Master Thesis
Master's Programme in Network Forensics, 60 credits

Industrial Internet of Things Edge Computing

Edge Forensics

Network Forensics, 15 credits

Halmstad 2018-06-20
Shooresh Sufiye
This Page Intentionally Left Blank
Contact Details: Shooresh Sufiye
Wallmarksgatan 2
302 35 Halmstad
shooreshsufiye@gmail.com

Supervisors
Dr. Alexey VINEL
Halmstad University
School of Information Technology
301 18 Halmstad
alexey.vinel@hh.se

Dr. Jens JACOBSSON
HMS Industrial Networks
Research and Development dept.
Stationsgatan 37, 302 50 Halmstad
jeja@hms.se

Examiner
Dr. Stefan AXELSSON
Halmstad University
School of Information Technology
301 18 Halmstad
stefan.axelsson@hh.se
Abstract

Internet of Things (IoT) is an upcoming prominent technology which is quickly growing. Not all IoT demands of computing resources can be satisfied by cloud, and obstacles are firmer when it comes to mobility and agility. Thus, edge computing as a suitable middleware can fill the gap between cloud and IoT devices. Refer to the latest researches, edge security is still evolving, and forensics is almost untouched. In this work, we attempt to study available technologies and materials then design and implement an edge computing application which addresses the challenge of log collection from different edge devices. The interaction between edge and cloud is in a fashion that cloud entity can perform log collection from heterogeneous edge devices belong to different owners. On the other hand, due to local computing on the logs, the edge device can trust cloud party. Results show that thanks to the crucial topological position of the edge devices, the concept of edge computing can easily solve similar cloud challenges.
Preface

This thesis is an original work by the author. No part of this thesis has been previously published. Any use of others’ works are explicitly cited.
Acknowledgements

I present this work to Doctor Gunnar Wiik who without his support this accomplishment would not have been possible.

I would also like to thank all of the teachers and colleagues who helped me to fulfill this research project, especially Olga Torstensson, Björn Sjögren, Despoina Giarimpampa, and Mohammed Abdulrazzaq.

Finally, I would like to thank my family who ever supported me with their genuine love.

Author

Shooresh Sufiye
# Table of Contents

Abstract i

Table of Contents v

List of Figures vi

List of Tables vii

1 Introduction 1
   1.1 Purpose of the Study 1
   1.2 Motivation 1
   1.3 Objectives 2
      1.3.1 Scientific Question 2
      1.3.2 Hypothesis 2
   1.4 Methodology 4
   1.5 Limitations and delimitations 6
      1.5.1 Edge or Fog 6
   1.6 Assumptions 6
   1.7 Thesis organization 7

2 Background 8
   2.1 Definitions 8
      2.1.1 Enigmatic Nature of Fog 9
   2.2 Study the Available Technologies 9
      2.2.1 Platforms 10
      2.2.2 Cloud Technologies 11
   2.3 Programming Languages 12
      2.3.1 Java 12
      2.3.2 C 14
      2.3.3 Lua 14
      2.3.4 Node.js 15
   2.4 Application Layer Protocols 15

3 Related Work 19
   3.1 Edge Security 19
   3.2 Real-time Challenges 20
      3.2.1 Identity Authentication 20
      3.2.2 Access Control 21
      3.2.3 Lightweight Protocol Design 21
      3.2.4 Intrusion Detection 21
      3.2.5 Trust Management 22
   3.3 Edge Computation Verification 22
   3.4 Chapter Summary 23
List of Figures

1.1 current log collection .................................................. 3
1.2 Proposed log collection with edge computing ......................... 5

2.1 Programming Languages Comparison Summary ....................... 15
2.2 Application Layer Protocols Comparison Summary .................. 16

4.1 Client Authentication ................................................. 29
4.2 Hash Generation Example ............................................ 30
4.3 Experimental Network Environment Topology ......................... 31
List of Tables

