.5 microseconds (6.7%). RSASig-3072 Verify takes 94 microseconds to execute on both object sizes. This is because, when the AES-128 based algorithms are applied, 16-B and 32-B are, respectively, padded to 32 B and 48 B, resulting in difference execution times. However, when the RSA based algorithms are applied, both 16-B and 32-B data are padded to 384 B, thus, yielding the same execution times.

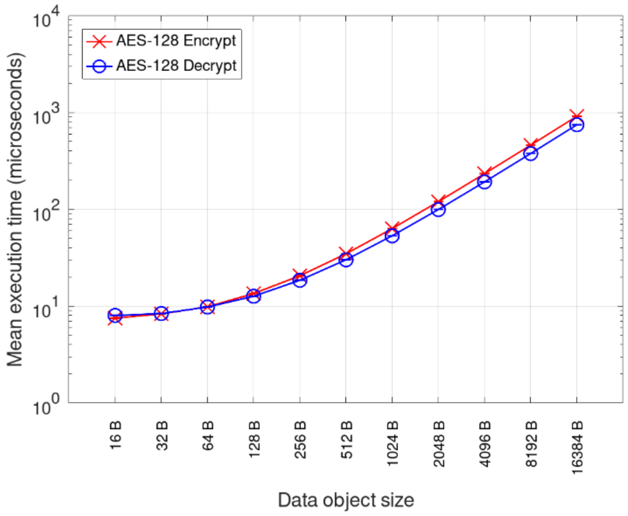


Figure 5.18: The comparisons of the execution times of AES-128 Encryption and Decryption on objects with varying sizes.

From Figure 5.18, we can make the following observations. The execution times of AES-128 Encrypt and Decrypt increase almost linearly as the size of objects increases. In addition, the rate of increase in the execution time of AES-128 Encrypt is higher than that of AES-128 Decrypt. For example, when the object size increases from 16 B to 16,364 B (an order of 10^3 increase), the execution times of AES-128 Encrypt and Decrypt increase from 7.5 microseconds and 8.0 microseconds to 910 microseconds and 750 microseconds, respectively. These approximately equal the increases of 120 times and 94 times, respectively. The reason is that the larger size of the objects increases the workload of the algorithms thus the execution times. This is consistent with the experimental results of the costs of cryptographic algorithms used in data authentication reported in [44].

These results indicate that. Asymmetric-key based algorithms are much more expensive than symmetric-key based algorithms. In other words, using the symmetric-key based algorithms for entity authentication is more efficient and can considerably lower the overhead cost introduced. For example, when the object size is 16 B, AES-128 Decrypt costs approximately 0.16% of RSAEnc-3072 Decrypt cost. When applying the same algorithm on objects of small sizes (16 B and 32 B), the difference in execution times is small. In addition, for AES-128 based algorithms, when the object size increases, the execution time of the algorithm also increases.

5.9.4.2 Exp2: Costs of Entity Authentication Protocols

The execution times of all the five protocols, and the three MIEA protocols (GP2A, GE2A, and SOA) against the size of credential packages (credentials established for subsequent authentication), are depicted in Figure 5.19 and Figure 5.20, respectively.

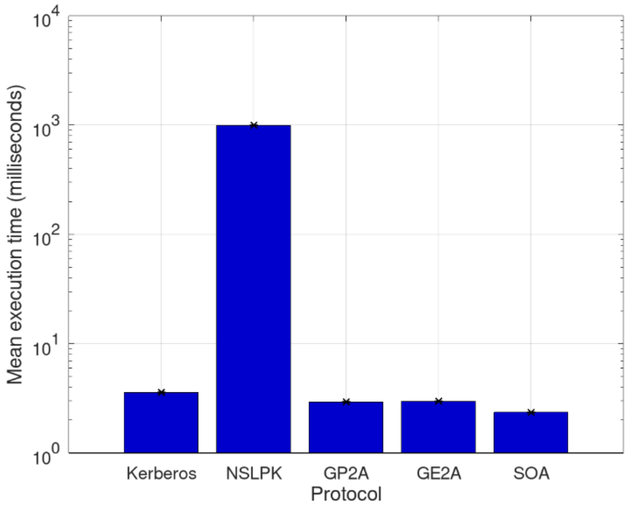


Figure 5.19: The comparisons of the execution times of all the protocols.

From Figure 5.19, it can be seen that NSLPK costs the highest and the MIEA protocols cost the lowest. In addition, among the MIEA protocols, SOA takes the shortest time to execute. The execution times of NSLPK (the highest) and SOA (the lowest) are, respectively, 1,000 milliseconds and 2.4 milliseconds, having a difference of 420 times. There are two reasons for such a large difference. The first is that NSLPK transmits the highest number of messages (i.e., 7 messages), whereas each of the MIEA protocols transmits only 3 messages; the more the messages, the higher communication overhead thus the cost. The second is that NSLPK performs a total of 10 expensive asymmetric-key operations, whereas SOA performs a total of 18 inexpensive symmetric-key operations. Although the number of operations performed in SOA is larger than that of NSLPK, the cost of the symmetric-key operations is much lower, as shown in the results of Exp1.

When comparing SOA with Kerberos, both of which are one-factor and symmetric-key based protocols, SOA introduces lower cost than Kerberos. This is because, in comparison with Kerberos, SOA markedly cut the communication cost from 5 messages to 3 messages (i.e., the reduction of 40%). In our testbed where a high-performance machine is used, the cost of a symmetric-key operation is usually a few microseconds, whereas the cost of a message transmission could be 50 microseconds[6] or higher. Hence, reducing the number of messages transmitted has larger impact on cost reduction.

[6] The time of a message transmission is measured by using 'ping', a tool commonly used to check reachability of machines in networks.

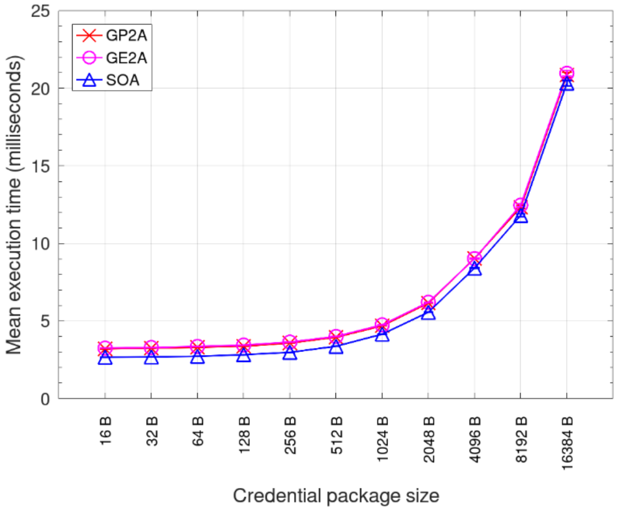


Figure 5.20: The comparisons of the execution times of GP2A, GE2A, and SOA against the size of credential packages.

From Figure 5.20, we can make the following observations. The execution times of the MIEA protocols increase as the credential package size increases. When the size of credential packages increases from 16 B to 512 B, the increase in the execution times is small. For example, when GP2A is applied, the execution times increase from 3.2 milliseconds to 4.0 milliseconds, which is an increase of 25%. However, when the size goes beyond 1,024 B, the execution times sharply increase, particularly when the credential package size is 8,192 B. For example, GP2A takes about 12 milliseconds and 21 milliseconds to execute when the credential package size increases from 8,192 B to 16,384 B, respectively; the execution time is almost doubled. This is because the increase in the credential package size increases the workload of cryptographic algorithms (thus computational overhead) and the volume of data to be transmitted via networks (thus communication overhead). When the size of credential packages is smaller than 512 B, the computational cost (i.e., tenths of milliseconds) has small impact on the execution time of the protocols, as can be seen in the figure. However, when the size goes beyond 8,192 B, the impact of the increased computational and communication becomes more apparent; the execution times of the protocol increase proportionally to the size of the credential packages.

Among the MIEA protocols, GE2A costs the highest whereas SOA costs the lowest. For example, when the size of credential packages is 16 B, GE2A and SOA takes about 3.3 milliseconds and 2.7 milliseconds to execute, respectively. The difference is larger when the credential package size is smaller. The largest difference occurs when the size is 64 B, which is about 24%. This is because GP2A and GE2A use a nested encryption approach to generate and verify challenges and responses, i.e., the generation and the verification of an authenticator each require two operations; whereas only one operation is required when SOA is applied. In addition, when GE2A is applied, a pairwise key shared between an initiator I and a respondent R is distributed via a ticket. R has to verify and decrypt the ticket to obtain the key before using the key for authentication, increasing the execution time of the protocol.

The above experimental results indicate that GP2A, GE2A, and SOA can achieve the same level of protection (in terms of efforts needed to break an authentication token) as that provided by NSLPK but at two-magnitude lower cost. In comparison with Kerberos, the MIEA protocols can provide stronger protections (with an exception that SOA uses only one factor for authentication) while introducing the same level of overhead cost.

5.10 Chapter Summary

This chapter has presented a novel approach, a multi-factor interaction based approach, to entity authentication for MR based CBDC-MPC and a novel entity authentication framework, the MIEA framework, that implements the approach. By conducting a critical analysis on the related work, we discover that none of the existing solutions can satisfy all the specified requirements as these solutions are designed for applications in different contexts. Most of the solutions are designed to protect against external threats, thus, the authentication is applied only at the gate level (i.e., before an entity is allowed to access the service); it is not required at the interaction level (during the course of a job execution). Some solutions are not efficient for this context as they make use of computationally expensive cryptographic operations at the object level and use many protocol messages to accomplish authentication. MIEA is designed to address the knowledge gap. In the design of MIEA, three main ideas have been used. The first idea is MIA in which critical interactions (the interactions that are used to establish credentials for subsequent authentication) are protected with two-factor authentication and non-critical interactions are protected with one-factor. This doubles the effort needed to break an authentication token used to protect the critical interaction. This idea allows us to achieve a stronger level of protection compared with the related work (e.g., Kerberos) that use only one factor. The second idea is DCS in which symmetric keys are distributed by distributed trustworthy components and the authentication between two components is carried out without a centralised authentication entity. Combined with the third idea called HKS, which constructs a key hierarchy and the keys in the higher level of the hierarchy are used to distribute the keys in the lower level of the hierarchy, the number of protocol messages needed to accomplish authentication is reduced to three. Compared with Kerberos and NSLPK without credential caching which require five and seven protocol messages, respectively, MIEA can significantly reduce the communication overhead cost in terms of the times needed for exchanging the protocol messages. The results of the performance evaluation show that the performance of MIEA is at the same level as that of Kerberos and is much lower than that of NSLPK (a difference of 420 times). This means that MIEA is as efficient as Kerberos, one of the most used entity authentication protocols. MIEA can provide protection to every interaction without using a centralised authentication server. The applicability of MIEA is not limited to MR based services, it should also be applicable to other distributed computing services (e.g., Apache Spark) that exhibit similar characteristics, e.g., multi-stage data processing. The approach to entity authentication and the design and evaluations of MIEA presented in this chapter is the second contribution (NC2) of this research work. The contribution answers the research questions (Q2) and (Q4).

The next chapter presents in detail a novel approach, a communication pattern based approach, to data authentication which provides the strongest protection to data authenticity and non-repudiation at the finest granularity throughout the whole cycle of a job execution while minimising overhead costs imposed on components and the underlying system.

Chapter 6
Communication Pattern based Data Authentication (CPDA) Framework

6.1 Chapter Introduction

This chapter presents a novel data authentication framework, called the Communication Pattern based Data Authentication (CPDA) framework, which is also part of the MDA framework. The CPDA framework aims to provide the strongest level of JobData authenticity protection (i.e., assuring data origin and integrity authentication, as well as non-repudiation of origin) at the finest granularity (at the object level), but with as less overhead cost as possible. The design of CPDA has exploited two main ideas. The first is AuthData and Communication Aggregation (ACA) in which the operations of AuthData generation and verification as well as communications transmitting the AuthData are aggregated. The aggregation methods are selected based on the communication patterns exhibited. The second idea is a Hybrid use of multiple cryptographic schemes with Segregation of Credentials (HYSC). Computationally less expensive mechanisms (i.e., hash functions and MACs) are used to protect individual objects that are transferred between untrustworthy and trustworthy entities whereas computationally more expensive mechanisms (digital signatures) are used to secure aggregated AuthData, thus, extending the protection to all data objects. Each of untrustworthy entities is assigned a different pairwise key for securing objects it produces, hence, accountability can be pinpointed to entities sharing the key. To demonstrate the effectiveness, the efficiency, and the scalability of the CPDA framework, the CPDA framework has been extensively evaluated both theoretically and experimentally. Theoretically analyses have been conducted by using both informal and formal methods. Experimental evaluations are carried out on a testbed consisting of five networked machines with a real-world dataset.

In detail, Section 6.2 critically analyses related data authentication solutions against the requirements specified in Section 4.5 and discusses what is missing. Sections 6.3, 6.4, and 6.5, respectively, give high-level ideas, notations and design assumptions used, and low-level description of CPDA. Sections 6.7, 6.8, and 6.9, respectively, present security analysis, theoretical, and experimental performance evaluations of CPDA and the most related solutions. Finally, Section 6.10 concludes the chapter.

6.2 Existing Data Authentication Solutions

Based on targeted systems, related data authentication solutions can also be largely classified into two groups: non-MR specific solutions and MR specific solutions.

6.2.1 Non-MR Specific Solutions

Depending on the cryptographic schemes used, non-MR specific solutions can be further classified into three groups: secret-share based, symmetric-key based, and asymmetric-key based.

In a secret-share based solution, AuthData are generated with a secret but verified with a secret-share derived from the secret. A single secret is divided into N secret-shares. Any k or more (out of N) secret-shares can be used to reconstruct the secret, but $k-1$ or fewer secret-shares cannot. Desmedt et al. [183] proposed such a scheme for multicast services where one producer sends a data object to multiple consumers. In this scheme, for each object, the producer uses two polynomials of degree $k-1$ (known only to the producer) to generate AuthData and N secret-shares. Each secret-share is distributed to a different consumer so that each consumer can independently verify the AuthData. Safavi-Naini and Wang [184][185] improved on the Desmedt's scheme by reducing the number of polynomials required to authenticate multiple objects. To authenticate m objects, the producer uses only $m+1$ polynomials, as opposed to $2*m$ polynomials as required by the Desmedt's scheme. This cuts the costs in generating and storing AuthData by half. Nonetheless, the secret-share based solutions incur a high level of computational overhead due to the cost of computing polynomials of degree $k-1$, especially when k is large.

Symmetric-key based data authentication solutions, such as MACs, are designed to counter external attacks. They do not provide non-repudiation protections, making them vulnerable to threats, e.g., tag forgeries, imposed by authorised insiders. To address this issue, the idea of asymmetry is used. There are two forms of asymmetry: information asymmetry and time asymmetry. With an information-asymmetry based scheme [186], a producer has a full view of a secret (a set of secret keys) whereas each consumer has only a partial view of the secret (a subset of the secret keys). The entire set of the secret keys is used to generate AuthData (tags), whereas a subset of the secret keys is used to verify the tags. A subset of the secret keys is made available for each consumer and these subsets are different from each other. Tag forgeries are countered by limiting the number of secret keys revealed to each consumer. This approach incurs a high level of computational as well as storage overheads, as multiple tags are processed (generated and verified) and multiple secret keys are required for the processing of such tags.

With a time-asymmetry based scheme, tag forgeries are countered by controlling when a secret key is used for generating tags and when the key is being made available for verifying the tags. In other words, the secret key is used to generate tags in one time period, and it is released for the verification of the tags in another time period. Examples of such schemes include Chained Stream Authentication (CSA) [187], Timed Efficient Stream Loss-tolerant Authentication (TESLA) [39][188][189], and μTESLA [190]. Although delaying the release of the keys does not introduce additional computational overhead, it increases the data processing time and offsets the benefit of parallel computations provided by distributed computing frameworks such as MR.

To ensure data authenticity and provide non-repudiation of origin, digital signatures are frequently used. With a digital signature based solution, two asymmetric keys (a private key and a public key) are used, respectively, for generating and verifying AuthData (signatures). As long as the public key is certified and the private key is kept secret, it is computationally infeasible for another entity, rather than the owner of the private key, to forge signatures. However, signature operations (generation and verification) are computationally expensive, much more expensive than MAC operations [45]. In addition, for the same security level, the lengths of asymmetric keys and signatures are usually much longer than those of symmetric keys and tags, respectively [191][192][193]. Therefore, using digital signatures to secure individual objects in Big Data applications is neither efficient nor scalable.

A number of data authentication schemes have been proposed with an intention to reduce the number of signatures used. These schemes employ a signature amortisation technique. Such a technique builds a chain of AuthData in a way that the AuthData of one object are linked to those of other objects. In this way, only a subset of the objects is signed but the protection is provided to the whole set of the objects. Related work in this category has been focusing on how to construct such AuthData chains so that the dependency among the objects and the amount of AuthData embedded in the objects can be reduced. In the method proposed by Gennaro and Rohatgi [194], a chain of AuthData is constructed by embedding the AuthData of one object in the preceding object and the first object is signed with a digital signature scheme. This method is not designed for applications where data are sent over unreliable networks as the loss of one object would make the succeeding objects unverifiable. A number of schemes have been proposed to address this limitation, and these are Efficient Multi-chained Stream Signature (EMSS) [39], p-Random Authentication [40], the piggybacking scheme [40], Golle and Modadugu's scheme [41], and Adaptive source Authentication protocol for multiCAST streams (A2Cast) [42]. The essence of these schemes is to embed the AuthData of one object in a number of other objects. In this way, the remaining objects will still be verifiable even if some of the objects are lost. Nonetheless, this is achieved by using redundancy and at the cost of increased communication and storage overheads.

To reduce the redundancy thus the overheads, a number of schemes employing error correction codes are proposed. These schemes are Signature Amortization using IDA (SAIDA) [195][196] and Data Authentication Model based on Reed-Solomon Error-correcting Code (DAM-RSEC) [197]. In these schemes, the hashes of the whole set of objects and the signature of the aggregated hashes are encoded with an error correcting code and the resulting code is split and embedded in the objects. In this way, the AuthData can be reconstructed from a subset of the objects and the amount of AuthData carried by each object is reduced.

All the above schemes are designed for multicast and broadcast services where there is only one data producer but multiple consumers. A major limitation of these schemes when being applied to the CBDC-MPC context is that they do not allow each of the objects to be independently verifiable. An exception is the scheme proposed by Wong and Lam [88]. This scheme allows the verification of individual objects while reducing the number of objects to be signed and verified. This is done by constructing a hash tree of the objects and signing only the hash of the root node of the tree. The verification of a particular object is done by using the hash of the object, the hashes along the path leading to the root node, and the signature.

6.2.2 MR Specific Solutions

Data authentication solutions specifically designed for MR applications can be largely classified into two groups, task-replication based and non-task-replication based.

As indicated by the name, task-replication based solutions use task replication to ensure the correctness of JobData that are generated during a job execution, thus providing data integrity protection. With a task-replication based solution, each data processing task (a map or reduce task) is assigned to multiple Workers and the outputs produced by these Workers are compared to detect inconsistencies. This approach has been used by a number of schemes published in literature, and these are Verification-based Integrity Assurance Framework for MR (VIAF) [46], Cross Cloud MapReduce (CCMR) [47], IntegrityMR [48], Verification-based Anti-collusive Worker Scheduling (VAWS) [49], and HAdoop Trust MANager (Hatman) [50]. However, the above schemes do not protect against repudiation of origin attacks. AssureMR [198] and CorrectMR [199] improve on this by making use of a Pedersen-Merkle-R-Tree based authenticated data structure and a digital signature scheme in addition to task replication. Task replication imposes a high level of resource requirements; it multiplies the computational resource required to process each task. In addition, the approach also depletes scalability. To lower the resource requirements, TrustMR [200], Trusted Sampling-based Third-party Result Verification (TS-TRV) [201], and Accountable MR [202][203] are proposed. These schemes replicate only a subset of the tasks, thus reducing the resource consumption. Nonetheless, they still introduce a high level of overhead cost and do not protect against repudiation of origin attacks. SecureMR [204] counters such attacks by employing task replication in conjunction with other measures, namely a commitment protocol, a verification protocol, and a digital signature scheme. However, like the earlier mentioned task-replication based schemes, the approach is still costly. More importantly, the task-replication based schemes mostly apply protections at the task level; they do not provide fine-grained, or object-level, protections.

Non-task-replication based solutions make use of cryptographic primitives and security protocols to protect the authenticity of JobData. The most notable solution is the one employed by Apache Hadoop [120]. In this solution, a number of security measures are taken [35]. To protect data-in-transit, it uses the Simple Authentication and Security Layer (SASL) framework, encryption schemes (e.g., AES), and Hypertext Transfer Protocol Secure (HTTPS) to, respectively, protect messages transmitted over Remote Procedure Call (RPC), Transmission Control Protocol over Internet Protocol (TCP/IP), and HTTP. However, these security measures are intended for countering external attacks. They do not provide data authenticity protection to data-at-rest, and they are intended for an MR service deployed in a single domain. In [43], Zhou et al. proposed a secure data processing system for distributed computing services, called Declarative Secure Distributed System (DS2). As a proof of concept, the system is used to implement an MR service with a data authentication facility, called Authenticated MapReduce. In this system, each JobData object produced by a Mapper is signed with a data authentication scheme, i.e., HMAC-SHA1 (MAC) or RSA-1024 (digital signature). Their experimental results show that, with the respective use of HMAC-SHA1 and RSA-1024, the query completion latency of a job execution is increased by 17.4% and 78.3%, in comparison with the case where no data authentication measure is used. This indicates that, when using a MAC scheme (HMAC-SHA1), the protection level is insufficient as non-repudiation of origin is not provided, but when applying the digital signature scheme to secure each individual object (in order to provide non-repudiation), a significant level of delay is added onto a job execution process and is highly inefficient.

6.2.3 What is Missing

The data authentication solutions discussed above are critically analysed against the requirements with regards to data authenticity and non-repudiation of origin protections, i.e., (FR1), (FR4), (FR5), (SR5), (SR6), (SR7), (PR1), and (PR2), specified in Section 4.5. The knowledge gaps are identified and summarised in Table 6.1. From this table, we can make the following observations.

- None of the existing data authentication solutions provides a full-cycle protection (FR1) of data authentication to MR based distributed computing in this CBDC-MPC context. Most of the non-MR specific solutions are either designed for broadcast/multicast applications or for data collection applications. In a broadcast/multicast application, a single producer produces and sends the same data object to multiple consumers. In a data collection application, there are multiple producers but a single consumer; the different producers produce data objects (typically containing different contents) but send them to the same consumer. These applications are different from MR based applications where a single job execution consists of multiple stages, each stage is characterised by a different communication pattern, and some pattern (i.e., the M2M pattern) involves multiple producers each producing different data objects for different consumers or multiple consumers each consumes different data objects that are produced by different producers. Existing MR specific solutions are mostly designed for addressing external threats. The issue of assuring non-repudiation of data origin in an MPC environment was not specifically considered in the design of these solutions.
- As mentioned in earlier chapters, multi-stage Big Data processing in this environment indicates that it is important to reduce processing delays as introduced by security protections as much as possible (PR1), while providing the full-cycle protection (FR1). Digitally signing every data object to protect non-repudiation of origin is not desirable to satisfy requirement (PR1), particularly in cases where (i) there are multiple producers each producing a different data object for a different consumer and (ii) there are multiple consumers each consumes a different data object produced by a different producer. This is because in case (i) it would require each producer to generate a separate digital signature for a data object destined to a different consumer. If there is a large number of consumers, then the producers would be prone to becoming a performance bottleneck. Similarly, in case (ii), it would require each consumer to verify multiple signatures each signed by a different producer. If there is a large number of producers, the consumers would be prone to becoming a performance bottleneck.
- Symmetric-key based solutions without applying any form of asymmetry are not applicable to our use case due to the lack of non-repudiation of origin protection. Information-asymmetry based solutions imposed a high-level of computational and communication overhead costs on entities and the underlying networks, respectively. Time-asymmetry based solutions introduce additional delays to the execution of a job due to the deferment of the release of verification keys. These

solutions are not suited to time-sensitive applications such as one addressed in this research project.

- Task-replication based solutions are designed to verify the correctness of the output of a job under an assumption that there are sufficient redundant resources allocated for the job. This assumption is not compatible with the use case considered in this research. In this use case, a data processing job is carried out on computation resources shared by multiple organisations. In other words, all the available resources are to be shared fairly to all the collaborative organisations and the resources that can be allocated for the job could be limited.

Table 6.1: Related data authentication solutions.

	Requirements							
Approaches	**(FR1)**	**(FR4)**	**(FR5)**	**(SR5)**	**(SR6)**	**(SR7)**	**(PR1)**	**(PR2)**
Non-MR specific								
Secret-share based [183][184]	×	√	√	√	√	∗	×	×
Symmetric-key with information-asymmetry based [186]	×	√	√	√	√	∗	∗	×
Symmetric-key with time-asymmetry based [187][39][190]	×	√	√	√	√	√	∗	√
Asymmetric-key with AuthData-chain based [39][194][40][41][42]	×	×	√	√	√	√	√	√
Asymmetric-key with error-correcting-code based [195][197]	×	×	√	√	√	√	√	√
Asymmetric-key with hash-tree based [88]	×	√	√	√	√	√	√	√
MR specific								
Task-replication without digital-signature based [46][47][48][49][50] [200][201][202]	×	√	×	×	√	×	×	×
Task-replication with digital-signature based [198][199][204]	×	√	×	√	√	√	×	×
Measures used in Apache Hadoop [35]	×	√	√	∗	∗	∗	×	√
DS2 [43]	×	√	√	√	√	√	×	√

- Notes:
- √: Requirement is addressed.
- ∗: Requirement can be addressed with additional plug-in modules or minor modifications, or there is room for improvement.
- ×: Requirement is not addressed.

6.3 High-level Ideas

In this section, we describe high-level ideas used in the design of CPDA. Two main ideas are used in the design of CPDA. The first is AuthData and Communication Aggregation (ACA). With this idea, we apply and maximise the use of aggregation to the generation and verification of AuthData as well as communications between components. AuthData aggregation lowers the computational overhead costs imposed on data processing components by reducing the number of objects to be signed and verified with an expensive cryptographic scheme (i.e., digital signature). Communication aggregation can reduce the number of communications (interactions) among the components, reducing network traffics thus communication overhead cost. Communication aggregation can be done by introducing a third-party aggregator (referred to as Aggregator). Depending on the communication pattern used, AuthData aggregation and communication aggregation can be applied separately, or in a hybrid manner, to maximise the benefits they both bring. We have thus adopted a communication pattern based approach, i.e., we identify and classify different communication patterns among the data processing components during different phases of a job execution and apply one or both of AuthData aggregation and communication aggregation accordingly.

The second idea is a Hybrid use of multiple cryptographic schemes with Segregation of Credentials (HYSC). As mentioned earlier, a MAC scheme is computationally more efficient but does not provide non-repudiation of data origin, whereas a digital signature scheme provides the non-repudiation protection but is computationally expensive. To provide all of these protections at the finest granularity but with minimal overhead, we apply the MAC scheme to AuthData tokens that are pairwise transmitted (between one producer and Aggregator) but apply the digital signature scheme to aggregated AuthData that are used by multiple consumers. This hybrid use of cryptographic scheme can ensure the accountability of producers. With regard to credential segregation, pairwise keys used by different producers are segregated. In other words, each producer uses a different key to generate AuthData. This narrows the scope of accountability to the two entities sharing a key.

In the following, we describe, at a high level, how the two ideas described above are implemented. With regard to the communication pattern based AuthData aggregation, as there are three communication patterns, i.e., the one-to-many (O2M) pattern taking place in the job submission phase, the many-to-many (M2M) pattern in the map phase, and the many-to-one (M2O) pattern in the reduce phase, three AuthData aggregation methods are designed, one for each pattern. The three methods are, respectively, called Tree based AuthData Aggregation (TreeAgg) for O2M, Hybrid AuthData Aggregation (HybridAgg) for M2M, and Flat AuthData Aggregation (FlatAgg) for M2O. Before describing HybridAgg, we explain TreeAgg and FlatAgg as these two methods are used as building blocks for the design of HybridAgg.

6.3.1 TreeAgg Method

In the O2M pattern, the producer has to generate AuthData for multiple objects. To minimize the cost in protecting objects, we should require the producer to perform only one signature signing operation, but the resulting AuthData should allow each consumer to verify the object assigned to it independently.

The TreeAgg method is designed to accomplish this function. With this method, a binary tree containing aggregated AuthData for the whole set of N objects is constructed. The tree consists of N leaf nodes and $N-1$ internal nodes layered at multiple levels. Each leaf node represents the AuthData of a different object. Each internal node at the next level up in the hierarchy represents aggregated AuthData derived from its children (child nodes). The internal node at the top level is called the root node. The root node is the aggregated AuthData (referred to as Root-AuthData) for the entire set of the objects. The signature is then signed on the Root-AuthData. To minimize the amount of AuthData needed to verify each individual object, we make this tree a balanced full binary tree[7] [88]. An example of an AuthData tree for 8 objects is shown in Figure 6.1.

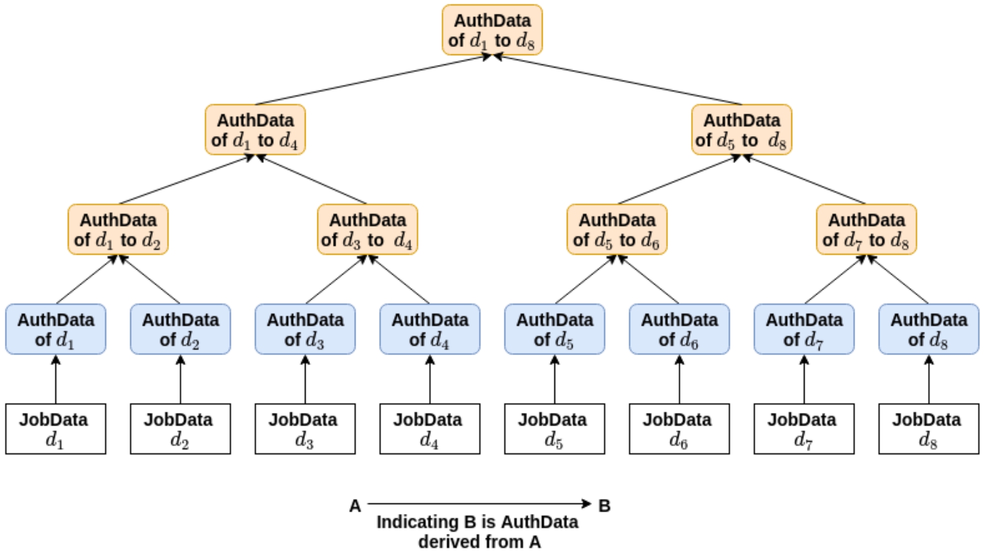


Figure 6.1: An AuthData tree for 8 objects.

AuthData needed for the verification of each object are object dependent, i.e., for different objects, their respective AuthData are different. This is because, as mentioned earlier, objects consumed by different consumers are typically different, and paths connecting each object to the root (Root-AuthData) of the tree are different. Furthermore, to minimise the size of each such AuthData thus minimising the computational and communication overheads, any redundant item in AuthData should be excluded. For these reasons, the AuthData associated to a particular object are constructed as the signature of the Root-AuthData (this token is the same for all of the objects) along with a set of object-specific AuthData tokens (these tokens are specifically tailored for each consumer). The object-specific AuthData tokens are Sibling-AuthData, i.e., the sibling nodes along the path from the leaf node (associated to the object) to the root node. The Sibling-AuthData for an object are illustrated in Figure 6.2. From the figure, we can see that the Sibling-AuthData for d_3 consist of three tokens: the AuthData of d_4, the AuthData of d_1 to d_2, and the AuthData of d_5 to d_8. Comparing with using all the leaf nodes to reconstruct the whole tree (thus the other seven nodes have to be transmitted along with the signature of Root-AuthData), our approach yields a reduction of 50% in communication overhead in terms of the number of tokens transmitted[8].

[7] A balanced full binary tree is a tree in which every internal node has exactly two child nodes and the left and the right subtrees of every node differ in height by no more than one. The height of such a tree for N objects is $\lceil \log N \rceil$.

[8] The tree-reconstruction approach requires 8 tokens (1 signature of the Root-AuthData and 7 Sibling-AuthData tokens) to be transmitted whereas our approach requires only 4 tokens (1 signature of the Root-AuthData and 3 Sibling-AuthData tokens) to be transmitted, thus the reduction of $4/8 = 50\%$.

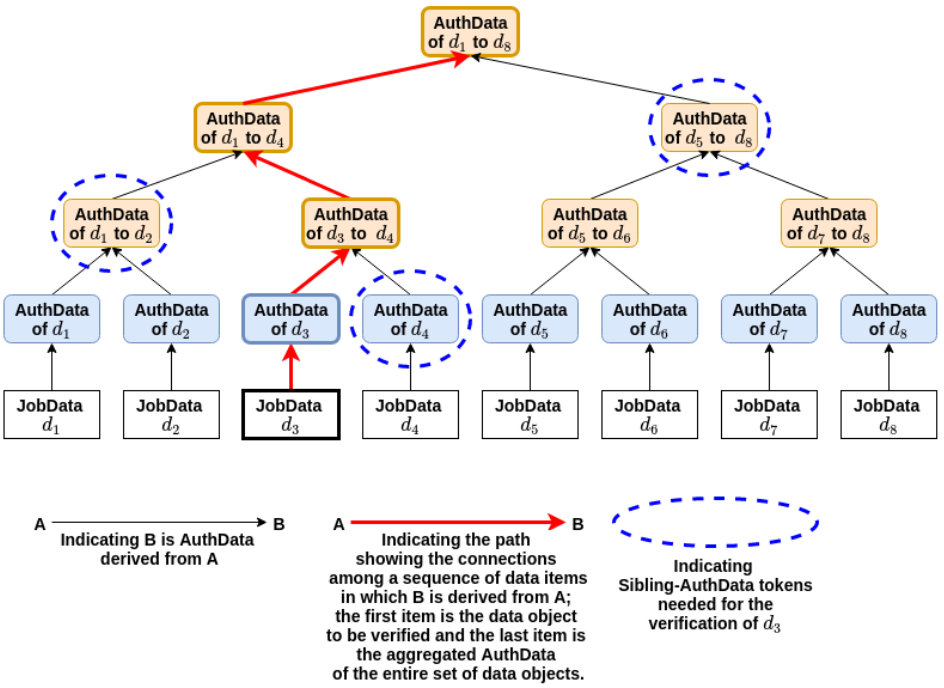
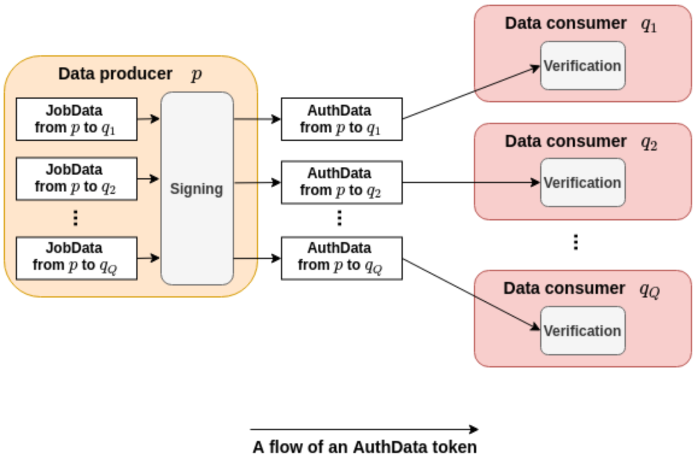
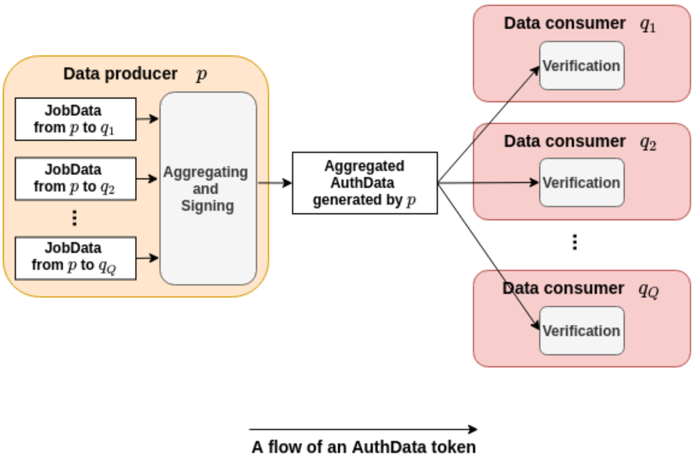

AuthData of d_1 to d_8
AuthData of d_1 to d_4
AuthData of d_5 to d_8
AuthData of d_1 to d_2
AuthData of d_3 to d_4
AuthData of d_5 to d_6
AuthData of d_7 to d_8
AuthData of d_1
AuthData of d_2
AuthData of d_3
AuthData of d_4
AuthData of d_5
AuthData of d_6
AuthData of d_7
AuthData of d_8
JobData d_1
JobData d_2
JobData d_3
JobData d_4
JobData d_5
JobData d_6
JobData d_7
JobData d_8
A
B
Indicating B is AuthData derived from A
A
B
Indicating the path showing the connections among a sequence of data items in which B is derived from A; the first item is the data object to be verified and the last item is the aggregated AuthData of the entire set of data objects.
Indicating Sibling-AuthData tokens needed for the verification of d_3

Figure 6.2: The Sibling-AuthData for d_3 in an AuthData tree for 8 objects.

Figure 6.3 contrasts the process and AuthData tokens sent by a producer to Q consumers with and without applying the TreeAgg method. As shown in the figure, when TreeAgg is not applied, the producer would need to sign the AuthData for each of the Q objects, respectively, before dispatching them to the consumers. This means that the producer needs to perform Q signature signing operations. In contrast, when TreeAgg is applied, the producer only needs to perform one AuthData aggregation operation and one signing operation.


Data producer p
JobData from p to q_1
JobData from p to q_2
⋮
JobData from p to q_Q
Signing
AuthData from p to q_1
AuthData from p to q_2
⋮
AuthData from p to q_Q
Data consumer q_1
Verification
Data consumer q_2
Verification
⋮
Data consumer q_Q
Verification
A flow of an AuthData token

(a)


Data producer p
JobData from p to q_1
JobData from p to q_2
JobData from p to q_Q
Aggregating and Signing
Aggregated AuthData generated by p
Data consumer q_1
Verification
Data consumer q_2
Verification
Data consumer q_Q
Verification
A flow of an AuthData token

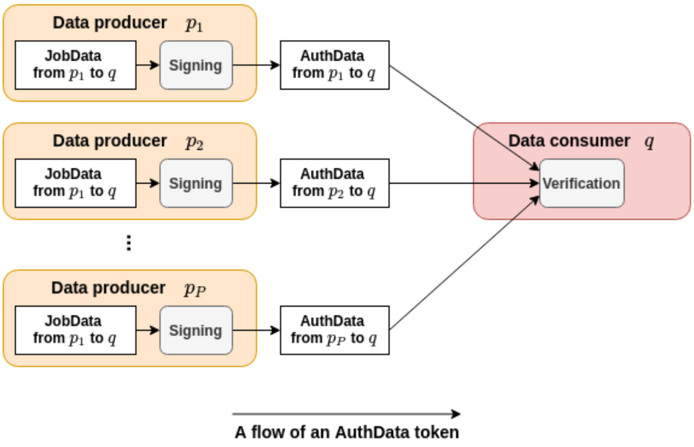
(b)

Figure 6.3: AuthData transmitted among components in the O2M pattern.
(a) Without the use of the TreeAgg method. (b) With the use of the TreeAgg method.

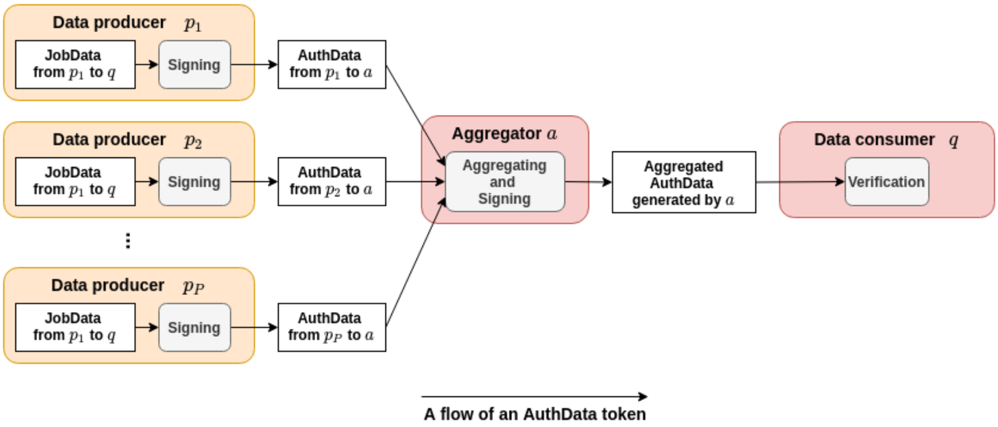
6.3.2 FlatAgg Method

In the M2O pattern, the consumer has to verify multiple objects. To minimize computational overhead incurred in verifying the objects, we should require the consumer to perform only one signature verification operation. To achieve this, we have introduced an idea of a third-party based aggregation method. The third party, called Aggregator, off-loads computational overhead away from the consumer as much as possible. It obtains and verifies AuthData generated and signed (with a MAC scheme) by different producers, then generates aggregated AuthData and signs (with a digital signature scheme) the aggregated AuthData before dispatching both the aggregated AuthData and the signature to the consumer. The size of the AuthData has also been reduced as much as possible to minimise bandwidth consumptions.

These measures have been captured in the FlatAgg method. Figure 6.4 illustrates the AuthData exchanged among the producers and the consumer with and without the use of the FlatAgg method.


Data producer p_1
JobData from p_1 to q
Signing
AuthData from p_1 to q
Data producer p_2
JobData from p_1 to q
Signing
AuthData from p_2 to q
Data producer p_P
JobData from p_1 to q
Signing
AuthData from p_P to q
Data consumer q
Verification
A flow of an AuthData token

(a)


Data producer p_1
JobData from p_1 to q
Signing
AuthData from p_1 to a
Data producer p_2
JobData from p_1 to q
Signing
AuthData from p_2 to a
⋮
Data producer p_P
JobData from p_1 to q
Signing
AuthData from p_P to a
Aggregator a
Aggregating and Signing
Aggregated AuthData generated by a
Data consumer q
Verification
A flow of an AuthData token

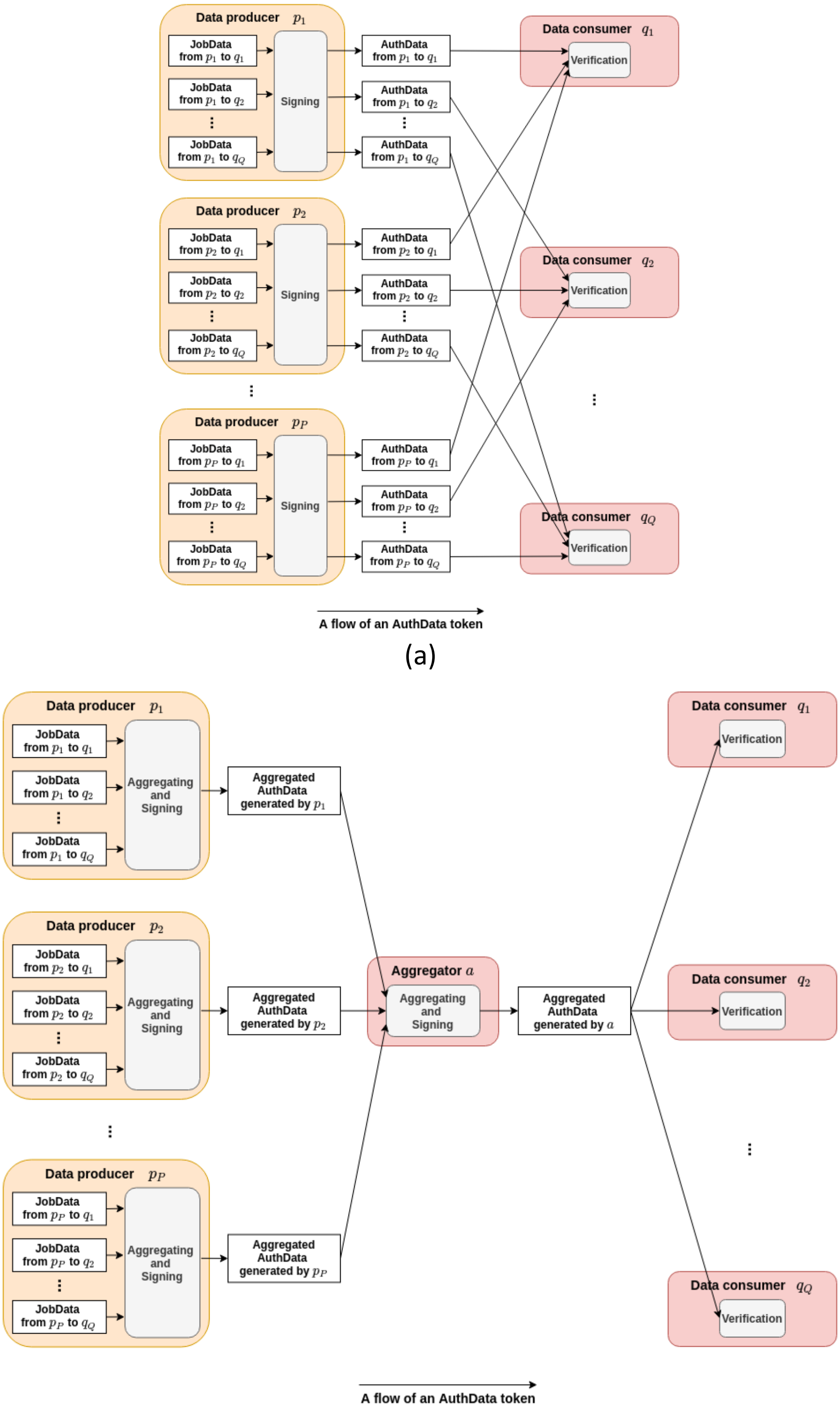
(b)

Figure 6.4: AuthData transmitted among components in the M2O pattern. (a) Without the use of the FlatAgg method. (b) With the use of the FlatAgg method.