5.1 Edge Computing Results Comparison .................. 35
List of Symbols, Abbreviations, and Nomenclature

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\lambda$</td>
<td>The eleventh letter of the Greek alphabet</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABE</td>
<td>Attribute-based Encryption</td>
</tr>
<tr>
<td>AI</td>
<td>Artificial Intelligence</td>
</tr>
<tr>
<td>BYOD</td>
<td>Bring Your Own Device</td>
</tr>
<tr>
<td>C-RAN</td>
<td>Cloud-Radio Access Network</td>
</tr>
<tr>
<td>CFM</td>
<td>Cloud-Fog Middleware</td>
</tr>
<tr>
<td>CP-ABE</td>
<td>Ciphertext-Policy Attribute-based Encryption</td>
</tr>
<tr>
<td>CS</td>
<td>Computer Science</td>
</tr>
<tr>
<td>DDS</td>
<td>Data Distribution Service</td>
</tr>
<tr>
<td>DoS</td>
<td>Denial-of-Service</td>
</tr>
<tr>
<td>DTLS</td>
<td>Datagram Transfer Layer Security</td>
</tr>
<tr>
<td>EDL</td>
<td>Eclipse Distribution License</td>
</tr>
<tr>
<td>EPL</td>
<td>Eclipse Public License</td>
</tr>
<tr>
<td>ETSI</td>
<td>European Telecommunications Standards Institute</td>
</tr>
<tr>
<td>F-RAN</td>
<td>Fog-Radio Access Network</td>
</tr>
<tr>
<td>FaaS</td>
<td>Fog as a Service</td>
</tr>
<tr>
<td>FC</td>
<td>Fog Cell</td>
</tr>
<tr>
<td>FCN</td>
<td>Fog control node</td>
</tr>
<tr>
<td>GC</td>
<td>Garbage Collector</td>
</tr>
</tbody>
</table>
HIDS Host-based Intrusion Detection Systems
HMS Hardware Meet Software
IaaS Infrastructure as a Service
ID Identity
IDS Intrusion Detection System
IoT Internet of Things
IP Internet Protocol
JRE Java Runtime Environment
JVM Java Virtual Machine
KP-ABE Key-Policy Attribute-based Encryption
LWM2M Light-Weight Machine to Machine
M2M Machine to Machine
MANET Mobile Ad Hoc Network
MEC Mobile Edge Computing
NIDS Network-based Intrusion Detection Systems
OMA Open Mobile Alliance
OMG Object Management Group
OS Operating System
OSGi Open Services Gateway initiative
PaaS Platform as a Service
PDA Personal Digital Assistants
QoS Quality of Service
RAN Radio Access Network
REST Representational state transfer
RFID Radio-Frequency Identification
SaaS Software as a Service
SDN Software Defined Network
TCP Transmission Control Protocol
TLS Transfer Layer Security
UDP User Datagram Protocol
V2G Vehicle-to-Grid
VANET Vehicular Ad Hoc Network
Chapter 1

Introduction

1.1 Purpose of the Study

IoT is the next promising revolutionary change in Information Technology. It will change the face of the planet soon [9]. Internet of Things is a term to address the devices such as; sensors vehicles, home furniture, and basically “Things” with embedded connectivity to the Internet. The growing number of those devices leads to the invention of other notions in order to automation of IoT.

Edge solution exponential growth shows that it is impossible to control IoT with a centralized architecture [20], [34]. Edge computing is a way to bring down distributed processing, storage, and control plan near to things in contrary to cloud processing which is semantically centralized, even though it is distributed in the cloud. In edge computing architecture role of smart edge devices such as gateways are vital. In this project, We are going to design and build an application to perform edge computing task for gateways. In this case, implement an edge computing solution on logs in a particular industrial hardware and software architecture made by HMS industrial networks company.

1.2 Motivation

The topic of edge computing is relatively a new topic and still evolving in its early levels. Many different standards, protocols, and procedures are introduced and proposing to develop different aspects of edge computing. There is a lake of study and achievements in security and the gap is more open when it comes to the forensic aspect. However, some studies already pointed out a few research areas.