6.3.3 HybridAgg Method

The M2M pattern can be viewed as the integration of the O2M and M2O patterns. Without any additional measures, each producer will need to sign Q objects, and each consumer will need to verify P objects. Furthermore, as there are AuthData to be transmitted between each pair of producer and consumer, there are up to $P * Q$ interactions taking place in this job execution phase, introducing a high-level of communication overhead cost. As mentioned earlier, to minimise the computational overhead cost, each producer should only perform one MAC signing operation, and each consumer should only perform one signature verification operation. To accomplish this, we apply two levels of AuthData aggregation, i.e., the intra-producer level aggregation and the inter-producer level aggregation. The intra-producer level aggregation is performed by each producer by using the TreeAgg method to aggregate the AuthData for the objects it produces, but only signing the Root-AuthData with a MAC scheme. The inter-producer level aggregation is performed by the Aggregator; as described in the FlatAgg method, it verifies and aggregates AuthData generated by different producers, and then signs the aggregated AuthData using a digital signature scheme. By introducing Aggregator, we can also apply communication aggregation. Each of the producers only sends AuthData it generates to Aggregator. Aggregator then dispatches the AuthData to each of the consumers. This cuts the number of interactions to only $P + Q$.

This idea has been implemented in the HybridAgg method. Figure 6.5 shows the flows of AuthData exchanged among components with and without the use of the HybridAgg method.


Data producer p_1
JobData from p_1 to q_1
JobData from p_1 to q_2
JobData from p_1 to q_Q
Signing
AuthData from p_1 to q_1
AuthData from p_1 to q_2
AuthData from p_1 to q_Q
Data producer p_2
JobData from p_2 to q_1
JobData from p_2 to q_2
JobData from p_2 to q_Q
AuthData from p_2 to q_1
AuthData from p_2 to q_2
AuthData from p_2 to q_Q
Data producer p_P
JobData from p_P to q_1
JobData from p_P to q_2
JobData from p_P to q_Q
AuthData from p_P to q_1
AuthData from p_P to q_2
AuthData from p_P to q_Q
Data consumer q_1
Data consumer q_2
Data consumer q_Q
Verification
A flow of an AuthData token
Aggregating and Signing
Aggregated AuthData generated by p_1
Aggregated AuthData generated by p_2
Aggregated AuthData generated by p_P
Aggregator a
Aggregated AuthData generated by a

(a)

(b)

Figure 6.5: AuthData exchanged among components in the M2M pattern. (a) Without the use of the HybridAgg method. (b) With the use of the HybridAgg method.

The second idea, i.e., a hybrid use of MAC and digital signature schemes in conjunction with the segregation of credentials, is implemented in the FlatAgg and HybridAgg methods. As explained in these methods, a MAC scheme is used to protect AuthData transferred between each producer and Aggregator and a digital signature scheme is used to protect aggregated AuthData dispatched by the Aggregator to consumers. For pairwise transmitted

AuthData, MAC can provide a sufficient level of protection. This is because the key used between a producer and the Aggregator is a pairwise key, the AuthData from each producer will be further aggregated and digitally signed by the Aggregator, and the Aggregator is trustworthy. If any fraudulent AuthData token is detected, its origin can be traced via the verification of the signature signed by the Aggregator and the verification of the tag signed (using a pairwise key) by the originator of the AuthData. The use of different pairwise keys captures the segregation of credentials.

6.4 Design Assumptions and Notations

In the design of the CPDA framework, we use the following design assumptions and notations.

6.4.1 Design Assumptions

The following assumptions are used in the design of CPDA.

(DAS1) Users are already authenticated prior to accessing the MR service.

(DAS2) The MR components allocated to a particular JobDomain are already authenticated prior to executing the job.

(DAS3) All the cryptographic keys that are used in data authentication are established when the MR components are authenticated; the public keys are certified and known to their respective users.

6.4.2 Notations

In addition to the notations listed in Table 5.2, additional notations used in the description of CPDA are shown in Table 6.2.

Table 6.2: Notations used in the description of CPDA.

Symbols	Meanings
P, Q	The numbers of producers, consumers
$k_{x,y}$	A pairwise key shared between x and y.
sk_x	A private key of x.
pk_x	A public key of x.
$d_{x,y}$	A data object produced by x and consumed by y.
$h_{x,y}$	The hash of $d_{x,y}$.
rh_x	A root hash of a hash tree constructed by x.
ch_x	A concatenated hash generated by x.
τ_o	A tag of an object o.
σ_o	A signature of an object o.
$sa_{x,y}$	The Sibling-AuthData token for $d_{x,y}$.
SA_x	A set of Sibling-AuthData tokens $\{sa_{x,1}, sa_{x,2}, \dots, sa_{x,C}\}$ for the objects generated by x.

6.5 CPDA in Detail

This section describes our novel data authentication solution, the CPDA framework. It gives an overview, and then the detailed description, of the framework. In the last subsection, it shows how the methods and protocols of the framework are collectively used to protect JobData throughout the whole cycle of a job execution. The algorithms implementing the methods used in the framework are formally described in Appendix C.

6.5.1 An Overview of the CPDA Architecture

As explained in Section 4.3.3, an MR job execution comprises three phases and each phase is characterised by a different communication pattern. Hence, the CPDA architecture consists of three modularised functional blocks, one for each job execution phase. These functional blocks are, respectively, the O2M block for the job submission phase, the M2M block for the map phase, and the M2O block for the reduce phase. An overview of the CPDA architecture is depicted in Figure 6.6.

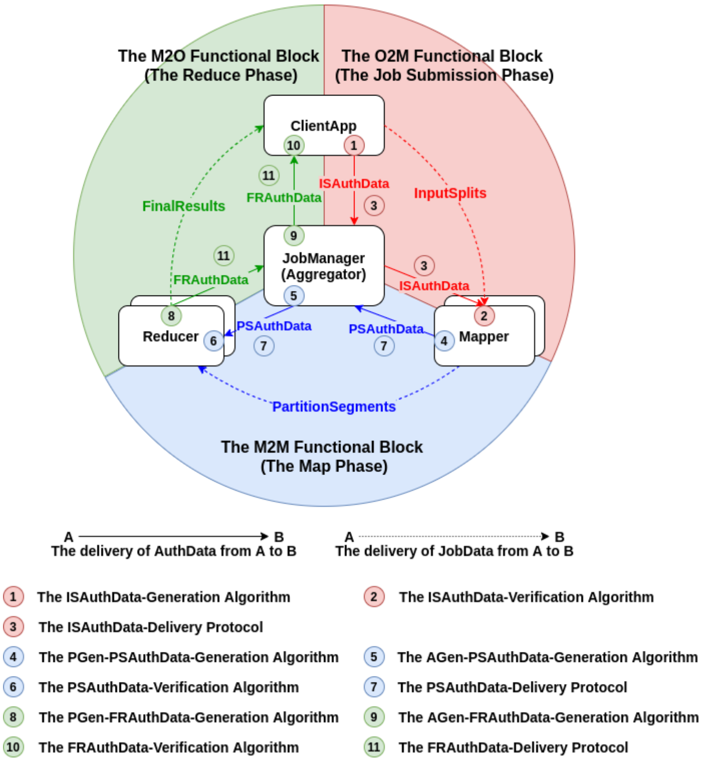

The M2O Functional Block
(The Reduce Phase)
The O2M Functional Block
(The Job Submission Phase)
ClientApp
10
1
11
FRAuthData
ISAuthData
3
InputSplits
FinalResults
9
JobManager
(Aggregator)
11
FRAuthData
3
ISAuthData
5
2
8
Reducer
6
PSAuthData
7
PSAuthData
7
4
Mapper
PartitionSegments
The M2M Functional Block
(The Map Phase)
A B
The delivery of AuthData from A to B
A B
The delivery of JobData from A to B
1 The ISAuthData-Generation Algorithm
2 The ISAuthData-Verification Algorithm
3 The ISAuthData-Delivery Protocol
4 The PGen-PSAuthData-Generation Algorithm
5 The AGen-PSAuthData-Generation Algorithm
6 The PSAuthData-Verification Algorithm
7 The PSAuthData-Delivery Protocol
8 The PGen-FRAuthData-Generation Algorithm
9 The AGen-FRAuthData-Generation Algorithm
10 The FRAuthData-Verification Algorithm
11 The FRAuthData-Delivery Protocol

Figure 6.6: An overview of the CPDA architecture.

As shown in the figure, each functional block consists of two AuthData generation algorithms (with the exception of the O2M block which has only one algorithm), one AuthData verification algorithm, and one AuthData delivery protocol. The AuthData generation algorithms are used to generate AuthData for JobData objects. They each utilise one of the three AuthData aggregation methods explained in Section 6.3. The AuthData verification algorithms are used to verify the authenticity of objects with the generated AuthData. The AuthData delivery protocols are used to deliver the AuthData from producers to Aggregator and from Aggregator to consumers. The delivery of AuthData is decoupled from the built-in JobData delivery mechanisms so that CPDA is not tightly bound to a specific MR implementation. In this way, it can be applied as an add-on and the modifications made to the underlying MR service are minimal.

Before describing the three functional blocks in detail, we first explain two AuthData aggregation algorithms (collectively implementing the three AuthData aggregation methods) and generic protocol message structure.

6.5.2 AuthData Aggregation Algorithms

The TreeAgg, FlatAgg, and HybridAgg methods can be realised by two AuthData aggregation algorithms, namely Hash-Tree based AuthData-Aggregation (HT-AuthData-Aggregation) and Hash-Concatenation based AuthData-Aggregation (HC-AuthData-Aggregation). TreeAgg is implemented by HT-AuthData-Aggregation, FlatAgg by HC-AuthData-Aggregation, and HybridAgg by both algorithms.

6.5.2.1 HT-AuthData-Aggregation Algorithm

The HT-AuthData-Aggregation algorithm uses a balanced full binary hash tree to aggregate AuthData for a set of Q objects produced by a producer x. It takes the hashes $h_{x,1}, h_{x,2}, \dots, h_{x,Q}$ of the objects as input and returns a root hash rh_x and a set of Sibling-AuthData tokens $SA_x = \{sa_{x,1}, sa_{x,2}, \dots, sa_{x,Q}\}$ as output. The algorithm constructs a hash tree ht by invoking the HT-Construction algorithm with the hashes and assigns the root hash to rh_x. Then, it iteratively invokes the SA-Extraction algorithm with each of $h_{x,1}, h_{x,2}, \dots, h_{x,Q}$ and appends the result to SA_x. Lastly, it returns rh_x and SA_x as output. The algorithm is detailed in Algorithm 6.1.1 (given in the Appendix C).

6.5.2.2 HC-AuthData-Aggregation Algorithm

The HC-AuthData-Aggregation algorithm is used by Aggregator α to generate aggregated AuthData for a set of P objects. These objects are consumed by a consumer y. It takes the hashes $h_{1,y}, h_{2,y}, \dots, h_{P,y}$ of the objects as input and returns a concatenated hash ch_α as output. This is done by concatenating all of the hashes and returning the resulting concatenated hash ch_α as output. The algorithm is detailed in Algorithm 6.1.2.

6.5.3 Protocol Message Structure and Format

The three AuthData-Delivery protocols, respectively, used in each of the job execution phases share a common transaction flow and message structure. For each AuthData delivery transaction, there are two protocol messages, namely an AuthData Delivery (ADD) message and an Acknowledgement (ACK) message. The ADD message is sent from an initiator to a respondent to transmit AuthData. The ACK message is conversely sent from the respondent back to the initiator to confirm the receipt of the ADD message. The exchange of these messages is depicted in Figure 6.7.

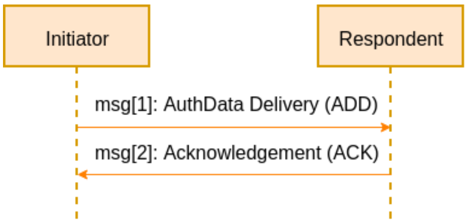

Initiator
Respondent
msg[1]: AuthData Delivery (ADD)
msg[2]: Acknowledgement (ACK)

Figure 6.7: A generic message transaction flow used in the AuthData-Delivery protocols of CPDA.

The structure of ADD messages and ACK messages is the same as those used in the entity authentication protocols of MIEA, as shown in Figure 5.7. It consists of a header and a payload. The descriptions of the fields contained in the header are summarised in Table 5.6. Unlike the messages used in MIEA, ADD and ACK messages use different PRO and MTYPE values, which are summarised in Table 6.3.

Table 6.3: Values for PRO and MTYPE of ADD and ACK messages.

Field	Numerical Value	Notation	Description
Protocol (PRO)	4	$ISAD$	ISAuthData-Delivery protocol
	5	$PSAD$	PSAuthData-Delivery protocol
	6	$FRAD$	FRAuthData-Delivery protocol
Message Type (MTYPE)	5	$ADD1$	ADD message sent from a producer to Aggregator (JobManager)
	6	$ADD2$	ADD message sent from Aggregator to a consumer
	7	ACK	ACK message acknowledging the preceding ADD message.

The payloads of different ADD messages have variable lengths. These will be further explained later on in each of the three functional blocks. For each ACK message, on the other hand, the payload contains only one item, i.e., the MID of the preceding ADD message, and has a fixed length. In other words, ACK messages of different protocols have the same format. Assuming that pro is the current protocol used; mid_1 and mid_2 are, respectively, the MIDs of the preceding ADD message and this ACK message; y and x are, respectively, the sender (a respondent) and the receiver (an initiator), the ACK message msg-ACK can be expressed as msg-ACK: $\{pro, mid_2, ACK, S(MID),\ id_y,\ did_y,\ id_x,\ did_x,\ mid_1\}$.

6.5.4 O2M Functional Block

The O2M functional block consists of the InputSplit AuthData-Generation (ISAuthData-Generation) algorithm, the InputSplit AuthData-Verification (ISAuthData-Verification) algorithm, and the InputSplit AuthData-Delivery (ISAuthData-Delivery) protocol. The ISAuthData-Generation algorithm is used by ClientApp to generate AuthData for M InputSplits. These AuthData are referred to as ISAuthData. The ISAuthData-Verification algorithm is used by each of the M Mappers for the verification of an InputSplit assigned to the Mapper. The ISAuthData-Delivery protocol is used to deliver the AuthData from ClientApp to JobManager and from JobManager to each of the Mappers. A high-level view of the O2M functional block is shown in Figure 6.8.

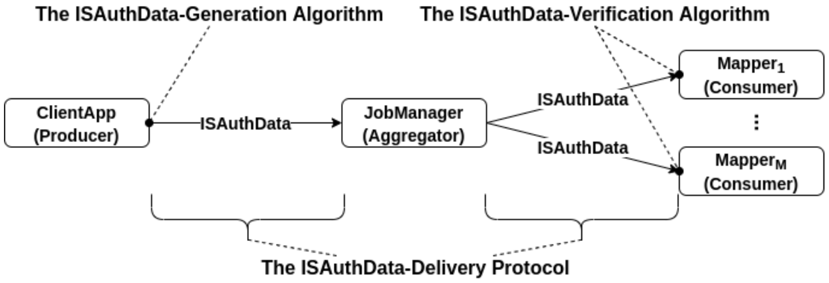


Figure 6.8: The components involved, and the algorithms and the protocol used, in the O2M functional block.

6.5.4.1 ISAuthData-Generation Algorithm

The ISAuthData-Generation algorithm uses the HT-AuthData-Aggregation algorithm (implementing TreeAgg) and a digital signature scheme to, respectively, generate and sign ISAuthData. It takes InputSplits $d_{c,m_1}, d_{c,m_2}, \dots, d_{c,m_M}$ (submitted by ClientApp c) and the private key sk_c as input and generates a signature σ_{rh_c} (of a root hash rh_c) and a set of Sibling-AuthData tokens $SA_c = \{sa_{c,m_1}, sa_{c,m_2}, \dots,\ sa_{c,m_M}\}$ as output. Firstly, it iteratively invokes the hash generation algorithm with each of the InputSplits to generate the hashes of the InputSplits and invokes the HT-AuthData-Aggregation algorithm with the hashes to obtain rh_c and SA_c. It then invokes the SIG-Signing algorithm with sk_c and rh_c to generate σ_{rh_c}. Lastly, it returns σ_{rh_c} and SA_c as output. The algorithm is detailed in Algorithm 6.2.1.

6.5.4.2 ISAuthData-Verification Algorithm

The ISAuthData-Verification algorithm verifies the authenticity of an InputSplit with ISAuthData (i.e., the signature of the root hash and the corresponding Sibling-AuthData token). It takes an InputSplit d_{c,m_a} (submitted by ClientApp c and assigned to Mapper m_a), the signature σ_{rh_c} (of the root hash rh_c), the Sibling-AuthData token sa_{c,m_a}, and the public key pk_c as input and returns the verification result sv as output. Firstly, it invokes the hash generation algorithm with d_{c,m_a} to generate the hash h'_{c,m_a} of the InputSplit and invokes the RA-Recovery algorithm with h'_{c,m_a} and sa_{c,m_a} to obtain rh'_c. It then invokes the SIG-Verification algorithm with pk_c, rh'_c, and σ_{rh_c} and returns the verification result sv as output. The algorithm is detailed in Algorithm 6.2.2.

6.5.4.3 ISAuthData-Delivery Protocol

As described earlier in Section 6.5.3, two messages, i.e., an ADD message and an ACK message, are used in each AuthData delivery transaction. For the delivery of the signature σ_{rh_c} and a set of Sibling-AuthData tokens $sa_{c,m_1}, \dots, sa_{c,m_M}$ from ClientApp c to JobManager jm, the ISAuthData-Delivery protocol uses two messages, msg-ISADD1 and msg-ISACK1. These two messages are, respectively, expressed as: msg-ISADD1: $\{ISAD, mid_1,\ ADD1,\ S(\sigma_{rh_c}) + S(\{sa_{c,m_1}, \dots ,\ sa_{c,m_M}\}),\ id_c, did_c,\ id_{jm}, did_{jm},\ \sigma_{rh_c},\ sa_{c,m_1},\ \dots ,\ sa_{c,m_M}\}$ and msg-ISACK1: $\{ISAD, mid_2,\ ACK,\ S(MID),\ id_{jm},\ did_{jm}, id_c, did_c, mid_1\}$.

For the delivery of the signature σ_{rh_c} and the respective Sibling-AuthData token sa_{c,m_a} from JobManager jm to each Mapper m_a, the ISAuthData-Delivery protocol also uses two messages, msg-ISADD2 and msg-ISACK2. These two messages are, respectively, expressed as: msg-ISADD2: $\{ISAD, mid_1,\ ADD2,\ S(\sigma_{rh_c}) + S(sa_{c,m_a}), id_{jm}, did_{jm}, id_{m_i}, did_{m_i}, \sigma_{rh_c}, sa_{c,m_a}\}$ and msg-ISACK2: $\{ISAD, mid_2,\ ACK,\ S(MID),\ id_{m_a}, did_{m_a}\ id_{jm}, did_{jm},\ mid_1\}$.

6.5.5 M2M Functional Block

The M2M functional block consists of the Producer-Generated PartitionSegment AuthData-Generation (PGen-PSAuthData-Generation) algorithm, the Aggregator-Generated PartitionSegment AuthData-Generation (AGen-PSAuthData-Generation) algorithm, the PartitionSegment AuthData-Verification (PSAuthData-Verification) algorithm, and the PartitionSegment AuthData-Delivery (PSAuthData-Delivery) protocol. The PGen-PSAuthData-Generation algorithm is used by each of the M Mappers to generate AuthData for a set of E PartitionSegments that are produced by the Mapper. These AuthData are referred to as PGen-PSAuthData. The AGen-PSAuthData-Generation algorithm is used by JobManager to generate aggregated AuthData for the PartitionSegments produced by all the Mappers. These AuthData are referred to as AGen-PSAuthData. The PSAuthData-Verification algorithm is used by each of the E Reducers to verify a set of M PartitionSegments that are assigned to the Reducer. The PSAuthData-Delivery protocol is used to deliver PGen-PSAuthData from each

Mapper to JobManager and to deliver PGen-PSAuthData and AGen-PSAuthData from JobManager to each Reducer. A high-level view of the M2M functional block is shown in Figure 6.9.

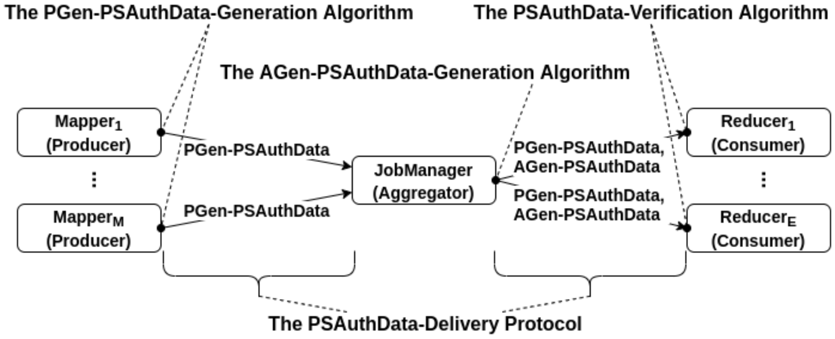

The PGen-PSAuthData-Generation Algorithm
The PSAuthData-Verification Algorithm
The AGen-PSAuthData-Generation Algorithm
Mapper$_1$ (Producer)
PGen-PSAuthData
JobManager (Aggregator)
PGen-PSAuthData, AGen-PSAuthData
Reducer$_1$ (Consumer)
Mapper$_M$ (Producer)
PGen-PSAuthData
PGen-PSAuthData, AGen-PSAuthData
Reducer$_E$ (Consumer)
The PSAuthData-Delivery Protocol

Figure 6.9: The components involved, and the algorithms and the protocol used, in the M2M functional block.

6.5.5.1 PGen-PSAuthData-Generation Algorithm

The PGen-PSAuthData-Generation algorithm uses the HT-AuthData-Aggregation algorithm (which implements the intra-producer level AuthData aggregation of HybridAgg) and a MAC scheme to, respectively, generate and sign PGen-PSAuthData. It takes the PartitionSegments $d_{m_a,r_1}, d_{m_a,r_2}, \dots, d_{m_a,r_E}$ (produced by a Mapper m_a) and the pairwise key $k_{m_a,jm}$ as input and generates a root hash rh_{m_a}, a tag $\tau_{rh_{m_a}}$ (of the root hash), and a set of Sibling-AuthData tokens $SA_{m_a} = \{sa_{m_a,r_1}, sa_{m_a,r_2}, \dots, sa_{m_a,r_E}\}$ as output. Firstly, it iteratively invokes the hash generation algorithm with each of the PartitionSegments to generate the hashes of the PartitionSegments and invokes the HT-AuthData-Aggregation algorithm with the hashes to obtain rh_{m_a} and SA_{m_a}. It then invokes the MAC-Signing algorithm with $k_{m_a,jm}$ and rh_{m_a} to generate $\tau_{rh_{m_a}}$. Lastly, it returns rh_{m_a}, $\tau_{rh_{m_a}}$, and SA_{m_a} as output. The algorithm is detailed in Algorithm 6.3.1.

6.5.5.2 AGen-PSAuthData-Generation Algorithm

The AGen-PSAuthData-Generation algorithm (used by JobManager) verifies PGen-PSAuthData generated by different producers and uses the HC-AuthData-Aggregation algorithm (which implements the inter-producer level AuthData aggregation of HybridAgg) and a digital signature scheme to, respectively, generate and sign AGen-PSAuthData. It takes the root hashes $rh_{m_1}, rh_{m_2}, \dots, rh_{m_M}$ (generated by different Mappers), the tags $\tau_{rh_{m_1}}, \tau_{rh_{m_2}}, \dots, \tau_{rh_{m_M}}$ (of the root hashes), the pairwise keys $k_{m_1,jm}, k_{m_2,jm}, \dots, k_{m_M,jm}$, and the private key sk_{jm} as input and generates a concatenated hash ch_{jm} and the signature $\sigma_{ch_{jm}}$ (of the concatenated hash) as output. Firstly, it iteratively invokes the MAC-Verify algorithm with each set of $k_{m_a,jm}$, rh_{m_a}, and $\tau_{rh_{m_a}}$ to verify the authenticity of rh_{m_a}, where $1 \leq a \leq M$. If all of the root hashes are authentic, then it invokes the HC-AuthData-Aggregation algorithm with the root hashes to generate ch_{jm}. Subsequently, it invokes the SIG-Signing algorithm with sk_{jm} and ch_{jm} to generate $\sigma_{ch_{jm}}$. It returns ch_{jm} and $\sigma_{ch_{jm}}$ as output. The algorithm is detailed in Algorithm 6.3.2.

6.5.5.3 PSAuthData-Verification Algorithm

The verification process consists of two phases. In the first phase, the authenticity of the concatenated hash ch_{jm} is verified against the signature $\sigma_{ch_{jm}}$. If the verification result is positive, then the process continues to the second phase. In the second phase, each PartitionSegment is verified against the respective Sibling-AuthData token and the respective root hash contained in ch_{jm}. The PSAuthData-Verification algorithm takes the PartitionSegments $d_{m_1,r_b}, d_{m_2,r_b}, \dots, d_{m_M,r_b}$ (assigned to a Reducer r_b), a set of Sibling-AuthData tokens $\{sa_{m_1,r_b}, sa_{m_2,r_b}, \dots, sa_{m_M,r_b}\}$, the concatenated hash ch_{jm}, the signature $\sigma_{ch_{jm}}$ (of the concatenated hash), and the public key pk_{jm} (of JobManager) as input and returns the verification result as output. It invokes the SIG-Verification algorithm with pk_{jm}, ch_{jm}, and $\sigma_{ch_{jm}}$ to verify the authenticity of ch_{jm}. If the result is negative, return negative; otherwise, proceed to the next step. It iteratively invokes the hash generation algorithm with each of the PartitionSegments to generate the hashes of the PartitionSegments and invokes the RA-Recovery algorithm with each of the hashes and the respective SiblingAuthData token to generate root hashes $rh'_{m_1}, rh'_{m_2}, \dots, rh'_{m_M}$. It compares each rh'_{m_a} with rh_{m_a} (extracted from ch_{jm}) and returns the comparison result. The algorithm is detailed in Algorithm 6.3.3.

6.5.5.4 PSAuthData-Delivery Protocol

The PSAuthData-Delivery protocol also uses two messages, i.e., an ADD message and an ACK message, to, respectively, deliver AuthData and acknowledge the receipt of the AuthData. For the delivery of a root hash rh_{m_a}, a tag $\tau_{rh_{m_a}}$, and a set of Sibling-AuthData tokens $\{sa_{m_a,r_1}, \dots, sa_{m_a,r_E}\}$ from each Mapper m_a to JobManager jm, the PSAuthData-Delivery protocol uses to messages, msg-PSADD1 and msg-PSACK1. These two messages are, respectively, expressed as: msg-PSADD1: $\{PSAD, mid_1, ADD1, S(rh_{m_a}) + S(\tau_{rh_{m_a}}) + S(\{sa_{m_a,r_1}, \dots, sa_{m_a,r_E}\}), id_{m_a}, did_{m_a}, id_{jm}, did_{jm}, rh_{m_a}, \tau_{rh_{m_a}}, sa_{m_a,r_1}, \dots, sa_{m_a,r_E}\}$ and msg-PSACK1: $\{PSAD, mid_2, ACK, S(MID), id_{jm}, did_{jm}, id_{m_a}, did_{m_a}, mid_1\}$.

For the delivery of the concatenated hash ch_{jm}, the signature $\sigma_{ch_{jm}}$ and a respective set of Sibling-AuthData tokens $\{sa_{m_1,r_b}, \dots, sa_{m_M,r_b}\}$ from JobManager jm to each Reducer r_b, the PSAuthData-Delivery protocol uses two messages, msg-PSADD2 and msg-PSACK2. These two messages are, respectively, expressed as: msg-PSADD2: $\{PSAD, mid_1, ADD2, S(ch_{jm}) + S(\sigma_{ch_{jm}}) + S(\{sa_{m_1,r_b}, \dots, sa_{m_M,r_b}\}), id_{jm}, did_{jm}, id_{r_b}, did_{r_b}, ch_{jm}, \sigma_{ch_{jm}}, sa_{m_1,r_b}, \dots, sa_{m_M,r_b}\}$ and msg-PSACK2: $\{PSAD, mid_2, ACK, S(MID), id_{r_b}, did_{r_b}, id_{jm}, did_{jm}, mid_1\}$.

6.5.6 M2O Functional Block

The M2O functional block consists of the Producer-Generated FinalResult AuthData-Generation (PGen-FRAuthData-Generation) algorithm, the Aggregator-Generated FinalResult AuthData-Generation (AGen-FRAuthData-Generation) algorithm, the FinalResult AuthData-Verification (FRAuthData-Verification) algorithm, and the FinalResult AuthData-Delivery (FRAuthData-Delivery) protocol. The PGen-FRAuthData-Generation algorithm is used by each of the E Reducers to generate AuthData for the FinalResult produced by the Reducer. These

AuthData are referred to as PGen-FRAuthData. The AGen-FRAuthData-Generation algorithm is used by JobManager to generate aggregated AuthData for FinalResults produced by all the Reducers. These AuthData are referred to as AGen-FRAuthData. The FRAuthData-Verification algorithm is used by ClientApp for the verification of the entire set of the E FinalResults. The FRAuthData-Delivery protocol is used to deliver PGen-FRAuthData from each Reducer to JobManager and to deliver AGen-FRAuthData from JobManager to ClientApp. A high-level view of the M2O functional block is shown in Figure 6.10.

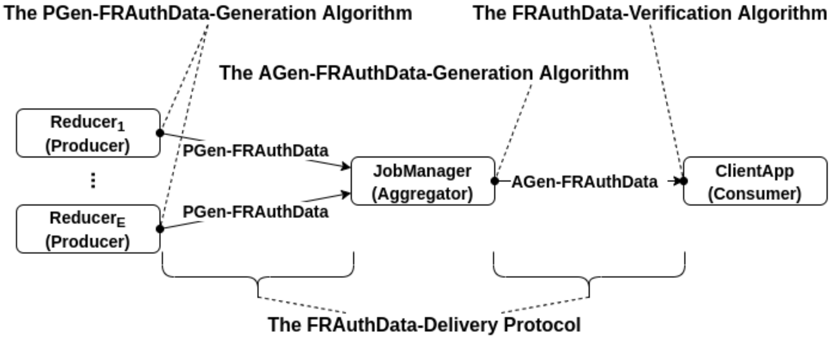

The PGen-FRAuthData-Generation Algorithm
The FRAuthData-Verification Algorithm
The AGen-FRAuthData-Generation Algorithm
Reducer$_1$ (Producer)
PGen-FRAuthData
⋮
Reducer$_E$ (Producer)
PGen-FRAuthData
JobManager (Aggregator)
AGen-FRAuthData
ClientApp (Consumer)
The FRAuthData-Delivery Protocol

Figure 6.10: The components involved, and the algorithms and the protocol, used in the M2O functional block.

6.5.6.1 PGen-FRAuthData-Generation Algorithm

The PGen-FRAuthData-Generation algorithm uses a hash function and a MAC scheme to, respectively, generate and sign AuthData for a FinalResult produced by a Reducer. It takes a FinalResult $d_{r_b,c}$ (produced by a Reducer r_b) and the pairwise key $k_{r_b,jm}$ as input and generates the hash $h_{r_b,c}$ (of the FinalResult) and the tag $\tau_{h_{r_b,c}}$ (of the hash) as output. Firstly, it invokes the hash generation algorithm with $d_{r_b,c}$ to obtain $h_{r_b,c}$. It then invokes the MAC-Signing function with $k_{r_b,jm}$ and $h_{r_b,c}$ to obtain $\tau_{h_{r_b,c}}$. Lastly, it returns $h_{r_b,c}$ and $\tau_{h_{r_b,c}}$ as output. The algorithm is detailed in Algorithm 6.4.1.

6.5.6.2 AGen-FRAuthData-Generation Algorithm

The AGen-FRAuthData-Generation algorithm verifies PGen-FRAuthData generated by different Reducers and uses the HC-AuthData-Aggregation algorithm (which implements FlatAgg) and a digital signature scheme to, respectively, generate and sign AGen-FRAuthData. It takes the hashes $h_{r_1,c}, h_{r_2,c}, \dots, h_{r_E,c}$ (of the FinalResults produced by all the Reducers), the tags $\tau_{h_{r_1,c}}, \tau_{h_{r_2,c}}, \dots, \tau_{h_{r_E,c}}$ (of the hashes), the pairwise keys $k_{r_1,jm}, k_{r_2,jm}, \dots, k_{r_E,jm}$, and the private key sk_{jm} as input and generates a concatenated hash ch_{jm} and the signature $\sigma_{ch_{jm}}$ (of the concatenated hash) as output. Firstly, it iteratively invokes the MAC-Verify algorithm with each set of $k_{r_b,jm}$, $h_{r_b,c}$, and $\tau_{h_{r_b,c}}$ to verify the authenticity of $h_{r_b,c}$, where $1 \leq b \leq E$. If all hashes are authentic, then it invokes the HC-AuthData-Aggregation algorithm with the hashes to generate ch_{jm}. Subsequently, it invokes the SIG-Signing algorithm with sk_{jm} and ch_{jm} to generate $\sigma_{ch_{jm}}$. It returns ch_{jm} and $\sigma_{ch_{jm}}$ as output. The algorithm is detailed in Algorithm 6.4.2.

6.5.6.3 FRAuthData-Verification Algorithm

Similar to PSAuthData-Verification, the verification process here also consists of two phases. In the first phase, the authenticity of the concatenated hash is verified, and, in the second phase, the hashes of the FinalResults are compared against the hashes contained in the concatenated hash. The FRAuthData-Verification algorithm takes the FinalResults $d_{r_1,c}, d_{r_2,c}, \ldots, d_{r_E,c}$ (consumed by ClientApp c), the concatenated hash ch_{jm}, the signature $\sigma_{ch_{jm}}$ (of the concatenated hash), and the public key pk_{jm} as input and returns the verification result as output. Firstly, it invokes the SIG-Verification function with pk_{jm}, ch_{jm}, and $\sigma_{ch_{jm}}$ to verify the authenticity of ch_{jm}. If the result is negative, return negative; otherwise, proceed to the next step. It iteratively invokes the hash generation algorithm with each of $d_{r_1,c}, d_{r_2,c}, \ldots, d_{r_E,c}$ to generate the hashes $h'_{r_1,c}, h'_{r_2,c}, \ldots, h'_{r_E,c}$. It compares each $h'_{r_b,c}$ with the respective $h_{r_b,c}$ extracted from ch_{jm}. It returns the comparison result as output. The algorithm is detailed in Algorithm 6.4.3.

6.5.6.4 FRAuthData-Delivery Protocol

Like the two AuthData-Delivery protocols explained earlier, the FRAuthData-Delivery protocol uses two messages, i.e., an ADD message and an ACK message, in each AuthData delivery transaction. For the delivery of a hash $h_{r_b,c}$ and a tag $\tau_{h_{r_b,c}}$ from each Reducer r_b to JobManager jm, the FRAuthData-Delivery protocol uses two messages, msg-FRADD1 and msg-FRACK1. These two messages are, respectively, expressed as: msg-FRADD1: $\{FRAD, mid_1,\ ADD1,\ S(h_{r_b,c}) + S(\tau_{h_{r_b,c}}),\ id_{r_b}, did_{r_b}, id_{jm}, did_{jm},\ h_{r_b,c},\ \tau_{h_{r_b,c}}\}$ and msg-FRACK1: $\{FRAD, mid_2,\ ACK,\ S(MID),\ id_{jm}, did_{jm},\ id_{r_b}, did_{r_b},\ mid_1\}$.

For the delivery of the concatenated hash ch_{jm} and the signature $\sigma_{ch_{jm}}$ from JobManager jm to ClientApp c, the FRAuthData-Delivery protocol uses two messages, msg-FRADD2 and msg-FRACK2. These two messages are, respectively, expressed as: msg-FRADD2: $\{FRAD, mid_1,\ ADD2,\ S(ch_{jm}) + S(\sigma_{ch_{jm}}),\ id_{jm}, did_{jm}, id_c, did_c,\ ch_{jm}, \sigma_{ch_{jm}}\}$ and msg-FRACK2: $\{FRAD, mid_2,\ ACK,\ S(MID),\ id_c, did_c,\ id_{jm}, did_{jm},\ mid_1\}$.

6.5.7 Putting Everything Together: CPDA in Action

The operation flow of CPDA when applied to a job execution is depicted as a sequence diagram shown in Figure 6.11. The sequence diagram highlights what and when the algorithms and the protocols are used by which components. Up on a successful execution of CPDA, (1) AuthData for all objects are generated and delivered to the respective consumers, (2) the authenticity (SR5) and (SR6) of each object can be verified against the related AuthData, and (3) producers cannot falsely deny producing their objects (SR7).

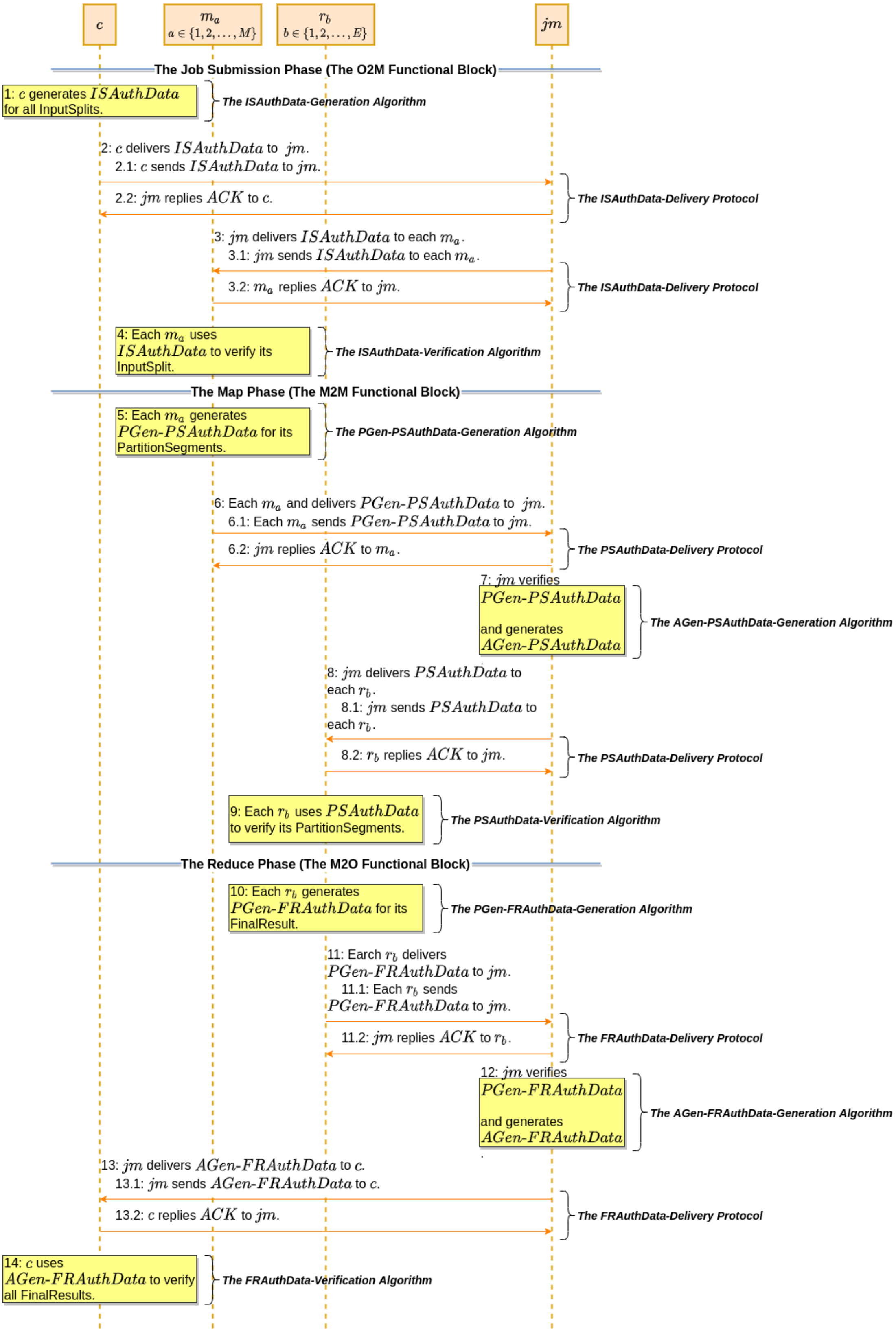

c
m_a $a \in \{1, 2, \ldots, M\}$
r_b $b \in \{1, 2, \ldots, E\}$
jm
The Job Submission Phase (The O2M Functional Block)
1: c generates $ISAuthData$ for all InputSplits.
The ISAuthData-Generation Algorithm
2: c delivers $ISAuthData$ to jm.
2.1: c sends $ISAuthData$ to jm.
2.2: jm replies ACK to c.
The ISAuthData-Delivery Protocol
3: jm delivers $ISAuthData$ to each m_a.
3.1: jm sends $ISAuthData$ to each m_a.
3.2: m_a replies ACK to jm.
The ISAuthData-Delivery Protocol
4: Each m_a uses $ISAuthData$ to verify its InputSplit.
The ISAuthData-Verification Algorithm
The Map Phase (The M2M Functional Block)
5: Each m_a generates $PGen\text{-}PSAuthData$ for its PartitionSegments.
The PGen-PSAuthData-Generation Algorithm
6: Each m_a and delivers $PGen\text{-}PSAuthData$ to jm.
6.1: Each m_a sends $PGen\text{-}PSAuthData$ to jm.
6.2: jm replies ACK to m_a.
The PSAuthData-Delivery Protocol
7: jm verifies $PGen\text{-}PSAuthData$ and generates $AGen\text{-}PSAuthData$
The AGen-PSAuthData-Generation Algorithm
8: jm delivers $PSAuthData$ to each r_b.
8.1: jm sends $PSAuthData$ to each r_b.
8.2: r_b replies ACK to jm.
The PSAuthData-Delivery Protocol
9: Each r_b uses $PSAuthData$ to verify its PartitionSegments.
The PSAuthData-Verification Algorithm
The Reduce Phase (The M2O Functional Block)
10: Each r_b generates $PGen\text{-}FRAuthData$ for its FinalResult.
The PGen-FRAuthData-Generation Algorithm
11: Earch r_b delivers $PGen\text{-}FRAuthData$ to jm.
11.1: Each r_b sends $PGen\text{-}FRAuthData$ to jm.
11.2: jm replies ACK to r_b.
The FRAuthData-Delivery Protocol
12: jm verifies $PGen\text{-}FRAuthData$ and generates $AGen\text{-}FRAuthData$.
The AGen-FRAuthData-Generation Algorithm
13: jm delivers $AGen\text{-}FRAuthData$ to c.
13.1: jm sends $AGen\text{-}FRAuthData$ to c.
13.2: c replies ACK to jm.
The FRAuthData-Delivery Protocol
14: c uses $AGen\text{-}FRAuthData$ to verify all FinalResults.
The FRAuthData-Verification Algorithm

Figure 6.11: The operations of the CPDA framework during the entire course of a job execution.

6.6 The Running Example

To demonstrate how CPDA works and to motivate the discussion, we further build on the running example described in Section 5.6 by applying CPDA to the job execution process. The example shows how the components of CPDA are used at different phases of the execution of the job to ensure the authenticity of JobData.

The result of the analysis in the running example is mission critical. If the result is contaminated and the organisations are misinformed, the organisations may misjudge the situation and make an inappropriate response, leading to potentially severe consequences. For example, if the potentially compromised machines are indeed compromised and these machines are not discovered during the security log analysis due to unauthorised alteration to the log files, the machines could be used as a backdoor to continue to cause harm to the systems of the organisations. For these reasons, the end users (i.e., the authorised employees of the respective organisations) should verify the authenticity of the analysis result to ensure that the output files produced by the distributed computing service (i.e., MR in this example) are processed by the authorised components and that the data used in the processing are from the expected sources and have not been tampered with. CPDA provides a data authentication service that protect the authenticity of JobData used, generated, and processed in every phase of a job execution, from when the input data are submitted to the MR services to when the output data are retrieved by the JobSubmitter. This is done by applying a different CPDA functional block to each phase of the job execution, i.e., the O2M functional block to the job submission phase, the M2M functional block to the map phase, and the M2O block to the reduce phase. Each of the functional blocks consists of three components: AuthData-Generation algorithms (except the O2M functional block which has only one algorithm), an AuthData-Verification algorithm, and an AuthData-Delivery protocol. The AuthData-Generation algorithms are used to generate AuthData (e.g., MAC tags and digital signatures) for JobData objects when the objects are produced by data producers (with the exception of InputSplits which are supplied by Users via ClientApps). The AuthData-Verification algorithms are used to verify the authenticity of the objects against the AuthData before the objects are consumed by data consumers. The AuthData-Delivery protocols are used to deliver AuthData from a data producer to the Aggregator (here it is $JobManager$) and from the Aggregator to a data consumer.

In the example, respective AuthData-Generation algorithms are used by ClientApps in step 4 (shown in Figure 4.8), Mappers in step 22, Reducers in step 26, and $JobManager$ in steps 23 and 27. Respective AuthData-Verification algorithms are used by $ClientApp^1$ in step 29, by Mappers in step 22, and by Reducers in step 25. Respective AuthData-Delivery protocols are used to transfer AuthData from ClientApps to $JobManager$ in step 11, from $JobManager$ to Mappers in step 20, from Mappers to $JobManager$ in step 23, from $JobManager$ to Reducers in step 24, from Reducers to JobManager in step 27, and from $JobManager$ to $ClientApp^1$ in step 28.

CPDA makes a hybrid use of symmetric-key and asymmetric-key cryptosystems for generating and verifying AuthData for protecting the authenticity of JobData objects. With regards to keys used, $ClientApp^1$, $ClientApp^2$, and $ClientApp^3$ each, respectively, generate a pair of private and public keys, i.e., sk_{c^1} and pk_{c^1} by $ClientApp^1$, sk_{c^2} and pk_{c^2} by $ClientApp^2$, and sk_{c^3} and pk_{c^3} by $ClientApp^3$. These keys are generated before the execution of the job. Each of the private keys is used for signing (generating AuthData for) the respective InputSplit by the respective ClientApp in step 4. Each of the public keys is used for verifying the assigned InputSplit by the respective Mapper in step 22. $Mapper_1$, $Mapper_2$, $Mapper_3$, $Reducer_1$, $Reducer_2$, and $Reducer_3$ are issued pairwise (symmetric) keys, i.e., $k_{m_1,jm}$, $k_{m_2,jm}$, $k_{m_3,jm}$, $k_{r_1,jm}$, $k_{r_2,jm}$, and $k_{r_3,jm}$, respectively, by $JobManager$ (keys for the Mappers are issued in step 20 and keys for the Reducers in step 24). The keys $k_{m_1,jm}$, $k_{m_2,jm}$, and $k_{m_3,jm}$ are used by the Mappers to sign the PartitionSegments they produce in step 22 and by $JobManager$ to verify AuthData (PGen-PSAuthData) generated by the Mappers in step 23. The keys $k_{r_1,jm}$, $k_{r_2,jm}$, and $k_{r_3,jm}$ are used by the Reducers to sign the FinalResults they produce in step 26 and by $JobManager$ to verify AuthData (PGen-FRAuthData) generated by the Reducers in step 27. $JobManager$ generates a pair of private and public keys, sk_{jm} and pk_{jm}, when $JobManager$ is launched in step 9. The private key sk_{jm} is used by $JobManager$ to generate AGen-PSAuthData for the PartitionSegments produced by the Mappers in step 23 and to generate AGen-FRAuthData for the FinalResults produced by the Reducers in step 27. The public key pk_{jm} is used by the Reducers to verify the assigned PartitionSegments in step 25 and by $ClientApp^1$ to verify the FinalResults in step 29. All the keys used in providing this data authentication service are distributed to the respective components by using the entity authentication service provided by MIEA.

In the following, we demonstrate how the different components of CPDA are used at different phases of the job execution. In this demonstration, we describe only operational steps related to data authentication (signing and verification) by $ClientApp^1$, $Mapper_1$, $Reducer_1$, and $JobManager$.

In the job submission phase, in step 4, after $ClientApp^1$ writes $InputSplit_{1,1}$ onto DFS^1, it signs $InputSplit_{1,1}$ by using the ISAuthData-Generation algorithm (explained in Section 6.5.4.1) with its private key sk_{c^1}, generating ISAuthData tokens $\sigma_{rh_{c^1}}$ (the signature of the root hash for $InputSplit_{1,1}$) and SA_{c^1} (a set of Sibling-AuthData token containing only one sa_{c^1,m_1}). The ISAuthData tokens $\sigma_{rh_{c^1}}$ and SA_{c^1} are sent from $ClientApp^1$ to $JobManager$ in step 11 and from $JobManager$ to $Mapper_1$ in step 20 by using the ISAuthData-Delivery protocol (explained in Section 6.5.4.3). The public key pk_{c^1} of $ClientApp^1$ is distributed from $ClientApp^1$ to $ResourceManager^1$ in step 5[9], from $ResourceManager^1$ to $JobManager$ in step 16, and from $JobManager$ to $Mapper_1$ in step 20. In step 20, $JobManager$ also sends a pairwise key $k_{m_1,jm}$ to $Mapper_1$ for signing the PartitionSegments produced by $Mapper_1$. In step 22, after $Mapper_1$ reads $InputSplit_{1,1}$ from DFS^1, it verifies the authenticity of $InputSplit_{1,1}$ by using the ISAuthData-Verification algorithm (explained in Section 6.5.4.2) with pk_{c^1}, $\sigma_{rh_{c^1}}$, and SA_{c^1} before executing its map task in the map phase.

In the map phase, in step 22, $Mapper_1$ performs its map tasks on $InputSplit_{1,1}$ and produces $PartitionSegment_{1,2}$ and $PartitionSegment_{1,3}$. It signs $PartitionSegment_{1,2}$

[9] ResourceManagers gets the public keys of the respective ClientApps in step 5. $ResourceManager^1$ collects all the public keys from the other ResourceManagers in step 15 before sending all the public keys to $JobManager$ in step 16.

and $PartitionSegment_{1,3}$ by using the PGen-PSAuthData-Generation algorithm (explained in Section 6.5.5.1) with $k_{m_1,jm}$, generating PGen-PSAuthData tokens rh_{m_1} (the root hash for the PartitionSegments), $\tau_{rh_{m_1}}$ (a MAC tag of the root hash), and SA_{m_1} (a set of Sibling-AuthData tokens containing sa_{m_1,r_2} and sa_{m_1,r_3}). The PGen-PSAuthData tokens rh_{m_1}, $\tau_{rh_{m_1}}$, and SA_{m_1} are sent from $Mapper^1$ to $JobManager$ in step 23 by using the PSAuthData-Delivery protocol (explained in Section 6.5.5.4). After $JobManager$ receives all the PGen-PSAuthData tokens generated by all the Mappers (including $Mapper_2$ and $Mapper_3$), $JobManager$ generates AGen-PSAuthData, ch_{jm} and $\sigma_{ch_{jm}}$, for all the PartitionSegments by using the AGen-PSAuthData-Generation algorithm (explained in Section 6.5.5.2) with its private key sk_{jm}, all the PGen-PSAuthData tokens, and the pairwise keys shared with the Mappers ($k_{m_1,jm}$, $k_{m_2,jm}$, and $k_{m_3,jm}$). In step 24, $JobManager$ sends the respective Sibling-AuthData token (sa_{m_2,r_1}) and AGen-PSAuthData (ch_{jm} and $\sigma_{ch_{jm}}$) to $Reducer_1$ by using the PSAuthData-Delivery protocol. It also sends its public key pk_{jm} and a pairwise key $k_{r_1,jm}$ to $Reducer_1$ by using the entity authentication service provided by MIEA. These keys are for $Reducer_1$ to verify the assigned PartitionSegments and to sign $FinalResult_{1,1}$, respectively. In step 25, after $Reducer_1$ reads $PartitionSegment_{2,1}$ from $WorkerManager_1^2$, it verifies the authenticity of $PartitionSegment_{2,1}$ by using the PSAuthData-Verification algorithm (explained in Section 6.5.5.3) with pk_{jm}, sa_{m_2,r_1}, ch_{jm}, and $\sigma_{ch_{jm}}$ before executing its reduce task in the reduce phase.

In the reduce phase, in step 26, $Reducer_1$ performs its reduce task on $PartitionSegment_{2,1}$ and produces $FinalResult_{1,1}$. It signs $FinalResult_{1,1}$ by using the PGen-FRAuthData-Generation algorithm (explained in Section 6.5.6.1) with $k_{r_1,jm}$, generating PGen-FRAuthData tokens h_{r_1,c^1} (the hash of $FinalResult_{1,1}$) and $\tau_{h_{r_1,c^1}}$ (the MAC tag of the hash). It then sends h_{r_1,c^1} and $\tau_{h_{r_1,c^1}}$ to $JobManager$ in step 27 by using the FRAuthData-Delivery protocol (explained in Section 6.5.6.4). After $JobManager$ receives all the PGen-FRAuthData tokens generated by all the Reducers (including $Reducer_2$ and $Reducer_3$), $JobManager$ generates AGen-FRAuthData, ch_{jm}^* and $\sigma_{ch_{jm}^*}$, for all the FinalResults by using the AGen-FRAuthData-Generation algorithm (explained in Section 6.5.6.2) with its private key sk_{jm}, all the PGen-FRAuthData tokens, and the pairwise keys shared with the Reducers ($k_{r_1,jm}$, $k_{r_2,jm}$, and $k_{r_3,jm}$). In step 28, $JobManager$ sends the AGen-PSAuthData (ch_{jm}^* and $\sigma_{ch_{jm}^*}$) to $ClientApp^1$ by using the FRAuthData-Delivery protocol. It also sends its public key pk_{jm} to $ClientApp^1$ for verifying all the FinalResults ($FinalResult_{1,1}$, $FinalResult_{2,1}$, and $FinalResult_{3,1}$). In other words, $ClientApp^1$ needs only one public key pk_{jm} to verify the authenticity of the result of the analysis job. In step 29, after $ClientApp^1$ reads all the FinalResults from DFS^1, it verifies the authenticity of the FinalResults by using the FRAuthData-Verification algorithm (explained in Section 6.5.6.3) with pk_{jm}, ch_{jm}^*, and $\sigma_{ch_{jm}^*}$ before presenting the FinalResults to $User^1$.

The detailed operational steps when both entity authentication (provided by MIEA) and data authentication (provided by CPDA) are applied to the job in the running example are elaborated in Section 7.3.

6.7 Security Analysis

The security of CPDA is analysed by using both an informal (property) method and a formal (complexity) analysis method. With the informal analysis method, we analyse CPDA against the security requirements (SR5), (SR6), and (SR7). The complexity analysis shows how much effort is required to successfully mount any of the attacks (T5) and (T6) against the system. The results are compared with those of the most related object based methods, i.e., the methods that secure individual objects by using a MAC scheme and a digital signature scheme, respectively. These methods are hereafter referred to as the MAC based scheme and the signature based scheme.

6.7.1 Informal Analysis

CPDA protects the authenticity of all the objects submitted or generated throughout the course of a job execution. In the job submission phase, each InputSplit can be verified against the respective Sibling-AuthData token and the signature of the root hash generated by ClientApp.

In the map phase, each PartitionSegment can be verified against the respective Sibling-AuthData token generated by the respective Mapper and the respective root hash contained in the concatenated hash which, in turn, is generated by JobManager. The authenticity of the concatenated hash is ensured by the signature generated by JobManager.

In the reduce phase, each FinalResult can be verified against the respective hash contained in the concatenated hash generated by JobManager. The authenticity of the concatenated hash, similarly, is ensured by the signature generated by JobManager.

6.7.1.1 Data Origin Authentication

Entities external to a job should not be able to inject a fraudulent object into the job. No entities should be able to falsify the origin of an object because it is computationally difficult to find an object that is different from an authentic one, but produces the same hash value, or to forge a new AuthData token (e.g., a tag or signature) for a fraudulent object. Hence, the CPDA framework satisfies the requirement of data origin authentication (SR5).

6.7.1.2 Data Integrity Protection

Any modifications made to any of the objects would change the hashes of the objects, thus different from when the objects are generated. When the tampered objects are verified against the respective AuthData, the result will be negative and such attempts will be detected. Therefore, the CPDA framework meets the requirement of data integrity protection (SR6).

6.7.1.3 Non-repudiation of Origin

In the job submission phase, non-repudiation is achieved by ClientApp signing the root hash of the hash tree. As only ClientApp knows the signature signing key and the signature verification key has been certified by a trusted entity (e.g., a certificate authority), any signature that has been positively verified must be from ClientApp.

In the map phase, JobManager provides a signature-protected concatenated hash containing authentic root hashes. The authenticity of the root hashes is ensured by the tags that are generated by the respective Mappers using their respective pairwise keys uniquely shared between each Mapper and JobManager. As JobManager is a trustworthy component and each pairwise key is only known by JobManager and the corresponding Mapper, it is hard for the Mapper to falsely deny that it has produced the PartitionSegments.

Similarly, in the reduce phase, JobManager provides a signature-protected concatenated hash containing authentic hashes. The authenticity of the hashes is, in turn, protected by using pairwise keys that are known only by JobManager and the respective Reducers. It is hard for each of the Reducers to falsely deny having produced the respective FinalResult. Therefore, the CPDA framework satisfies the requirement of non-repudiation of origin (SR7).

6.7.1.4 The Comparisons of the Security Properties

There are some differences in the security properties offered by the MAC based and the signature based schemes. Here, in Table 6.4, we provide a summary of the security properties provided by CPDA and these two schemes along with the use of JobManager, a trusted third party (TTP). The result shows that CPDA satisfies all of the specified security requirements, and it provides the same level of security protection as that provided by digitally signing all the data objects individually.

Table 6.4: The comparisons of security properties achieved by CPDA, the MAC based scheme, and the signature based scheme.

Security Requirement	MAC	MAC with TTP	Signature	CPDA
(SR5) Data origin authentication	√	√	√	√
(SR6) Data integrity protection	√	√	√	√
(SR7) Non-repudiation of origin	×	√	√	√

6.7.2 Complexity Analysis

The strengths of the security protections provided by CPDA are analysed in terms of computational complexity required to successfully mount a data injection attack (T5) and a data tampering attack (T6), respectively. In the following, we first give a list of notations used in the analysis, then the security strengths of cryptographic schemes (a hash function and a digital signature scheme) in addition to those discussed in Section 5.7.3.2, before comparing the strength of CPDA with those of the MAC based and signature based schemes.

6.7.2.1 Notations

Table 6.5 shows the notations used in this analysis; all of the lengths are expressed in bits.

Table 6.5: Notations used in the complexity analysis of CPDA.

Symbol	Meaning
L_h	Hash length
L_d	Object length
L_τ	MAC tag length
L_σ	Signature length
L_k	Secret key length
L_{sk}	Private key length
l	Security level

6.7.2.2 The Strength of Cryptographic Schemes

In Section 5.7.3.2, we have discussed the strengths of a number of cryptographic schemes, including a MAC scheme. In this section, we further analyse the strengths of a hash function and a digital signature scheme in terms of computational complexity needed to break an authentication token. The complexity is also expressed as 2^n. Cryptanalytical attacks on hash functions are omitted as these attacks can also be mitigated by using a scheme with no known vulnerabilities.

Attacks on hash functions can be classified into preimage attacks (finding a preimage of a given hash), second preimage attacks (given a preimage, finding a second preimage that produces the same hash), and collision attacks (finding two different preimages that produce the same hash). The complexities of launching a preimage attack and a second preimage attack are 2^{L_h} [178], whereas the computational complexity of launching a collision attack is $2^{L_h/2}$ [178]. Hence, the minimum complexity needed to successfully mount an attack on a hash is $2^{L_h/2}$.

For digital signatures, there exist signature forgery attacks that are more efficient than exhaustive search of new data objects, private keys, or signatures [205]. In other words, the computational complexity required to break a signature is much less than $2^{\min(L_d, L_{sk}, L_\sigma)}$. Rather, such complexity is usually expressed by using a notion of security levels, i.e., 2^l where l is a specified security level. A number of organisations, such as NIST [191], ENISA [206], and IETF [207], have estimated key lengths needed to achieve different security levels. For example, according to NIST [191], 3072-bit RSA and 256-bit ECDSA could be used to achieve a security level of 128-bit.

6.7.2.3 Data Injection Attacks

When CPDA is applied, to inject a fraudulent object into a job execution without being detected, an adversary may (1) find a new object that would yield the same hash as an existing object; (2) find a new object and a new Sibling-AuthData token that would produce the same root hash as the existing ones; or (3) forge a new tag or signature. For (1) and (2), finding a new object that produces the same hash requires a complexity of 2^d, or an attacker may perform one of preimage, second preimage, and collusion attacks, which requires a minimum computational complexity of $2^{L_h/2}$. For (3), forging a new tag and a new signature, respectively, requires computational complexities of $2^{\min(L_d, L_k, L_\tau)}$ and 2^l. Therefore, the complexity of successfully launching a data injection attack is $2^{\min\left(L_d, \frac{L_h}{2}, L_k, L_\tau, l\right)}$.

6.7.2.4 Data Tampering Attacks

To tamper with an existing object without being detected, an adversary may modify an existing object in a way that the modified object yields the same hash as an existing object, or generate fraudulent AuthData (e.g., Sibling-AuthData tokens, tags, and signatures) for the modified object. Besides finding a new object that produces the same AuthData token, a successful data tampering attack requires compromising a hash, a tag, or a signature. Therefore, the complexity of successfully launching a data tampering attack is also $2^{\min\left(L_d, \frac{L_h}{2}, L_k, L_\tau, l\right)}$.

6.7.2.5 The Comparisons of the Security Strengths

The strengths of the MAC based and signature based schemes are equal to the strengths of the underlying cryptographic schemes, i.e., $2^{\min(L_d, L_k, L_\tau)}$ and 2^l, respectively. The strengths of CPDA and these two schemes are summarised in Table 6.6.

Table 6.6: The comparisons of the security strengths of CPDA, the MAC based scheme, and the signature based scheme.

Attacks	MAC	Signature	CPDA
(T5) Data injection attacks	$2^{\min(L_d,L_k,L_\tau)}$	2^l	$2^{\min\left(L_d,\frac{L_h}{2},L_k,L_\tau,l\right)}$
(T6) Data tampering attacks	$2^{\min(L_d,L_k,L_\tau)}$	2^l	$2^{\min\left(L_d,\frac{L_h}{2},L_k,L_\tau,l\right)}$

6.8 Performance Evaluation

The overheads introduced by CPDA are theoretically evaluated in two aspects, computational overhead and communication overhead. The results are then compared with the overheads introduced by the MAC based and signature based schemes.

6.8.1 Notations

Table 6.7 shows the notations used in this performance evaluation.

Table 6.7: Notations used in performance evaluation of CPDA.

Symbols	Meanings
M, E	The numbers of Mappers, Reducers
OS_h, OL_h	Hash operation on a small object, a large object
OS_{ms}, OL_{ms}	MAC-Signing operation on a small object, a large object
OS_{mv}, OL_{mv}	MAC-Verification operation on a small object, a large object
OS_{ss}, OL_{ss}	SIG-Signing operation on a small object, a large object
OS_{sv}, OL_{sv}	SIG-Verification operation on a small object, a large object
L_{hd}	The header length of an ADD message
L_{ack}	The total length of an ACK message
L_h, L_τ, L_σ	The lengths of a hash, a tag, a signature

6.8.2 Computational Overheads

The computational overheads are evaluated in terms of the number of cryptographic operations performed by each of the CPDA components. Non-cryptographic operations (such as tree traversal and hash concatenation) are omitted as their costs (in terms of execution times) are negligible in comparison with those of cryptographic operations. The cryptographic operations are classified into five groups: hash generation (OS_h, OL_h), MAC-Signing (OS_{ms}, OL_{ms}), MAC-Verification (OS_{mv}, OL_{mv}), SIG-Signing (OS_{ss}, OL_{ss}), and SIG-Verification (OS_{sv}, OL_{sv}). As the cost of an operation is also affected by the size of an object, we count the operations performed on small objects (OS_h, OS_{ms}, OS_{mv}, OS_{ss}, and OS_{sv}) and on (potentially) large objects (OL_h, OL_{ms}, OL_{mv}, OL_{ss}, and OL_{sv}), separately.

6.8.2.1 CPDA Framework

In the job submission phase, two data authentication algorithms are used, ISAuthData-Generation and ISAuthData-Verification. ISAuthData-Generation is executed by ClientApp. It contains two sets of operations: one is for constructing a hash tree for M InputSplits ($M * OL_h + (M-1) * OS_h$), and the other is for signing the root hash with a digital signature scheme (OS_{ss}). Hence, the total number of operations is $M * OL_h + (M-1) * OS_h + OS_{ss}$. ISAuthData-Verification is executed by each Mapper. It contains three sets of operations, respectively, for computing the hash of its InputSplit (OL_h), for recovering the root hash from the hash ($\lceil \log M \rceil * OS_h$), and for verifying the root hash against the signature (OS_{sv}). Hence, the total number of operations is $OL_h + \lceil \log M \rceil * OS_h + OS_{sv}$.

In the map phase, three data authentication algorithms are used, PGen-PSAuthData-Generation, AGen-PSAuthData-Generation, and PSAuthData-Verification. PGen-PSAuthData-Generation is executed by each Mapper. It contains two sets of operations, respectively, for constructing a hash tree for E PartitionSegments ($E * OL_h + (E - 1) * OS_h$), and for signing the root hash with a MAC scheme (OS_{ms}). Hence, the total number of operations is $E * OL_h + (E - 1) * OS_h + OS_{ms}$. AGen-PSAuthData-Generation is executed by JobManager. It contains two sets of operations, respectively, for verifying the authenticity of M root hashes against the respective tags ($M * OS_{mv}$), and for signing the concatenated hash with a digital signature scheme (OL_{ss}). Hence, the total number of operations is $OL_{ss} + M * OS_{mv}$. PSAuthData-Verification is executed by each Reducer. It contains three sets of operations, respectively, for verifying the authenticity of the concatenated hash against the signature (OL_{sv}), for computing the hashes of M PartitionSegments ($M * OL_h$), and for recovering M root hashes from the hashes ($M * \lceil \log E \rceil * OS_h$). Hence, the total number of operations is $M * OL_h + OL_{sv} + M * \lceil \log E \rceil * OS_h$.

In the reduce phase, three data authentication algorithms are used, PGen-FRAuthData-Generation, AGen-FRAuthData-Generation, and FRAuthData-Verification. PGen-FRAuthData-Generation is executed by each Reducer. It contains two sets of operations, respectively, for computing the hash of its FinalResult (OL_h) and for signing the resulting hash with a MAC scheme (OS_{ms}). Hence, the total number of operations is $OL_h + OS_{ms}$. AGen-FRAuthData-Generation is executed by JobManager. It contains two sets of operations, respectively, for verifying E hashes against the respective tags ($E * OS_{mv}$) and for signing the concatenated hash with a digital signature scheme (OL_{ss}). Hence, the total number of operations is $OL_{ss} + E * OS_{mv}$. FRAuthData-Verification is executed by ClientApp and contains two sets of operations, respectively, for verifying the authenticity of the concatenated hash against the signature (OL_{sv}) and for computing the hashes of E FinalResults ($E * OL_h$). Hence, the total number of operations is $E * OL_h + OL_{sv}$.

6.8.2.2 MAC based and Signature based Schemes

With the MAC based scheme, each object is signed and verified individually using a MAC scheme. Similarly, with the signature based scheme, each object is individually protected by using a digital signature scheme. Hence, the number of operations performed by an individual component is equal to the number of objects to be protected.

With the MAC based scheme, in the job submission phase, ClientApp signs M InputSplits (ISAuthData-Generation). Each Mapper verifies one InputSplit (ISAuthData-Verification). Therefore, the numbers of operations performed by ClientApp and each Mapper are $M * OL_{ms}$ and OL_{mv}, respectively.

In the map phase, each Mapper signs E PartitionSegments (PGen-PSAuthData-Generation). Each Reducer verifies M PartitionSegments (PSAuthData-Verification). Therefore, the numbers of operations performed by each Mapper and each Reducer are $E * OL_{ms}$ and $M * OL_{mv}$, respectively.

In the reduce phase, each Reducer signs one FinalResult (PGen-FRAuthData-Generation). ClientApp verifies E FinalResults (FRAuthData-Verification). Therefore, the numbers of operations performed by each Reducer and ClientApp are OL_{ms} and $E * OL_{mv}$, respectively.

Using the same method with the signature based scheme, the numbers of operations performed by each of the components are the same as those using the MAC based scheme. The only difference lies in the cost of each signing and verification operation; here the operation is a signature operation, rather than a MAC operation.

6.8.2.3 *The Comparisons of the Computational Overheads*

The computational overheads when different data authentication solutions are applied are summarised in Table 6.8. The operations performed on large objects are highlighted in red. The result shows that CPDA reduces the number of expensive signature signing and verifying operations performed by each data processing component to one and these operations are performed on aggregated AuthData (root hashes and concatenated hashes) which are usually smaller than non-aggregated ones. This is achieved at a cost of additional operations imposed on JobManager. We anticipate that the level of reduction by CPDA should increase as the number of objects increases owing to a more significant level of decrease in expensive operations performed on large objects.

Table 6.8: The comparisons of the computational overheads imposed on individual components by different data authentication solutions.

The Job Submission Phase				
Component	**Algorithm**	**MAC**	**Signature**	**CPDA**
ClientApp	ISAuthData-Generation	$M * OL_{ms}$	$M * OL_{ss}$	$M * OL_h + (M-1) * OS_h + OS_{ss}$
Each Mapper	ISAuthData-Verification	OL_{mv}	OL_{sv}	$OL_h + \lceil \log M \rceil * OS_h + OS_{sv}$
The Map Phase				
Component	**Algorithm**	**MAC**	**Signature**	**CPDA**
Each Mapper	PGen-PSAuthData-Generation	$E * OL_{ms}$	$E * OL_{ss}$	$E * OL_h + (E-1) * OS_h + OS_{ms}$
JobManager	AGen-PSAuthData-Generation	-	-	$OL_{ss} + M * OS_{mv}$
Each Reducer	PSAuthData-Verification	$M * OL_{mv}$	$M * OL_{sv}$	$M * OL_h + OL_{sv} + M * \lceil \log E \rceil * OS_h$
The Reduce Phase				
Component	**Algorithm**	**MAC**	**Signature**	**CPDA**
Each Reducer	PGen-FRAuthData-Generation	OL_{ms}	OL_{ss}	$OL_h + OS_{ms}$
JobManager	AGen-FRAuthData-Generation	-	-	$OL_{ss} + E * OS_{mv}$
ClientApp	FRAuthData-Verification	$E * OL_{mv}$	$E * OL_{sv}$	$E * OL_h + OL_{sv}$

6.8.3 Communication Overheads

The communication overheads are evaluated in terms of the number and the sizes of messages exchanged between components. As explained in Section 6.5.3, each AuthData delivery transaction consists of two messages, one ADD message and one ACK message. The total size of an ADD message is equal to the sum of the size of the header (L_{hd}) and the size of the payload. The size of the payload is dependent on the number and sizes of AuthData tokens (L_h, L_τ, and L_σ) contained in the payload. The size of an ACK message is L_{ack}. For comparison, it is assumed that, for the cases where the MAC based and signature based schemes are used, AuthData sent from producers to consumers are also through JobManager, in the same way as the case for CPDA.

6.8.3.1 CPDA Framework

In the job submission phase, ClientApp sends one ADD message containing one signature and M instances of Sibling-AuthData (each containing up to $\lceil \log M \rceil$ hashes) to JobManager. Hence, the size of the message is $L_{hd} + L_\sigma + M * \lceil \log M \rceil * L_h$. JobManager replies with an ACK message with the size of L_{ack} to ClientApp. It then sends one ADD message containing one signature and one instance of Sibling-AuthData to each Mapper. The size of the message is $L_{hd} + L_\sigma + \lceil \log M \rceil * L_h$. Each Mapper replies with an ACK message with the size of L_{ack} to JobManager.

In the map phase, each Mapper sends one ADD message containing one tag, one root hash, and E instances of Sibling-AuthData (each containing up to $\lceil \log E \rceil$ hashes) to JobManager. The size of the message is $L_{hd} + L_\tau + L_h + E * \lceil \log E \rceil * L_h$. JobManager replies with an ACK message with the size of L_{ack} to each Mapper. After it generates AGen-PSAuthData, it sends one ADD message containing one signature, one concatenated hash (containing M root hashes), and M instances of Sibling-AuthData to each Reducer. The size of the message is $L_{hd} + L_\sigma + M * L_h + M * \lceil \log E \rceil * L_h$. Each Reducer replies with an ACK message with the size of L_{ack} to JobManager.

In the reduce phase, each Reducer sends one ADD message containing one tag and one hash to JobManager. The size of the message is $L_{hd} + L_\tau + L_h$. JobManager replies with an ACK message with the size of L_{ack} to each Reducer. After it generates AGen-FRAuthData, it sends one ADD message containing one signature and one concatenated hash (containing E hashes) to ClientApp. The size of the message is $L_{hd} + L_\sigma + E * L_h$. ClientApp replies with an ACK message with the size of L_{ack} to JobManager.

6.8.3.2 MAC based and Signature based Schemes

In the MAC based and signature based schemes, the size of the payload of each ADD message is dependent on the number and the size of authentication tokens to be delivered.

With the MAC based scheme, in the job submission phase, ClientApp sends one ADD message containing M tags to JobManager and JobManager replies with an ACK message to ClientApp. The sizes of these messages are respectively $L_{hd} + M * L_\tau$ and L_{ack}. JobManager sends one ADD message containing one tag to each Mapper and each Mapper replies with an ACK message to JobManager. The sizes of these messages are respectively $L_{hd} + L_\tau$ and L_{ack}.

In the map phase, each Mapper sends one ADD message containing E tags to JobManager and JobManager replies with an ACK message to each Mapper. The sizes of these messages are respectively $L_{hd} + E * L_\tau$ and L_{ack}. JobManager sends one ADD message containing M tags to each Reducer and each Reducer replies with an ACK message to JobManager. The sizes of these messages are respectively $L_{hd} + M * L_\tau$ and L_{ack}.

In the reduce phase, each Reducer sends one ADD message containing one tag to JobManager and JobManager replies with an ACK message to each Reducer. The sizes of these messages are respectively $L_{hd} + L_\tau$ and L_{ack}. JobManager sends one ADD message containing E tags to ClientApp and ClientApp replies with an ACK message to JobManager. The sizes of these messages are respectively $L_{hd} + E * L_\tau$ and L_{ack}.

Similarly, with the digital signature based scheme, the number of messages exchanged between components and the number of items contained in each of the messages are the same as those by using the MAC based scheme. The only difference is that the sizes of authentication tokens contained in the payloads of the messages used in these two schemes are different (i.e., L_σ rather than L_τ).

6.8.3.3 The Comparisons of the Communication Overheads

The communication overheads when different data authentication solutions are applied are shown in Table 6.9. The result shows that, the three solutions introduce the same numbers of messages. However, among the three solutions, the sizes of the ADD messages used in CPDA are the largest. We argue that the impact of the increased payload size to the underlying networks is insignificant as an ADD message is much smaller than a JobData object. For example, in the job submission phase when CPDA is applied, assuming that M = 1000, $L_h = 256$ bits, and $L_\sigma = 3072$ bits, the payload size of an ADD message that JobManager sends to each Mapper is equal to $3072 + \lceil \log 1000 \rceil * 256 = 5632$ bits = 704 B which is much smaller than the size of a 128-MiB InputSplit. Moreover, with CPDA, it is possible to reduce the communication overhead introduced by using signature caching. One copy of the same signature can be sent and cached on each WorkerNode rather than sending multiple copies to different Workers hosted on the same WorkerNode. The signature caching technique is not applicable to the MAC based and signature based schemes as AuthData tokens for different objects are different.

Table 6.9: The comparisons of the communication overheads introduced by different data authentication solutions.

The Job Submission Phase			
Interactions	**MAC**	**Signature**	**CPDA**
Between ClientApp and JobManager	1*ADD: $L_{hd} + M * L_\tau$ 1*ACK: L_{ack}	1*ADD: $L_{hd} + M * L_\sigma$ 1*ACK: L_{ack}	1*ADD: $L_{hd} + L_\sigma + M * \lceil \log M \rceil * L_h$ 1*ACK: L_{ack}
Between JobManager and each Mapper	1*ADD: $L_{hd} + L_\tau$ 1*ACK: L_{ack}	1*ADD: $L_{hd} + L_\sigma$ 1*ACK: L_{ack}	1*ADD: $L_{hd} + L_\sigma + \lceil \log M \rceil * L_h$ 1*ACK: L_{ack}
The Map Phase			
Interactions	**MAC**	**Signature**	**CPDA**
Between each Mapper and JobManager	1*ADD: $L_{hd} + E * L_\tau$ 1*ACK: L_{ack}	1*ADD: $L_{hd} + E * L_\sigma$ 1*ACK: L_{ack}	1*ADD: $L_{hd} + L_\tau + L_h + E * \lceil \log E \rceil * L_h$ 1*ACK: L_{ack}
Between JobManager and each Reducer	1*ADD: $L_{hd} + M * L_\tau$ 1*ACK: L_{ack}	1*ADD: $L_{hd} + M * L_\sigma$ 1*ACK: L_{ack}	1*ADD: $L_{hd} + L_\sigma + M * L_h + M * \lceil \log E \rceil * L_h$ 1*ACK: L_{ack}
The Reduce Phase			
Interactions	**MAC**	**Signature**	**CPDA**
Between each Reducer and JobManager	1*ADD: $L_{hd} + L_\tau$ 1*ACK: L_{ack}	1*ADD: $L_{hd} + L_\sigma$ 1*ACK: L_{ack}	1*ADD: $L_{hd} + L_\tau + L_h$ 1*ACK: L_{ack}
Between JobManager and ClientApp	1*ADD: $L_{hd} + E * L_\tau$ 1*ACK: L_{ack}	1*ADD: $L_{hd} + E * L_\sigma$ 1*ACK: L_{ack}	1*ADD: $L_{hd} + L_\sigma + E * L_h$ 1*ACK: L_{ack}

6.9 Experimental Evaluation

The performance of CPDA is experimentally evaluated when applied to MR job executions on a real-system testbed. For benchmarking, we compare the results with those of the MAC based and signature based schemes. In the following, we first explain methodology and evaluation metrics, then describe testbed setup and parameters used, before reporting our experimental results.

6.9.1 Methodology and Evaluation Metrics

The performance of CPDA is influenced by computational (operational costs imposed on components) as well as communication overheads (volume of traffics transmitted via networks for AuthData delivery). To evaluate such overhead costs, we have implemented three data authentication services (CDPA, the MAC based scheme, and the signature based scheme) and applied them to an MR service deployed on a cluster of machines. The evaluation consists of three experiments, Exp1, Exp2, and Exp3. Exp1 evaluates the costs of the cryptographic algorithms used, i.e., hash generation, MAC-Signing, MAC-Verification, SIG-Signing, and SIG-Verification. Exp2 evaluates the costs of data authentication algorithms, i.e., AuthData-Generation and AuthData-Verification algorithms, imposed on individual MR components. Exp3 evaluates the performance of the data authentication services when applied to job executions.

The costs of the cryptographic algorithms, the data authentication algorithms, and performance of the data authentication services are measured in terms of the execution times of the algorithms and jobs, respectively. For each particular set of parameter values, we collect multiple samples of execution times to calculate statistical values (i.e., mean values and standard error of the mean).

6.9.2 Testbed Setup

Our testbed consists of an MR service and the three data authentication services deployed on five networked machines. In the following, we describe the software and hardware of the testbed.

6.9.2.1 Software

Figure 6.12 depicts the software architecture of our testbed. It consists of a (simplified) MR service and three data authentication services. The interactions between MR components are shown as solid arrowed lines. The invocations of the three data authentication services are shown as dashed arrowed lines.

The MR service is implemented using MapReduce Lite [208] which is developed by Tencent. It provides job submission, task scheduling, and task execution functions. It has two types of components, Scheduler and Worker, which are written in Python. Scheduler performs the functions of both ClientApp and JobManager whereas each Worker performs the function of either a Mapper or a Reducer. To execute a task, Worker calls external data processing functions, i.e., map and reduce functions, and these functions are written in C++ and supplied by users. Workers are executed as application processes and they can be run on a single machine or multiple distributed machines. To enable data authentication for

MapReduce Lite, we have made a number of improvements to allow the invocations of the data authentication services and the transmission of AuthData.

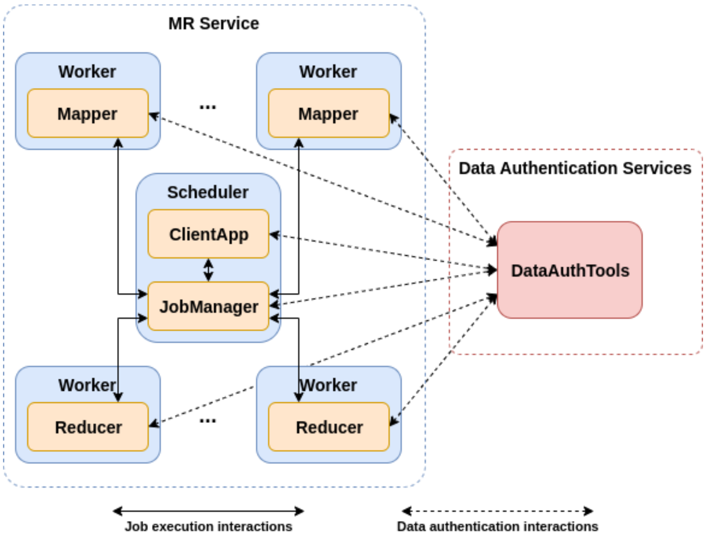


Figure 6.12: The software architecture of our testbed.

The three data authentication services are implemented as a single executable file, called DataAuthTools, which is written in C++. The cryptographic functions are implemented using the Botan cryptographic library [180]. In these implementations, we have selected (1) SHA-256 for the hash scheme; (2) HMAC with SHA-256 and 128-bit keys (referred to as HMAC-128) for the MAC scheme; and (3) RSA with SHA-256, 3072-bit keys, and the PSS padding scheme (referred to as RSA-3072) for the digital signature scheme. These schemes are chosen as they are widely accepted by academia and in industries. Examples where these schemes are used include the Transport Layer Security (TLS) protocol [148], the Internet Protocol Security (IPSec) protocol suite [209], and the Secure Shell (SSH) protocol [210]. The key and token sizes are set to achieve a sufficient level of security protection, which is 128 bits as recommended by NIST [191].

The specifications of the underlying operating system, the C/C++ compiler, the Python interpreter, and the cryptographic library used are given in Table 6.10.

Table 6.10: Software specifications.

Component	Specification
Operating system	Linux Manjaro 18.0.0 Illyria Kernel: 4.14.81-1-MANJARO x86_64
C/C++ compiler	gcc 8.2.1
Python interpreter	python 3.7.1
Cryptographic library	botan 2.8.0

6.9.2.2 Hardware

The testbed consists of five machines, labelled as PC1 through to PC5. PC1 is used to conduct Exp1 and Exp2 whereas all the PCs are used to conduct Exp3. The same set of software is installed on all machines. The hardware specifications of the machines are summarised in Table 6.11.

Table 6.11: Hardware specifications.

Machine	Components and Specifications
PC1	CPU: Quad Core Intel Core i7-6700, 64-bit, max 4.0 GHz RAM: 16 GB HDD: 1 TB
PC2	CPU: Quad Core Intel Core i5-3470, 64-bit, max 3.6 GHz RAM: 8 GB HDD: 500 GB
PC3	CPU: Quad Core Intel Core i5-3470, 64-bit, max 3.6 GHz RAM: 8 GB HDD: 500 GB
PC4	CPU: Quad Core Intel Core i7-2600, 64-bit, max 3.8 GHz RAM: 8 GB HDD: 500 GB
PC5	CPU: Dual Core Intel Core i3-2100, 64-bit, max 3.1 GHz RAM: 4 GB HDD: 250 GB

All the machines (PC1 through to PC5) are connected to a LAN via a 100-Mbps switch, as shown in Figure 6.13. ClientApp and JobManager are hosted on PC1, whereas Workers (Mappers and Reducers) are hosted on all of the machines. The distribution of the Workers is 25%, 20%, 20%, 20%, and 15%, respectively. The distribution is made based on the specifications of the machines.

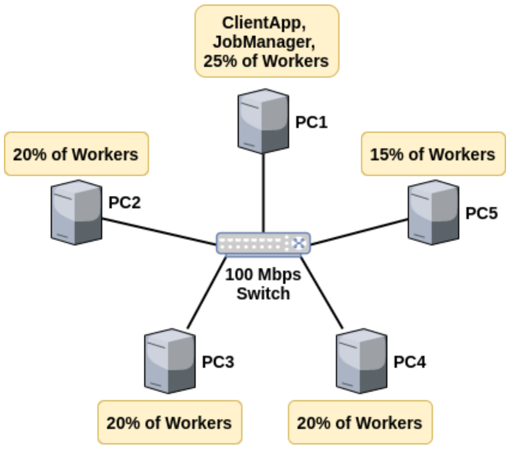


Figure 6.13: Network topology and the deployment of MR components on the testbed.

6.9.3 Parameters and Configurations

The sizes of data objects used in the experiments are expressed in bytes (B). For ease of presentation, we use a binary unit prefix to express multiples of units. This binary prefix signifies a multiplication by a power of 2, i.e., 1 KiB (kibibyte) refers to 2^{10} B = 1024 B and 1 MiB (mebibyte) refers to 2^{10} KiB = 1048576 B.

In Exp1, we measure the execution times of cryptographic algorithms performed on objects of different sizes. Each mean execution time is obtained from 1,000 samples. The sizes of the objects range from 32 B (the size of a hash) to 128 MiB (the size of an InputSplit) with an increment of twofold. The input data used are randomly generated binary data.

In Exp2, we measure the execution times of the algorithms used in implementing the three data authentication services with varying object sizes and varying numbers of Mappers and Reducers. Each mean execution time is obtained from 100 samples. The sizes of input objects are, respectively, 1 MiB, 16 MiB, and 128 MiB for ISAuthData-Generation and ISAuthData-Verification; 128 KiB, 1 MiB, and 16 MiB for PGen-PSAuthData-Generation, PSAuthData-Verification, PGen-FRAuthData-Generation, and FRAuthData-Verification; and

32 B (for each hash or root hash) for AGen-PSAuthData-Generation and AGen-FRAuthData-Generation. The object sizes are set based on the following considerations: (1) InputSplits are usually large (e.g., 128 MiB); (2) PartitionSegments and FinalResults are usually smaller than the InputSplits; and (3) the sizes of hashes and root hashes are fixed (32 B). The numbers of Mappers and Reducers are set to 1, 10, 20, …, 100. Like Exp1, the input data used in Exp2 are also randomly generated binary data.