According to surveys and state-of-the-art researches and articles, one of the challenges in edge computing security and forensics is collecting data from log files in general [20]. In a lot of forensics, scenarios log files
are the last and usually the only chance of the analysts to find valuable evidence. Hence, I found the log file forensics in an edge computing domain interesting, beneficial, and almost untouched.

1.3 Objectives

We can recount the biggest challenges in log collection as follows:

1. Enormous central storage and processing resources needed
2. Transmission volume
3. Latency from edge to cloud
4. Privacy and forensic issue, all of the logs must transfer to cloud. It means devices owners (ISPs, CSPs, and users) must share their logs with us!

An overview of the current method to collect logs from a network device generally illustrated in figure 1.1.

Legal issue rises when enterprises and individuals do not interested nor allow collecting log files from their devices by a third party because log files may contain confidential information. Hence, from a non-technical aspect to gather logs from network devices is also challenging.

1.3.1 Scientific Question

Taking the aforementioned gaps and obstacles into consideration the plausible question is:

- Is it possible to collect logs from edge devices belonging to heterogeneous owners efficiently without disclosure of confidential data?

1.3.2 Hypothesis

By utilizing edge computing we can categorize different logs with different access permissions. And collect logs of a particular category from network devices in general, in this case, from fog devices. This way first we preserved the confidentiality of logs and second, we saved an enormous amount of unwanted network traffic due to the transmission of all the logs which we don’t need necessarily. Third, we saved central resources including storage and processing in central cloud clusters.

Our hypothesis to solve the aforementioned challenges is to do computing tasks on log files in the edge or more persistently in fog devices including:
Figure 1.1: current log collection
• Categorize and store logs,

• Accept requests from authenticated client(s),

• Find interesting logs according to the request, and

• Return only those interesting logs.

Certainly, in more advanced levels of investigation, if for instance a central IDS interested in logs of a particular device it can request for complete log files of that device selectively and receive them.

Finally, to prevent a flood occurrence to the client when a device finished computation tasks and prepared the interesting logs to send to the client. It registers itself in a queue on the client side. By sending a small "Result-Is-Ready" message including an ID. Then the client can request for prepared results of each device in turn. So that client doesn’t receive a flood of logs from millions of routers.

This way we prevent something and use advantages of fog computing to achieve

1. Low and predictable latency

2. Significant reduction of bandwidth consumption

3. More privacy

4. Context awareness

5. Less transmission of data

The consequences of the proposed method is completely described in Results Chapter.

1.4 Methodology

In this section, we are going to explain the methodology and algorithm we used to perform the edge computing task on log collection at the edge of the network for further investigation in the central cloud. The program is written in Java with a non-complex structure in order to enable further adoptions of the code to functional programming languages such as C which have no support for Object concept.

In the proposed edge computing log collection method, first, we need to know what we are looking for in the logs. To do so, a query is generated in the cloud application and send to the "smart edge device" which basically is a router with running JRE. The edge application validates the other party and then accepts the query. Then it searches through log files and selects records matching the query and forms an answer to the query including those interesting logs. And finally, send back the interesting logs to the cloud party through a secure connection. The edge application methodology is illustrated in figure three in a very simplified way.
Figure 1.2: Proposed log collection with edge computing
1.5 Limitations and delimitations

One of the biggest challenges for this study was a lack of references and related works. Due to the recentness of the main subject of edge computing, few works have done in security aspects which have some concepts in common with forensics on the edge. When it comes to forensics it is relatively remained untouched. As a consequence, it is very hard to provide a solution for the chosen problem especially when there are no other works to compare and evaluate our work with them.

The aim of this project is basically about edge computing and how to efficiently collect logs from many devices (thousands or tens of thousands) it is not possible to test the proposed solution in a real scale network. Hence, we implemented the two parts (peers) of the application on an edge-enabled gateway and the other application hosted on Amazon cloud to perform a real cloud-edge communication between those two applications. Though, the one to one communication is fully implemented in a real actual network over the Internet.