In Exp3, we measure the execution times of jobs without data authentication and with each of the three data authentication services, respectively, given varying numbers of Mappers and Reducers. The time is measured from when ClientApp starts performing ISAuthData-Generation to when ClientApp finishes performing FRAuthData-Verification. Each mean execution time is obtained from 25 samples. The numbers of Mappers and Reducers used are set to 5, 50, 100, 200 and 5, 40, 80, 120, 160, 200, respectively. It is worth noting that, although deploying 400 Workers on a testbed of 5 machines (due to hardware accessibility) may be unusual in practice, the purpose of the experiment is to compare the performances of different data authentication services when being applied to an MR job using the same MR service and the same set of hardware. The wide range in Worker scaling allows us to see the trends in the performances of different data authentication services against the number of Workers. In this way, we can anticipate what the performance of our solution would be like when applying it to a larger scale MR service. We use the MR job described in Section 4.2 for the experiment. In this job, the map tasks (executed by Mappers) are to filter the weather data and output temperature values observed by each of the weather stations. For this task, each of the Mappers scans its InputSplit line-by-line and output a list of key-value pairs of a weather station ID and a temperature value. The reduce tasks (executed by Reducers) are to find the highest temperature value observed by each of the weather stations. Each of the Reducers the merged PartitionSegments key-by-key. For each key (weather station ID), the Reducer scans a list of values (temperature values) to find the highest value. It then outputs a list of key-value pairs of a weather station ID and the highest temperature value. The input data for the job are GHCN-Daily version 3.25 provided by NCEI[10] [211][212]. The input data are divided into multiple 128-MiB InputSplits. The InputSplits are stored on all of the machines prior to a job submission and the FinalResults are stored on the machine hosting ClientApp (i.e., PC1).

All of the input data for all the experiments are stored in RAM to minimize I/O overhead. In Exp3, due to the large size of weather data, 10-year data (approximately 12 GiB) cannot fit into RAMs of PC2 to PC5. As the content of input data should not affect how jobs are executed, we use a symbolic link approach to create a set of 10-year data from a smaller set of data. With this approach, we divide 2 years (2016 and 2017) of data into 10 of 128-MiB InputSplits. The remaining InputSplits are symbolic links pointing to the 10 InputSplits in a round-robin fashion. In this way, all InputSplits (including symbolic links) can be stored in RAMs of all the machines.

Like the experimental evaluations of MIEA (reported in Section 5.9), we also use standard error of the mean to estimate the error of sample means. According to our experiments, the results (the execution times) sampled in Exp1 are more dispersed than those in Exp2 and

[10] In particular, we use files stored in the “by_year” directory on the website.

Exp3, respectively. Therefore, to get more accurate results, Exp1 requires more samples than those of Exp2 and Exp3, respectively. The justification for the chosen sample sizes (1,000 for Exp1, 100 for Exp2, and 25 for Exp3) is that, with these sample sizes, the uncertainties of the mean execution times in terms of the relative standard error of the mean are lower than 1.5%. Again, a slight increase in the sample sizes will considerably increase experimental times, which does not justify a marginal gain of accuracy.

6.9.4 Experimental Results

This section reports the experimental results and discuss our findings.

6.9.4.1 Exp1: Costs of Cryptographic Algorithms

The execution times of SHA-256, HMAC-128, and RSA-3072 on objects with varying sizes are depicted in Figure 6.14.

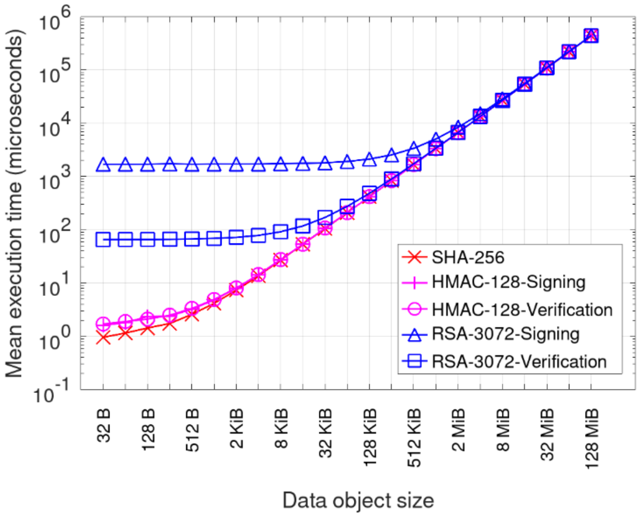


Figure 6.14: The comparisons of the execution times of SHA-256, HMAC-128, and RSA-3072 on objects with varying sizes.

From the figure, we can make the following observations. The mean execution times for SHA-256, HMAC-128-Signing and HMAC-128-Verification have similar values and they increase almost linearly as the size of the objects increases. For example, when the object size increases from 32B to 128 MiB (an order of 10^6 increase), the execution time increases from less than 2 microseconds, to about 440,000 microseconds (an order of 10^5 increase). This is because the larger size of the objects increases the workload of the algorithms thus the execution times.

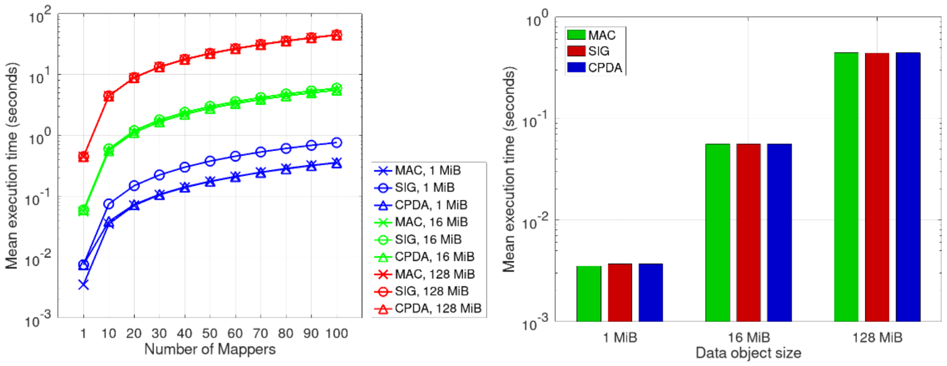
With regard to RSA-3072-Signing and RSA-3072-Verification, their execution times are of similar values when the size of each object goes beyond 512 KiB, and the values and trend are similar to those of the hash function and MAC based algorithms mentioned above. In other words, the differences in the costs of these algorithms are insignificant. However, when the object sizes are small, the execution time of RSA-3072-Signing is much higher than that of RSA-3072-Verification, and they do not change much when the object size is smaller than 32 KiB. For example, when the object size is 32B, RSA-3072-Signing takes about 1,700 microseconds to execute whereas for RSA-3072-Verification, the value is 65 microseconds. This means that, for small sized objects, RSA-3072-Signing is 26 times more expensive than

RSA-3072-Verification. This is because of the difference in their internal operations. The executions of these two algorithms are mostly influenced by two internal operations, (1) hash generation applied on the object and (2) a signature operation (signing or verification) applied on the resulting hash. The execution time of (1) increases when the object size increases but the execution time of (2) is fixed (as the hash size is fixed). When the object size is small (e.g., 32 B), the execution time of (2) is much longer than that of (1) (due to difficulty of computing asymmetric-key algorithms). In addition, RSA SIG-Signing is much more computationally expensive than SIG-Verification. However, the execution time of (1) surpasses that of (2) when the object size goes beyond a certain threshold (e.g., 512 KiB) and becomes the dominant cost of the signature based algorithms; the larger the object sizes, the closer the execution times of hash function, MAC based, and signature based algorithms thus the smaller difference in the execution times.

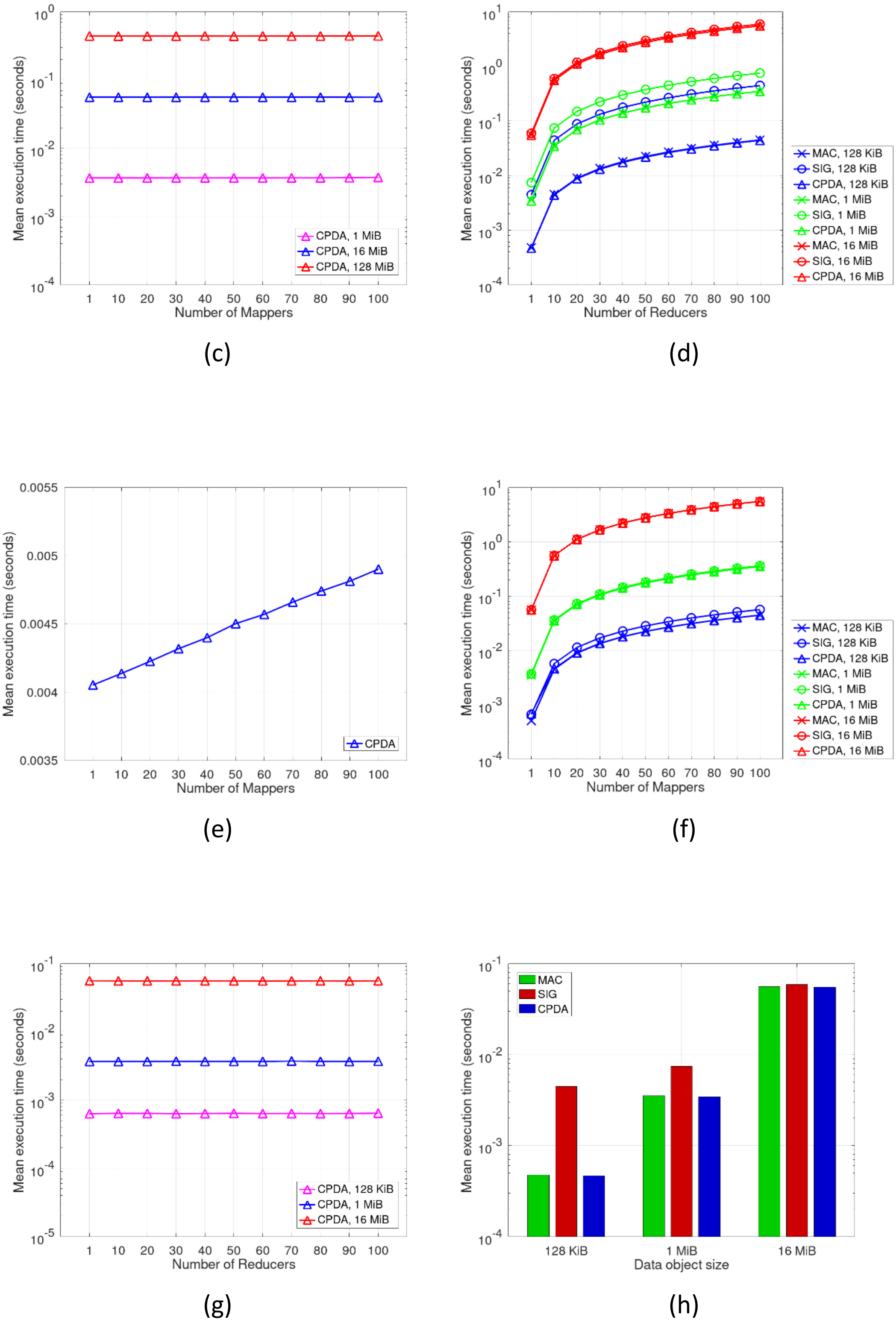
These results lead to the following findings: (1) the overhead introduced by data authentication solutions constructed based on these cryptographic algorithms should increase when the object sizes increase; (2) the hybrid approach to data authentication solutions, which minimises the use of digital signatures combined with the use of hash functions and MACs assisted with pairwise keys shared with a trusted third party, can bring significant reduction in computational overheads in providing data authentication; and (3) this reduction is more significant when the sizes of the objects to be protected are smaller, for example, when the object size is 32B, the hash function and MAC cost approximately 0.1% and 2.6% of RSA based signature signing and verification costs, respectively.

6.9.4.2 Exp2: Costs of Data Authentication Algorithms

The execution times of AuthData-Generation and AuthData-Verification algorithms used in the MAC based, signature (SIG) based, and CPDA are compared under different parameter value settings in terms of object sizes and the numbers of Workers used. The experimental results are shown in Figure 6.15.



(a) (b)



Mean execution time (seconds)
Number of Mappers
CPDA, 1 MiB
CPDA, 16 MiB
CPDA, 128 MiB
Number of Reducers
MAC, 128 KiB
SIG, 128 KiB
CPDA, 128 KiB
MAC, 1 MiB
SIG, 1 MiB
CPDA, 1 MiB
MAC, 16 MiB
SIG, 16 MiB
CPDA, 16 MiB
CPDA
0.0055
0.005
0.0045
0.004
0.0035
1
10
20
30
40
50
60
70
80
90
100
MAC
SIG
128 KiB
1 MiB
16 MiB
Data object size

(c)

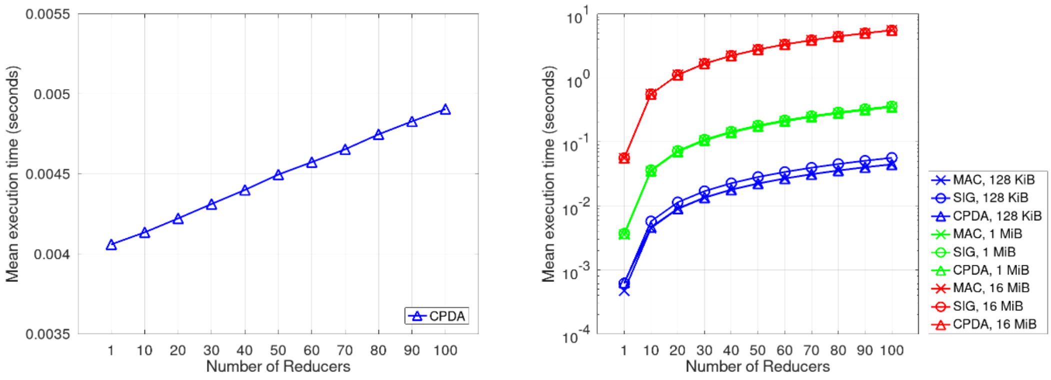
(d)

(e)

(f)

(g)

(h)



(i) (j)

Figure 6.15: The comparisons of the execution times of the data authentication algorithms used in the MAC based, signature (SIG) based and CPDA.
(a) ISAuthData-Generation. (b) ISAuthData-Verification with $M = 1$.
(c) ISAuthData-Verification of CPDA. (d) PGen-PSAuthData-Generation with $M = 1$.
(e) AGen-PSAuthData-Generation. (f) PSAuthData-Verification with $R = 1$.
(g) PSAuthData-Verification of CPDA with $M = 1$. (h) PGen-FRAuthData-Generation.
(i) AGen-FRAuthData-Generation. (j) FRAuthData-Verification.

Figure 6.15(a) shows the execution times of ISAuthData-Generation (executed by ClientApp) against the size of objects and number of Mappers. From the figure, we can see that the execution times increase as the object size increases, but the differences among the execution times are disappearing when the object sizes are large. For example, with 100 Mappers and 1-MiB object, the mean execution times of the MAC based, signature based, and CPDA are respectively 0.35 seconds, 0.76 seconds, and 0.35 seconds; whereas, with objects of 128-MiB size, all the mean execution times are approximately 44 seconds. This is because the execution times of ISAuthData-Generation are dependent on the execution times of the underlying cryptographic algorithms. The execution times of these algorithms increase as the size of the object increases and the difference among the execution times of different algorithms disappears when the object size is sufficiently large (as explained in Exp1).

In addition, the execution times also increase as the number of Mappers increases. This is because the increase in Mappers increases the number of InputSplits to be signed and verified, hence, the increase in execution times.

The most important observation from these results is that CPDA markedly outperforms the signature based when the size of the objects is small. For example, as reported above, given 100 Mappers, the mean execution times for the MAC based, the signature based, and CPDA are, respectively, 0.35 seconds, 0.76 seconds, and 0.35 seconds. This shows that CPDA gives a similar performance as the MAC based method. It is 53% more efficient than the signature based method.

Figure 6.15(b) shows the execution times of ISAuthData-Verification using the three different methods as against different object sizes. This verification operation is performed by a single Mapper. From the results, it can be seen that the three methods introduce a similar level of costs at any given object size and the costs only increase as the object size increases. For example, with 1-MiB sized objects, the mean execution times are 0.004 seconds, whereas, with 128-MiB sized objects, the mean execution times are 0.4 seconds. The costs of the algorithms used in three different solutions are at the same level because: (1) the costs of the algorithms are dependent on the numbers, and costs, of the underlying cryptographic operations used (as summarised in Table 6.8); (2) all the three solutions each perform one cryptographic operation (but different class of cryptographic operation) on the assigned InputSplit (which is large), with an exception of CPDA which introduces additional operations (i.e., root hash recovery and SIG-Verification on the root hash) on small objects; (3) as discussed in Exp1, when the object size is large, the three classes of cryptographic operations introduce the same level of costs, i.e., the differences in costs among different classes are very small; and (4) the costs of operations on small objects are negligible in comparison with those of operations on large objects when InputSplits are sufficiently large (e.g., 1 MiB).

Figure 6.15(c) shows the execution times of ISAuthData-Verification using CPDA with objects of different sizes and different numbers of Mappers. From the figure, it can be seen that the size of objects has a major effect on the execution times, the larger the size of the objects, the longer the execution time. This result is within our expectation, as the cost of hash generation increases as the object (InputSplit) size increases. However, for any given object size, the increase in execution times caused by the increase in Mappers is negligible. This is consistent with our theoretical analysis result, i.e., an increase in the number of Mappers would lead to an increase in the overhead of the method by $\lceil \log M \rceil * OS_h$ = $\lceil \log 100 \rceil * 1$ microseconds = 7 microseconds, which is negligible compared with the values of OL_h = 440,000 microseconds and OS_{sv} = 66 microseconds[11].

Figure 6.15(d) shows the execution times of PGen-PSAuthData-Generation (performed by a Mapper) against different object sizes and different numbers of Reducers. The trend in the results is very similar to the that in Figure 6.15(a), with an exception that here, in this figure, there are larger performance gaps between CPDA and the signature based and the gaps expand to medium (1-MiB) sized objects. This means that CPDA performs better in comparison with the signature based method with regard to this algorithm. The reason for this is that, in this algorithm, a MAC scheme is used to sign each root hash and the reduction in costs is larger than the ISAuthData-Generation algorithm.

Figure 6.15(e) shows the execution times of AGen-PSAuthData-Generation (performed by JobManager) against different numbers of Mappers. The execution times increase as the number of Mappers increases; they increased from 0.004 seconds to 0.005 seconds (an increase of 25%) when the number of Mappers increased from 1 to 100 (an order of two-magnitude increase). This indicates that, with regard to this algorithm, CPDA is highly scalable as the increase in the execution time is a fraction of the increased number of Mappers. As explained in Section 6.8.2, the increase in the number of Mappers increases the times needed for verifying the root hashes using MAC-Verify and the size of concatenated hash to be digitally signed thus the increase in the execution times of the algorithm. However, such increase is small compared to the execution time of SIG-Signing.

Figure 6.15(f) shows the execution times of PSAuthData-Verification (performed by a Reducer) against different object sizes and different numbers of Mappers. The trend is similar

[11] Here, OS_h is a hash operation on 64-byte data (a concatenation of two hashes), OL_h is a hash operation on 128-MiB data, and OS_{sv} is a SIG-Verification operation on 32-byte data (with padding to 3072 bits = 384 B).

to that shown in Figure 6.15(a) with an exception that the gaps between the results of CPDA and those of the signature based are smaller here. Although CPDA greatly reduce the number of signature verification operations used in this algorithm, the cost incurred in hash generation and MAC-Verify is close to that incurred in RSA SIG-Verification thus small reduction in the execution times.

Figure 6.15(g) shows the execution times of PSAuthData-Verification using CPDA (performed by a Reducer) against object sizes with the use of one Mapper and varying numbers of Reducers. The results show the similar patterns as those in Figure 6.15(c).

Figure 6.15(h) shows the execution times of PGen-FRAuthData-Generation (performed by a Reducer) against object sizes. Similar to the results shown in Figure 6.15(b), the execution times increase as the object size increases. In addition, as explained in Figure 6.15(d), as CPDA uses MAC scheme to sign each root hash, which is much cheaper than the digital signature based method, we get a larger cut in execution times in PGen-FRAuthData-Generation.

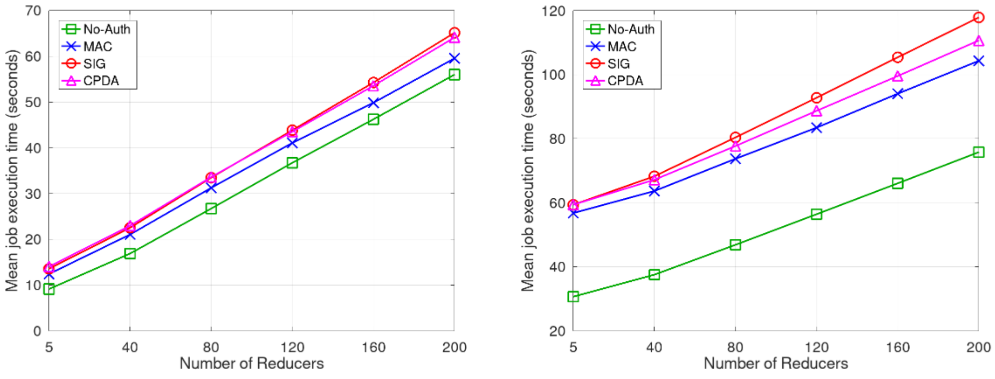
Figure 6.15(i) shows the execution times of AGen-FRAuthData-Generation (performed by JobManager) against the number of Reducers. The trend is similar to that in Figure 6.15(e).

Figure 6.15(j) shows the execution times of FRAuthData-Verification (performed by ClientApp) against object sizes with different numbers of Reducers. It exhibits the same trend as that in Figure 6.15(f).

The results from Exp2 show that the costs incurred by the CPDA algorithms are remarkably close to those introduced by the MAC based algorithms. CPDA is markedly more efficient than the signature based method, particularly when the data objects to be protected are of smaller sizes and the quantities of the objects are large (i.e., large number of producers and consumers). This is due to the reduction in expensive operations by CPDA and large differences in computational costs among different classes of cryptographic operations when applied to small objects. The cost reduction benefits all the data processing components in the system. The largest reduction occurs in PGen-PSAuthData-Generation; in comparison with the signature based method, bringing a cost reduction of 90%.

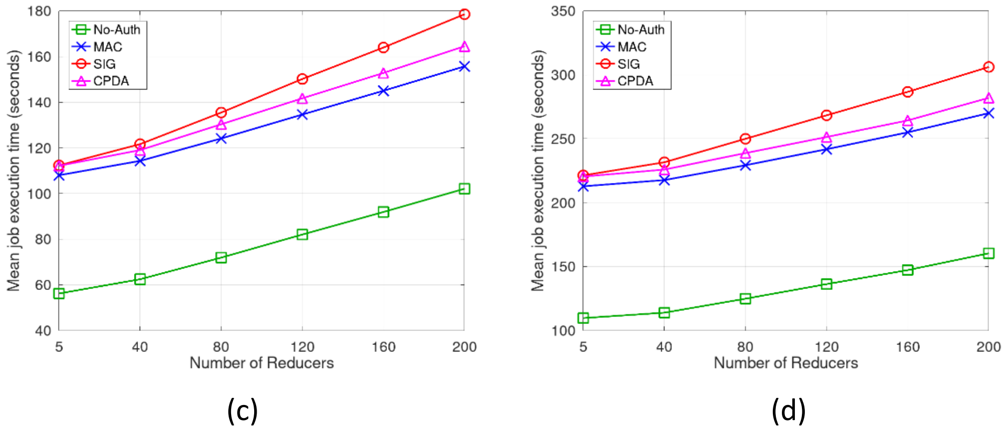
6.9.4.3 Exp3: Performance of Data Authentication Services

The times taken to execute jobs (job execution times) under four different conditions, i.e., without any data authentication (No-Auth) and with each of the three data authentication services (MAC based, signature (SIG) based and CPDA) are investigated against varying numbers of Workers (Mappers and Reducers). The results are depicted in Figure 6.16.



(a)

(b)



(c)

(d)

Figure 6.16: The comparisons of the execution times of MR jobs with and without data authentication.
(a) $M = 5$. (b) $M = 50$. (c) $M = 100$. (d) $M = 200$.

Based on the results shown in the figures, we can make the following observations. Firstly, as the numbers of Mappers and Reducers increase, the job execution times in all the four cases (without data authentication and with each of the three services applied) increase. For example, when 5 Mappers are used, with 5 Reducers, the job execution times are approximately 9 seconds (No-Auth), 12 seconds (MAC), 13 seconds (SIG), and 14 seconds (CPDA), but with 200 Reducers, the corresponding values are 56 seconds, 60 seconds, 65 seconds, and 64 seconds. The rates of increase for the four cases are respectively, 6.2 (No-Auth), 5 (MAC), 5 (SIG), and 4.5 (CPDA). The reason for the increase in job execution times as the number of Workers (one or both of the Mappers and Reducers) increases is that when the number of these Workers increases, the number of objects to be signed and verified also increases, and this introduces additional overhead costs (e.g., process initialisation, memory allocation, and inter-process communication). These overhead costs offset the benefit of task parallelism due to the limited number of tasks that can be executed concurrently.

The second observation is that, among the three data authentication services, the MAC based service adds the smallest amount of delay whereas the signature based service adds the largest. The more the Workers that are used, the smaller the gaps between the CPDA and MAC based service and the larger the gaps between the CPDA and signature based service, which means the bigger the benefit CPDA brings in terms of cutting down execution times. For example, when 5 Mappers and 200 Reducers are used, the differences in job execution times between CPDA and the MAC based service and between CPDA and the signature based service are, respectively, 4.6 and 1 seconds. However, when 200 Mappers and 200 Reducers are used, these values are, respectively, 12 and 24 seconds. This means that, in this setting, CPDA cuts down up to two thirds of the additional overhead cost while still providing the same level of protection as that of the signature based service. This reduction is significant, as, in some application contexts such as security threat analysis or intrusion detection, a minor reduction in a job execution time means sooner production of analysis results, which, in turn, means an earlier intrusion detection and a faster reaction (or a mitigating response).

The above experimental results indicate that, at the cost closer to that of the MAC based service, CPDA can provide the same security protections as the signature based service that is stronger in security protection but computationally much more expensive than the MAC based service. The more Workers that are used, the closer they are.

6.10 Chapter Summary

This chapter has presented a novel approach, a communication pattern based approach, to data authenticity and non-repudiation of origin protections for MR based CBDC-MPC and a novel data authentication framework, the CPDA framework, that implements the approach. The critical analysis on the related work indicates that there are rooms for improvements in existing solutions with regard to efficient provisioning of the protections in the context. None of the solutions are designed to provide protection to every data object generated and processed during a job execution. Some solutions (i.e., symmetric-key based without a form of asymmetry) are efficient but do not provide non-repudiation of origin protection. Some solutions (e.g., secret-share based and task-replication based) introduce a high level of overhead cost which may hinder the performance of the underlying services. Via literature research, we discovered that the overhead cost introduced by expensive asymmetric-key operations can be reduced by using a signature amortisation technique, and this inspired the design of CPDA. The design of CPDA makes use of two main ideas. The first is ACA in which aggregation is applied to AuthData generation and verification as well as communications among MR components. The aggregation of AuthData generation and verification reduces number of expensive cryptographic operations to be performed by data processing components, thus reducing computational overhead cost. The aggregation of communications reduces the number of communications among the components by using a third-party aggregator, reducing network traffics thus communication overhead cost. The second idea is HYSC which makes a hybrid use of multiple cryptographic primitives. The computationally less expensive scheme (i.e., the MAC scheme) is applied to protect JobData that are pairwise transmitted and the computationally more expensive scheme (i.e., the digital signature scheme) is applied to protect JobData that are used by multiple data consumers. This hybrid use of cryptographic scheme can ensure the accountability of data producers while minimising the computational overhead cost. The security analysis and performance evaluation have been conducted on CPDA and the most related object-based solutions. The results show that CPDA can provide the same level of protections as the MAC based scheme with TTP and the signature based scheme. In addition, the overhead cost introduced by CPDA is lower than that of the signature based scheme (a maximum reduction of 66%) and the overhead cost grows closer to that of the MAC based scheme when CPDA is applied to a larger scale of MR service. CPDA is also suited to other applications, such as wireless sensors networks and IoT applications, that can be characterised by some or all of the O2M, M2M, and M2O patterns. The approach to data authentication and the design and evaluations of CPDA presented in this chapter is the third contribution (NC3) of this research work. The contribution answers the research questions (Q3) and (Q4).

The next chapter presents a working example to demonstrate how a cyberthreat analysis job is carried out by using the MR framework in the CBDC-MPC context, and how the MDA framework is applied to the job execution to provide authentication protections.

Chapter 7
The Detailed Operational Steps for the Running Example

7.1 Chapter Introduction

This chapter provides detailed operational steps for the running example to illustrate how an example MR based cyberthreat analysis job is executed without our MDA framework and how the job is executed when the MDA framework is applied. The chapter explains all the operational steps for the job execution. This complements the example job execution flow given in Section 4.6. It then explains how different components of the MDA framework are used to achieve both entity and data authentication protections at the finest granularity (i.e., at every interaction and for every JobData object) throughout the entire cycle of the job execution. It also highlights how credentials used for the authentication are established on the MR components and how AuthData are transmitted during the job execution. This complements the descriptions of the running example given in Sections 5.6 and 6.6.

In detail, Section 7.2 gives a step-by-step description of the job execution flow for the running example. Section 7.3 explains in detail the operations of the MDA framework when being applied to the job execution. Section 7.4 concludes the chapter.

7.2 Job Execution Flow

This section explains the operational steps for the execution of the job in the running example described in Section 4.6. The operational steps are based on the operational steps (GM-1) through to (GM-29) explained in Section 4.3.4. In the following, the steps (EF-1) through to (EF-7) are captured in Figure 4.8(a), (EF-8) through to (EF-19) in Figure 4.8(b), (EF-20) through to (EF-23) in Figure 4.8(c), and (EF-24) through to (EF-29) in Figure 4.8(d).

(EF-1) $User^1$, as the JobSubmitter, sends a request for security log files, $File_2$ and $File_3$, for the job to $User^2$ and $User^3$ via existing secure communication channels. The request contains a reference ID with a value of 0001 for the job. $User^2$ and $User^3$ receive and approve the request and send a confirmation back to $User^1$.

(EF-2) $User^1$, $User^2$, and $User^3$, respectively, use $ClientApp^1$, $ClientApp^2$, and $ClientApp^3$ to send a request with the reference ID for a new job ID and a path to write the security log files and job configuration files (e.g., the size of each InputSplit and the number of Reducers) to $ResourceManager^1$, $ResourceManager^2$, and $ResourceManager^3$, respectively.

(EF-3) $ResourceManager^1$, $ResourceManager^2$, and $ResourceManager^3$, respectively, receive and accept the request sent from $ClientApp^1$, $ClientApp^2$, and $ClientApp^3$. Each of the ResourceManagers replies the respective ClientApp with a job ID with a value of 0001 and a path with a value of "/Job/0001".

(EF-4) $ClientApp^1$, $ClientApp^2$, and $ClientApp^3$, respectively, receive the reply from $ResourceManager^1$, $ResourceManager^2$, and $ResourceManager^3$.

$ClientApp^1$, $ClientApp^2$, and $ClientApp^3$, respectively, write $File_1$, $File_2$, and $File^3$ and the job configuration files to the path "/Job/0001" on the respective DFS clusters. The writing process involves three further steps.

a. $ClientApp^1$, $ClientApp^2$, and $ClientApp^3$, respectively, send a request for writing the data (the security log files and the job configuration files) to "/Job/0001" to $NameManager^1$, $NameManager^2$, and $NameManager^3$.

b. $NameManager^1$ receives and accepts the request. $NameManager^1$ determines that the data can be stored on $DataStore_1^1$. Therefore, $NameManager^1$ replies $ClientApp^1$ to write the data to $DataStore_1^1$. Similarly, $NameManager^2$ and $NameManager^3$, receives the requests and, respectively, reply $ClientApp^2$ and $ClientApp^3$ to write the data to $DataStore_1^2$ and $DataStore_1^3$.

c. $ClientApp^1$, $ClientApp^2$, and $ClientApp^3$ receive the reply from the respective NameManager and, respectively, contact and write the data to $DataStore_1^1$, $DataStore_1^2$, and $DataStore_1^3$. $File_1$, $File_2$, and $File_3$ are, respectively, used as $InputSplit_{1,1}$, $InputSplit_{2,2}$, and $InputSplit_{3,3}$.

(EF-5) Once each of the ClientApps finishes writing the data, it notifies the respective ResourceManager of the writing completion.

(EF-6) $ResourceManager^1$ sends a request for inquiring the status of data writing to $ResourceManager^2$ and $ResourceManager^3$.

(EF-7) When the writing of data by $ClientApp^2$ and $ClientApp^3$ completes, $ResourceManager^2$ and $ResourceManager^3$, respectively, reply $ResourceManager^1$ of the completion of writing.

(EF-8) $ResourceManager^1$ sends a request for launching $JobManager$ for the job to $WorkerManager_1^1$ (managing $WorkerNode_1^1$). Along the request, $ResourceManager^1$ sends a list of NameManagers ($NameManager^1$, $NameManager^2$, and $NameManager^3$) for J$obManager$ to contact and the path "/Job/0001" to read the job configuration files.

(EF-9) $WorkerManager_1^1$ starts $JobManager$ and pass the data to $JobManager$.

(EF-10) Once $JobManager$ is launched, $ResourceManager^1$ notifies $ClientApp^1$ to contact $JobManager$.

(EF-11) $ClientApp^1$ contacts $JobManager$. Throughout the job execution, $ClientApp^1$ will periodically contacts $JobManager$ to inquire the status of job.

(EF-12) $JobManager$ reads the job configuration files from the DFS clusters. The reading process involves three further steps.

a. $JobManager$ sends a request for reading the job configuration files from the path "/Job/0001" to $NameManager^1$, $NameManager^2$, and $NameManager^3$.

b. $NameManager^1$, $NameManager^2$, and $NameManager^3$ receive and accept the request and, respectively, reply $JobManager$ to read the job configuration files from $DataStore_1^1$, $DataStore_1^2$, and $DataStore_1^3$.

c. $JobManager$ receives the reply from the NameManagers and read the job configuration files from the respective DataStores.

(EF-13) $JobManager$ decides the number of Workers (Mappers and Reducers) needed for the job based on the configuration files. In this case, 3 Mappers ($Mapper_1$, $Mapper_2$, and $Mapper_3$) and 3 Reducers ($Reducer_1$, $Reducer_2$, and $Reducer_3$) will be used. $JobManager$ then sends a request for worker allocation (3 Mappers and 3 Reducers) to $ResourceManager^1$.

(EF-14) $ResourceManager^1$ receives the request. As $WorkerNode_2^1$ can host only 1 Mapper and 1 Reducer, $ResourceManager^1$ sends a request for worker allocation (2 Mappers and 2 Reducers) to $ResourceManager^2$ and $ResourceManager^3$.

(EF-15) $ResourceManager^2$ and $ResourceManager^3$ receive the request. After they examine their available resources, $ResourceManager^2$ and $ResourceManager^3$, respectively, reply $ResourceManager^1$ that $WorkerNode_1^2$ and $WorkerNode_1^3$ each can host 1 Mapper and 1 Reducer.

(EF-16) $ResourceManager^1$ receives the reply from $ResourceManager^2$ and $ResourceManager^3$. $ResourceManager^1$ replies $JobManager$ that each of $WorkerNode_2^1$, $WorkerNode_1^2$, and $WorkerNode_1^3$ can host 1 Mapper and 1 Reducer, a total of 3 Mappers and 3 Reducers.

(EF-17) $JobManager$ sends a request for launching 1 Mapper and 1 Reducer to each of $WorkerNode_2^1$, $WorkerNode_1^2$, and $WorkerNode_1^3$. The request is handled by the WorkerManager of each WorkerNode, i.e., $WorkerManager_2^1$, $WorkerManager_1^2$, and $WorkerManager_1^3$.

(EF-18) Each of $WorkerManager_2^1$, $WorkerManager_1^2$, and $WorkerManager_1^3$ starts 1 Mapper and 1 Reducer on its WorkerNode, i.e., $Mapper_1$ and $Reducer_1$ on $WorkerNode_2^1$, $Mapper_2$ and $Reducer_2$ on $WorkerNode_1^2$, and $Mapper_3$ and $Reducer_3$ on $WorkerNode_1^3$.

(EF-19) Each of $Mapper_1$, $Mapper_2$, $Mapper_3$, $Reducer_1$, $Reducer_2$, and $Reducer_3$ contacts $JobManager$. During the executions of map and reduce tasks, the Mappers and Reducers will periodically report the progress of the task execution to $JobManager$.

(EF-20) $JobManager$ issues a command to $Mapper_1$, $Mapper_2$, and $Mapper_3$ to start map tasks and to specify the locations of the assigned InputSplits, i.e., $InputSplit_{1,1}$ for $Mapper_1$ is in DFS^1, $InputSplit_{2,2}$ for $Mapper_2$ is in DFS^2, and $InputSplit_{3,3}$ for $Mapper_3$ is in DFS^3.

(EF-21) $Mapper_1$, $Mapper_2$, and $Mapper_3$ receive the command and read the assigned InputSplits from the respective DFS clusters. The reading processing involves three further steps.

a. $Mapper_1$ sends a request for reading $InputSplit_{1,1}$ from the path "/Job/0001" to $NameManager^1$. Similarly, $Mapper_2$ and $Mapper_3$ each send a request to $NameManager^2$ and $NameManager^3$, respectively.

b. $NameManager^1$ receives and accepts the request. It replies $Mapper_1$ to read $InputSplit_{1,1}$ from $DataStore_1^1$. Similarly, $NameManager^2$ replies $Mapper_2$ to read $InputSplit_{2,2}$ from $DataStore_1^2$ and $NameManager^3$ replies $Mapper_3$ to read $InputSplit_{3,3}$ from $DataStore_1^3$.

c. $Mapper_1$ receives the reply from $NameManager^1$. It then contacts and reads $InputSplit_{1,1}$ from $DataStore_1^1$. Similarly, $Mapper_2$ reads $InputSplit_{2,2}$ from $DataStore_1^2$ and $Mapper_3$ reads $InputSplit_{3,3}$ from $DataStore_1^3$.

(EF-22) $Mapper_1$, $Mapper_2$, and $Mapper_3$ perform the map tasks on the assigned InputSplits. Each of the Mappers scans the entries in the respective InputSplit. If an entry shows that the source IP address is one of the compromised machines and the destination port number is 22, the Mapper will output a key-value pair of destination and source IP addresses, expressed as $\{Destination\ IP, Source\ IP\}$. For $Mapper_1$, three key-value pairs are output, {10.2.0.201, 10.1.0.101}, {10.3.0.201, 10.1.0.101}, and {10.3.0.202, 10.1.0.102}. The output will be partitioned into PartitionSegments[12].

The key-value pairs will be partitioned based on the key (the destination IP address), i.e., the IP address block 10.1.0.0/16 will be in the first PartitionSegment (for $Reducer_1$), the IP address block 10.2.0.0/16 in the second PartitionSegment (for $Reducer_2$), and the IP address block 10.3.0.0/16 in the third PartitionSegment (for $Reducer_3$). As a result, $Mapper_1$ produces an IntermediateResult ($IntermediateResult_1$) containing 2 PartitionSegments, $PartitionSegment_{1,2}$ containing {10.2.0.201, 10.1.0.101} and $PartitionSegment_{1,3}$ containing {10.3.0.201, 10.1.0.101; 10.3.0.202, 10.1.0.102}. Using the same method, $Mapper_2$ and $Mapper_3$, respectively, produce $IntermediateResult_2$ and $IntermediateResult_3$. $IntermediateResult_2$ contains 2 PartitionSegments, $PartitionSegment_{2,1}$ containing {10.1.0.201, 10.2.0.101} and $PartitionSegment_{2,2}$ containing {10.2.0.203, 10.2.0.101}. $IntermediateResult_3$ contains 2 PartitionSegments, $PartitionSegment_{3,2}$ containing {10.2.0.201, 10.3.0.101} and $PartitionSegment_{3,3}$ containing {10.3.0.202, 10.3.0.101}. The IntermediateResults are stored in the local storage of the respective WorkerNodes. The contents of the IntermediateResults are summarised in Table 7.1.

Table 7.1: Output produced by the Mappers.

Mapper	Output file	$PartitionSegment_{i,1}$ (for $Reducer_1$)	$PartitionSegment_{i,2}$ (for $Reducer_2$)	$PartitionSegment_{i,3}$ (for $Reducer_3$)

$Mapper_1$	$IntermediateResult_1$	-	10.2.0.201, 10.1.0.101;	10.3.0.201, 10.1.0.101;
				10.3.0.202, 10.1.0.102;

$Mapper_2$	$IntermediateResult_2$	10.1.0.201, 10.2.0.101;	10.2.0.203, 10.2.0.101;	-
$Mapper_3$	$IntermediateResult_3$	-	10.2.0.201, 10.3.0.101;	10.3.0.202, 10.3.0.101;

(EF-23) Each of $Mapper_1$, $Mapper_2$, and $Mapper_3$ notifies $JobManager$ when its map task finishes and its IntermediateResult is ready.

(EF-24) Once all the map tasks complete, $JobManager$ issues a command to $Reducer_1$, $Reducer_2$, and $Reducer_3$ to start reduce tasks and specify the locations of the assigned PartitionSegments. The locations for the PartitionSegments are as follows: $PartitionSegment_{2,1}$ for $Reducer_1$ is stored in $WorkerNode_1^2$, $PartitionSegment_{1,2}$, $PartitionSegment_{2,2}$, and $PartitionSegment_{3,2}$ for

[12] The implementation of the partition function is MR implementation dependent. In this work, it is assumed that the partition is carried out by the MR service after each of the Mappers carries out its map task.

$Reducer_2$ are stored in $WorkerNode_2^1$, $WorkerNode_1^2$, and $WorkerNode_1^3$, respectively, and $PartitionSegment_{1,3}$ and $PartitionSegment_{3,3}$ for $Reducer_3$ are stored in $WorkerNode_2^1$ and $WorkerNode_1^3$, respectively.

(EF-25) $Reducer_1$, $Reducer_2$, and $Reducer_3$ receive the command and retrieve the assigned PartitionSegments.

a. If the PartitionSegments are stored in the local storage of its WorkerNode, the Reducer can retrieve the PartitionSegments locally and no inter-node data transfer is required. In this example, $Reducer_2$ and $Reducer_3$ can, respectively, retrieve $PartitionSegment_{2,2}$ and $PartitionSegment_{3,3}$ from the local storage of their nodes.

b. If the PartitionSegments are stored on the other WorkerNodes, the Reducer sends requests for the PartitionSegments to the WorkerManagers of the respective WorkerNodes. Once the requests are received and approved, the Reducers can retrieve the requested PartitionSegments. In this example, $Reducer_1$ has to send a request for a PartitionSegment to $WorkerManager_1^2$, $Reducer_2$ to $WorkerManager_2^1$ and $WorkerManager_1^3$, and $Reducer_3$ to $WorkerManager_2^1$.

The assigned PartitionSegments will be processed (merged) by the merge function[13]. This is done by grouping the values (the source IP addresses) corresponding to the same key (the destination IP address) together. Hence, the input of $Reducer_1$, $Reducer_2$, and $Reducer_3$ are, respectively, {10.1.0.201, [10.2.0.101]}, {10.2.0.201, [10.1.0.101, 10.3.0.101]; 10.2.0.203, [10.2.0.101]}, and {10.3.0.201, [10.1.0.101]; 10.3.0.202, [10.1.0.102, 10.3.0.101]}. This is summarised in Table 7.2.

Table 7.2: Input used by the Reducers.

Reducer	Input (merged PartitionSegments)
$Reducer_1$	10.1.0.201, [10.2.0.101];

$Reducer_2$	10.2.0.201, [10.1.0.101, 10.3.0.101];
	10.2.0.203, [10.2.0.101];

$Reducer_3$	10.3.0.201, [10.1.0.101];
	10.3.0.202, [10.1.0.102, 10.3.0.101];

(EF-26) $Reducer_1$, $Reducer_2$, and $Reducer_3$ perform the reduce tasks on the respective merged PartitionSegments. $Reducer_1$ processes its input key by key. For each key, it counts the number of source IP addresses (i.e., how many times the machine had been connected to by the compromised machines) and outputs a key-value pair of destination IP address and connection count, expressed as $\{Destination\,IP$, $Count\}$. Hence, $Reducer_1$ produces an output file, $FinalResult_{1,1}$, containing {10.1.0.201, 1}. Using the same method, $Reducer_2$ produces $FinalResult_{2,1}$ containing {10.2.0.201, 2; 10.2.0.203, 1} and $Reducer_3$ produces $FinalResult_{3,1}$ containing {10.3.0.201, 1; 10.3.0.202, 2}. This is summarised in Table 7.3.

[13] The implementation of the merge function is MR implementation dependent. In this work, it is assumed that the merge function is carried out by the MR service before each of the Reducers carries out its reduce task.

Table 7.3: Output produced by the Reducers.

Reducer	Output file	Content
$Reducer_1$	$FinalResult_{1,1}$	10.1.0.201, 1;

$Reducer_2$	$FinalResult_{2,1}$	10.2.0.201, 2;
		10.2.0.203, 1;

$Reducer_3$	$FinalResult_{3,1}$	10.3.0.201, 1;
		10.3.0.202, 2;

After the reduce tasks finish, $Reducer_1$, $Reducer_2$, and $Reducer_3$ writes $FinalResult_{1,1}$, $FinalResult_{2,1}$, and $FinalResult_{3,1}$ to DFS^1. The writing process involves three further steps.

a. $Reducer_1$, $Reducer_2$, and $Reducer_3$, respectively, send a request for writing the output files ($FinalResult_{1,1}$, $FinalResult_{2,1}$, and $FinalResult_{3,1}$) to "/Job/0001" to $NameManager^1$.

b. $NameManager^1$ receives and accepts the request. It replies $Reducer_1$, $Reducer_2$, and $Reducer_3$ to write the FinalResults to $DataStore_1^1$.

c. $Reducer_1$, $Reducer_2$, and $Reducer_3$ receive the reply from $NameManager^1$. The Reducers then, respectively, contact and write the data to $DataStore_1^1$.

(EF-27) Each of $Reducer_1$, $Reducer_2$, and $Reducer_3$ notifies $JobManager$ when its reduce task finishes and the FinalResult is written to DFS^1.

(EF-28) Once $JobManager$ receives the notifications from all the Reducers, it notifies $ClientApp^1$ that FinalResults are ready for retrieval.

(EF-29) $ClientApp^1$ receives the notification from $JobManager$ and reads the FinalResults stored on DFS^1 for $User^1$. The reading process involves three further steps.

a. $ClientApp^1$ sends a request for reading all the FinalResults of the job ($FinalResult_{1,1}$, $FinalResult_{2,1}$, and $FinalResult_{3,1}$) from the path "/Job/0001" to $NameManager^1$.

b. $NameManager^1$ receives and accepts the request. It replies $ClientApp^1$ to read the requested data from $DataStore_1^1$.

c. $ClientApp^1$ receives the reply from $NameManager^1$. It contacts $DataStore_1^1$ and reads $FinalResult_{1,1}$, $FinalResult_{2,1}$, and $FinalResult_{3,1}$. The execution of the job is complete successfully.

It is worth mentioning that the MR framework can be used to execute data analysis jobs with various levels of sophistication; this is dependent on the problem to be analysed, the MR implementations used, and the software codes of the Mappers and Reducers. One of the more sophisticated data analysis jobs than our working example described above is a job used to trace back to the origin of the attacks mounted on a compromised machine. The job is executed in multiple rounds of executions, each round consists of a separate pair of map and reduce phases. The output from an earlier round is used as the input of the next round. For example, the first round is used to identify machines that have ever connected to any of the compromised machines. The subsequent rounds are used to identify machines that have made connections to the machines identified in the previous rounds. The execution continues until there are no more machines to trace back.

The next section explains in detail how the MDA framework is applied to the working example described above to protect the authenticity of JobData used, processed, and generated by the MR services during the job execution.

7.3 MDA in Action

This section explains in detail how the MDA framework provides entity and data authentication protections to the job in the running example. It complements the demonstration given in Section 5.6 and Section 6.6. Based on the operational steps (EF-1) through to (EF-29) described earlier, this section explains the operational steps when MDA is applied to the job execution, showing how different components of the MDA framework are used at different stages of the job execution to protect every JobData object at every interaction.

In each of the operational steps, we describe which MDA components are applied, what credentials are used for each authentication instance, and what credentials are distributed and established for subsequent authentication. We use the same assumptions as those given in Section 5.4.1 and Section 6.4.1. The certification and verification processes of the public keys are omitted. The notations used in describing the keys and entities involved in the authentication process are shown in Table 5.2 (Section 5.4.2) and Table 6.2 (Section 6.4.2). Keys that are established on components prior to the execution of the job are summarised in Table 7.4. These keys are established by using existing mechanisms, such as MR service-level authentication services.

Table 7.4: Credentials established prior to the execution of the job.

Keys	Components involved	When the keys are being established
ok^1, pmk_{c^1,rm^1}	c^1 and rm^1	c^1 is registered to rm^1
ok^2, pmk_{c^2,rm^2}	c^2 and rm^2	c^2 is registered to rm^2
ok^3, pmk_{c^3,rm^3}	c^3 and rm^3	c^3 is registered to rm^3
pik^1, pck^1 $pmk_{wm_1^1,rm^1}, slk_{wm_1^1,rm^1}$	wm_1^1 and rm^1	wm_1^1 is registered to rm^1
pik^1, pck^1 $pmk_{wm_2^1,rm^1}, slk_{wm_2^1,rm^1}$	wm_2^1 and rm^1	wm_2^1 is registered to rm^1
dfk^1, slk_{nm^1,rm^1}	nm^1 and rm^1	nm^1 is registered to rm^1
dfk^2, slk_{nm^2,rm^2}	nm^2 and rm^2	nm^2 is registered to rm^2
dfk^3, slk_{nm^3,rm^3}	nm^3 and rm^3	nm^3 is registered to rm^3
$dfk^1, slk_{ds_1^1,nm^1}$	ds_1^1 and nm^1	ds_1^1 is registered to nm^1
$dfk^2, slk_{ds_1^2,nm^2}$	ds_1^2 and nm^2	ds_1^2 is registered to nm^2
$dfk^3, slk_{ds_1^3,nm^3}$	ds_1^3 and nm^3	ds_1^3 is registered to nm^3
pmk_{rm^1,rm^2}	rm^1 and rm^2	Collaboration is established
pmk_{rm^1,rm^3}	rm^1 and rm^3	Collaboration is established
pmk_{rm^2,rm^3}	rm^2 and rm^3	Collaboration is established
sk_{c^1}, pk_{c^1}	c^1	Before job submission
sk_{c^2}, pk_{c^2}	c^2	Before job submission
sk_{c^3}, pk_{c^3}	c^3	Before job submission

(MF-1) Prior to sending a request for security log files ($File_2$ and $File_3$) as described in (EF-1), $User^1$ authenticates to $User^2$ and $User^3$ by using existing authentication services. After the users are successfully authenticated, $User^1$ can then proceed to sending the request to $User^2$ and $User^3$. A JobDomain key jk is generated and distributed to all the users.

(MF-2) $User^1$, $User^2$, and $User^3$, respectively, authenticate to $ClientApp^1$, $ClientApp^2$, and $ClientApp^3$ by using an existing authentication service (e.g., an operating system-level authentication service). Before $User^1$, $User^2$, and $User^3$ (through their ClientApps) can send a request to start a new job and to write the input data (the security log files) and job configuration files to $ResourceManager^1$, $ResourceManager^2$, and $ResourceManager^3$ as described in (EF-2), ClientApps and the respective ResourceManagers should be mutually authenticated. The authentication of each pair of $ClientApp^1$ and $ResourceManager^1$, $ClientApp^2$ and $ResourceManager^2$, and $ClientApp^3$ and $ResourceManager^3$ is done by using the GP2A protocol. The operational steps for the GP2A protocol are given in Section 5.5.4.1. The keys used for the entity authentication are the respective OrgDomain key and the primary key shared between a ClientApp and the respective ResourceManager, i.e., ok^1 and pmk_{c^1,rm^1} are used by $ClientApp^1$, ok^2 and pmk_{c^2,rm^2} by $ClientApp^2$, and ok^3 and pmk_{c^3,rm^3} by $ClientApp^3$. Each of the ClientApps sends jk for $ResourceManager^1$ to authenticate to $ResourceManager^2$ and $ResourceManager^3$, a secondary key (sck_{c^1,rm^1} by $ClientApp^1$, sck_{c^2,rm^2} by $ClientApp^2$, and sck_{c^3,rm^3} by $ClientApp^3$) for subsequent authentication between the ClientApp and the respective ResourceManager, and a session key (ssk_{c^1,rm^1} by $ClientApp^1$, ssk_{c^2,rm^2} by $ClientApp^2$, and ssk_{c^3,rm^3} by $ClientApp^3$) for protecting data exchanged during the session.

(MF-3) $ResourceManager^1$, $ResourceManager^2$, and $ResourceManager^3$, respectively, authenticate to $ClientApp^1$, $ClientApp^2$, and $ClientApp^3$ before sending replies as described in (EF-3). As each of the ResourceManagers has been interacted with the respective ClientApp, the authentication is carried out by using the SOA protocol with the secondary key (sck_{c^1,rm^1} by $ResourceManager^1$, sck_{c^2,rm^2} by $ResourceManager^2$, and sck_{c^3,rm^3} by $ResourceManager^3$) established in (MF-2). The operational steps of the SOA protocol are given in Section 5.5.4.3. Each of the ResourceManagers prepares a DFS-C key, a primary key, and a ticket (containing the encrypted primary key) for the respective ClientApp to authenticate to the corresponding NameManager. $ResourceManager^1$ also gives a DPS-C key pck^1 to $ClientApp^1$ for authentication to $JobManager$. Hence, $ResourceManager^1$ sends pck^1, dfk^1, pmk_{c^1,nm^1}, $tkt_{c^1,nm^1}^{rm^1}$, and ssk_{rm^1,c^1} to $ClientApp^1$; $ResourceManager^2$ sends dfk^2, pmk_{c^2,nm^2}, $tkt_{c^2,nm^2}^{rm^2}$, and ssk_{rm^2,c^2} to $ClientApp^2$; and $ResourceManager^3$ sends dfk^3, pmk_{c^3,nm^3}, $tkt_{c^3,nm^3}^{rm^3}$, and ssk_{rm^3,c^3} to $ClientApp^3$.

(MF-4) As described in (EF-4), the writing of security log files and the job configuration files to the respective DFS cluster involves three interactions.

a. Each of the ClientApps authenticates to the respective NameManager by using the GE2A protocol. The operational steps of the GE2A protocol are given in Section 5.5.4.2. For each authentication instance, the DFS-C key and the primary key established in (MF-3) are used, i.e., dfk^1 and pmk_{c^1,nm^1} by $ClientApp^1$, dfk^2 and pmk_{c^2,nm^2} by $ClientApp^2$, and dfk^3 and pmk_{c^3,nm^3} by $ClientApp^3$. It is worth noting that the primary key for authentication between a ClientApp and a NameManager is contained in the ticket sent from the ClientApp to the NameManager in the first protocol message (the CH message). During the authentication, $ClientApp^1$ sends sck_{c^1,nm^1} and ssk_{c^1,nm^1} to $NameManager^1$, $ClientApp^2$ sends sck_{c^2,nm^2} and ssk_{c^2,nm^2} to $NameManager^2$, and $ClientApp^3$ sends sck_{c^3,nm^3} and ssk_{c^3,nm^3} to $NameManager^3$.

b. Before each of the NameManagers sends a reply back to the corresponding ClientApp, it authenticates to the respective ClientApp by using the SOA protocol with the secondary key established in (MF-4(a)), i.e., sck_{c^1,nm^1} by $NameManager^1$, sck_{c^2,nm^2} by $NameManager^2$, and sck_{c^3,nm^3} by $NameManager^3$. Each of the NameManagers also generates a primary key and a ticket for the respective ClientApp to authenticate to the corresponding DataStore. Hence, $NameManager^1$ sends pmk_{c^1,ds_1^1}, $tkt_{c^1,ds_1^1}^{nm^1}$, and ssk_{nm^1,c^1} to $ClientApp^1$, $NameManager^2$ sends pmk_{c^2,ds_1^2}, $tkt_{c^2,ds_1^2}^{nm^2}$, and ssk_{nm^2,c^2} to $ClientApp^2$, and $NameManager^3$ sends pmk_{c^3,ds_1^3}, $tkt_{c^3,ds_1^3}^{nm^3}$, and ssk_{nm^3,c^3} to $ClientApp^3$.

c. Each of the ClientApps authenticates to the respective DataStore. The entity authentication for each pair of ClientApp and the respective DataStore is carried out by using the GE2A protocol with the DFS-C key established in (MF-3) and the primary key established in (MF-4(b)). These keys are dfk^1 and pmk_{c^1,ds_1^1} used by $ClientApp^1$, dfk^2 and pmk_{c^2,ds_1^2} by $ClientApp^2$, and dfk^3 and pmk_{c^3,ds_1^3} by $ClientApp^3$. During the authentication, only a session key is transmitted. Hence, $ClientApp^1$, $ClientApp^2$, and $ClientApp^3$, respectively, send ssk_{c^1,ds_1^1}, ssk_{c^2,ds_1^2}, and ssk_{c^3,ds_1^3} to $DataStore_1^1$, $DataStore_1^2$, and $DataStore_1^3$.

After the input data are written to the respective DataStore and divided into InputSplits, each of the ClientApps generates AuthData for the InputSplit it provides, ensuring the authenticity of the InputSplit. This is done by using the ISAuthData-Generation algorithm (explained in Section 6.5.4.1) with the private key (an asymmetric key) of the respective ClientApp. These private keys are sk_{c^1} for $ClientApp^1$, sk_{c^2} for $ClientApp^2$, and sk_{c^3} for $ClientApp^3$. The algorithm generates AuthData tokens for the respective InputSplits, i.e., $\sigma_{rh_{c^1}}$ and SA_{c^1} for $InputSplit_{1,1}$ by $ClientApp^1$, $\sigma_{rh_{c^2}}$ and SA_{c^2} for $InputSplit_{2,2}$ by $ClientApp^2$, and $\sigma_{rh_{c^3}}$ and SA_{c^3} for $InputSplit_{3,3}$ by $ClientApp^3$.

(MF-5) Each of the ClientApps authenticates to the respective ResourceManager before notifying of the completion of data writing as described in (EF-5). The authentication process and the keys used are as described in (MF-3). The difference is that a new session key for the session and the public key of the ClientApp (for InputSplit verification by Mappers) are transmitted from the ClientApp to the respective ResourceManager. In other words, $ClientApp^1$, $ClientApp^2$, and $ClientApp^3$, respectively, send ssk_{c^1,rm^1} and pk_{c^1}, ssk_{c^2,rm^2} and pk_{c^2}, and ssk_{c^3,rm^3} and pk_{c^3} to $ResourceManager^1$, $ResourceManager^2$, and $ResourceManager^3$.

(MF-6) $ResourceManager^1$ authenticates to $ResourceManger^2$ and $ResourceManager^3$ before sending a request for inquiring status of data writing as described in (EF-6). The entity authentication is done by using the GP2A protocol with the JobDomain key jk established in (MF-2) and the primary key established prior to the job execution. $ResourceManager^1$ uses jk and pmk_{rm^1,rm^2} to authenticate to $ResourceManager^2$ and jk and pmk_{rm^1,rm^3} to $ResourceManager^3$. It then sends secondary keys and session keys to the other ResourceManagers, i.e., sck_{rm^1,rm^2} and ssk_{rm^1,rm^2} to $ResourceManager^2$ and sck_{rm^1,rm^3} and ssk_{rm^1,rm^3} to $ResourceManager^3$.

(MF-7) When the input data ($File_2$ and $File_3$) and job configuration files are written to the DFS clusters, $ResourceManager^2$ and $ResourceManager^3$ authenticate to $ResourceManager^1$ before notifying of the completion of data writing as described in (EF-7). The entity authentication is done by using the SOA protocol with the secondary key established in (MF-6). The keys used are sck_{rm^1,rm^2} by $ResourceManager^2$ and sck_{rm^1,rm^3} by $ResourceManager^3$. Each of $ResourceManager^2$ and $ResourceManager^3$ prepares the DFS-C key, a new primary key, and a respective ticket for $JobManager$ to authenticate to $NameManager^2$ and $NameManager^3$, respectively. $ResourceManager^2$ and $ResourceManager^3$, respectively, send dfk^2, pmk_{jm,nm^2}, $tkt^{rm2}_{jm,nm2}$, and ssk_{rm^2,rm^1} and dfk^3, pmk_{jm,nm^3}, $tkt^{rm^3}_{jm,nm^3}$, and ssk_{rm^3,rm^1} to $ResourceManager^1$.

(MF-8) $ResourceManager^1$ authenticates to $WorkerManager^1_1$ before sending a request for launching $JobManager$ as described in (EF-8). The entity authentication is carried out by using the GP2A protocol with the DPS-I key pik^1 and the primary key $pmk_{wm^1_1,rm^1}$ established prior to the job execution. $ResourceManager^1$ generates a new primary key $pmk_{rm^1,jm}$ for $JobManager$ to authenticate to $ResourceManager^1$, prepares DFS-C keys (dfk^1, dfk^2, and dfk^3), primary keys (pmk_{jm,nm^1}, pmk_{jm,nm^2}, and pmk_{jm,nm^3}), and tickets ($tkt^{rm^1}_{jm,nm^1}$, $tkt^{rm^2}_{jm,nm^2}$, and $tkt^{rm^3}_{jm,nm^3}$) for $JobManager$ to authenticate to the NameManagers. The keys and tickets are obtained in (MF-7). $ResourceManager^1$ also generates a session key ssk_{rm^1,wm^1_1} for this session. All the keys and tickets are transmitted from $ResourceManager^1$ to $WorkerManager^1_1$.

(MF-9) $WorkerManager_1^1$ starts $JobManager$ as described in (EF-9). The entity authentication is done locally using an existing method. It also passes the keys and tickets obtained in (MF-8) to $JobManager$. $JobManager$ generates a pair of a private key sk_{jm} and a public key pk_{jm}. The keys are, respectively, used for signing and verifying aggregated AuthData tokens for PartitionSegments and FinalResults.

(MF-10) $ResourceManager^1$ authenticates to $ClientApp^1$ to notify that $JobManager$ has been launched as described in (EF-10). The entity authentication is done by using the SOA protocol with the secondary key sck_{c^1,rm^1} established in (MF-2). $ResourceManager^1$ generates a new primary key $pmk_{c^1,jm}$ and a ticket $tkt_{c^1,jm}^{rm^1}$ for $ClientApp^1$ to authenticate to $JobManager$. $ResourceManager^1$ sends $pmk_{c^1,jm}$, $tkt_{c^1,jm}^{rm^1}$, and a new session key ssk_{rm^1,c^1} to $ClientApp^1$.

(MF-11) In addition to the operational step (EF-11), $ClientApp^1$ sends a request for ISAuthData tokens (for verifying the authenticity of $InputSplit_{2,2}$ and $InputSplit_{3,3}$) to $ClientApp^2$ and $ClientApp^3$. $ClientApp^2$ and $ClientApp^3$ accept the requests and reply $ClientApp^1$ with $\sigma_{rh_{c^2}}, SA_{c^2}$ and $\sigma_{rh_{c^3}}, SA_{c^3}$. These exchanges are done through secure communication channels.

Next, $ClientApp^1$ authenticates to $JobManager$ before inquiring the status of the job as described in (EF-11). The entity authentication is carried out by using the GE2A protocol with the DPS-C key pck^1 established in (MF-3) and the primary key $pmk_{c^1,jm}$ established in (MF-10). $ClientApp^1$ sends a secondary key $sck_{c^1,jm}$ and a session key $ssk_{c^1,jm}$ to $JobManager$. Lastly, $ClientApp^1$ sends all the ISAuthData tokens (including $\sigma_{rh_{c^1}}, SA_{c^1}$) to $JobManager$ by using the ISAuthData-Delivery protocol (explained in Section 6.5.4.3).

(MF-12) Before $JobManager$ can read the job configuration files from all the DFS clusters as described in (EF-12), there are three interactions, hence, three entity authentication instances. The authentication process is similar to that of (MF-4)).

a. $JobManager$ authenticates to $NameManger^1$, $NameManager^2$, and $NameManager^3$ before sending a request for reading the job configuration files. The authentication is done by using the GE2A protocol with the DFS-C key and the primary key established in (MF-9). $JobManager$ uses dfk^1 and pmk_{jm,nm^1} to authenticate to $NameManager^1$, dfk^2 and pmk_{jm,nm^2} to $NameManager^2$, and dfk^3 and pmk_{jm,nm^3} to $NameManager^3$. Unlike (MF-4(a)), in addition to a secondary key and a session key, $JobManager$ also generates a ticket sealing key for each of the NameManagers. The ticket sealing keys are used for the generation and verification of tickets issued by $JobManager$ for Mappers and Reducers to authenticate to the NameManagers. Hence, $JobManager$ sends sck_{jm,nm^1}, $tsk_{nm^1,jm}$, and ssk_{jm,nm^1} to $NameManager^1$, sck_{jm,nm^2}, $tsk_{nm^2,jm}$, and ssk_{jm,nm^2} to $NameManager^2$, and sck_{jm,nm^3}, $tsk_{nm^3,jm}$, and ssk_{jm,nm^3} to $NameManager^3$.

b. $NameManager^1$, $NameManager^2$, and $NameManager^3$ authenticate to $JobManager$ before sending a reply to $JobManager$. The entity authentication is done by using the SOA protocol with the secondary key established in (MF-12(a)), i.e., sck_{jm,nm^1} by $NameManager^1$, sck_{jm,nm^2} by $NameManager^2$, and sck_{jm,nm^3} by $NameManager^3$. For authentication to the respective DataStores, the NameManagers prepare and send the primary keys and tickets to $JobManager$. Hence, in addition to session keys, the keys sent to $JobManager$ are pmk_{jm,ds_1^1}, $tkt_{jm,ds_1^1}^{nm^1}$, and $ssk_{nm^1,jm}$ by $NameManager^1$, pmk_{jm,ds_1^2}, $tkt_{jm,ds_1^2}^{nm^2}$, and $ssk_{nm^2,jm}$ by $NameManager^2$, and pmk_{jm,ds_1^3}, $tkt_{jm,ds_1^3}^{nm^3}$, and $ssk_{nm^3,jm}$ by $NameManager^3$.

c. $JobManager$ authenticates to $DataStore_1^1$, $DataStore_1^2$, and $DataStore_1^3$ before reading the job configuration files. The entity authentication is done by using the GE2A protocol with the DFS-C key established in (MF-9) and the primary key established in (MF-12(b)). $JobManager$ uses dfk^1 and pmk_{jm,ds_1^1} to authenticate to $DataStore_1^1$, dfk^2 and pmk_{jm,ds_1^2} to $DataStore_1^2$, and dfk^3 and pmk_{jm,ds_1^3} to $DataStore_1^3$. $JobManager$ sends ssk_{jm,ds_1^1} to $DataStore_1^1$, ssk_{jm,ds_1^2} to $DataStore_1^2$, and ssk_{jm,ds_1^3} to $DataStore_1^3$.

(MF-13) $JobManager$ authenticates to $ResourceManager^1$ before sending a request for Worker allocation as described in (EF-13). The entity authentication is done by using the GP2A protocol with the DPS-I key pik^1 and the primary key $pmk_{rm^1,jm}$ established in (MF-9). During the authentication, $JobManager$ sends a secondary key $sck_{rm^1,jm}$ and a session key ssk_{jm,rm^1} to $ResourceManager^1$.

(MF-14) $ResourceManager^1$ authenticates to $ResourceManager^2$ and $ResourceManager^3$ before sending a request for Worker allocation as described in (EF-14). The entity authentication is done by using the SOA protocol with the secondary keys (sck_{rm^1,rm^2} for authentication to $ResourceManager^2$ and sck_{rm^1,rm^3} to $ResourceManager^3$) established in (MF-6). $ResourceManager^1$ sends a new session key ssk_{rm^1,rm^2} to $ResourceManager^2$ and a new session key ssk_{rm^1,rm^3} to $ResourceManager^3$.

(MF-15) $ResourceManager^2$ and $ResourceManager^3$ authenticate to $ResourceManager^1$ before sending a reply to the Worker allocation request as described in (EF-15). Like (MF-14), the entity authentication is done by using the SOA protocol with sck_{rm^1,rm^2} by $ResourceManager^2$ and sck_{rm^1,rm^3} by $ResourceManager^3$. $ResourceManager^2$ and $ResourceManager^3$ prepare and send $ResourceManager^1$ the DPS-C keys, the primary keys, and the tickets for $JobManager$ to authenticate to the respective WorkerManagers ($WorkerManager_1^2$ and $WorkerManager_1^3$). They also send the public keys pk_{c^2} and pk_{c^3} established in (MF-5) to $ResourceManager^1$. The keys (including pk_{c^1}) will be distributed to the respective Mappers (via $JobManager$) for verifying the authenticity of the InputSplits. Hence, along with session keys, $ResourceManager^2$

sends pck^2, pmk_{jm,wm_1^2}, $tkt_{jm,wm_1^2}^{rm^2}$, pk_{c^2}, and ssk_{rm^2,rm^1} to $ResourceManager^1$, $ResourceManager^3$ sends pck^3, pmk_{jm,wm_1^3}, $tkt_{jm,wm_1^3}^{rm^3}$, pk_{c^3}, and ssk_{rm^3,rm^1} to $ResourceManager^1$.

(MF-16) $ResourceManager^1$ authenticates to $JobManager$ before sending a reply with a list of available WorkerNodes as described in (EF-16). The entity authentication is done by using the SOA protocol with the secondary key $sck_{rm^1,jm}$ established in (MF-13). $ResourceManager^1$ prepares a primary key pmk_{jm,wm_2^1} and a ticket $tkt_{jm,wm_2^1}^{rm^1}$ for $JobManager$ to authenticate to $WorkerManger_2^1$. Along with pmk_{jm,wm_2^1}, $tkt_{jm,wm_2^1}^{rm^1}$, and pk_{c^1} (established in (MF-5)), $ResourceManager^1$ sends pck^2, pck^3, pmk_{jm,wm_1^2}, pmk_{jm,wm_1^3}, $tkt_{jm,wm_1^2}^{rm^2}$, $tkt_{jm,wm_1^3}^{rm^3}$, pk_{c^2}, and pk_{c^3} obtained from $ResourceManager^2$ and $ResourceManager^3$ in (MF-15), and a new session key $ssk_{rm^1,jm}$ to $JobManager$.

(MF-17) $JobManager$ authenticates to $WorkerManager_2^1$, $WorkerManager_1^2$, and $WorkerManager_1^3$ before sending a request for launching Mappers and Reducers as described in (EF-17). The entity authentication is done by using the GE2A protocol with the DPS-C key (with the exception of the authentication to $WorkerManager_2^1$ where the DPS-I key pik^1 is used) and the primary key established in (MF-16). These keys are pik^1 and pmk_{jm,wm_2^1} for authentication to $WorkerManager_2^1$, pck^2 and pmk_{jm,wm_1^2} for authentication to $WorkerManager_1^2$, and pck^3 and pmk_{jm,wm_1^3} for authentication to $WorkerManager_1^3$. $JobManager$ prepares the DPS-C key pck^1 and the primary keys pmk_{jm,m_1}, pmk_{jm,m_2}, pmk_{jm,m_3}, pmk_{jm,r_1}, pmk_{jm,rm_2}, and pmk_{jm,rm_3} for $Mapper_1$, $Mapper_2$, $Mapper_3$, $Reducer_1$,. $Reducer_2$, and $Reducer_3$ to authenticate to $JobManager$, respectively. It also prepares the ticket sealing keys $tsk_{wm_2^1,jm}$, $tsk_{wm_1^2,jm}$, and $tsk_{wm_1^3,jm}$ for the generation and verification of tickets issued by $JobManager$ to Reducers for authentication to the respective WorkerManagers (to retrieve the assigned PartitionSegments). These keys and session keys ssk_{jm,wm_2^1}, ssk_{jm,wm_1^2}, and ssk_{jm,wm_1^3} are, respectively, sent to $WorkerManager_2^1$, $WorkerManager_1^2$, and $WorkerManager_1^3$.

(MF-18) $WorkerManager_2^1$, $WorkerManager_1^2$, and $WorkerManager_1^3$, respectively, start $Mapper_1$ and $Reducer_1$, $Mapper_2$ and $Reducer_2$, and $Mapper_3$ and $Reducer_3$ as described in (EF-18). Each of the WorkerManagers authenticates to the respective Mapper and Reducer by using an existing method. Each of the WorkerManagers embeds the group key (the DPS-I key or the DPS-C key) and the primary keys obtained in (MF-17) to the respective Mapper and Reducer by using an existing method (such as container template and shared memory). In other words, pik^1 and pmk_{jm,m_1} are given to $Mapper_1$ and pik^1 and pmk_{jm,r_1} to $Reducer_1$ by $WorkerManager_2^1$; pck^1 and pmk_{jm,m_2} are given to $Mapper_2$ and pck^1 and pmk_{jm,r_2} to $Reducer_2$ by $WorkerManager_1^2$; and pck^1 and pmk_{jm,m_3} are given to $Mapper_3$ and pck^1 and pmk_{jm,r_3} to $Reducer_3$ by $WorkerManager_1^3$.

(MF-19) $Mapper_1$, $Mapper_2$, $Mapper_3$, $Reducer_1$, $Reducer_2$, and $Reducer_3$ authenticate to $JobManager$ before reporting their status as described in (EF-19). The entity authentication is done by using the GP2A protocol with the group key (the DPS-I key pik^1 or the DPS-C key pck^1) and the primary key established in (MF-18). To authenticate to $JobManager$, $Mapper_1$ uses pik^1 and pmk_{jm,m_1}, $Mapper_2$ uses pck^1 and pmk_{jm,m_2}, $Mapper_3$ uses pck^1 and pmk_{jm,m_3}, $Reducer_1$ uses pik^1 and pmk_{jm,r_1}, $Reducer_2$ uses pck^1 and pmk_{jm,r_2}, and $Reducer_3$ uses pck^1 and pmk_{jm,r_3}. Each of the Workers sends a secondary key and a session key to $JobManager$, i.e., sck_{jm,m_1} and $ssk_{m_1,jm}$ are sent by $Mapper_1$, sck_{jm,m_2} and $ssk_{m_2,jm}$ by $Mapper_2$, sck_{jm,m_3} and $ssk_{m_3,jm}$ by $Mapper_3$, sck_{jm,r_1} and $ssk_{r_1,jm}$ by $Reducer_1$, sck_{jm,r_2} and $ssk_{r_2,jm}$ by $Reducer_2$, and sck_{jm,r_3} and $ssk_{r_3,jm}$ by $Reducer_3$.

(MF-20) $JobManager$ authenticates to $Mapper_1$, $Mapper_2$, and $Mapper_3$ before issuing a command to start map tasks and giving the location of the assigned InputSplits as described in (EF-20). The entity authentication is done by using the SOA protocol with the secondary key (sck_{jm,m_1} for $Mapper_1$, sck_{jm,m_2} for $Mapper_2$, and sck_{jm,m_3} for $Mapper_3$) established in (MF-19). $JobManager$ prepares the DFS-C keys (dfk^1, dfk^2, and dfk^3) and generates new primary keys (pmk_{m_1,nm^1}, pmk_{m_2,nm^2}, and pmk_{m_3,nm^3}) and tickets ($tkt^{jm}_{m_1,nm^1}$, $tkt^{jm}_{m_2,nm^2}$, and $tkt^{jm}_{m_3,nm^3}$) for the Mappers to authenticate to the respective NameManagers. It prepares pk_{c^1}, pk_{c^2}, and pk_{c^3} (obtained in (MF-16)) for the respective Mappers to verify the assigned InputSplits. It also generates new pairwise keys $k_{m_1,jm}$, $k_{m_2,jm}$, and $k_{m_3,jm}$ for the respective Mappers to sign the PartitionSegments they produce. Hence, $JobManager$ sends dfk^1, pmk_{m_1,nm^1}, $tkt^{jm}_{m_1,nm^1}$, pk_{c^1}, $k_{m_1,jm}$, and ssk_{jm,m_1} to $Mapper_1$; dfk^2, pmk_{m_2,nm^2}, $tkt^{jm}_{m_2,nm^2}$, pk_{c^2}, $k_{m_2,jm}$, and ssk_{jm,m_2} to $Mapper_2$; and dfk^3, pmk_{m_3,nm^3}, $tkt^{jm}_{m_3,nm^3}$, pk_{c^3}, $k_{m_3,jm}$, and ssk_{jm,m_3} to $Mapper_3$.

In addition, $JobManager$ also sends all the ISAuthData tokens to the respective Mappers (i.e., $\sigma_{rh_{c^1}}$and SA_{c^1} to $Mapper_1$, $\sigma_{rh_{c^2}}$and SA_{c^2} to $Mapper_2$, and $\sigma_{rh_{c^3}}$and SA_{c^3} to $Mapper_3$) by using the ISAuthData-Delivery protocol.

(MF-21) Before $Mapper_1$, $Mapper_2$, and $Mapper_3$ can read the assigned InputSplits from the respective DFS clusters as described in (EF-21), for each of the Mappers, three instances of entity authentication are taking place. The authentication process is similar to that of (MF-4).

a. $Mapper_1$, $Mapper_2$, and $Mapper_3$, respectively, authenticate to $NameManager^1$, $NameManager^2$, and $NameManager^3$. The entity authentication is done by using the GE2A protocol with the DFS-C key and the primary key obtained in (MF-20). The keys dfk^1 and pmk_{m_1,nm^1} are used by $Mapper_1$, dfk^2 and pmk_{m_2,nm^2} by $Mapper_2$, and dfk^3 and pmk_{m_3,nm^3} by $Mapper_3$. $Mapper_1$ sends sck_{m_1,nm^1} and ssk_{m_1,nm^1} to $NameManager^1$,

$Mapper_2$ sends sck_{m_2,nm^2} and ssk_{m_2,nm^2} to $NameManager^2$, and $Mapper_3$ sends sck_{m_3,nm^3} and ssk_{m_3,nm^3} to $NameManager^3$.

b. $NameManager^1$, $NameManager^2$, and $NameManager^3$, respectively, authenticate to $Mapper^1$, $Mapper^2$, and $Mapper^3$. The entity authentication is done by using the SOA protocol with the secondary key established in (MF-21(a)), i.e., sck_{m_1,nm^1} is used by $NameManager^1$, sck_{m_2,nm^2} by $NameManager^2$, and sck_{m_3,nm^3} by $NameManager^3$. The NameManagers generates primary keys and tickets for the Mappers to authenticate to the respectively DataStores. The keys and the tickets are transmitted along with session keys. Hence, $NameManager^1$ sends pmk_{m_1,ds_1^1}, $tkt_{m_1,ds_1^1}^{nm^1}$, and ssk_{nm^1,m_1} to $Mapper_1$; $NameManager^2$ sends pmk_{m_2,ds_1^2}, $tkt_{m_2,ds_1^2}^{nm^2}$, and ssk_{nm^2,m_2} to $Mapper_2$; and $NameManager^3$ sends pmk_{m_3,ds_1^3}, $tkt_{m_3,ds_1^3}^{nm^3}$, and ssk_{nm^3,m_3} to $Mapper_3$.

c. $Mapper_1$, $Mapper_2$, and $Mapper_3$, respectively, authenticate to $DataStore_1^1$, $DataStore_1^2$, and $DataStore_1^3$. The entity authentication is done by using the GE2A protocol with the DFS-C key established in (MF-20) and the primary key established in (MF-21(b)). The keys dfk^1 and pmk_{m_1,ds_1^1} are used by $Mapper_1$; dfk^2 and pmk_{m_2,ds_1^2} by $Mapper_2$; and dfk^3 and pmk_{m_3,ds_1^3} by $Mapper_3$. $Mapper_1$ sends ssk_{m_1,ds_1^1} to $DataStore_1^1$, $Mapper_2$ sends ssk_{m_2,ds_1^2} to $DataStore_1^2$, and $Mapper_3$ sends ssk_{m_3,ds_1^3} to $DataStore_1^3$.

(MF-22) Before performing the map tasks as described in (EF-22), $Mapper_1$, $Mapper_2$, and $Mapper_3$, respectively, verify the authenticity of $InputSplit_{1,1}$, $InputSplit_{2,2}$, and $InputSplit_{3,3}$ by using the ISAuthData-Verification algorithm (explained in Section 6.5.4.2) with the public keys of the respective ClientApps established in (MF-20) against ISAuthData tokens received in (MF-20). The key pk_{c^1} is used for the verification of $InputSplit_{1,1}$ against $\sigma_{rh_{c^1}}$ and SA_{c^1}; pk_{c^2} for $InputSplit_{2,2}$ against $\sigma_{rh_{c^2}}$ and SA_{c^2}; and pk_{c^3} for $InputSplit_{3,3}$ against $\sigma_{rh_{c^3}}$ and SA_{c^3}. If the verifications of all the InputSplits are positive, the Mappers can perform the map tasks on the assigned InputSplits.

When each of the Mappers finishes its map task and produces an IntermediateResult (containing multiple PartitionSegments), it signs its PartitionSegments by using the PGen-PSAuthData-Generation algorithm (explained in Section 6.5.5.1) with the pairwise key shared with $JobManager$ established in (MF-20). PGen-PSAuthData tokens are generated by the algorithm. $Mapper_1$ signs $PartitionSegment_{1,2}$ and $PartitionSegment_{1,3}$ with $k_{m_1,jm}$ to generate rh_{m_1}, $\tau_{rh_{m_1}}$, and SA_{m_1}; $Mapper_2$ signs $PartitionSegment_{2,1}$ and $PartitionSegment_{2,2}$ with $k_{m_2,jm}$ to generate rh_{m_2}, $\tau_{rh_{m_2}}$, and SA_{m_2}; and $Mapper_3$ signs $PartitionSegment_{3,2}$ and $PartitionSegment_{3,3}$ with $k_{m_3,jm}$ to generate rh_{m_3}, $\tau_{rh_{m_3}}$, and SA_{m_3}. The PGen-PSAuthData tokens will be used to generate AGen-PSAuthData tokens by $JobManager$.

(MF-23) $Mapper_1$, $Mapper_2$, and $Mapper_3$ authenticate to $JobManager$ before notifying of the completion of the map tasks as described in (EF-23). The entity authentication is done by using the SOA protocol with the secondary key (i.e., sck_{jm,m_1} by $Mapper_1$, sck_{jm,m_2} by $Mapper_2$, and sck_{jm,m_3} by $Mapper_3$) established in (MF-19). $Mapper_1$, $Mapper_2$, and $Mapper_3$, respectively, send session keys $ssk_{m_1,jm}$, $ssk_{m_2,jm}$, and $ssk_{m_3,jm}$ to $JobManager$.

Each of the Mappers also sends the PGen-PSAuthData tokens generated in (MF-22) to $JobManager$ by using the PSAuthData-Delivery protocol (explained in Section 6.5.5.4). $JobManager$ invokes the AGen-PSAuthData-Generation algorithm (explained in Section 6.5.5.2) with the PGen-PSAuthData tokens (rh_{m_1}, rh_{m_2}, rh_{m_3}, $\tau_{rh_{m_1}}$, $\tau_{rh_{m_2}}$, $\tau_{rh_{m_3}}$, SA_{m_1}, SA_{m_2}, and SA_{m_3}), the pairwise keys ($k_{m_1,jm}$, $k_{m_2,jm}$, and $k_{m_3,jm}$ established in (MF-20)), and the private key sk_{jm} (generated in (MF-9)) to generate AGen-PSAuthData tokens, ch_{jm} and $\sigma_{ch_{jm}}$. The PGen-PSAuthData tokens together with the AGen-PSAuthData tokens are used for the verification of PartitionSegments by the Reducers.

(MF-24) $JobManager$ authenticates to $Reducer_1$, $Reducer_2$, and $Reducer_3$. The entity authentication is done by using the SOA protocol with the secondary key (sck_{jm,r_1} for $Reducer_1$, sck_{jm,r_2}, for $Reducer_2$, and sck_{jm,r_3} for $Reducer_3$) established in (MF-19). $JobManager$ prepares the DPS-C keys (pck^1, pck^2, and pck^3) and generates new primary keys (pmk_{r_1,wm_1^2}, pmk_{r_2,wm_2^1}, pmk_{r_2,wm_1^3}, and pmk_{r_3,wm_2^1},) and tickets ($tkt^{jm}_{r_1,wm_1^2}$, $tkt^{jm}_{r_2,wm_2^1}$, $tkt^{jm}_{r_2,wm_1^3}$, and $tkt^{jm}_{r_3,wm_2^1}$) for the Reducers to authenticate to the respective WorkerManagers. It prepares the DFS-C key dfk^1 and generates new primary keys (pmk_{r_1,nm^1}, pmk_{r_2,nm^1}, and pmk_{r_3,nm^1}) and tickets ($tkt^{jm}_{r_1,nm^1}$, $tkt^{jm}_{r_2,nm^1}$, and $tkt^{jm}_{r_3,nm^1}$) for the Reducers to authenticate to $NameManager^1$. It prepares its public key pk_{jm} (generated in (MF-9)) for the Reducers to verify the assigned PartitionSegments. It also generates new pairwise keys $k_{r_1,jm}$, $k_{r_2,jm}$, and $k_{r_3,jm}$ for the respective Reducers to sign the FinalResults they produce. Hence, $JobManager$ sends pck^2, pmk_{r_1,wm_1^2}, $tkt^{jm}_{r_1,wm_1^2}$, dfk^1, pmk_{r_1,nm^1}, $tkt^{jm}_{r_1,nm^1}$, pk_{jm}, $k_{r_1,jm}$, and ssk_{jm,r_1} to $Reducer_1$; pck^1, pck^3, pmk_{r_2,wm_2^1}, pmk_{r_2,wm_1^3}, $tkt^{jm}_{r_2,wm_2^1}$, $tkt^{jm}_{r_2,wm_1^3}$, dfk^1, pmk_{r_2,nm^1}, $tkt^{jm}_{r_2,nm^1}$, pk_{jm}, $k_{r_2,jm}$, and ssk_{jm,r_2} to $Reducer_2$; and pck^1, pmk_{r_3,wm_2^1}, $tkt^{jm}_{r_3,wm_2^1}$, dfk^1, pmk_{r_3,nm^1}, $tkt^{jm}_{r_3,nm^1}$, pk_{jm}, $k_{r_3,jm}$, and ssk_{jm,r_3} to $Reducer_3$.

$JobManager$ also sends all the PGen-PSAuthData tokens to the respective Reducers (i.e., sa_{m_2,r_1} to $Reducer_1$, sa_{m_1,r_2}, sa_{m_2,r_2}, and sa_{m_3,r_2} to $Reducer_2$, and sa_{m_1,r_3} and sa_{m_3,r_3} to $Reducer_3$) and AGen-PSAuthData tokens (ch_{jm} and $\sigma_{ch_{jm}}$) to all the Reducers by using the PSAuthData-Delivery protocol. $JobManager$ then issues a command to all the Reducers to start the reduce tasks as described in (EF-24).

(MF-25) As described in (EF-25), each of the Reducers has to retrieve PartitionSegments from different WorkerManagers. The entity authentication is done by using the GE2A protocol with DPS-C key and the primary key established in (MF-24). $Reducer_1$ uses pck^2 and pmk_{r_1,wm_1^2} to authenticate to $WorkerManager_1^2$. $Reducer_2$ uses pck^1 and pmk_{r_2,wm_2^1} and pck^3 and pmk_{r_2,wm_1^3} to authenticate to $WorkerManager_2^1$ and $WorkerManager_1^3$, respectively. $Reducer_3$ uses pck^1 and pmk_{r_3,wm_2^1} to authenticate to $WorkerManager_2^1$. During the authentication, session keys are transmitted. A session key ssk_{r_1,wm_1^2} is sent from $Reducer_1$ to $WorkerManager_1^2$, ssk_{r_2,wm_2^1} from $Reducer_2$ to $WorkerManager_2^1$, ssk_{r_2,wm_1^3} from $Reducer_2$ to $WorkerManager_1^3$ and ssk_{r_3,wm_2^1} from $Reducer_3$ to $WorkerManager_2^1$.