1.5.1 Edge or Fog

Generally, three major solutions proposed for edge computing problem so far. Cloudlet, Mobile Edge Computing (MEC), and Fog Computing. However, due to the immaturity of the context, to distinguish between edge computing and fog computing is not so easy, and generally, some of the terms have overlapped areas. Terminology is not consistent yet and from text to text people have different definitions for the same terms. Since, Fog works with the cloud, whereas edge is defined by the exclusion of cloud [50], finally, we decide to choose fog computing as the atmosphere of this project. That is a part of the bigger notion of edge computing meaning decentralize processing, storage, and control from the central cloud to edge.

1.6 Assumptions

Fortunately the final designated results are achieved in the project and one assumption which we cannot actually implement is expansion of the solution from one gateway to many which are designed and planned to how to deal with many (thousands of gateways) without occurrence of a denial of service (for instance) to the central node which collects logs.

Another assumption is, when we want to test full log files collection in order to compare with the proposed edge computing method we assume that the edge device let the remote host access all of the logs, which is obviously not happens in real-world incidents and also is in conflict with many security measures.
1.7 Thesis organization

In this document, we first review different philosophy and architectures proposed for edge computing in the literature and then, choose a suitable programming language, protocols, and technologies to implement this application.

In chapter two we study and investigate the latest available technologies, programming languages, application protocols, and standards in the domain.

In chapter three we will study the state-of-the-art researches as well as describing some important terminology and definitions.

In chapter four, we summarize the implementation of the applications developed in Java programming language as well as some of the important configurations and settings.

In the fifth and the last chapter, we have results, discussion, conclusion, suggestions, and future works and possibilities.

Finally, The full code and explanations for each block and its functionality brought in Appendix.
Chapter 2

Background

In this chapter we will review and determine some important terminology and their definitions, then we study the state-of-the-art practically available technologies, programming languages, and application layer protocols in Edge and Fog domain.

2.1 Definitions

Cloud Computing: Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction [31].

Fog Computing: Fog computing is proposed to enable computing directly at the edge of the network, which can deliver new applications and services especially for the future of Internet[2], [32].

Fog node: commercial edge routers are advertising processor speed, number of cores and built-in network storage. Those routers have the potential to become new servers. In fog computing, facilities or infrastructures that can provide resources for services at the edge of the network are called fog nodes. They can be resource-poor devices such as set-top- boxes, access points, routers[33].

Edge: In CISCO’s seven-layer architecture the very first layer which typically consists of computing nodes, e.g., smart controllers, sensors, RFID readers, etc., and different versions of RFID tags[34].

Cloud-fog Middleware (CFM): Cloud middleware that handles the service deployment in the cloud and the data propagation to cloud services[1].

Fog control node (FCN): Control node that orchestrates subjacent fog colonies consisting of other fog control nodes and fog cells[1].

8
Fog cell (FC): Node connected to fog control nodes and IoT devices to execute services and propagate data to a parent fog control node[1].

Fog data: Models and Utility classes shared amongst the other components. Host monitor: Monitoring application that monitors the host and sends the data to a corresponding Redis database for further processing.

Many enterprises, organizations, communities are active in the field of research, standardize, and define protocols and principles for IoT and edge computing. Meanwhile, some frameworks and technologies developed during the last years, depending on location and use, at cloud side, we have Microservices, Docker container, Resin.io, Azure, Thingsworx, MindSphere, Apache EDGENT... At the edge, we have influxDB, iFog, AWS Greengrass, different SDKs for different hardware, different OSes[10], [48]. Eclipse IoT project, Watson IoT, OpenFog, and Azure are some of the role players who trying to standardize the domain from different aspects.

2.1.1 Enigmatic Nature of Fog

Even though it mentioned in the first chapter, but we need to re-emphasize that due to the immaturity of the context, to distinguish between edge computing and fog computing is not so easy, and generally some of the terms have overlapped areas. also, it is good to mention, the reason behind naming the Fog is because there is no clear boundary in real fog in nature and in the network it is still the case.