For each of the Reducers, the assigned PartitionSegments are verified before the PartitionSegments are merged. The verification of the PartitionSegments is done by using the PSAuthData-Verification algorithm with the public key pk_{jm} of $JobManager$ against the AGen-PSAuthData tokens (ch_{jm} and $\sigma_{ch_{jm}}$) and the respective PGen-PSAuthData tokens (sa_{m_2,r_1} is used by $Reducer_1$; sa_{m_1,r_2}, sa_{m_2,r_2}, and sa_{m_3,r_2} by $Reducer_2$; and sa_{m_1,r_3} and sa_{m_3,r_3} by $Reducer_3$) obtained in (MF-24).

(MF-26) $Reducer_1$, $Reducer_2$, and $Reducer_3$ perform the reduce tasks on the merged PartitionSegments and generate the FinalResults ($FinalResult_{1,1}$, $FinalResult_{2,1}$, and $FinalResult_{3,1}$, respectively), as described in (EF-26). Each of the Reducers signs the FinalResult it produces by using the PGen-FRAuthData-Generation algorithm (explained in Section 6.5.6.1) with the pairwise key shared with $JobManager$ established in (MF-24). PGen-FRAuthData tokens are generated by the algorithm. $Reducer_1$ signs $FinalResult_{1,1}$ with $k_{r_1,jm}$ to generate h_{r_1,c^1} and $\tau_{h_{r_1,c^1}}$. $Reducer_2$ signs $FinalResult_{2,1}$ with $k_{r_2,jm}$ to generate h_{r_2,c^1} and $\tau_{h_{r_2,c^1}}$. $Reducer_3$ signs $FinalResult_{3,1}$ with $k_{r_3,jm}$ to generate h_{r_3,c^1} and $\tau_{h_{r_3,c^1}}$. The PGen-FRAuthData tokens will be used to generate the AGen-FRAuthData tokens by $JobManager$.

Following the generation of the FinalResults, all the Reducers write the FinalResults to DFS^1. Three instances of entity authentication are taking place.

a. $Reducer_1$, $Reducer_2$, and $Reducer_3$ authenticate to $NameManager^1$. The entity authentication is done by using the GE2A protocol with the DFS-C key dfk^1 and the primary key (pmk_{r_1,nm^1} by $Reducer_1$, pmk_{r_2,nm^1} by $Reducer_2$, and pmk_{r_3,nm^1} by $Reducer_3$) established in (MF-24). $Reducer_1$, $Reducer_2$, and $Reducer_3$, respectively, send sck_{r_1,nm^1} and ssk_{r_1,nm^1}, sck_{r_2,nm^1} and ssk_{r_2,nm^1}, and sck_{r_3,nm^1} and ssk_{r_3,nm^1} to $NameManager^1$.
b. $NameManager^1$ authenticates to $Reducer_1$, $Reducer_2$, and $Reducer_3$. The entity authentication is done by using the SOA protocol with the secondary key (sck_{r_1,nm^1} for $Reducer_1$, sck_{r_2,nm^1} for $Reducer_2$, and sck_{r_3,nm^1} for $Reducer_3$) established in (MF-26(a)). $NameManager^1$ prepares new primary keys and tickets for the Reducers to authenticate to $DataStore_1^1$. It then sends the primary keys, the tickets, and session keys to the Reducers, i.e., pmk_{r_1,ds_1^1},

$tkt_{r_1,ds_1^1}^{nm^1}$, and ssk_{nm^1,r_1} to $Reducer_1$; pmk_{r_2,ds_1^1}, $tkt_{r_2,ds_1^1}^{nm^1}$, and ssk_{nm^1,r_2} to $Reducer_2$; and pmk_{r_3,ds_1^1}, $tkt_{r_3,ds_1^1}^{nm^1}$, and ssk_{nm^1,r_3} to $Reducer_3$.

c. $Reducer_1$, $Reducer_2$, and $Reducer_3$ authenticate to $DataStore_1^1$. The entity authentication is done by using the GE2A protocol with the DFS-C key dfk^1 established in (MF-24) and the primary key (pmk_{r_1,ds_1^1} by $Reducer_1$, pmk_{r_2,ds_1^1} by $Reducer_2$, and pmk_{r_3,ds_1^1} by $Reducer_3$) established in (MF-26(b)). $Reducer_1$, $Reducer_2$, and $Reducer_3$, respectively, send ssk_{r_1,ds_1^1}, ssk_{r_2,ds_1^1}, and ssk_{r_3,ds_1^1} to $DataStore_1^1$.

(MF-27) $Reducer_1$, $Reducer_2$, and $Reducer_3$ authenticate to $JobManager$ before notifying of the completion of the reduce tasks as described in (EF-27). The entity authentication is done by using the SOA protocol with the secondary key ($sck_{r_1,jm}$ by $Reducer_1$, $sck_{r_2,jm}$ by $Reducer_2$, and $sck_{r_3,jm}$ by $Reducer_3$) established in (MF-19). $Reducer_1$, $Reducer_2$, and $Reducer_3$, respectively, send $ssk_{r_1,jm}$, $ssk_{r_2,jm}$, and $ssk_{r_3,jm}$ to $JobManager$.

Each of the Reducers sends PGen-FRAuthData tokens generated in (MF-26) to $JobManager$. The PGen-FRAuthData tokens are delivered by using the FRAuthData-Delivery protocol (explained in Section 6.5.6.4). $JobManager$ invokes the AGen-FRAuthData-Generation algorithm (explained in Section 6.5.6.2) with the PGen-FRAuthData tokens (h_{r_1,c^1}, h_{r_2,c^1}, h_{r_3,c^1}, $\tau_{h_{r_1,c^1}}$, $\tau_{h_{r_2,c^1}}$, and $\tau_{h_{r_3,c^1}}$), the pairwise keys ($k_{r_1,jm}$, $k_{r_2,jm}$, and $k_{r_3,jm}$) established in (MF-24), and the private key sk_{jm} (generated in (MF-9)) to generate AGen-FRAuthData tokens, ch_{jm}^* and $\sigma_{ch_{jm}^*}$. The AGen-FRAuthData tokens are used for the verification of FinalResults by $ClientApp^1$.

(MF-28) $JobManager$ authenticates to $ClientApp^1$ before notifying of the readiness of the FinalResults as described in (EF-28). The entity authentication is done by using the SOA protocol with the secondary key $sck_{c^1,jm}$ established in (MF-11). $JobManager$ sends its public key pk_{jm} (generated in (MF-9)) for $ClientApp^1$ to verify the authenticity of the FinalResults and a new session key ssk_{jm,c^1} to $ClientApp^1$. It also sends the AGen-FRAuthData tokens (ch_{jm}^* and $\sigma_{ch_{jm}^*}$) generated in (MF-27) to $ClientApp^1$ by using the FRAuthData-Delivery protocol.

(MF-29) Before $ClientApp^1$ reads the FinalResults stored in DFS^1 as described in (EF-29), three instances of entity authentication are taking place.

a. $ClientApp^1$ authenticates to $NameManager^1$. The entity authentication is done by using the GE2A protocol with the DFS-C key dfk^1 and the primary key pmk_{c^1,nm^1} established in (MF-3). $ClientApp^1$ sends a secondary key sck_{c^1,nm^1} and ssk_{c^1,nm^1} to $NameManager^1$.

b. $NameManager^1$ authenticates to $ClientApp^1$. The entity authentication is done by using the SOA protocol with the secondary key sck_{c^1,nm^1} established in (MF-29(a)). $NameManager^1$ prepares a new primary key pmk_{c^1,ds_1^1} and a new ticket $tkt_{c^1,ds_1^1}^{nm^1}$ for $ClientApp^1$ to authenticate to $DataStore_1^1$. Hence,

$NameManager^1$ sends pmk_{c^1,ds_1^1}, $tkt_{c^1,ds_1^1}^{nm^1}$, and a new session key ssk_{nm^1,c^1} to $ClientApp^1$.

c. $ClientApp^1$ authenticates to $DataStore_1^1$. The entity authentication is done by using the GE2A protocol with the DFS-C key dfk^1 established in (MF-3) and the primary key pmk_{c^1,ds_1^1} established in (MF-29(b)). $ClientApp^1$ sends a session key ssk_{c^1,ds_1^1} to $DataStore_1^1$.

After all the FinalResults ($FinalResult_{1,1}$, $FinalResult_{2,1}$ and $FinalResult_{3,1}$) are retrieved, $ClientApp^1$ verifies the authenticity of the FinalResults. This is done by using the FRAuthData-Verification algorithm (explained in Section 6.5.6.3) with the public key pk_{jm} of $JobManager$ against the AGen-FRAuthData tokens (ch_{jm}^* and $\sigma_{ch_{jm}^*}$) obtained in (MF-28). If the verification of all the FinalResults is positive, $User^1$ is assured that the output of the job is authentic and has not been tampered with by unauthorised entities.

Table 7.5 summarises the entities, credentials, and AuthData involved in authenticating entities and JobData when MDA is applied to the job execution of the working example.

Table 7.5: The summary of entities, credentials, and AuthData involved in authentication when MDA is applied to the job execution of the working example.

Operational step	Entities / Components / JobData objects	Authentication	Protocol / Algorithm / Method	Keys used for authentication[1]	Keys transmitted / AuthData tokens generated
(MF-1)	$User^1, User^2, User^3$	Entity authentication	Existing method (EXT)	-	jk
(MF-2)	$ClientApp^1$, $ResourceManager^1$	Entity authentication	GP2A	ok^1, pmk_{c^1,rm^1}	$jk, sck_{c^1,rm^1}, ssk_{c^1,rm^1}$
	$ClientApp^2$, $ResourceManager^2$	Entity authentication	GP2A	ok^2, pmk_{c^2,rm^2}	$jk, sck_{c^2,rm^2}, ssk_{c^2,rm^2}$
	$ClientApp^3$, $ResourceManager^3$	Entity authentication	GP2A	ok^3, pmk_{c^3,rm^3}	$jk, sck_{c^3,rm^3}, ssk_{c^3,rm^3}$
(MF-3)	$ResourceManager^1$, $ClientApp^1$	Entity authentication	SOA	sck_{c^1,rm^1}	$pck^1, dfk^1, pmk_{c^1,nm^1}, tkt_{c^1,nm^1}^{rm^1}$, ssk_{rm^1,c^1}
	$ResourceManager^2$, $ClientApp^2$	Entity authentication	SOA	sck_{c^2,rm^2}	$dfk^2, pmk_{c^2,nm^2}, tkt_{c^2,nm^2}^{rm^2}$, ssk_{rm^2,c^2}
	$ResourceManager^3$, $ClientApp^3$	Entity authentication	SOA	sck_{c^3,rm^3}	$dfk^3, pmk_{c^3,nm^3}, tkt_{c^3,nm^3}^{rm^3}$, ssk_{rm^3,c^3}
(MF-4)	$ClientApp^1$, $NameManager^1$	Entity authentication	GE2A	dfk^1, pmk_{c^1,nm^1}	$sck_{c^1,nm^1}, ssk_{c^1,nm^1}$
	$ClientApp^2$, $NameManager^2$	Entity authentication	GE2A	dfk^2, pmk_{c^2,nm^2}	$sck_{c^2,nm^2}, ssk_{c^2,nm^2}$
	$ClientApp^3$, $NameManager^3$	Entity authentication	GE2A	dfk^3, pmk_{c^3,nm^3}	$sck_{c^3,nm^3}, ssk_{c^3,nm^3}$
	$NameManager^1$, $ClientApp^1$	Entity authentication	SOA	sck_{c^1,nm^1}	$pmk_{c^1,ds_1^1}, tkt_{c^1,ds_1^1}^{nm^1}, ssk_{nm^1,c^1}$
	$NameManager^2$, $ClientApp^2$	Entity authentication	SOA	sck_{c^2,nm^2}	$pmk_{c^2,ds_1^2}, tkt_{c^2,ds_1^2}^{nm^2}, ssk_{nm^2,c^2}$
	$NameManager^3$, $ClientApp^3$	Entity authentication	SOA	sck_{c^3,nm^3}	$pmk_{c^3,ds_1^3}, tkt_{c^3,ds_1^3}^{nm^3}, ssk_{nm^3,c^3}$

Operational step	Entities / Components / JobData objects	Authentication	Protocol / Algorithm / Method	Keys used for authentication[1]	Keys transmitted / AuthData tokens generated
	$ClientApp^1$, $DataStore_1^1$	Entity authentication	GE2A	dfk^1, pmk_{c^1,ds_1^1}	ssk_{c^1,ds_1^1}
	$ClientApp^2$, $DataStore_1^2$	Entity authentication	GE2A	dfk^2, pmk_{c^2,ds_1^2}	ssk_{c^2,ds_1^2}
	$ClientApp^3$, $DataStore_1^3$	Entity authentication	GE2A	dfk^3, pmk_{c^3,ds_1^3}	ssk_{c^3,ds_1^3}
	$InputSplit_{1,1}$	Data authentication	ISAuthData-Generation	sk_{c^1}	$\sigma_{rh_{c^1}}$, SA_{c^1}
	$InputSplit_{2,2}$	Data authentication	ISAuthData-Generation	sk_{c^2}	$\sigma_{rh_{c^2}}$, SA_{c^2}
	$InputSplit_{3,3}$	Data authentication	ISAuthData-Generation	sk_{c^3}	$\sigma_{rh_{c^3}}$, SA_{c^3}
(MF-5)	$ClientApp^1$, $ResourceManager^1$	Entity authentication	SOA	sck_{c^1,rm^1}	ssk_{rm^1,c^1}, pk_{c^1}
	$ClientApp^2$, $ResourceManager^2$	Entity authentication	SOA	sck_{c^2,rm^2}	ssk_{rm^2,c^2}, pk_{c^2}
	$ClientApp^3$, $ResourceManager^3$	Entity authentication	SOA	sck_{c^3,rm^3}	ssk_{rm^3,c^3}, pk_{c^3}
(MF-6)	$ResourceManager^1$, $ResourceManager^2$	Entity authentication	GP2A	jk, pmk_{rm^1,rm^2}	sck_{rm^1,rm^2}, ssk_{rm^1,rm^2}
	$ResourceManager^1$, $ResourceManager^3$	Entity authentication	GP2A	jk, pmk_{rm^1,rm^3}	sck_{rm^1,rm^3}, ssk_{rm^1,rm^3}
(MF-7)	$ResourceManager^2$, $ResourceManager^1$	Entity authentication	SOA	sck_{rm^1,rm^2}	dfk^2, pmk_{jm,nm^2}, $tkt_{jm,nm^2}^{rm^2}$, ssk_{rm^2,rm^1}
	$ResourceManager^3$, $ResourceManager^1$	Entity authentication	SOA	sck_{rm^1,rm^3}	dfk^3, pmk_{jm,nm^3}, $tkt_{jm,nm^3}^{rm^3}$, ssk_{rm^3,rm^1}
(MF-8)	$ResourceManager^1$, $WorkerManager_1^1$	Entity authentication	GP2A	pik^1, $pmk_{wm_1^1,rm^1}$	dfk^1, dfk^2, dfk^3, $pmk_{rm^1,jm}$, pmk_{jm,nm^1}, pmk_{jm,nm^2}, pmk_{jm,nm^3}, $tkt_{jm,nm^1}^{rm^1}$, $tkt_{jm,nm^2}^{rm^2}$, $tkt_{jm,nm^3}^{rm^3}$, ssk_{rm^1,wm_1^1}
(MF-9)	$WorkerManager_1^1$, $JobManager$	Entity authentication	EXT	-	dfk^1, dfk^2, dfk^3, $pmk_{rm^1,jm}$, pmk_{jm,nm^1}, pmk_{jm,nm^2}, pmk_{jm,nm^3}, $tkt_{jm,nm^1}^{rm^1}$, $tkt_{jm,nm^2}^{rm^2}$, $tkt_{jm,nm^3}^{rm^3}$
(MF-10)	$ResourceManager^1$, $ClientApp^1$	Entity authentication	SOA	sck_{c^1,rm^1}	$pmk_{c^1,jm}$, $tkt_{c^1,jm}^{rm^1}$, ssk_{rm^1,c^1}
(MF-11)	$ClientApp^1$, $JobManager$	Entity authentication	GE2A	pck^1, $pmk_{c^1,jm}$	$sck_{c^1,jm}$, $ssk_{c^1,jm}$
(MF-12)	$JobManager$, $NameManager^1$	Entity authentication	GE2A	dfk^1, pmk_{jm,nm^1}	sck_{jm,nm^1}, $tsk_{nm^1,jm}$, ssk_{jm,nm^1}
	$JobManager$, $NameManager^2$	Entity authentication	GE2A	dfk^2, pmk_{jm,nm^2}	sck_{jm,nm^2}, $tsk_{nm^2,jm}$, ssk_{jm,nm^2}
	$JobManager$, $NameManager^3$	Entity authentication	GE2A	dfk^3, pmk_{jm,nm^3}	sck_{jm,nm^3}, $tsk_{nm^3,jm}$, ssk_{jm,nm^3}
	$NameManager^1$, $JobManager$	Entity authentication	SOA	sck_{jm,nm^1}	pmk_{jm,ds_1^1}, $tkt_{jm,ds_1^1}^{nm^1}$, $ssk_{nm^1,jm}$
	$NameManager^2$, $JobManager$	Entity authentication	SOA	sck_{jm,nm^2}	pmk_{jm,ds_1^2}, $tkt_{jm,ds_1^2}^{nm^2}$, $ssk_{nm^2,jm}$
	$NameManager^3$, $JobManager$	Entity authentication	SOA	sck_{jm,nm^3}	pmk_{jm,ds_1^3}, $tkt_{jm,ds_1^3}^{nm^3}$, $ssk_{nm^3,jm}$

Operational step	Entities / Components / JobData objects	Authentication	Protocol / Algorithm / Method	Keys used for authentication[1]	Keys transmitted / AuthData tokens generated
	$JobManager$, $DataStore_1^1$	Entity authentication	GE2A	dfk^1, pmk_{jm,ds_1^1}	ssk_{jm,ds_1^1}
	$JobManager$, $DataStore_1^2$	Entity authentication	GE2A	dfk^2, pmk_{jm,ds_1^2}	ssk_{jm,ds_1^2}
	$JobManager$, $DataStore_1^3$	Entity authentication	GE2A	dfk^3, pmk_{jm,ds_1^3}	ssk_{jm,ds_1^3}
(MF-13)	$JobManager$, $ResourceManager^1$	Entity authentication	GP2A	pik^1, $pmk_{rm^1,jm}$	$sck_{rm^1,jm}$, ssk_{jm,rm^1}
(MF-14)	$ResourceManager^1$, $ResourceManager^2$	Entity authentication	SOA	sck_{rm^1,rm^2}	ssk_{rm^1,rm^2}
	$ResourceManager^1$, $ResourceManager^3$	Entity authentication	SOA	sck_{rm^1,rm^3}	ssk_{rm^1,rm^3}
(MF-15)	$ResourceManager^2$, $ResourceManager^1$	Entity authentication	SOA	sck_{rm^1,rm^2}	pck^2, pmk_{jm,wm_1^2}, $tkt_{jm,wm_1^2}^{rm^2}$, pk_{c^2}, ssk_{rm^2,rm^1}
	$ResourceManager^3$, $ResourceManager^1$	Entity authentication	SOA	sck_{rm^1,rm^3}	pck^3, pmk_{jm,wm_1^3}, $tkt_{jm,wm_1^3}^{rm^3}$, pk_{c^3}, ssk_{rm^3,rm^1}
(MF-16)	$ResourceManager^1$, $JobManager$	Entity authentication	SOA	$sck_{rm^1,jm}$	pck^2, pck^3, pmk_{jm,wm_2^1} pmk_{jm,wm_1^2}, pmk_{jm,wm_1^3}, $tkt_{jm,wm_2^1}^{rm^1}$, $tkt_{jm,wm_1^2}^{rm^2}$, $tkt_{jm,wm_1^3}^{rm^3}$, pk_{c^1}, pk_{c^2}, pk_{c^3}, $ssk_{rm^1,jm}$
(MF-17)	$JobManager$, $WorkerManager_2^1$	Entity authentication	GE2A	pik^1, pmk_{jm,wm_2^1}	pmk_{jm,m_1}, pmk_{jm,r_1}, $tsk_{wm_2^1,jm}$, ssk_{jm,wm_2^1}
	$JobManager$, $WorkerManager_1^2$	Entity authentication	GE2A	pck^2, pmk_{jm,wm_1^2}	pck^1, pmk_{jm,m_2}, pmk_{jm,r_2}, $tsk_{wm_1^2,jm}$, ssk_{jm,wm_1^2}
	$JobManager$, $WorkerManager_1^3$	Entity authentication	GE2A	pck^3, pmk_{jm,wm_1^3}	pck^1, pmk_{jm,m_3}, pmk_{jm,r_3}, $tsk_{wm_1^3,jm}$, ssk_{jm,wm_1^3}
(MF-18)	$WorkerManager_2^1$, $Mapper_1$	Entity authentication	EXT	-	pik^1, pmk_{jm,m_1}
	$WorkerManager_2^1$, $Reducer_1$	Entity authentication	EXT	-	pik^1, pmk_{jm,r_1}
	$WorkerManager_1^2$, $Mapper_2$	Entity authentication	EXT	-	pck^1, pmk_{jm,m_2}
	$WorkerManager_1^2$, $Reducer_2$	Entity authentication	EXT	-	pck^1,pmk_{jm,r_2}
	$WorkerManager_1^3$, $Mapper_3$	Entity authentication	EXT	-	pck^1, pmk_{jm,m_3}
	$WorkerManager_1^3$, $Reducer_3$	Entity authentication	EXT	-	pck^1, pmk_{jm,r_3}
(MF-19)	$Mapper_1$, $JobManager$	Entity authentication	GP2A	pik^1, pmk_{jm,m_1}	sck_{jm,m_1}, $ssk_{m_1,jm}$
	$Mapper_2$, $JobManager$	Entity authentication	GP2A	pck^1, pmk_{jm,m_2}	sck_{jm,m_2}, $ssk_{m_2,jm}$
	$Mapper_3$, $JobManager$	Entity authentication	GP2A	pck^1, pmk_{jm,m_3}	sck_{jm,m_3}, $ssk_{m_3,jm}$
	$Reducer_1$, $JobManager$	Entity authentication	GP2A	pik^1, pmk_{jm,r_1}	sck_{jm,r_1},$ssk_{r_1,jm}$
	$Reducer_2$, $JobManager$	Entity authentication	GP2A	pck^1, pmk_{jm,r_2}	sck_{jm,r_2}, $ssk_{r_2,jm}$
	$Reducer_3$, $JobManager$	Entity authentication	GP2A	pck^1, pmk_{jm,r_3}	sck_{jm,r_3}, $ssk_{r_3,jm}$

Operational step	Entities / Components / JobData objects	Authentication	Protocol / Algorithm / Method	Keys used for authentication[1]	Keys transmitted / AuthData tokens generated
(MF-20)	$JobManager$, $Mapper_1$	Entity authentication	SOA	sck_{jm,m_1}	dfk^1, pmk_{m_1,nm^1}, tkt_{m_1,nm^1}^{jm}, pk_{c^1}, $k_{m_1,jm}$, ssk_{jm,m_1}
	$JobManager$, $Mapper_2$	Entity authentication	SOA	sck_{jm,m_2}	dfk^2, pmk_{m_2,nm^2}, tkt_{m_2,nm^2}^{jm}, pk_{c^2}, $k_{m_2,jm}$, ssk_{jm,m_2}
	$JobManager$, $Mapper_3$	Entity authentication	SOA	sck_{jm,m_3}	dfk^3, pmk_{m_3,nm^3}, tkt_{m_3,nm^3}^{jm}, pk_{c^3}, $k_{m_3,jm}$, ssk_{jm,m_3}
(MF-21)	$Mapper_1$, $NameManager^1$	Entity authentication	GE2A	dfk^1, pmk_{m_1,nm^1}	sck_{m_1,nm^1}, ssk_{m_1,nm^1}
	$Mapper_2$, $NameManager^2$	Entity authentication	GE2A	dfk^2, pmk_{m_2,nm^2}	sck_{m_2,nm^2}, ssk_{m_2,nm^2}
	$Mapper_3$, $NameManager^3$	Entity authentication	GE2A	dfk^3, pmk_{m_3,nm^3}	sck_{m_3,nm^3}, ssk_{m_3,nm^3}
	$NameManager^1$, $Mapper_1$	Entity authentication	SOA	sck_{m_1,nm^1}	pmk_{m_1,ds_1^1}, $tkt_{m_1,ds_1^1}^{nm^1}$, ssk_{nm^1,m_1}
	$NameManager^2$, $Mapper_2$	Entity authentication	SOA	sck_{m_2,nm^2}	pmk_{m_2,ds_1^2}, $tkt_{m_2,ds_1^2}^{nm^2}$, ssk_{nm^2,m_2}
	$NameManager^3$, $Mapper_3$	Entity authentication	SOA	sck_{m_3,nm^3}	pmk_{m_3,ds_1^3}, $tkt_{m_3,ds_1^3}^{nm^3}$, ssk_{nm^3,m_3}
	$Mapper_1$, $DataStore_1^1$	Entity authentication	GE2A	dfk^1, pmk_{m_1,ds_1^1}	ssk_{m_1,ds_1^1}
	$Mapper_2$, $DataStore_1^2$	Entity authentication	GE2A	dfk^2, pmk_{m_2,ds_1^2}	ssk_{m_2,ds_1^2}
	$Mapper_3$, $DataStore_1^3$	Entity authentication	GE2A	dfk^3, pmk_{m_3,ds_1^3}	ssk_{m_3,ds_1^3}
(MF-22)	$InputSplit_{1,1}$	Data authentication	ISAuthData-Verification	pk_{c^1}	-
	$InputSplit_{2,2}$	Data authentication	ISAuthData-Verification	pk_{c^2}	-
	$InputSplit_{3,3}$	Data authentication	ISAuthData-Verification	pk_{c^3}	-
	$PartitionSegment_{1,2}$, $PartitionSegment_{1,3}$	Data authentication	PGen-PSAuthData-Generation	$k_{m_1,jm}$	rh_{m_1}, $\tau_{rh_{m_1}}$, SA_{m_1}
	$PartitionSegment_{2,1}$, $PartitionSegment_{2,2}$	Data authentication	PGen-PSAuthData-Generation	$k_{m_2,jm}$	rh_{m_2}, $\tau_{rh_{m_2}}$, SA_{m_2}
	$PartitionSegment_{3,2}$, $PartitionSegment_{3,3}$	Data authentication	PGen-PSAuthData-Generation	$k_{m_3,jm}$	rh_{m_2}, $\tau_{rh_{m_2}}$, SA_{m_2}
(MF-23)	$Mapper_1$, $JobManager$	Entity authentication	SOA	sck_{jm,m_1}	$ssk_{m_1,jm}$
	$Mapper_2$, $JobManager$	Entity authentication	SOA	sck_{jm,m_2}	$ssk_{m_2,jm}$
	$Mapper_3$, $JobManager$	Entity authentication	SOA	sck_{jm,m_3}	$ssk_{m_3,jm}$
	rh_{m_1}, rh_{m_2}, rh_{m_3}, $\tau_{rh_{m_1}}$, $\tau_{rh_{m_2}}$, $\tau_{rh_{m_3}}$, SA_{m_1}, SA_{m_2}, SA_{m_3}	Data authentication	AGen-PSAuthData-Generation	$k_{m_1,jm}$, $k_{m_2,jm}$, $k_{m_3,jm}$, sk_{jm}	ch_{jm}, $\sigma_{ch_{jm}}$
(MF-24)	$JobManager$, $Reducer_1$	Entity authentication	SOA	sck_{jm,r_1}	pck^2, pmk_{r_1,wm_1^2}, $tkt_{r_1,wm_1^2}^{jm}$, dfk^1, pmk_{r_1,nm^1}, tkt_{r_1,nm^1}^{jm}, pk_{jm}, $k_{r_1,jm}$, ssk_{jm,r_1}

Operational step	Entities / Components / JobData objects	Authentication	Protocol / Algorithm / Method	Keys used for authentication[1]	Keys transmitted / AuthData tokens generated
	$JobManager$, $Reducer_2$	Entity authentication	SOA	sck_{jm,r_2}	pck^1, pck^3, pmk_{r_2,wm_2^1}, pmk_{r_2,wm_1^3}, $tkt_{r_2,wm_2^1}^{jm}$, $tkt_{r_2,wm_1^3}^{jm}$, dfk^1, pmk_{r_2,nm^1}, tkt_{r_2,nm^1}^{jm}, pk_{jm}, $k_{r_2,jm}$, ssk_{jm,r_2}
	$JobManager$, $Reducer_3$	Entity authentication	SOA	sck_{jm,r_3}	pck^1, pmk_{r_3,wm_2^1}, $tkt_{r_3,wm_2^1}^{jm}$, dfk^1, pmk_{r_3,nm^1}, tkt_{r_3,nm^1}^{jm}, pk_{jm}, $k_{r_3,jm}$, ssk_{jm,r_3}
(MF-25)	$Reducer_1$, $WorkerManager_1^2$	Entity authentication	GE2A	pck^2, pmk_{r_1,wm_1^2}	ssk_{r_1,wm_1^2}
	$Reducer_2$, $WorkerManager_2^1$	Entity authentication	GE2A	pck^1, pmk_{r_2,wm_2^1}	ssk_{r_2,wm_2^1}
	$Reducer_2$, $WorkerManager_1^3$	Entity authentication	GE2A	pck^3, pmk_{r_2,wm_1^3}	ssk_{r_2,wm_1^3}
	$Reducer_3$, $WorkerManager_2^1$	Entity authentication	GE2A	pck^1, pmk_{r_3,wm_2^1}	ssk_{r_3,wm_2^1}
	$PartitionSegment_{2,1}$	Data authentication	PSAuthData-Verification	pk_{jm}	-
	$PartitionSegment_{1,2}$, $PartitionSegment_{2,2}$, $PartitionSegment_{3,2}$	Data authentication	PSAuthData-Verification	pk_{jm}	-
	$PartitionSegment_{1,3}$, $PartitionSegment_{3,3}$	Data authentication	PSAuthData-Verification	pk_{jm}	-
(MF-26)	$FinalResult_{1,1}$	Data authentication	PGen-FRAuthData-Generation	$k_{r_1,jm}$	h_{r_1,c^1}, $\tau_{h_{r_1,c^1}}$
	$FinalResult_{2,1}$	Data authentication	PGen-FRAuthData-Generation	$k_{r_2,jm}$	h_{r_2,c^1}, $\tau_{h_{r_2,c^1}}$
	$FinalResult_{3,1}$	Data authentication	PGen-FRAuthData-Generation	$k_{r_3,jm}$	h_{r_3,c^1}, $\tau_{h_{3,c^1}}$
	$Reducer_1$, $NameManager^1$	Entity authentication	GE2A	dfk^1, pmk_{r_1,nm^1}	sck_{r_1,nm^1}, ssk_{r_1,nm^1}
	$Reducer_2$, $NameManager^1$	Entity authentication	GE2A	dfk^1, pmk_{r_2,nm^1}	sck_{r_2,nm^1}, ssk_{r_2,nm^1}
	$Reducer_3$, $NameManager^1$	Entity authentication	GE2A	dfk^1, pmk_{r_3,nm^1}	sck_{r_3,nm^1}, ssk_{r_3,nm^1}
	$NameManager^1$, $Reducer_1$	Entity authentication	SOA	sck_{r_1,nm^1}	pmk_{r_1,ds_1^1}, $tkt_{r_1,ds_1^1}^{nm^1}$, ssk_{nm^1,r_1}
	$NameManager^1$, $Reducer_2$	Entity authentication	SOA	sck_{r_2,nm^1}	pmk_{r_2,ds_1^1}, $tkt_{r_2,ds_1^1}^{nm^1}$, ssk_{nm^1,r_2}
	$NameManager^1$, $Reducer_3$	Entity authentication	SOA	sck_{r_3,nm^1}	pmk_{r_3,ds_1^1}, $tkt_{r_3,ds_1^1}^{nm^1}$, ssk_{nm^1,r_3}
	$Reducer_1$, $DataStore_1^1$	Entity authentication	GE2A	dfk^1, pmk_{r_1,ds_1^1}	ssk_{r_1,ds_1^1}
	$Reducer_2$, $DataStore_1^1$	Entity authentication	GE2A	dfk^1, pmk_{r_2,ds_1^1}	ssk_{r_2,ds_1^1}
	$Reducer_3$, $DataStore_1^1$	Entity authentication	GE2A	dfk^1, pmk_{r_3,ds_1^1}	ssk_{r_3,ds_1^1}
(MF-27)	$Reducer_1$, $JobManager$	Entity authentication	SOA	sck_{jm,r_1}	$ssk_{r_1,jm}$

Operational step	Entities / Components / JobData objects	Authentication	Protocol / Algorithm / Method	Keys used for authentication[1]	Keys transmitted / AuthData tokens generated
	$Reducer_2$, $JobManager$	Entity authentication	SOA	sck_{jm,r_2}	$ssk_{r_2,jm}$
	$Reducer_3$, $JobManager$	Entity authentication	SOA	sck_{jm,r_3}	$ssk_{r_3,jm}$
	h_{r_1,c^1}, h_{r_2,c^1}, h_{r_3,c^1}, $\tau_{h_{r_1,c^1}}$, $\tau_{h_{r_2,c^1}}$, $\tau_{h_{r_3,c^1}}$	Data authentication	AGen-FRAuthData-Generation	$k_{r_1,jm}$, $k_{r_2,jm}$, $k_{r_3,jm}$, sk_{jm}	ch^*_{jm}, $\sigma_{ch^*_{jm}}$
(MF-28)	$JobManager$, $ClientApp^1$	Entity authentication	SOA	$sck_{c_1,jm}$	pk_{jm}, ssk_{jm,c^1}
(MF-29)	$ClientApp^1$, $NameManager^1$	Entity authentication	GE2A	dfk^1, pmk_{c^1,nm^1}	sck_{c^1,nm^1}, ssk_{c^1,nm^1}
	$NameManager^1$, $ClientApp^1$	Entity authentication	SOA	sck_{c^1,nm^1}	pmk_{c^1,ds_1^1}, $tkt^{nm^1}_{c^1,ds_1^1}$, ssk_{nm^1,c^1}
	$ClientApp^1$, $DataStore_1^1$	Entity authentication	GE2A	dfk^1, pmk_{c^1,ds_1^1}	ssk_{c^1,ds_1^1}
	$FinalResult_{1,1}$, $FinalResult_{2,1}$, $FinalResult_{3,1}$	Data authentication	FRAuthData-Verification	pk_{jm}	-

Note: 1 – The times and establishment methods for keys for entity authentication are summarised in Table 5.5.

7.4 Chapter Summary

This chapter has used the running example to first illustrate how the job is executed using MR based services without our MDA framework, and then with our MDA framework, in a CBDC-MPC setting. The job execution flow of MR is not application-specific, so it could support a wide range of applications, including those in which data used are mission critical and may be sensitive. By applying the MDA framework, all the entities involved in the job execution are authenticated, and these entity authentications are applied whenever the entities are involved in an interaction to send, receive, or process data, regardless of when the interaction is taking place during the job execution cycle. In addition, the MDA framework also provides data authentication protection, and the protection is applied to all the data used and generated in the entire job execution cycle. The chapter has explained in detail how different components of the MDA framework are applied in each of the operational steps of the job execution to facilitate these entity and data authentication protections. The example presented in this chapter has demonstrated that the MDA framework is an authentication solution that is suited to MR based data processing in a CBDC-MPC context.

The next chapter concludes this thesis and presents future work.

Chapter 8
Conclusions and Future Work

With the advancement of Big Data processing and cloud computing, there is a growing trend for CBDC-MPC. In this context, there are open security issues and challenges that have yet to be addressed. This thesis investigates how to achieve effective, efficient, and scalable authentication to support secure CBDC-MPC using distributed computing services. This chapter summarises the work presented in this thesis, highlighting the contributions and findings. It also gives recommendations for future work.

8.1 Contributions

The contributions of this thesis are summarised on a chapter-by-chapter basis as follows.

Chapter 4: Generic Use Case Model and MDA Framework

In this chapter, a generic use case model for CBDC-MPC has been formulated and the architecture of our novel effective, efficient, and scalable authentication framework, the Multi-domain Decentralised Authentication (MDA) framework, for MR based CBDC-MPC has been described.

The CBDC-MPC model is formulated based on an extreme version of collaborative data analysis using distributed computing services. All the data, the data processing services, and the underlying infrastructure are assumed to be from different administrative domains and there is minimal trust among the entities involved. In formulating the use case, two system architectures and five Big Data processing models have been examined. Compared with SA-SC (all the components of a data processing service are hosted in a single cloud), SA-MC (the components of a service are hosted in different cloud) resembles a trend for utility computing as it gives a higher level of flexibility to service consumers and providers. With greater flexibility, it presents a broader set of security challenges. This implies that an authentication solution designed for SA-MC should also be able to address security challenges faced by SA-SC. Based on our analysis, different Big Data processing models have many common characteristics. This means that a solution designed for one model should also be applicable to the other models. Although some models, e.g., Apache Spark, perform better than MR under certain conditions and settings, MR is one of the most used Big Data processing models and there are extensive supports and documents available to users and service providers. As a result, the SA-MC architecture and the MR model have been chosen for the construction of the CBDC-MPC model. The model shows in detail the entities involved in a job execution and how these components interconnect to accomplish the job. The model can not only be used to serve the design of our authentication solution, but also help with threat analyses and the designs of other security solutions for other applications that exhibit similar characteristics.

Using the model, we have analysed where in the system that attacks could be mounted and how the attacks may be countered with as less overhead as possible. For this purpose, the classifications of MR components involved, data used, and interactions taking place have been carried out, and communication patterns identified. Threats and attacks with regard to violation of entity identity and data authenticity protection have been identified. A set of requirements have been specified to counter the identified threats. The observations on the model along with the specified requirements have been used to guide the design of the MDA framework. The MDA framework consists of the Multi-factor Interaction based Entity Authentication (MIEA) framework and the Communication Pattern based Data Authentication (CPDA) framework. MIEA and CPDA, respectively, provide an entity authentication facility and a data authentication facility to support secure job execution in the context. Different from other existing authentication solutions, MDA is specifically designed for the CBDC-MPC context, and it provides a strong security protection (both content authenticity and origin non-repudiation) at the finest granularity level with minimal impacts on the performance of the underlying system. Although we have chosen MR as the underlying distributed computing service framework, MDA can be applied to any distributed computing service frameworks so long as they possess the characteristics of MR, namely, multi-stage data processing, the use of multiple data producers and data consumers, and the use of any or all of the communication patterns captured in MDA. In addition, owing to the modular design of MDA, components of MDA can be applied to other applications as needed. They can also be used with other security services, e.g., with an authorisation service to provide access control, or with an auditing service as part of a detective security measure. Although the main ideas used in the design of MDA have been used or applied in other fields or contexts, the application of these ideas in the CBDC-MPC context is novel.

Chapter 5: MIEA Framework

In this chapter, a novel approach, an interaction based approach, to entity authentication for CBDC-MPC has been proposed, implemented, and evaluated.

The proposed approach provides entity authentication protection to every interaction taking place during the whole cycle of a job execution. This is done by utilising three main ideas: (1) the idea of Multi-factor Interaction based Authentication (MIA) in which credentials and authentication methods are selected based on the risk level tagged to each of the interactions; (2) the idea of a Decentralised approach with Combined use of group-and-entity-dependent Symmetric keys (DCS) in which the distribution and verification of credentials are done by distributed entities while maximising the use of computationally-efficient symmetric-key cryptosystems; and (3) the idea of a Hierarchical Key Structure (HKS) in which the distribution of keys is based on a hierarchical structure, keys in a higher level of the structure are used to securely distribute keys in a lower level of the structure.

The approach has been implemented in the design of a novel entity authentication framework, the MIEA framework. Compared with entity authentication solutions that provide only gate-level protection, MIEA provides protection at a much finer granularity (the interaction level) which covers the entire cycle of a data processing job. This is to deter threats and attacks caused by both outsiders and insiders. Compared with symmetric-key based solutions, such as Kerberos, MIEA uses two-factor authentication to protect critical interactions, thus, making breaking authentication tokens harder. It also minimises the number of messages exchanged to facilitate the authentication, thus, lowering communication overhead introduced. Compared with asymmetric-key based solutions, such as NSLPK, MIEA uses symmetric-key operations, which is significantly cheaper computationally than their asymmetric-key based counterparts, thus introducing a lower level of computational overhead cost. To demonstrate the effectiveness and the efficiency of the MIEA framework, the framework has been extensively evaluated by using both theoretical and experimental methods. Informal, symbolic, and complexity analysis methods have been used to evaluate the security properties and strengths of MIEA. The results show that the MIEA framework satisfies all the specified security requirements with regard to entity authentication (i.e., (SR1), (SR2), (SR3), and (SR4)) and the strengths of the protections are dependent on the parameter values (e.g., key lengths). In theoretical performance evaluation, the computational and communication overheads introduced by MIEA have been analysed in terms of the number of cryptographic operations performed and the volume of AuthData transmitted over networks, respectively. The results have been compared with those of the most related entity authentication solutions, i.e., the Kerberos and the NSLPK protocols. The results show that, for computational overhead, MIEA introduces the highest number of cryptographic operations (all of which are symmetric-key based and are less computationally expensive). Regarding communication overhead, the number of protocol messages exchanged when MIEA is applied is only 3, fewer than those of Kerberos and NSLPK without ticket and public key caching. However, the messages used in the MIEA protocols have larger payload sizes compared with those of Kerberos and NSLPK. Experimental evaluations have been conducted on a real-system testbed. The results show that the execution times of the MIEA protocols (particularly, the SOA protocol) are the shortest due to the smallest number of transmitted messages and the use of computationally less expensive symmetric-key cryptosystems. In addition, the performance of MIEA is dependent on the sizes of message payloads, the larger the payload (i.e., an entity may interact with many other entities, thus, more credentials to be transmitted for subsequent authentication), the longer the execution times. The results of the evaluations indicate that MIEA outperforms other related solutions for MR based data processing under the parameters and settings used in the experiments. It provides a stronger level of entity authentication protection but at no higher cost, than Kerberos, one of the most used entity authentication solutions. The identities of entities can be established and verified, laying a groundwork for other security services. The more efficient the entity authentication process, the sooner the data processing tasks can start, the sooner the job can finish, the sooner the output can be produced, the more the jobs can be processed in a given time for a given resource setting.

Chapter 6: CPDA Framework

In this chapter, a novel approach, a communication pattern based approach, to data authentication for MR based CBDC-MPC has been proposed, implemented, and evaluated.

The proposed approach protects the authenticity (encompassing origin authentication and integrity protection) of all the JobData and achieves accountability (by providing non-repudiation of origin) at the finest granularity (i.e., at the object level) while being highly efficient and scalable. This is accomplished by using two main ideas: (1) the idea of AuthData and Communication Aggregation (ACA) which reduces the number of objects to be signed and verified with computationally-expensive cryptographic algorithms and aggregates the communications transmitting the AuthData; and (2) the idea of a Hybrid use of multiple cryptographic schemes with Segregation of Credentials (HYSC) which further cuts the overhead introduced by maximising the use of computationally-inexpensive cryptographic algorithms and improves accountability by using a different pairwise key for each pair of untrustworthy and trustworthy components.

The approach has been implemented in the design of a novel data authentication framework, the CPDA framework. Compared with symmetric-key based solutions without any form of asymmetry, CPDA can provide non-repudiation of origin protection which is necessary to hold entities accountable. Data producers cannot falsely deny having produced their data, thus, preventing fraudulent data injection and tampering. Compared with the secret-share based, task-replication based, and asymmetric-key based without signature amortisation solutions, CPDA requires less computation resources and introduces a lower level of overhead cost. To evaluate the security protections provided by and the performance of CPDA, theoretical analyses and experimental evaluations have been carried out. The results have been compared with those of the most related object-level solutions, i.e., the schemes that secure individual objects with a MAC and a digital signature, respectively. An informal analysis method and a complexity analysis method have been used to analyse the security properties and strength of CPDA. The results show that CPDA can achieve all the specified security requirements with regard to data authentication (i.e., (SR5), (SR6), and (SR7)) as the strongest solution, i.e., the signature based scheme, but with less overhead cost introduced. The theoretical performance evaluation of CPDA has been carried out by analysing the number of cryptographic operations performed by individual components and the volume of traffics transmitted for AuthData delivery. The results show that, in comparison with the signature based scheme, CPDA can bring a significant reduction in computational overhead cost imposed on data processing components by cutting down the number of expensive cryptographic operations on large objects to one. This is achieved at a cost of additional cryptographic operations imposed on Aggregator and a larger message size (more items in the payload). Experimental evaluations have been conducted on a real-system testbed (consisting of 5 networked machines running up to 400 Workers) with a real-world weather dataset. The results show that (1) CPDA is significantly more efficient compared to the signature based scheme; (2) the cost incurred by CPDA is close to those of the MAC based scheme; and (3) the reduction in overhead costs brought by CPDA is significant, particularly when CPDA is applied to a large-scale job execution involving a large quantity of objects with small size. The evaluation results show that CPDA provides the same level of protection (origin authentication, integrity protection, non-repudiation of origin) as that of the most secure object based solution (i.e., the solution that digitally signs and verifies individual data objects) at the finest level of granularity (the object level) but with performance closer to the MAC based solution. The strongest level of data authenticity protection ensures that the JobData used throughout the execution of the job are produced by the claimed entities and are not contaminated by unauthorised entities. In other words, the output of the job is authentic. This is particularly important for mission-critical jobs or applications. As JobData objects are individually verifiable, there are no dependency among data consumers and the data consumers can start their tasks as soon as the assigned objects are ready. A lower level of overhead cost introduced in signing and verifying objects means a shorter delay is added to the execution of the job. CDPA is highly scalable, thus, it is suited to Big Data processing applications.

8.2 Conclusions

From this research, we can draw the following conclusions:

- Designing an authentication solution for CBDC in an MPC environment is a challenging task, as for CBDC, the solution should be highly efficient and scalable and, for the MPC environment, the design of the solution should assume the components are less trustworthy. This implies that the security protection provided should be the strongest, but the overhead cost introduced should be the lowest. A strong level of protection usually incurs a high level of overhead cost. Applying the same level of strong protection to every interaction and every data object is neither efficient nor practical. The design of an authentication solution for large-scale Big Data processing, which is the case for this work, should balance the trade-off between the level of protection needed and the overhead cost introduced. The way we balanced this trade-off is that: an appropriate level of protection is applied to a different point in the data processing flow; a stronger level of protection with a higher level of overhead cost is applied to more-critical points, whereas a weaker level of protection with a lower level of overhead cost is applied to less-critical points.
- The design of an authentication solution that takes into account of the characteristics of the underlying distributed computing system brings much benefit in terms of supporting effective, efficient, and scalable authentication in a large-scale distributed computing setting. These characteristics help us identify the weak points of the system, the level of protection required, and how to improve efficiency and scalability. This is captured in the ideas of MIA (discussed in Section 5.3) where stronger security protection with higher overhead cost is only applied to interactions (i.e., initial interactions) experiencing a higher level of risks, and ACA (discussed in Section 6.3) where a different aggregation method for AuthData and communications is selected based on a communication pattern exhibited. Furthermore, by applying decentralised authentication, which is in alignment with the characteristics of distributed computing, we can make good use of resource parallelism offered by the underlying system to evenly balance the workload imposed on the system and improve on service resilience.
- A hybrid use of asymmetric key and symmetric key cryptosystems allows us to achieve strong security protection while lowering the computational overhead cost introduced. As can be seen in the design of CPDA, by applying a digital signature scheme only on aggregated AuthData tokens (which are smaller in size and quantity compared with JobData objects), we can extend the strong security protection of the digital signature scheme to all JobData objects but with a fraction of computational overhead cost introduced.

- Each of the components of the MDA framework, i.e., the MIEA framework and the CPDA framework, offers respective merits and demerits in terms of the reduction in overhead cost when applied to small-scale and large-scale MR services, respectively. For small-scale MR services, the overhead cost introduced by the MIEA framework is at the same level as that of Kerberos and is only 0.25% of that of NSLPK. On the other hand, the CPDA framework can bring a larger cut in overhead cost when applied to large-scale MR services. CPDA can cut the delay in job execution time by two thirds compared with the signature based scheme. These results show that the MDA framework are highly efficient and scalable in supporting secure distributed computing in this context of CBDC-MPC. It is worth noting that MDA may not be the best framework for distributed computing services in some contexts. For example, in a setting where an organisation has full control over the system, an existing solution such as Kerberos can provide a sufficient level of protection more efficiently.
- Owing to the modular design of the MDA framework, different components of MDA can be applied together or separately to distributed computing services. The components of MDA can work with other security services as long as the security services support the required functions (e.g., credential distribution). MDA can be applied as add-on modules so minimal modifications to the system are required. For these reasons, we believe that MDA can be applied to a wider range of distributed computing services.
- While we try our best to generalise our MDA framework as much as possible so that it can be applied in a broader set of applications in similar contexts. The performance of MDA when deployed for production may differ from the results reported in this thesis. This is because the performance of MDA is not only dependent on the scale of the underlying distributed computing services but also other factors, e.g., jobs to be processed, infrastructures used, and how the MDA framework is implemented. In all the experimental evaluations presented in this thesis, we have clearly specified which parameter values are used. The experimental results reported in this thesis should only be interpreted based on the specified sets of parameters.

8.3 Future Work

The following recommendations are given as directions for future work.

- We may evaluate the performance of the MDA framework under different settings, i.e., (1) different implementations of MR services (e.g., Apache Hadoop); (2) different jobs (e.g., cyber threat analysis and biomedical research); (3) larger scale of MR services (e.g., thousands of Workers); and (4) different infrastructures (e.g., public clouds). These could give us more empirical evidence and make the evaluation results more conclusive.
- We may investigate how the MDA could be applied to other distributed computing frameworks, such as Flink [131], Spark [116], and Storm [111]. Some of the characteristics exhibited by these frameworks may not be captured in MR, which may affect the applicability, effectiveness, efficiency, and scalability of the MDA

framework. In addition, these frameworks may also present other challenging issues that are yet to be addressed.

- The MDA framework could be extended to support other security properties (such as authorisation), thus, providing a more comprehensive security protection against unauthorised access to data and system. This could be done by integrating other security measures (e.g., access control to support authorisation) and technologies (e.g., blockchain to provide verifiable security records).
- In this research work, we take a customised approach in the design of our authentication solution, i.e., the design of our solution is tailored in line with the characteristics of the underlying systems, to achieve effective, efficient, and scalable authentication. The findings from this work also support that the approach can indeed achieve the aim of this research. However, there is also a growing trend for Security-as-a-Service which implements a generalised approach. It would be interesting to investigate what are merits and demerits of both approaches when applied to distributed computing in the same context.

References

[1] S. Chakrabarty, P. LaMontagne, D. S. Marcus, and M. Milchenko, “Preprocessing of clinical neuro-oncology MRI studies for big data applications,” in *Medical Imaging 2020: Imaging Informatics for Healthcare, Research, and Applications*, 2020, p. 8, doi: 10.1117/12.2548371.

[2] S. Wolfert, L. Ge, C. Verdouw, and M.-J. Bogaardt, “Big Data in Smart Farming – A review,” *Agric. Syst.*, vol. 153, pp. 69–80, May 2017, doi: 10.1016/j.agsy.2017.01.023.

[3] S. E. Bibri, “The IoT for smart sustainable cities of the future: An analytical framework for sensor-based big data applications for environmental sustainability,” *Sustain. Cities Soc.*, vol. 38, pp. 230–253, Apr. 2018, doi: 10.1016/j.scs.2017.12.034.

[4] N. Thomas, “Cyber Security in East Asia: Governing Anarchy,” *Asian Secur.*, vol. 5, no. 1, pp. 3–23, Jan. 2009, doi: 10.1080/14799850802611446.

[5] Y. Zhang, F. Patwa, and R. Sandhu, “Community-Based Secure Information and Resource Sharing in AWS Public Cloud,” in *2015 IEEE Conference on Collaboration and Internet Computing (CIC)*, 2015, pp. 46–53, doi: 10.1109/CIC.2015.42.

[6] M. Böhm, S. Leimeister, C. Riedl, and H. Krcmar, “Cloud Computing – Outsourcing 2.0 or a new Business Model for IT Provisioning?,” in *Application Management*, Wiesbaden: Gabler, 2011, pp. 31–56.

[7] S. Dhar, “From outsourcing to Cloud computing: evolution of IT services,” *Manag. Res. Rev.*, vol. 35, no. 8, pp. 664–675, Jul. 2012, doi: 10.1108/01409171211247677.

[8] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, “Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility,” *Futur. Gener. Comput. Syst.*, vol. 25, no. 6, pp. 599–616, Jun. 2009, doi: 10.1016/j.future.2008.12.001.

[9] E. Brynjolfsson, P. Hofmann, and J. Jordan, “Cloud Computing and Electricity: Beyond the Utility Model,” *Commun. ACM*, vol. 53, no. 5, p. 32, May 2010, doi: 10.1145/1735223.1735234.

[10] V. Chang, “Towards data analysis for weather cloud computing,” *Knowledge-Based Syst.*, vol. 127, pp. 29–45, Jul. 2017, doi: 10.1016/j.knosys.2017.03.003.

[11] V. Lakshmanan and T. W. Humphrey, “A MapReduce Technique to Mosaic Continental-Scale Weather Radar Data in Real-Time,” *IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens.*, vol. 7, no. 2, pp. 721–732, Feb. 2014, doi: 10.1109/JSTARS.2013.2282040.

[12] C. Zhang, H. De Sterck, A. Aboulnaga, H. Djambazian, and R. Sladek, “Case Study of Scientific Data Processing on a Cloud Using Hadoop,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 5976 LNCS, 2010, pp. 400–415.

[13] A. AlMahmoud, E. Damiani, H. Otrok, and Y. Al-Hammadi, “Spamdoop: A Privacy-Preserving Big Data Platform for Collaborative Spam Detection,” *IEEE Trans. Big Data*, vol. 5, no. 3, pp. 293–304, Sep. 2019, doi: 10.1109/TBDATA.2017.2716409.

[14] W. Zhao and G. White, “A Collaborative Information Sharing Framework for Community Cyber Security,” *2012 IEEE Int. Conf. Technol. Homel. Secur.*, pp. 457–462, 2012, doi: 10.1109/THS.2012.6459892.

[15] J. Luo, M. Wu, D. Gopukumar, and Y. Zhao, “Big Data Application in Biomedical Research and Health Care: A Literature Review,” *Biomed. Inform. Insights*, vol. 8, p. BII.S31559, Jan. 2016, doi: 10.4137/BII.S31559.

[16] S. Dolev, P. Florissi, E. Gudes, S. Sharma, and I. Singer, “A Survey on Geographically Distributed Big-Data Processing Using MapReduce,” *IEEE Trans. Big Data*, vol. 5, no. 1, pp. 60–80, Mar. 2019, doi: 10.1109/TBDATA.2017.2723473.

[17] M. Mattess, R. N. Calheiros, and R. Buyya, “Scaling MapReduce applications across hybrid clouds to meet soft deadlines,” *Proc. - Int. Conf. Adv. Inf. Netw. Appl. AINA*, pp. 629–636, 2013, doi: 10.1109/AINA.2013.51.

[18] C.-Y. Wang, T.-L. Tai, S. Jui-Shing, C. Jyh-Biau, and S. Ce-Kuen, “Federated MapReduce to Transparently Run Applications on Multicluster Environment,” in *2014 IEEE International Congress on Big Data*, 2014, pp. 296–303, doi: 10.1109/BigData.Congress.2014.50.

[19] A. Iordache, C. Morin, N. Parlavantzas, E. Feller, and P. Riteau, “Resilin: Elastic MapReduce over multiple clouds,” *Proc. - 13th IEEE/ACM Int. Symp. Clust. Cloud, Grid Comput. CCGrid 2013*, pp. 261–268, 2013, doi: 10.1109/CCGrid.2013.48.

[20] I. A. T. Hashem, I. Yaqoob, N. B. Anuar, S. Mokhtar, A. Gani, and S. Ullah Khan, “The rise of ‘big data’ on cloud computing: Review and open research issues,” *Inf. Syst.*, vol. 47, pp. 98–115, Jan. 2015, doi: 10.1016/j.is.2014.07.006.

[21] E. Huedo, R. S. Montero, R. Moreno, I. M. Llorente, A. Levin, and P. Massonet, “Interoperable Federated Cloud Networking,” *IEEE Internet Comput.*, vol. 21, no. 5, pp. 54–59, 2017, doi: 10.1109/MIC.2017.3481337.

[22] E. Gaetani, L. Aniello, R. Baldoni, F. Lombardi, A. Margheri, and V. Sassone, “Blockchain-based database to ensure data integrity in cloud computing environments,” 2017.

[23] A. Singh and K. Chatterjee, “Cloud security issues and challenges: A survey,” *J. Netw. Comput. Appl.*, vol. 79, no. November 2016, pp. 88–115, Feb. 2017, doi: 10.1016/j.jnca.2016.11.027.

[24] “Marriott International Notifies Guests of Property System Incident.” [Online]. Available: https://news.marriott.com/news/2020/03/31/marriott-international-notifies-guests-of-property-system-incident. [Accessed: 30-Jun-2021].

[25] “Trade Secret Theft.” [Online]. Available: https://www.fbi.gov/news/stories/two-guilty-in-theft-of-trade-secrets-from-ge-072920. [Accessed: 30-Jun-2021].

[26] “Ex-Cisco Engineer Pleads Guilty in Insider Threat Case.” [Online]. Available: https://www.bankinfosecurity.com/ex-cisco-engineer-pleads-guilty-in-insider-threat-case-a-14917. [Accessed: 30-Jun-2021].

[27] *Data Protection Act 2018*. UK.

[28] *UK General Data Protection Regulation*. .

[29] A. Rubens, C. Rigney, S. Willens, and W. A. Simpson, “Remote Authentication Dial In User Service (RADIUS),” no. 2865. RFC Editor, Jun-2000, doi: 10.17487/rfc2865.

[30] V. Fajardo, J. Arkko, J. Loughney, and G. Zorn, “Diameter Base Protocol,” no. 6733. RFC Editor, Oct-2012, doi: 10.17487/rfc6733.

[31] G. Lowe, “Breaking and fixing the Needham-Schroeder Public-Key Protocol using FDR,” 1996, pp. 147–166.

[32] “Shibboleth.” [Online]. Available: https://shibboleth.net/. [Accessed: 14-Feb-2020].

[33] “Access Policy Manager.” [Online]. Available: https://www.f5.com/products/security/access-policy-manager. [Accessed: 14-Feb-2020].

[34] C. Neuman, T. Yu, S. Hartman, and K. Raeburn, “The Kerberos Network Authentication Service (V5),” Jul. 2005.

[35] B. Spivey and J. Echeverria, *Hadoop Security: Protecting your big data platform*. “ O’Reilly Media, Inc.,” 2015.

[36] I. Lahmer, “Towards a Virtual Domain based Authentication Solution for the MapReduce Application,” the University of Manchester, 2018.

[37] H. Krawczyk, M. Bellare, and R. Canetti, “HMAC: Keyed-Hashing for Message Authentication,” Feb. 1997.

[38] M. J. Dworkin, “Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication,” Gaithersburg, MD, 2016.

[39] A. Perrig, R. Canetti, J. D. Tygar, and Dawn Song, “Efficient authentication and signing of multicast streams over lossy channels,” in *Proceeding 2000 IEEE Symposium on Security and Privacy. S&P 2000*, 2000, pp. 56–73, doi: 10.1109/SECPRI.2000.848446.

[40] S. Miner and J. Staddon, “Graph-based authentication of digital streams,” in *Proceedings 2001 IEEE Symposium on Security and Privacy. S&P 2001*, 2001, pp. 232–246, doi: 10.1109/SECPRI.2001.924301.

[41] P. Golle and N. Modadugu, “Authenticating Streamed Data in the Presence of Random Packet Loss,” *Proc. Symp. Netw. Distrib. Syst. Secur. (NDSS 2001)*, pp. 13–22, 2001.

[42] Y. Challal, H. Bettahar, and A. Bouabdallah, “A/sup 2/cast: an adaptive source authentication protocol for multicast streams,” in *Proceedings. ISCC 2004. Ninth International Symposium on Computers And Communications (IEEE Cat. No.04TH8769)*, vol. 1, pp. 363–368, doi: 10.1109/ISCC.2004.1358431.

[43] W. Zhou *et al.*, “Towards a data-centric view of cloud security,” in *Proceedings of the second international workshop on Cloud data management - CloudDB ’10*, 2010, p. 25, doi: 10.1145/1871929.1871934.

[44] S. Sirapaisan, N. Zhang, and Q. He, “Communication Pattern Based Data Authentication (CPDA) Designed for Big Data Processing in a Multiple Public Cloud Environment,” *IEEE Access*, vol. 8, pp. 107716–107748, 2020, doi: 10.1109/ACCESS.2020.3000989.

[45] “Speed Comparison of Popular Crypto Algorithms.” [Online]. Available: https://www.cryptopp.com/benchmarks.html. [Accessed: 25-Oct-2019].

[46] Y. Wang and J. Wei, “VIAF: Verification-based integrity assurance framework for MapReduce,” *Proc. - 2011 IEEE 4th Int. Conf. Cloud Comput. CLOUD 2011*, pp. 300–307, 2011, doi: 10.1109/CLOUD.2011.33.

[47] Y. Wang, J. Wei, and M. Srivatsa, “Cross Cloud MapReduce: A Result Integrity Check Framework on Hybrid Clouds,” *Int. J. Cloud Comput.*, vol. 1, no. 1, pp. 26–39, 2013.

[48] Y. Wang, J. Wei, M. Srivatsa, Y. Duan, and W. Du, “IntegrityMR: Integrity assurance framework for big data analytics and management applications,” in *2013 IEEE International Conference on Big Data*, 2013, pp. 33–40, doi: 10.1109/BigData.2013.6691780.

[49] Y. Ding, H. Wang, L. Wei, S. Chen, H. Fu, and X. Xu, “VAWS: Constructing trusted open computing system of mapreduce with verified participants,” *IEICE Trans. Inf. Syst.*, vol. E97-D, no. 4, pp. 721–732, 2014, doi: 10.1587/transinf.E97.D.721.

[50] S. M. Khan and K. W. Hamlen, “Hatman: Intra-cloud trust management for Hadoop,” *Proc. - 2012 IEEE 5th Int. Conf. Cloud Comput. CLOUD 2012*, pp. 494–501, 2012, doi: 10.1109/CLOUD.2012.64.

[51] R. Neisse, G. Steri, and I. Nai-Fovino, “A Blockchain-based Approach for Data Accountability and Provenance Tracking,” in *Proceedings of the 12th International Conference on Availability, Reliability and Security*, 2017, vol. Part F1305, pp. 1–10, doi: 10.1145/3098954.3098958.

[52] R. K. Kodali, S. Yerroju, and B. Y. K. Yogi, “Blockchain Based Energy Trading,” in *TENCON 2018 - 2018 IEEE Region 10 Conference*, 2018, vol. 2018-Octob, no. October, pp. 1778–1783, doi: 10.1109/TENCON.2018.8650447.

[53] M. A. Mustafa, Ning Zhang, G. Kalogridis, and Zhong Fan, “DEP2SA: A Decentralized Efficient Privacy-Preserving and Selective Aggregation Scheme in Advanced Metering Infrastructure,” *IEEE Access*, vol. 3, pp. 2828–2846, 2015, doi: 10.1109/ACCESS.2015.2506198.

[54] A. Naureen and N. Zhang, “A Comparative Study of Data Aggregation Approaches for Wireless Sensor Networks,” in *Proceedings of the 12th ACM Symposium on QoS and Security for Wireless and Mobile Networks*, 2016, pp. 125–128, doi: 10.1145/2988272.2988285.

[55] R. Ali, A. K. Pal, S. Kumari, M. Karuppiah, and M. Conti, “A secure user authentication and key-agreement scheme using wireless sensor networks for agriculture monitoring,” *Futur. Gener. Comput. Syst.*, vol. 84, pp. 200–215, Jul. 2018, doi: 10.1016/j.future.2017.06.018.

[56] B. Furht and F. Villanustre, “Introduction to Big Data,” in *Big Data Technologies and Applications*, Cham: Springer International Publishing, 2016, pp. 3–11.

[57] M. Chen, S. Mao, and Y. Liu, “Big Data: A Survey,” *Mob. Networks Appl.*, vol. 19, no. 2, pp. 171–209, Apr. 2014, doi: 10.1007/s11036-013-0489-0.

[58] M. Birjali, A. Beni-Hssane, and M. Erritali, “Analyzing Social Media through Big Data using InfoSphere BigInsights and Apache Flume,” *Procedia Comput. Sci.*, vol. 113, pp. 280–285, 2017, doi: 10.1016/j.procs.2017.08.299.

[59] L. G. Rios and J. A. I. Diguez, “Big Data Infrastructure for analyzing data generated by Wireless Sensor Networks,” in *2014 IEEE International Congress on Big Data*, 2014, pp. 816–823, doi: 10.1109/BigData.Congress.2014.142.

[60] L. Dandurand and O. Serrano, “Towards Improved Cyber Security Information Sharing,” *5th Int. Conf. Cyber Confl.*, p. 16, 2013, doi: 10.1109/HICSS.2014.252.

[61] P. Hui *et al.*, “Towards efficient collaboration in cyber security,” *2010 Int. Symp. Collab. Technol. Syst. CTS 2010*, pp. 489–498, 2010, doi: 10.1109/CTS.2010.5478473.

[62] X. Liu, N. Iftikhar, and X. Xie, “Survey of real-time processing systems for big data,” in *Proceedings of the 18th International Database Engineering & Applications Symposium on - IDEAS ’14*, 2014, pp. 356–361, doi: 10.1145/2628194.2628251.

[63] K. Hwang, J. Dongarra, and G. C. Fox, *Distributed and Cloud Computing: From Parallel Processing to the Internet of Things*, 1st ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2011.

[64] I. Foster, Y. Zhao, I. Raicu, and S. Lu, “Cloud Computing and Grid Computing 360-Degree Compared,” in *2008 Grid Computing Environments Workshop*, 2008, pp. 1–10, doi: 10.1109/GCE.2008.4738445.

[65] I. Brandic and S. Dustdar, “Grid vs Cloud — A Technology Comparison,” *it - Inf. Technol.*, vol. 53, no. 4, pp. 173–179, Jul. 2011, doi: 10.1524/itit.2011.0640.

[66] F. Oesterle, S. Ostermann, R. Prodan, and G. J. Mayr, “Experiences with distributed computing for meteorological applications: grid computing and cloud computing,” *Geosci. Model Dev.*, vol. 8, no. 7, pp. 2067–2078, Jul. 2015, doi: 10.5194/gmd-8-2067-2015.

[67] P. F. Hsu, S. Ray, and Y. Y. Li-Hsieh, “Examining cloud computing adoption intention, pricing mechanism, and deployment model,” *Int. J. Inf. Manage.*, vol. 34, no. 4, pp. 474–488, 2014, doi: 10.1016/j.ijinfomgt.2014.04.006.

[68] P. M. Mell and T. Grance, “The NIST definition of cloud computing,” Gaithersburg, MD, 2011.

[69] “AWS Customer Success.” [Online]. Available: https://aws.amazon.com/solutions/case-studies/. [Accessed: 13-Jul-2017].

[70] S. Barnum, “Standardizing cyber threat intelligence information with the Structured Threat Information eXpression (STIX^{TM}),” *MITRE Corp. July*, pp. 1–20, 2014.

[71] R. Zhang and L. Liu, “Security Models and Requirements for Healthcare Application Clouds,” in *2010 IEEE 3rd International Conference on Cloud Computing*, 2010, pp. 268–275, doi: 10.1109/CLOUD.2010.62.

[72] Mélissa Gaillard and S. Pandolfi, “CERN Data Centre passes the 200-petabyte milestone,” Jul. 2017.

[73] J. Urbani, S. Kotoulas, J. Maassen, F. Van Harmelen, and H. Bal, “WebPIE: A Web-scale Parallel Inference Engine using MapReduce,” *J. Web Semant.*, vol. 10, pp. 59–75, Jan. 2012, doi: 10.1016/j.websem.2011.05.004.

[74] G. J. Chen *et al.*, “Realtime Data Processing at Facebook,” *Proc. 2016 Int. Conf. Manag. Data*, pp. 1087–1098, 2016, doi: 10.1145/2882903.2904441.

[75] A. Thusoo, Z. Shao, and S. Anthony, “Data warehousing and analytics infrastructure at facebook,” *… Manag. data*, p. 1013, 2010, doi: 10.1145/1807167.1807278.

[76] A. J. Duncan, S. Creese, and M. Goldsmith, “Insider attacks in cloud computing,” *Proc. 11th IEEE Int. Conf. Trust. Secur. Priv. Comput. Commun. Trust. - 11th IEEE Int. Conf. Ubiquitous Comput. Commun. IUCC-2012*, pp. 857–862, 2012, doi: 10.1109/TrustCom.2012.188.

[77] W. R. Claycomb and A. Nicoll, “Insider threats to cloud computing: Directions for new research challenges,” *Proc. - Int. Comput. Softw. Appl. Conf.*, pp. 387–394, 2012, doi: 10.1109/COMPSAC.2012.113.

[78] “Top Threats to Cloud Computing: Egregious Eleven,” 2019. [Online]. Available: https://cloudsecurityalliance.org/artifacts/top-threats-to-cloud-computing-egregious-eleven/. [Accessed: 30-Jun-2021].

[79] G. S. Sadasivam, K. A. Kumari, and S. Rubika, “A novel authentication service for hadoop in cloud environment,” *IEEE Cloud Comput. Emerg. Mark. CCEM 2012 - Proc.*, pp. 23–28, 2012, doi: 10.1109/CCEM.2012.6354591.

[80] N. Somu, A. Gangaa, and V. S. Shankar Sriram, “Authentication service in hadoop using one time pad,” *Indian J. Sci. Technol.*, vol. 7, no. April, pp. 56–62, 2014.

[81] “Active Directory Federation Services.” [Online]. Available: https://docs.microsoft.com/en-us/windows-server/identity/active-directory-federation-services. [Accessed: 14-Feb-2020].

[82] X. Huang, Y. Xiang, E. Bertino, J. Zhou, and L. Xu, “Robust multi-factor authentication for fragile communications,” *IEEE Trans. Dependable Secur. Comput.*, vol. 11, no. 6, pp. 568–581, 2014, doi: 10.1109/TDSC.2013.2297110.

[83] W. Liu, A. S. Uluagac, and R. Beyah, “MACA: A privacy-preserving multi-factor cloud authentication system utilizing big data,” in *2014 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2014, pp. 518–523, doi: 10.1109/INFCOMW.2014.6849285.

[84] I. Lahmer and N. Zhang, “Towards a Virtual Domain Based Authentication on MapReduce,” *IEEE Access*, vol. 4, pp. 1658–1675, 2016, doi: 10.1109/ACCESS.2016.2558456.

[85] Q. H. Dang, “Secure Hash Standard,” Gaithersburg, MD, Jul. 2015.

[86] M. J. Dworkin, “SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions,” Gaithersburg, MD, Jul. 2015.

[87] “The BLAKE2 Cryptographic Hash and Message Authentication Code (MAC),” Nov. 2015.

[88] Chung Kei Wong and S. S. Lam, “Digital signatures for flows and multicasts,” in *Proceedings Sixth International Conference on Network Protocols (Cat. No.98TB100256)*, 1999, vol. 7, no. 4, pp. 198–209, doi: 10.1109/ICNP.1998.723740.

[89] R. C. Merkle, “Method of providing digital signatures,” US4309569A, 1982.

[90] R. C. Merkle, “A Certified Digital Signature,” in *Advances in Cryptology — CRYPTO’ 89 Proceedings*, vol. 435 LNCS, New York, NY: Springer New York, 1990, pp. 218–238.

[91] H. Krawczyk and P. Eronen, “HMAC-based Extract-and-Expand Key Derivation Function (HKDF),” May 2010.

[92] “Specification for the Advanced Encryption Standard (AES).” 2001.

[93] B. Schneier, “Description of a new variable-length key, 64-bit block cipher (Blowfish),” 1994, pp. 191–204.

[94] R. L. Rivest, M. J. B. Robshaw, R. Sidney, and Y. L. Yin, “The RC6 Block Cipher,” in *in First Advanced Encryption Standard (AES) Conference*, 1998, p. 16.

[95] R. L. Rivest, A. Shamir, and L. Adleman, “A method for obtaining digital signatures and public-key cryptosystems,” *Commun. ACM*, vol. 21, no. 2, pp. 120–126, Feb. 1978, doi: 10.1145/359340.359342.

[96] T. Elgamal, “A public key cryptosystem and a signature scheme based on discrete logarithms,” *IEEE Trans. Inf. Theory*, vol. 31, no. 4, pp. 469–472, Jul. 1985, doi: 10.1109/TIT.1985.1057074.

[97] T. Krovetz, Ed., “UMAC: Message Authentication Code using Universal Hashing,” Mar. 2006.

[98] “Digital Signature Standard (DSS),” Gaithersburg, MD, Jul. 2013.

[99] “Amazon Web Services (AWS).” [Online]. Available: https://aws.amazon.com/. [Accessed: 23-Sep-2020].

[100] “Digital Ocean.” [Online]. Available: https://www.digitalocean.com/. [Accessed: 16-Feb-2017].

[101] “Amazon Elastic Compute Cloud (EC2).” [Online]. Available: https://aws.amazon.com/ec2/. [Accessed: 23-Sep-2020].

[102] “Amazon Simple Storage Service (S3).” [Online]. Available: https://aws.amazon.com/s3/. [Accessed: 23-Sep-2020].

[103] “Amazon EMR.” [Online]. Available: https://aws.amazon.com/emr/. [Accessed: 01-Sep-2020].

[104] T. Gunarathne, T. L. Wu, J. Qiu, and G. Fox, “MapReduce in the clouds for science,” *Proc. - 2nd IEEE Int. Conf. Cloud Comput. Technol. Sci. CloudCom 2010*, vol. 2, pp. 565–572, 2010, doi: 10.1109/CloudCom.2010.107.

[105] T. Gunarathne, B. Zhang, T. L. Wu, and J. Qiu, “Scalable parallel computing on clouds using Twister4Azure iterative MapReduce,” *Futur. Gener. Comput. Syst.*, vol. 29, no. 4, pp. 1035–1048, 2012, doi: 10.1016/j.future.2012.05.027.

[106] A. N. Toosi, R. N. Calheiros, and R. Buyya, “Interconnected Cloud Computing Environments,” *ACM Comput. Surv.*, vol. 47, no. 1, pp. 1–47, May 2014, doi: 10.1145/2593512.

[107] “How We Extended CloudFlare’s Performance and Security Into Mainland China.” [Online]. Available: https://blog.cloudflare.com/how-we-extended-cloudflares-performance-and-security-into-mainland-china/. [Accessed: 16-Dec-2020].

[108] “GraphLab.” [Online]. Available: https://turi.com/. [Accessed: 29-Nov-2020].

[109] S. Melnik *et al.*, “Dremel: interactive analysis of web-scale datasets,” *Proc. VLDB Endow.*, vol. 3, no. 1–2, pp. 330–339, Sep. 2010, doi: 10.14778/1920841.1920886.

[110] G. Malewicz *et al.*, “Pregel: A System for Large-Scale Graph Processing,” in *Proceedings of the 2010 international conference on Management of data - SIGMOD ’10*, 2010, p. 135, doi: 10.1145/1807167.1807184.

[111] “Apache Storm.” [Online]. Available: https://storm.apache.org/. [Accessed: 11-Nov-2020].

[112] J. Dean and S. Ghemawat, “MapReduce: Simplified Data Processing on Large Clusters,” *Commun. ACM*, vol. 51, no. 1, pp. 107–113, 2008, doi: 10.1145/1327452.1327492.

[113] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, “Dryad: Distributed Data-Parallel Programs from Sequential Building Blocks,” in *Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007 - EuroSys ’07*, 2007, p. 59, doi: 10.1145/1272996.1273005.

[114] V. Borkar, M. Carey, R. Grover, N. Onose, and R. Vernica, “Hyracks: A flexible and extensible foundation for data-intensive computing,” in *2011 IEEE 27th International Conference on Data Engineering*, 2011, pp. 1151–1162, doi: 10.1109/ICDE.2011.5767921.

[115] D. Warneke and O. Kao, “Nephele: efficient parallel data processing in the cloud,” in *Proceedings of the 2nd Workshop on Many-Task Computing on Grids and Supercomputers - MTAGS ’09*, 2009, pp. 1–10, doi: 10.1145/1646468.1646476.

[116] “Apache Spark.” [Online]. Available: https://spark.apache.org/. [Accessed: 11-Nov-2020].

[117] J. Ekanayake, S. Pallickara, and G. Fox, “MapReduce for Data Intensive Scientific Analyses,” in *2008 IEEE Fourth International Conference on eScience*, 2008, pp. 277–284, doi: 10.1109/eScience.2008.59.

[118] O. Boykin, S. Ritchie, I. O ’connell, and J. Lin, “Summingbird: A Framework for Integrating Batch and Online MapReduce Computations,” *Proc. VLDB Endow.*, vol. 7, no. 13, pp. 1441--1451, 2014, doi: 10.14778/2733004.2733016.

[119] M. Zaharia *et al.*, “Apache Spark,” *Commun. ACM*, vol. 59, no. 11, pp. 56–65, Oct. 2016, doi: 10.1145/2934664.

[120] “Apache Hadoop.” [Online]. Available: https://hadoop.apache.org/. [Accessed: 01-Sep-2020].

[121] “Hortonworks Data Platform (HDP).” [Online]. Available: http://hortonworks.com/products/data-center/hdp/. [Accessed: 08-Feb-2017].

[122] “MapR Converged Data Platform.” [Online]. Available: https://www.mapr.com/products/mapr-converged-data-platform. [Accessed: 09-Feb-2017].

[123] V. Kumar, H. Andrade, B. Gedik, and K.-L. Wu, “DEDUCE: at the intersection of MapReduce and stream processing,” in *Proceedings of the 13th International Conference on Extending Database Technology - EDBT ’10*, 2010, p. 657, doi: 10.1145/1739041.1739120.

[124] Y. Zhang, Q. Gao, L. Gao, and C. Wang, "iMapReduce: A Distributed Computing Framework for Iterative Computation," *J. Grid Comput.*, vol. 10, no. 1, pp. 47–68, Mar. 2012, doi: 10.1007/s10723-012-9204-9.

[125] Y. Y. M. I. D. Fetterly, M. Budiu, Ú. Erlingsson, and P. K. G. J. Currey, "DryadLINQ: A system for general-purpose distributed data-parallel computing using a high-level language," *Proc. LSDS-IR*, vol. 8, 2009.

[126] B. He *et al.*, "Comet: batched stream processing for data intensive distributed computing," in *Proceedings of the 1st ACM symposium on Cloud computing - SoCC '10*, 2010, p. 63, doi: 10.1145/1807128.1807139.

[127] "Hyracks." [Online]. Available: https://code.google.com/archive/p/hyracks/. [Accessed: 01-Dec-2020].

[128] "Apache AsterixDB." [Online]. Available: https://asterixdb.apache.org/index.html. [Accessed: 01-Dec-2020].

[129] D. Battré, S. Ewen, F. Hueske, O. Kao, V. Markl, and D. Warneke, "Nephele/PACTs: A Programming Model and Execution Framework for Web-Scale Analytical Processing," in *Proceedings of the 1st ACM symposium on Cloud computing - SoCC '10*, 2010, p. 119, doi: 10.1145/1807128.1807148.

[130] P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, S. Haridi, and K. Tzoumas, "Apache Flink: Stream and batch processing in a single engine," *Bull. IEEE Comput. Soc. Tech. Comm. Data Eng.*, vol. 36, no. 4, 2015.

[131] "Apache Flink." [Online]. Available: https://flink.apache.org/. [Accessed: 11-Nov-2020].

[132] V. K. Vavilapalli *et al.*, "Apache Hadoop yarn: Yet another resource negotiator," *Annu. Symp. Cloud Comput.*, p. 5, 2013, doi: 10.1145/2523616.2523633.

[133] T. White, *Hadoop: The Definitive Guide 4rd edition*. 2015.

[134] T. C. Group, "Trusted Computing." [Online]. Available: https://trustedcomputinggroup.org/trusted-computing/. [Accessed: 03-May-2019].

[135] Savitha and Vijaya, "Mining of Web Server Logs in a Distributed Cluster Using Big Data Technologies," *Int. J. Adv. Comput. Sci. Appl.*, vol. 5, no. 1, pp. 137–142, 2014.

[136] M. Kumar and M. Hanumanthappa, "Scalable intrusion detection systems log analysis using cloud computing infrastructure," *2013 IEEE Int. Conf. Comput. Intell. Comput. Res.*, pp. 1–4, 2013, doi: 10.1109/ICCIC.2013.6724158.

[137] J. Therdphapiyanak and K. Piromsopa, "Applying Hadoop for log analysis toward distributed IDS," *Proc. 7th Int. Conf. Ubiquitous Inf. Manag. Commun. - ICUIMC '13*, pp. 1–6, 2013, doi: 10.1145/2448556.2448559.

[138] B. C. Neuman and T. Ts'o, "Kerberos: an authentication service for computer networks," *IEEE Commun. Mag.*, vol. 32, no. 9, pp. 33–38, Sep. 1994, doi: 10.1109/35.312841.

[139] J. T. Kohl, B. C. Neuman, and T. Y. Ts'o, "The evolution of the Kerberos authentication service," 1994, pp. 78–94.

[140] J. Hughes and E. Maler, “Security Assertion Markup Language (SAML) V2. 0 Technical Overview,” 2008.

[141] C. Metz, “AAA protocols: Authentication, authorization, and accounting for the internet,” *IEEE Internet Comput.*, vol. 3, no. 6, pp. 75–79, 1999, doi: 10.1109/4236.807015.

[142] A. Hosia, “Comparison between RADIUS and Diameter,” *Changes*, vol. 1, p. 2, 2003.

[143] “eduroam.” [Online]. Available: https://www.eduroam.org/. [Accessed: 12-Feb-2020].

[144] K. Wierenga and L. Florio, “Eduroam: past, present and future,” *Comput. Methods Sci. Technol.*, vol. 11, no. 2, pp. 169–173, 2005, doi: 10.12921/cmst.2005.11.02.169-173.

[145] J. Liu, S. Jiang, and L. Hicks, “Introduction to Diameter Get the next generation AAA protocol.” pp. 1–12, 2006.

[146] R. Stewart, Ed., “Stream Control Transmission Protocol,” Sep. 2007.

[147] S. Kent and K. Seo, “Security Architecture for the Internet Protocol,” Dec. 2005.

[148] T. Dierks and E. Rescorla, “The Transport Layer Security (TLS) Protocol Version 1.2 - RFC5246,” Aug. 2008.

[149] P. V. Mockapetris, “Domain names - concepts and facilities,” Nov. 1987.

[150] J. Zhao *et al.*, “A security framework in G-Hadoop for big data computing across distributed Cloud data centres,” *J. Comput. Syst. Sci.*, vol. 80, no. 5, pp. 994–1007, 2014, doi: 10.1016/j.jcss.2014.02.006.

[151] L. Wang *et al.*, “G-Hadoop: MapReduce across distributed data centers for data-intensive computing,” *Futur. Gener. Comput. Syst.*, vol. 29, no. 3, pp. 739–750, 2013, doi: 10.1016/j.future.2012.09.001.

[152] A. Freier, P. Karlton, and P. Kocher, “The Secure Sockets Layer (SSL) Protocol Version 3.0 - RFC6101,” Aug. 2011.

[153] Z. Quan, D. Xiao, D. Wu, C. Tang, and C. Rong, “TSHC: Trusted Scheme for Hadoop Cluster,” in *2013 Fourth International Conference on Emerging Intelligent Data and Web Technologies*, 2013, pp. 344–349, doi: 10.1109/EIDWT.2013.66.

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

[155] A. Ruan and A. Martin, “TMR: Towards a trusted MapReduce infrastructure,” *Proc. - 2012 IEEE 8th World Congr. Serv. Serv. 2012*, pp. 141–148, 2012, doi: 10.1109/SERVICES.2012.28.

[156] I. Khalil, Z. Dou, and A. Khreishah, “TPM-Based Authentication Mechanism for Apache Hadoop,” vol. 152, J. Tian, J. Jing, and M. Srivatsa, Eds. Cham: Springer International Publishing, 2015, pp. 105–122.

[157] C. A. Meadows and C. A. Meadows, “Formal verification of cryptographic protocols: A survey,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 917, J. Pieprzyk and R. Safavi-Naini, Eds. Berlin, Heidelberg: Springer, 1995, pp. 133–150.

[158] C. Meadows, “Formal methods for cryptographic protocol analysis: emerging issues and trends,” *IEEE J. Sel. Areas Commun.*, vol. 21, no. 1, pp. 44–54, Jan. 2003, doi: 10.1109/JSAC.2002.806125.

[159] R. M. Needham and M. D. Schroeder, “Using encryption for authentication in large networks of computers,” *Commun. ACM*, vol. 21, no. 12, pp. 993–999, Dec. 1978, doi: 10.1145/359657.359659.

[160] G. Lowe, “Casper: A compiler for the analysis of security protocols,” *J. Comput. Secur.*, vol. 6, no. 1–2, pp. 53–84, 1998.

[161] B. Roscoe, “Model-checking CSP.” Prentice-Hall, 1994.

[162] R. Kemmerer, C. Meadows, and J. Millen, “Three systems for cryptographic protocol analysis,” *J. Cryptol.*, vol. 7, no. 2, pp. 79–130, Jun. 1994, doi: 10.1007/BF00197942.

[163] L. Viganò, “Automated Security Protocol Analysis With the AVISPA Tool,” *Electron. Notes Theor. Comput. Sci.*, vol. 155, no. 1, pp. 61–86, May 2006, doi: 10.1016/j.entcs.2005.11.052.

[164] V. Cortier, S. Kremer, and B. Warinschi, “A Survey of Symbolic Methods in Computational Analysis of Cryptographic Systems,” *J. Autom. Reason.*, vol. 46, no. 3–4, pp. 225–259, Apr. 2011, doi: 10.1007/s10817-010-9187-9.

[165] B. Blanchet, “Security Protocol Verification: Symbolic and Computational Models,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 7215 LNCS, 2012, pp. 3–29.

[166] D. L. Dill, A. J. Drexler, A. J. Hu, and C. H. Yang, “Protocol Verification as a Hardware Design Aid.,” in *ICCD*, 1992, vol. 92, pp. 522–525.

[167] Dawn Xiaodong Song, “Athena: a new efficient automatic checker for security protocol analysis,” in *Proceedings of the 12th IEEE Computer Security Foundations Workshop*, pp. 192–202, doi: 10.1109/CSFW.1999.779773.

[168] C. J. F. Cremers, “The Scyther Tool: Verification, Falsification, and Analysis of Security Protocols,” in *Computer Aided Verification*, Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 414–418.

[169] B. Blanchet, M. Abadi, and C. Fournet, “Automated verification of selected equivalences for security protocols,” *J. Log. Algebr. Program.*, vol. 75, no. 1, pp. 3–51, Feb. 2008, doi: 10.1016/j.jlap.2007.06.002.

[170] S. Meier, B. Schmidt, C. Cremers, and D. Basin, “The TAMARIN Prover for the Symbolic Analysis of Security Protocols,” 2013, pp. 696–701.

[171] C. A. R. Hoare, *Communicating Sequential Processes*. Prentice Hall, 1985.

[172] C. J. F. Cremers, P. Lafourcade, and P. Nadeau, “Comparing State Spaces in Automatic Security Protocol Analysis,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 5458 LNCS, 2009, pp. 70–94.

[173] “The AVISPA Project.” [Online]. Available: http://www.avispa-project.org/. [Accessed: 14-Jun-2020].

[174] “Security Protocols Open Repository.” [Online]. Available: http://www.lsv.fr/Software/spore/. [Accessed: 14-Jun-2020].

[175] B. LaMacchia, K. Lauter, and A. Mityagin, “Stronger Security of Authenticated Key Exchange,” in *Provable Security*, Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 1–16.

[176] C. Cremers and S. Mauw, *Operational Semantics and Verification of Security Protocols*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012.

[177] W. Stallings, *Cryptography and Network Security: Principles and Practice*. Pearson Education, 2017.

[178] A. Menezes, P. van Oorschot, and S. Vanstone, *Handbook of Applied Cryptography*, vol. 19964964. CRC Press, 1996.

[179] B. Lakhe, “Introducing Hadoop Security,” in *Practical Hadoop Security*, Berkeley, CA: Apress, 2014, pp. 37–47.

[180] “Botan: Crypto and TLS for Modern C++.” [Online]. Available: https://botan.randombit.net/. [Accessed: 21-Aug-2020].

[181] “Users of Botan.” [Online]. Available: https://github.com/randombit/botan/wiki/Users. [Accessed: 21-Aug-2020].

[182] E. Barker, “Recommendation for key management:,” Gaithersburg, MD, May 2020.