Terminology is not consistent yet and from text to text authors provide different definitions for same terms. Since, Fog works with the cloud, whereas edge is defined by the exclusion of cloud [50], finally, we decide to choose fog computing as the ecosystem of this project. That is a part of the bigger notion of edge computing meaning decentralize processing, storage, and control from the central cloud.

2.2 Study the Available Technologies

Available Methods: There are Three major technologies available to perform edge computing; Cloudlet, Mobile Edge Computing, and Fog Computing [14]. Each of those technologies is defined as:

-Cloudlet is a mobility-enhanced small-scale cloud data center that is located at the edge of the internet, it represents middle tier of the 3-layer hierarchy architecture. A cloudlet is a trusted, resource-rich computer or cluster of computers that is well-connected to the internet and available for use by nearby mobile devices [15].

-Mobile Edge Computing provides an IT service environment and cloud-computing capabilities at the edge of the mobile network, within the Radio Access Network (RAN) and in close proximity to mobile
subscribers [45].

Fog computing is a system-level horizontal architecture that distributes resources and services of computing, storage, control and networking anywhere along the continuum from Cloud to Things [49]. Refer to the definitions provided above, we can see that our project fits the most in Fog computing. However, due to the indiscernible nature of fog, differentiate between edge computing and fog computing is a bit tricky and case dependent, and in some texts, they overlapped one another [13]. We can say it is a fog computing at the edge of the network the closest to end devices. Or the very lowest layer of Fog. Hence, we conclude the right category for this project is fog computing, because it enables high-performance, interoperability and security in a multi-vendor fog computing based ecosystem [14], which are all requirements for an industrial IoT, as well as our preliminary goals of the project. Fog computing is standardized the most by OpenFog consortium group affiliated with IEEE. An OpenFog architecture is applicable across many different vertical-markets including transportation, agriculture, smart-cities, smart-buildings, hospitality, etc. [49], as well as Industry 4.0.

2.2.1 Platforms

Technologies Employed in Fog Computing Domain To address not only the need for scheduling processes but also separate cloud network data with underlying resources (for security reasons), virtualization is a good practice. Virtualization can achieve via using Virtual Machines or Containers. Which containers seem to be a better practice refer to [46]. In figure 2 we can see differences between the architecture of the virtual machine and container.

A container is essentially a packaged self-contained, ready-to-deploy set of parts of applications, that might even include middleware and business logic in the form of binaries and libraries to run the applications [47].

The basic ideas of containerization are: (1) A lightweight portable runtime environment, (2) The capability to develop, test and deploy applications to a large number of servers, and (3) The capability to interconnect containers. Containers address concerns at the cloud PaaS and IaaS level [46]. Which makes containers more interoperable across different platforms and services. Container tools like Docker are frameworks built around container engines. Thanks to recent Linux distros kernel mechanisms such as namespaces and cgroups Docker containers can Isolate processes on a shared OS [47].
2.2.2 Cloud Technologies

Fog computing provides the missing link in the cloud-to-thing continuum. Fog architectures selectively move to compute, storage, communication, control, and decision making closer to the network edge where data is being generated in order solve the limitations in current infrastructure to enable mission-critical, data.dense use cases [50].

OpenFog

Implementation of Fog Computing in OpenFog framework for a smart building use case [50]. One smart building can have thousands of sensors. Using the hierarchical design of the OpenFog Reference Architecture, each floor, wing, or even individual room could contain its own fog node. A fog node could be responsible for:

- Performing emergency monitoring and response functions.
- Performing building security functions.
- Controlling climate and lighting.
- Providing a more robust compute and storage infrastructure for building residents to support smartphones, tablets and desktop computers.

Locally stored operational history can be aggregated and sent to the cloud for large-scale analytics. This analytics can be applied to machine learning to create optimized models, which are then downloaded to the local fog infrastructure for execution [50].

Microsoft Azure

Microsoft is another key player in the cloud market and has a hand in fog and edge for tomorrow. Microsoft Azure already provide a verity of cloud services which some of them can be adapted to edge. Microsoft also provides Azure IoT Edge which extends cloud intelligence and analytics to edge devices—Azure IoT Edge is a fully managed service that delivers cloud intelligence locally by deploying and running artificial intelligence (AI), Azure services, and custom logic directly on cross-platform IoT devices. Run your IoT solution securely and at scale—whether in the cloud or offline[40].