[183] Y. Desmedt, Y. Frankel, and M. Yung, “Multi-receiver/multi-sender network security: efficient authenticated multicast/feedback,” in *[Proceedings] IEEE INFOCOM ’92: The Conference on Computer Communications*, 1992, pp. 2045–2054 vol.3, doi: 10.1109/INFCOM.1992.263476.

[184] R. Safavi-Naini and H. Wang, “New results on multi-receiver authentication codes,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 1403, 1998, pp. 527–541.

[185] R. Safavi-Naini and H. Wang, “Multireceiver Authentication Codes: Models, Bounds, Constructions, and Extensions,” *Inf. Comput.*, vol. 151, no. 1–2, pp. 148–172, May 1999, doi: 10.1006/inco.1998.2769.

[186] R. Canetti, J. Garay, G. Itkis, D. Micciancio, M. Naor, and B. Pinkas, “Multicast security: a taxonomy and some efficient constructions,” in *IEEE INFOCOM ’99. Conference on Computer Communications. Proceedings. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. The Future is Now (Cat. No.99CH36320)*, 1999, pp. 708–716 vol.2, doi: 10.1109/INFCOM.1999.751457.

[187] F. Bergadano, D. Cavagnino, and B. Crispo, “Individual single source authentication on the MBONE,” in *2000 IEEE International Conference on Multimedia and Expo. ICME2000. Proceedings. Latest Advances in the Fast Changing World of Multimedia (Cat. No.00TH8532)*, 2000, vol. 1, no. c, pp. 541–544, doi: 10.1109/ICME.2000.869659.

[188] A. Perrig, R. Canetti, J. D. Tygar, and D. Song, “The TESLA Broadcast Authentication Protocol,” *CryptoBytes*, vol. 5, no. 2, pp. 2–13, 2002.

[189] A. Perrig, R. Canetti, D. Song, and J. D. Tygar, “Efficient and secure source authentication for multicast,” in *Network and Distributed System Security Symposium, NDSS*, 2001, vol. 1, no. 2001, pp. 35–46.

[190] A. Perrig *et al.*, “SPINS: Security Protocols for Sensor Networks SPINS : Security Protocols for Sensor Networks,” *Wirel. Networks*, vol. 8, no. September, pp. 521–534, 2002, doi: 10.1023/A:1016598314198.

[191] E. Barker, “Recommendation for Key Management Part 1: General,” Gaithersburg, MD, Jan. 2016.

[192] “Elliptic Curve Cryptography - OpenSSLWiki.” [Online]. Available: https://wiki.openssl.org/index.php/Elliptic_Curve_Cryptography. [Accessed: 28-May-2018].

[193] “Keylength - Cryptographic Key Length Recommendation.” [Online]. Available: https://www.keylength.com/en/. [Accessed: 28-May-2018].

[194] R. Gennaro and P. Rohatgi, “How to Sign Digital Streams,” *Inf. Comput.*, vol. 165, no. 1, pp. 100–116, Feb. 2001, doi: 10.1006/inco.2000.2916.

[195] J. M. Park, E. K. P. Chong, and H. J. Siegel, “Efficient multicast packet authentication using signature amortization,” in *Proceedings 2002 IEEE Symposium on Security and Privacy*, 2002, vol. 2002-Janua, pp. 227–240, doi: 10.1109/SECPRI.2002.1004374.

[196] J. M. Park, E. K. P. Chong, and H. J. Siegel, “Efficient multicast stream authentication using erasure codes,” *ACM Trans. Inf. Syst. Secur.*, vol. 6, no. 2, pp. 258–285, May 2003, doi: 10.1145/762476.762480.

[197] J. Xu, X. Zhou, J. Han, F. Li, and F. Zhou, “Data Authentication Model Based on Reed-solomon Error-correcting Codes in Wireless Sensor Networks,” *IETE Tech. Rev.*, vol. 30, no. 3, p. 191, 2013, doi: 10.4103/0256-4602.113496.

[198] B. Zhang, B. Dong, and W. H. Wang, “AssureMR: Verifiable SQL Execution on MapReduce,” in *2018 IEEE 34th International Conference on Data Engineering (ICDE)*, 2018, no. 1, pp. 1228–1231, doi: 10.1109/ICDE.2018.00117.

[199] B. Zhang, B. Dong, and H. Wang, “CorrectMR: Authentication of Distributed SQL Execution on MapReduce,” *IEEE Trans. Knowl. Data Eng.*, vol. PP, no. c, pp. 1–1, 2019, doi: 10.1109/TKDE.2019.2935968.

[200] H. Ulusoy, M. Kantarcioglu, and E. Pattuk, “TrustMR: Computation integrity assurance system for MapReduce,” in *2015 IEEE International Conference on Big Data (Big Data)*, 2015, pp. 441–450, doi: 10.1109/BigData.2015.7363785.

[201] Y. Ding, H. Wang, P. Shi, H. Fu, C. Guo, and M. Zhang, "Trusted sampling-based result verification on mass data processing," *Proc. - 2013 IEEE 7th Int. Symp. Serv. Syst. Eng. SOSE 2013*, pp. 391–396, 2013, doi: 10.1109/SOSE.2013.65.

[202] Z. Xiao and Y. Xiao, "Accountable MapReduce in cloud computing," *Comput. Commun. Work. (INFOCOM WKSHPS), 2011 IEEE Conf.*, pp. 1082–1087, 2011, doi: 10.1109/INFCOMW.2011.5928788 M4 - Citavi.

[203] Z. Xiao and Y. Xiao, "Achieving Accountable MapReduce in cloud computing," *Futur. Gener. Comput. Syst.*, vol. 30, no. 1, pp. 1–13, 2014, doi: 10.1016/j.future.2013.07.001.

[204] W. Wei, J. Du, T. Yu, and X. Gu, "SecureMR: A service integrity assurance framework for MapReduce," *Proc. - Annu. Comput. Secur. Appl. Conf. ACSAC*, pp. 73–82, 2009, doi: 10.1109/ACSAC.2009.17.

[205] A. K. Lenstra, "Key Length: Contribution to The Handbook of Information Security," pp. 1–32, 2004.

[206] M. Abdalla *et al.*, "Algorithms, key size and parameters report 2014," 2014.

[207] H. Orman and P. Hoffman, "Determining Strengths For Public Keys Used For Exchanging Symmetric Keys," Apr. 2004.

[208] "MapReduce Lite." [Online]. Available: https://github.com/wangkuiyi/mapreduce-lite. [Accessed: 28-Aug-2018].

[209] S. Kelly and S. Frankel, "Using HMAC-SHA-256, HMAC-SHA-384, and HMAC-SHA-512 with IPsec," May 2007.

[210] O. Sury, "Use of the SHA-256 Algorithm with RSA, Digital Signature Algorithm (DSA), and Elliptic Curve DSA (ECDSA) in SSHFP Resource Records," Apr. 2012.

[211] M. J. Menne, I. Durre, R. S. Vose, B. E. Gleason, and T. G. Houston, "An Overview of the Global Historical Climatology Network-Daily Database," *J. Atmos. Ocean. Technol.*, vol. 29, no. 7, pp. 897–910, Jul. 2012, doi: 10.1175/JTECH-D-11-00103.1.

[212] M. J. Menne *et al.*, "Global Historical Climatology Network - Daily (GHCN-Daily), Version 3.25," 2012. [Online]. Available: http://doi.org/10.7289/V5D21VHZ. [Accessed: 27-Aug-2018].

Appendix A
Symbolic Analysis Source Codes

The contents of the three SPDL files (gp2a.spdl, ge2a.spdl, and soa.spdl) used for symbolic analysis of the MIEA protocols are shown as follows.

gp2a.spdl

```
/*
 * Group key and Pre-shared primary key Two-factor Authentication (GP2A) protocol
 */
// -- Custom types
usertype Protocol;
usertype MessageID;
usertype MessageType;
usertype PayloadSize;
usertype DomainID;
usertype JobID;
usertype ActionRequest;
usertype InitiatorClass;
usertype Credential;      // keys exchanged in an RP message

// -- Functions
// for the built-in encryption function: {y}x means a data item y encrypted with a key x
const DID: Function;      // DID(x) returns the DomainID of x
hashfunction HKDF;     // HKDF(x, y) returns the key derived from x and y

// -- Authentication keys;
secret pmk: Function;     // pre-shared primary key
secret gk: Function;      // group key

// -- Static constants
const GP2A: Protocol;
const CH: MessageType;
const RC: MessageType;
const RP: MessageType;

const psize1: PayloadSize;
const psize2: PayloadSize;
const psize3: PayloadSize;

// -- Macros (for brevity)
// I and R are, respectively, used as idI and idR
// authenticators containing challenges and responses and encrypted with authentication keys
macro auth1 = {{n1}pmk(I, R)}gk;
macro auth2 = {{n1, n2}pmk(I, R)}gk;
macro auth3 = {{n2}pmk(I, R)}gk;

// MAC key and credential encryption key
macro mkIR = HKDF(pmk(I, R), gk);
macro ckIR = HKDF(pmk(I, R), n2);

// an encryption based approach is used to generate MAC tags
macro tag1 = {mid1, jid, req, icl, auth1}mkIR;
macro tag2 = {mid2, mid1, auth2}mkIR;
macro tag3 = {mid3, mid2, auth3, {creds}ckIR}mkIR;

macro msg1 = (GP2A, mid1, CH, psize1, I, DID(I), R, DID(R), jid, req, icl, auth1, tag1);
macro msg2 = (GP2A, mid2, RC, psize2, R, DID(R), I, DID(I), mid1, auth2, tag2);
macro msg3 = (GP2A, mid3, RP, psize3, I, DID(I), R, DID(R), mid2, auth3, {creds}ckIR, tag3);

protocol GP2A-protocol(I,R)
{
    // -- Roles
    role I    // for initiators
    {
        // -- Fresh constants
        fresh mid1: MessageID;
        fresh jid: JobID;
        fresh req: ActionRequest;
        fresh icl: InitiatorClass;
        fresh n1: Nonce;

        fresh mid3: MessageID;
        fresh creds: Credential;

        // -- Variables
        var mid2: MessageID;
        var n2: Nonce;

        // -----------------------------------------
        // Step 1: Send a CH message (msg1).
        send_msg1(I, R, msg1);

        // -----------------------------------------
        // Step 3: Receive the RC message (msg2),
        //         and send an RP message (msg3).
        recv_msg2(R, I, msg2);
        send_msg3(I, R, msg3);

        // -----------------------------------------
        // -- Claims
        claim_I1(I, Nisynch);
        claim_I2(I, Secret, gk);
        claim_I3(I, Secret, pmk(I, R));
        claim_I4(I, Secret, mkIR);
        claim_I5(I, Secret, ckIR);
        claim_I6(I, Secret, n1);
        claim_I7(I, Secret, n2);
        claim_I8(I, Secret, creds);
    }

    role R    // for respondents
    {
        // -- Fresh constants
        fresh mid2: MessageID;
        fresh n2: Nonce;

        // -- Variables
        var mid1: MessageID;
        var jid: JobID;
        var req: ActionRequest;
        var icl: InitiatorClass;
        var n1: Nonce;

        var mid3: MessageID;
        var creds: Credential;

        // -----------------------------------------
        // Step 2: Receive the CH message (msg1),
        //         and send an RC message (msg2).
        recv_msg1(I, R, msg1);
        send_msg2(R, I, msg2);

        // -----------------------------------------
        // Step 4: Receive the RP message (msg3).
        recv_msg3(I, R, msg3);

        // -----------------------------------------
        // -- Claims
        claim_R1(R, Nisynch);
        claim_R2(R, Secret, gk);
        claim_R3(R, Secret, pmk(I, R));
        claim_R4(R, Secret, mkIR);
        claim_R5(R, Secret, ckIR);
        claim_R6(R, Secret, n1);
        claim_R7(R, Secret, n2);
        claim_R8(R, Secret, creds);
    }
}
```

ge2a.spdl

```
/*
 * Group key and Encapsulated primary key Two-factor Authentication (GE2A) protocol
 */
// -- Custom types
usertype Protocol;
usertype MessageID;
usertype MessageType;
usertype PayloadSize;
usertype DomainID;
usertype JobID;
usertype ActionRequest;
usertype InitiatorClass;
usertype Credential;      // keys exchanged in an RP message
usertype Timestamp;
usertype PrimaryKey;

// -- Functions
// for the built-in encryption function: {y}x means a data item y encrypted with a key x
```

```
const DID: Function;      // DID(x) returns the DomainID of x
hashfunction HKDF;     // HKDF(x, y) returns the key derived from x and y

// -- Authentication keys;
// for the built-in secret key: k(x, y) means a secret key shared between x and y
secret slk: Function;     // sealing key
secret gk: Function;      // group key

// -- Static constants
const GE2A: Protocol;
const CH: MessageType;
const RC: MessageType;
const RP: MessageType;

const psize1: PayloadSize;
const psize2: PayloadSize;
const psize3: PayloadSize;

// -- Macros (for brevity)
// I and R are, respectively, used as idI and idR
macro tkt = {I, DID(I), R, DID(R), Z, DID(Z), jid, req, gt, et, pmkIR}slk(R, Z);

// messages for transmitting pmkIR and tkt
macro msg0A = ({jid, req, R}k(I, Z));
macro msg0B = ({pmkIR}k(I, Z), tkt);

// authenticators containing challenges and responses and encrypted with authentication keys
macro auth1 = {{n1}pmkIR}gk;
macro auth2 = {{n1, n2}pmkIR}gk;
macro auth3 = {{n2}pmkIR}gk;

// MAC key and credential encryption key
macro mkIR = HKDF(pmkIR, gk);
macro ckIR = HKDF(pmkIR, n2);

// an encryption based approach is used to generate MAC tags
macro tag1 = {mid1, jid, req, icl, auth1, tkt}mkIR;
macro tag2 = {mid2, mid1, auth2}mkIR;
macro tag3 = {mid3, mid2, auth3, {creds}ckIR}mkIR;

macro msg1 = (GE2A, mid1, CH, psize1, I, DID(I), R, DID(R), jid, req, icl, auth1, tkt, tag1);
macro msg2 = (GE2A, mid2, RC, psize2, R, DID(R), I, DID(I), mid1, auth2, tag2);
macro msg3 = (GE2A, mid3, RP, psize3, I, DID(I), R, DID(R), mid2, auth3, {creds}ckIR, tag3);

protocol GE2A-protocol(I,R,Z)
{
    // -- Roles
    role I    // for initiators
    {
        // -- Fresh constants
        fresh mid1: MessageID;
        fresh jid: JobID;
        fresh req: ActionRequest;
        fresh icl: InitiatorClass;
        fresh n1: Nonce;

        fresh mid3: MessageID;
        fresh creds: Credential;

        // -- Variables
        var mid2: MessageID;
        var n2: Nonce;

        var et: Timestamp;
        var gt: Timestamp;
        var pmkIR: PrimaryKey;   // issued by Z

        // ------------------------------------------
        // Step 0-A: Send a message (msg0A) to Z to request a ticket for authentication to R.
        send_msg0A(I, Z, msg0A);

        // ------------------------------------------
        // Step 0-C: Recieve a message (msg0B).
        recv_msg0B(Z, I, msg0B);

        // ------------------------------------------
        // Step 1: Send a CH message (msg1).
        send_msg1(I, R, msg1);

        // ------------------------------------------
        // Step 3: Receive the RC message (msg2),
        //         and send an RP message (msg3).
        recv_msg2(R, I, msg2);
        send_msg3(I, R, msg3);

        // ------------------------------------------
        // -- Claims
        claim_I1(I, Nisynch);
        claim_I2(I, Secret, gk);
        claim_I3(I, Secret, pmkIR);
        claim_I4(I, Secret, mkIR);
        claim_I5(I, Secret, ckIR);
        claim_I6(I, Secret, n1);
        claim_I7(I, Secret, n2);
        claim_I8(I, Secret, creds);
    }

    role R    // for respondents
    {
        // -- Fresh constants
        fresh mid2: MessageID;
        fresh n2: Nonce;

        // -- Variables
        var mid1: MessageID;
        var jid: JobID;
        var req: ActionRequest;
        var icl: InitiatorClass;
        var n1: Nonce;

        var mid3: MessageID;
        var creds: Credential;

        var gt: Timestamp;
        var et: Timestamp;
        var pmkIR: PrimaryKey;   // contained in tkt in msg1

        // -----------------------------------------
        // Step 2: Receive the CH message (msg1),
        //         and send an RC message (msg2).
        recv_msg1(I, R, msg1);
        send_msg2(R, I, msg2);

        // -----------------------------------------
        // Step 4: Receive the RP message (msg3).
        recv_msg3(I, R, msg3);

        // -----------------------------------------
        // -- Claims
        claim_R1(R, Nisynch);
        claim_R2(R, Secret, gk);
        claim_R3(R, Secret, pmkIR);
        claim_R4(R, Secret, mkIR);
        claim_R5(R, Secret, ckIR);
        claim_R6(R, Secret, n1);
        claim_R7(R, Secret, n2);
        claim_R8(R, Secret, creds);
    }

    role Z    // for a trusted third party
    {
        // -- Fresh constants
        fresh gt: Timestamp;
        fresh et: Timestamp;
        fresh pmkIR: PrimaryKey;    // for authentication between I and R

        // -- Variables
        var jid: JobID;
        var req: ActionRequest;

        // -----------------------------------------
        // Step 0-B: Recieve the request message (msg0A)
        //           and reply a message (msg0B) containing pmkIR and tkt back to I.
        recv_msg0A(I, Z, msg0A);
        send_msg0B(Z, I, msg0B);

        // -----------------------------------------
        // -- Claims
        claim_Z0(Z, Secret, pmkIR);
    }
}
```

soa.spdl

```
/*
 * Secondary key One-factor Authentication (SOA) protocol
 */
// -- Custom types
usertype Protocol;
usertype MessageID;
usertype MessageType;
usertype PayloadSize;
usertype DomainID;
usertype JobID;
usertype ActionRequest;
usertype InitiatorClass;
usertype Credential;      // keys exchanged in an RP message

// -- Functions
// for the built-in encryption function: {y}x means a data item y encrypted with a key x
const DID: Function;      // DID(x) returns the DomainID of x
hashfunction HKDF;     // HKDF(x, y) returns the key derived from x and y

// -- Authentication key;
secret sck: Function;     // secondary key

// -- Static constants
const SOA: Protocol;
const CH: MessageType;
const RC: MessageType;
const RP: MessageType;

const psize1: PayloadSize;
const psize2: PayloadSize;
const psize3: PayloadSize;

// -- Macros (for brevity)
// I and R are, respectively, used as idI and idR
// authenticators containing challenges and responses and encrypted with authentication keys
macro auth1 = {n1}sck(I, R);
macro auth2 = {n1, n2}sck(I, R);
macro auth3 = {n2}sck(I, R);

// MAC key and credential encryption key
macro mkIR = HKDF(sck(I, R));
macro ckIR = HKDF(sck(I, R), n2);

// an encryption based approach is used to generate MAC tags
macro tag1 = {mid1, jid, req, icl, auth1}mkIR;
macro tag2 = {mid2, mid1, auth2}mkIR;
macro tag3 = {mid3, mid2, auth3, {creds}ckIR}mkIR;

macro msg1 = (SOA, mid1, CH, psize1, I, DID(I), R, DID(R), jid, req, icl, auth1, tag1);
macro msg2 = (SOA, mid2, RC, psize2, R, DID(R), I, DID(I), mid1, auth2, tag2);
macro msg3 = (SOA, mid3, RP, psize3, I, DID(I), R, DID(R), mid2, auth3, {creds}ckIR, tag3);

protocol SOA-protocol(I,R)
{
    // -- Roles
    role I    // for initiators
    {
        // -- Fresh constants
        fresh mid1: MessageID;
        fresh jid: JobID;
        fresh req: ActionRequest;
        fresh icl: InitiatorClass;
        fresh n1: Nonce;

        fresh mid3: MessageID;
        fresh creds: Credential;

        // -- Variables
        var mid2: MessageID;
        var n2: Nonce;

        // -----------------------------------------
        // Step 1: Send a CH message (msg1).
        send_msg1(I, R, msg1);

        // -----------------------------------------
        // Step 3: Receive the RC message (msg2),
        //         and send an RP message (msg3).
        recv_msg2(R, I, msg2);
        send_msg3(I, R, msg3);

        // -----------------------------------------
        // -- Claims
        claim_I1(I, Nisynch);
        claim_I3(I, Secret, sck(I, R));
        claim_I4(I, Secret, mkIR);
        claim_I5(I, Secret, ckIR);
        claim_I6(I, Secret, n1);
        claim_I7(I, Secret, n2);
        claim_I8(I, Secret, creds);
    }

    role R    // for respondents
    {
        // -- Fresh constants
        fresh mid2: MessageID;
        fresh n2: Nonce;

        // -- Variables
        var mid1: MessageID;
        var jid: JobID;
        var req: ActionRequest;
        var icl: InitiatorClass;
        var n1: Nonce;

        var mid3: MessageID;
        var creds: Credential;

        // ------------------------------------------
        // Step 2: Receive the CH message (msg1),
        //         and send an RC message (msg2).
        recv_msg1(I, R, msg1);
        send_msg2(R, I, msg2);

        // ------------------------------------------
        // Step 4: Receive the RP message (msg3).
        recv_msg3(I, R, msg3);

        // ------------------------------------------
        // -- Claims
        claim_R1(R, Nisynch);
        claim_R3(R, Secret, sck(I, R));
        claim_R4(R, Secret, mkIR);
        claim_R5(R, Secret, ckIR);
        claim_R6(R, Secret, n1);
        claim_R7(R, Secret, n2);
        claim_R8(R, Secret, creds);
    }
}
```

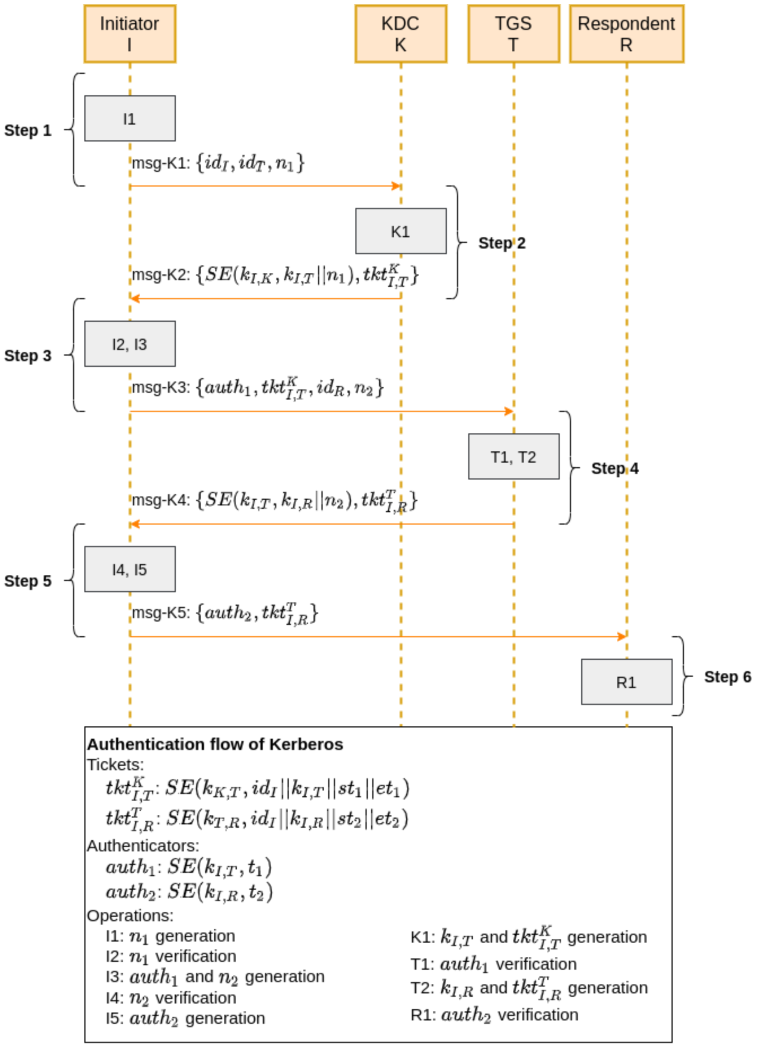
Appendix B

The Execution Flows of the Kerberos and NSLPK Protocols

This section explains the operational steps of Kerberos [34][138][139] and NSLPK [31]. It highlights authentication flows and describes how AuthData are generated and transmitted.

B.1 Kerberos

There are four entities involved in each authentication instance: an initiator I, a Key Distribution Center (KDC) server K, a Ticket-Granting Service (TGS) server T, and a respondent R. K is a server that issues (mid-level) credentials to I for authentication to T, whereas T is a server that issues (bottom-level) credentials to I for authentication to R. The Kerberos protocol consists of 6 operational steps and there are a total of 5 messages exchanged, as shown in Figure B.1.



Initiator I
KDC K
TGS T
Respondent R
Step 1
I1
msg-K1: $\{id_I, id_T, n_1\}$
K1
Step 2
msg-K2: $\{SE(k_{I,K}, k_{I,T}||n_1), tkt^K_{I,T}\}$
Step 3
I2, I3
msg-K3: $\{auth_1, tkt^K_{I,T}, id_R, n_2\}$
T1, T2
Step 4
msg-K4: $\{SE(k_{I,T}, k_{I,R}||n_2), tkt^T_{I,R}\}$
Step 5
I4, I5
msg-K5: $\{auth_2, tkt^T_{I,R}\}$
R1
Step 6
Authentication flow of Kerberos
Tickets:
$tkt^K_{I,T}$: $SE(k_{K,T}, id_I||k_{I,T}||st_1||et_1)$
$tkt^T_{I,R}$: $SE(k_{T,R}, id_I||k_{I,R}||st_2||et_2)$
Authenticators:
$auth_1$: $SE(k_{I,T}, t_1)$
$auth_2$: $SE(k_{I,R}, t_2)$
Operations:
I1: n_1 generation
I2: n_1 verification
I3: $auth_1$ and n_2 generation
I4: n_2 verification
I5: $auth_2$ generation
K1: $k_{I,T}$ and $tkt^K_{I,T}$ generation
T1: $auth_1$ verification
T2: $k_{I,R}$ and $tkt^T_{I,R}$ generation
R1: $auth_2$ verification

Figure B.1: The message transaction flow of Kerberos.

Step 1: In $I1$, I generates a nonce n_1 ($I1$) and sends a request for credentials, i.e., a pairwise key $k_{I,T}$ (for authentication between I and T) and a ticket (encrypted pairwise key) $tkt_{I,T}^{K}$, to K. The message (msg-K1) contains the ID of I id_I, the ID of T id_T and n_1, expressed as: msg-K1: $\{id_I, id_T, n_1\}$.

Step 2: Upon receiving msg-K1, in $K1$, K generates $k_{I,T}$ and $tkt_{I,T}^{K}$, and sends these items back to I. To ensure that the reply message (msg-K2) is not replayed, K encrypts $k_{I,T}$ and n_1 with a pairwise key $k_{I,K}$ shared between I and K. The message contains the encrypted $k_{I,T}$ and n_1 along with $tkt_{I,T}^{K}$, expressed as: msg-K2: $\{SE(k_{I,K}, k_{I,T}||n_1),\ tkt_{I,T}^{K}\}$.

Step 3: After receiving msg-K2, in $I2$, I decrypts the encrypted pairwise key and nonce, expressed as $\left(k_{I,T}\ ||\ n_1'\right) = SD(k_{I,K}, SE(k_{I,K}, k_{I,T}||n_1))$, and checks n_1' with n_1 (generated in **Step 1**). If the result is positive, in $I3$, I generates an authenticator $auth_1$, expressed as $auth_1 = SE(k_{I,T}, t_1)$ where t_1 is a current timestamp, and a nonce n_2, and then sends a request for credentials, i.e., a pairwise key $k_{I,R}$ (for authentication between I and R) and a ticket (encrypted pairwise key) $tkt_{I,R}^{T}$, to T. The message msg-K3 contains $auth_1$, $tkt_{I,T}^{K}$, the ID of R id_R, and n_2, expressed as msg-K3: $\{auth_1, tkt_{I,T}^{K}, id_R, n_2\}$.

Step 4: Upon receiving msg-K3, in $T1, T$ decrypts $tkt_{I,T}^{K}$ to obtain $k_{I,T}$ and the related data, $(id_I||k_{I,T}||st_1||et_1)$ where st_1 and et_1 are, respectively, the creation and expiry times of the key. T checks whether the ticket is for I and the key is not expired. T then uses $k_{I,T}$ to verify $auth_1$ by decrypting $auth_1$ and checking the timestamp, $t_1 = SD(k_{I,T}, auth_1)$. If t_1 is fresh, in $T2$, T prepares the requested credentials for I similar to $K1$ in **Step 2**. The message msg-K4 is expressed as: msg-K4: $\{SE(k_{I,T}, k_{I,R}||n_2),\ tkt_{I,R}^{T}\}$.

Step 5: After receiving msg-K4, in $I4$, I verifies the received $k_{I,R}$ and n_2', and in $I5$, sends a request for service access to R. The processes of $I4$ and $I5$ are similar to $I2$ and $I3$ in **Step 3**, respectively, but the content of the message is different. The generation of $auth_2$ is expressed as $auth_2 = SE(k_{I,R}, t_2)$ where t_2 is a current timestamp. The message msg-K5 is expressed as: msg-K5 $\{auth_2, tkt_{I,R}^{T}\}$.

Step 6: Upon receiving msg-K5, in $R1$, R verifies $auth_2$ using the process similar to $T1$ in **Step 4**.

If the protocol is successfully executed and all verifications are positive, I and R are mutually authenticated. It is worth noting that, when I wants to communicate with a new respondent (e.g., a new service server), I does not have to obtain a new pairwise key and a new ticket from K again as long as $tkt_{I,T}^{K}$ is not expired. I may use $k_{I,T}$ and $tkt_{I,T}^{K}$ to request credentials for the new respondent from T. In this case, **Step 1** and **Step 2** can be skipped and only 3 messages (msg-K3, msg-K4, and msg-K5) are used for authentication.

B.2 NSLPK

There are three entities involved in each authentication instance: an initiator I, a trusted key server Z, and a respondent R. Z certifies (by generating signatures) and issues the public key of I to R, and the public key of R to I. It is assumed that the public key of Z (pk_Z) is certified and known to I and R. The NSLPK protocol consists of 8 operational steps and there are a total of 7 messages exchanged, as shown in Figure B.2.

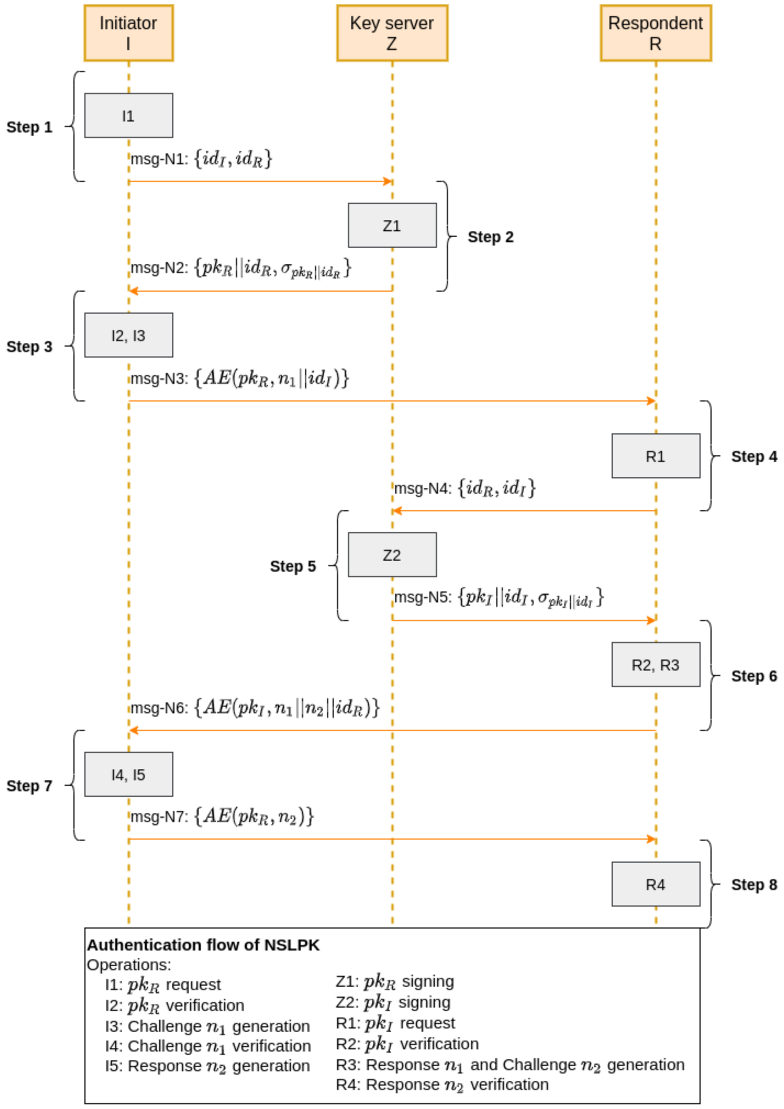

Initiator
I
Key server
Z
Respondent
R
Step 1
I1
msg-N1: $\{id_I, id_R\}$
Z1
Step 2
msg-N2: $\{pk_R||id_R, \sigma_{pk_R||id_R}\}$
Step 3
I2, I3
msg-N3: $\{AE(pk_R, n_1||id_I)\}$
R1
Step 4
msg-N4: $\{id_R, id_I\}$
Step 5
Z2
msg-N5: $\{pk_I||id_I, \sigma_{pk_I||id_I}\}$
R2, R3
Step 6
msg-N6: $\{AE(pk_I, n_1||n_2||id_R)\}$
Step 7
I4, I5
msg-N7: $\{AE(pk_R, n_2)\}$
R4
Step 8
Authentication flow of NSLPK
Operations:
I1: pk_R request
I2: pk_R verification
I3: Challenge n_1 generation
I4: Challenge n_1 verification
I5: Response n_2 generation
Z1: pk_R signing
Z2: pk_I signing
R1: pk_I request
R2: pk_I verification
R3: Response n_1 and Challenge n_2 generation
R4: Response n_2 verification

Figure B.2: The message transaction flow of NSLPK.

Step 1: In $I1$, I sends a request for the public key of R pk_R to Z. The message (msg-N1) contains the ID of I id_I and the ID of R id_R, expressed as: msg-N1: $\{id_I, id_R\}$.

Step 2: Upon receiving msg-N1, in $Z1$, Z signs pk_R || id_R with its private key sk_Z and sends a reply message back to I. The signing process is expressed as $\sigma_{pk_R\,||\,id_R} = SS(sk_Z,$ pk_R || $id_R)$. The message (msg-N2) contains the concatenation of $pk_R||id_R$ and a signature, expressed as: msg-N2: $\{pk_R \,||\, id_R, \sigma_{pk_R\,||\,id_R}\}$.

Step 3: After receiving msg-N2, in $I2$, I verifies the signature with the public key of Z pk_Z and obtains pk_R. The verification process is expressed as $sv = SV(pk_Z,\ pk_R \,||\, id_R\,,$ $\sigma_{pk_R\,||\,id_R})$. If the verification is positive, in $I3$, I generates a nonce n_1, encrypts n_1 || id_I (a challenge) with pk_R, and sends a message to R. The message (msg-N3) is expressed as: msg-N3: $\{AE(pk_R, n_1 \,||\, id_I)\}$.

Step 4: Upon receiving msg-N3, in $R1$, R decrypts the content of the message with its private key sk_R to obtain n_1 and id_I. The decryption process is expressed as $(n_1 \,||\, id_I) = AD(sk_R,\ AE(pk_R,\ n_1 \,||\, id_I))$. R then uses the same method as $I1$ in **Step 1**

to sends a request for the public key of I pk_I to Z. The message (msg-N4) is expressed as: msg-N4: $\{id_R, id_I\}$.

Step 5: Upon receiving msg-N4, in $Z2$, Z replies a message containing pk_I using the same method as $Z1$ in **Step 2**. The message (msg-N5) is expressed as: msg-N5: $\{pk_I \mid\mid id_I, \sigma_{pk_I \mid\mid id_I}\}$.

Step 6: After receiving msg-N5, in $R2$, R verifies the signature and obtains pk_I using the same method as $I2$ in **Step 3**. In $R3$, R generates a nonce n_2, encrypts $n_1 \mid\mid n_2 \mid\mid id_R$ (used as a response and a new challenge) with pk_I, and sends a reply message back to I. The message (msg-N6) is expressed as: msg-N6: $\{AE(pk_I, n_1 \mid\mid n_2 \mid\mid id_R)\}$.

Step 7: Upon receiving msg-N6, in $I4$, I decrypts the content of the message with its private key sk_I to obtain n_1' and n_2. If n_1' equals n_1 (generated in $I3$ in **Step 3**), then I is assured of the identity of R and I proceeds to $I5$. In $I5$, I sends a reply message containing n_2 (used as a response) encrypted with pk_R back to R. The message (msg-N7) is expressed as: msg-N7: $\{AE(pk_R, n_2)\}$.

Step 8: After receiving msg-N7, in $R4$, R decrypts the content of the message with sk_R to obtain n_2'. If n_2' equals n_2 (generated in $R3$ in **Step 6**), then R is assured of the identity of I and the protocol is successfully terminated.

At the end of the execution of the protocol, if all verifications are positive, then I and R are positively authenticated to each other. It is worth noting that I and R may cache the public key of the other entity. In this way, msg-N1, msg-N2, msg-N4, and msg-N5 can be omitted in subsequent authentication instances. In other words, only 3 messages (msg-N3, msg-N6, and msg-N7) are needed for subsequent authentication.

Appendix C
Algorithms Implementing the Methods of CPDA

The algorithms implementing the methods used in the design of CPDA are formally described in the following.

Algorithm 6.1.1: HT-AuthData-Aggregation

```
algorithm $HTAA(h_{x,1}, h_{x,2}, \dots, h_{x,Q})$
    $ht = HTC(h_{x,1}, h_{x,2}, \dots, h_{x,Q})$
    $rh_x =$ the value contained in the root node of $ht$
    for $i = 1 \ to \ Q$ do
        $sa_{x,i} = SAE(ht, h_{x,i})$
    end for
    $SA_x = \{sa_{x,1}, sa_{x,2}, \dots, sa_{x,Q}\}$
    return $\{rh_x, SA_x\}$
end algorithm
```

Algorithm 6.1.2: HC-AuthData-Aggregation

```
algorithm $HCAA(h_{1,y}, h_{2,y}, \dots, h_{P,y})$
    $ch_\alpha = h_{1,y}||h_{2,y}|| \dots ||h_{P,y}$
    return $ch_\alpha$
end algorithm
```

Algorithm 6.2.1: ISAuthData-Generation

```
algorithm $ISADG(d_{c,m_1}, d_{c,m_2}, \dots, d_{c,m_M}, sk_c)$
    for $i = 1 \ to \ M$ do
        $h_{c,m_i} = H(d_{c,m_i})$
    end for
    $\{rh_c, SA_c\} = HTAA(h_{c,m_1}, h_{c,m_2}, \dots, h_{c,m_M})$
    $\sigma_{rh_c} = SS(sk_c, rh_c)$
    return $\{\sigma_{rh_c}, SA_c\}$
end algorithm
```

Algorithm 6.2.2: ISAuthData-Verification

```
algorithm $ISADV(d_{c,m_a}, \sigma_{rh_c}, sa_{c,m_a}, pk_c)$
    $h'_{c,m_a} = H(d_{c,m_a})$
    $rh'_c = RAR(h'_{c,m_a}, sa_{c,m_a})$
    $sv = SV(pk_c, rh'_c, \sigma_{rh_c})$
    return $sv$
end algorithm
```

Algorithm 6.3.1: PGen-PSAuthData-Generation

```
algorithm $PPSADG(d_{m_a,r_1}, \dots, d_{m_a,r_E}, k_{m_a,jm})$
    for $j = 1 \ to \ E$ do
        $h_{m_a,r_j} = H(d_{m_a,r_j})$
    end for
    $\{rh_{m_a}, SA_{m_a}\} = HTAA(h_{m_a,r_1}, h_{m_a,r_2}, \dots, h_{m_a,r_E})$
    $\tau_{rh_{m_a}} = MS(k_{m_a,jm}, rh_{m_a})$
    return $\{rh_{m_a}, \tau_{rh_{m_a}}, SA_{m_a}\}$
end algorithm
```

Algorithm 6.3.2: AGen-PSAuthData-Generation

```
algorithm $APSADG(rh_{m_1}, \dots, rh_{m_M}, \tau_{rh_{m_1}}, \dots , \tau_{rh_{m_M}}, k_{m_1,jm}, \dots , k_{m_M,jm}, sk_{jm})$
  for $i = 1\ to\ M$ do
    $mv = MV(k_{m_i,jm}, rh_{m_i}, \tau_{rh_{m_i}})$
    if $mv$ is $negative$ then
      throw Exception($negative, i$)
    end if
  end for
  $ch_{jm} = HCAA(rh_{m_1}, rh_{m_2}, \dots, rh_{m_M})$
  $\sigma_{ch_{jm}} = SS(sk_{jm}, ch_{jm})$
  return $\{ch_{jm}, \sigma_{ch_{jm}}\}$
end algorithm
```

Algorithm 6.3.3: PSAuthData-Verification

```
algorithm $PSADV(d_{m_1,r_b}, \dots, d_{m_M,r_b}, sa_{m_1,r_b}, \dots , sa_{m_M,r_b}, ch_{jm}, \sigma_{ch_{jm}}, pk_{jm})$
  $sv = SV(pk_{jm}, ch_{jm}, \sigma_{ch_{jm}})$
  if $sv$ is $negative$ then
    return $\{negative, "ch_{jm}"\}$
  end if
  Extract $\{rh_{m_1}, \dots, rh_{m_M}\}$ from $ch_{jm}$
  $INDICES = \{\}$
  for $i = 1\ to\ M$ do
    $h'_{m_i,r_b} = H(d_{m_i,r_b})$
    $rh'_{m_i} = RAR(h'_{m_i,r_b}, sa_{m_i,r_b})$
    if $rh'_{m_i} ! = rh_{m_i}$ then
      Add $i$ to $INDICES$
    end if
  end for
  if $INDICES$ is $empty$ then
    return $\{positive\}$
  else
    return $\{negative, INDICES\}$
  end if
end algorithm
```

Algorithm 6.4.1: PGen-FRAuthData-Generation

```
algorithm $PFRADG(d_{r_b,c}, k_{r_b,jm})$
  $h_{r_b,c} = H(d_{r_b,c})$
  $\tau_{h_{r_b,c}} = MS(k_{r_b,jm}, h_{r_b,c})$
  return $\{h_{r_b,c}, \tau_{h_{r_b,c}}\}$
end algorithm
```

Algorithm 6.4.2: AGen-FRAuthData-Generation

```
algorithm $AFRADG(h_{r_1,c}, \dots, h_{r_E,c}, \tau_{h_{r_1,c}}, \dots, \tau_{h_{r_E,c}}, k_{r_1,jm}, \cdots, k_{r_E,jm}, sk_{jm})$
  for $j = 1\ to\ E$ do
    $mv = MV(k_{r_j,jm}, h_{r_j,c}, \tau_{h_{r_j,c}})$
    if $mv$ is $negative$ then
      throw Exception($negative, j$)
    end if
  end for
  $ch_{jm} = HC(h_{r_1,c}, h_{r_2,c}, \dots, h_{r_E,c})$
  $\sigma_{ch_{jm}} = SS(sk_{jm}, ch_{jm})$
  return $\{ch_{jm}, \sigma_{ch_{jm}}\}$
end algorithm
```

Algorithm 6.4.3: FRAuthData-Verification

```
algorithm $FRADV(d_{r_1,c}, \dots, d_{r_E,c}, ch_{jm}, \sigma_{ch_{jm}}, pk_{jm})$
  $sv = SV(pk_{jm}, ch_{jm}, \sigma_{ch_{jm}})$
  if $sv$ is $negative$ then
    return $\{negative, "ch_{jm}"\}$
  end if
  Extract $\{h_{r_1,c}, \dots, h_{r_E,c}\}$ from $ch_{jm}$
  $INDICES = \{\}$
  for $j = 1\ to\ E$ do
    $h'_{r_j,c} = H(d_{r_j,c})$
    if $h'_{r_j,c}! = h_{r_j,c}$ then
      Add $j$ to $INDICES$
    end if
  end for
  if $INDICES$ is $empty$ then
    return $\{positive\}$
  else
    return $\{negative, INDICES\}$
  end if
end algorithm
```