AWS Greengrass

AWS Greengrass is software that extends AWS cloud capabilities to local devices, making it possible for them to collect and analyze data closer to the source of information, while also securely communicating with
each other on local networks. More specifically, developers who use AWS Greengrass can author server-less code (AWS Lambda functions) in the cloud and conveniently deploy it to devices for local execution of applications [11].

Google

With Android OS and other online Google services no place for any doubt about Google's position in the IT world. Google prepared for IoT edge with "Cloud IoT Edge". It extends Google Cloud's powerful data processing and machine learning to billions of edge devices, such as robotic arms, wind turbines, and oil rigs, so they can act on the data from their sensors in real time and predict outcomes locally. Cloud IoT Edge can run on Android Things or Linux-based operating systems. Cloud IoT Edge is composed of two runtime components, Edge IoT Core and Edge ML, and also take advantage of Google's purpose-built hardware accelerator ASIC chip, Edge TPU [41].

Siemens MindSphere

Other companies like Siemens that are not originally IT companies but have good experience and credit in the industry also targeting the edge of the cloud with MindSphere. MindSphere is the cloud-based, open IoT operating system from Siemens that connects your products, plants, systems, and machines, enabling you to harness the wealth of data generated by the Internet of Things (IoT) with advanced analytics.

MindSphere delivers a wide range of device and enterprise system connectivity protocol options, industry applications, advanced analytics and an innovative development environment that utilizes both Siemens' open Platform-as-a-Service (PaaS) capabilities along with access to AWS cloud services. MindSphere is basically using AWS as infrastructure but the difference is in business logic and the industrial community that MindShpere mostly designed for [42].

2.3 Programming Languages

To implement our solution we need a programming language compatible and perhaps designed for the purpose we are going to use for. Bellow, I study four different programming languages and their characteristics in the edge computing domain.

2.3.1 Java

Write once run anywhere. That's true especially when it comes to heterogeneous IoT. Some of the benefits of Java are: first is skills – there are lots of Java developers out there, and that is an important factor
when selecting technology. Second is maturity and stability – when you have devices which are going to be remotely managed and provisioned for a decade[43], Java’s stability and care about backward compatibility plays a very important role. The third is the scale of the Java ecosystem – thousands of companies already base their business on Java and already it is Java Virtual Machine is installed on more than three billion devices worldwide.

Gateways are pretty much what OSGi (Open Services Gateway initiative) was originally designed for. The ability to do extensible, services-based architectures, remote management, provisioning and life-cycle management on a device gateway is extremely important, and OSGi provides a mature platform for doing exactly that.

- Java made for limited resources, 8 MB RAM, 500 MHz, and using -Xmx we can set heap space down to 9.5 MB (basic JVM).

- Portability Virtual Machine of Java makes Java independent of hardware.

- Dynamic Compilation, with two compilers and caching parts of code it has better performance than regular interpreter and in some cases it can outperform native code, due to aggressive optimization.

- Memory Management, Handles memory with garbage collector (also better for memory leaks)

  General heap space, most of objects are short-lived

  Time it takes to allocate a Java object is 6 machine instruction

- No explicit pointers, buffer overrun is not possible (point to next to end of the array), you cannot manipulate pointer, reference to objects cannot be changed arithmetically.

- Secure Built-in, designed to run distributed, JDK 7 and 8 have security issues because of browser plugins, but IoT we do not have browser.

- Comprehensive library.

- Common interface techs GPIO to work with pins and I/Os

- Multithreading built-in and concurrency utilities

- Flexible deployment, JRE can be 11 to 54 MBs.

- built-in remote management JMX (management extensions), Manage beans, manage applications and the JVM.

- Good tools such as eclipse (pioneer in IoT), Inteliij, NetBeans [51]. Disadvantages of Java
• Java compiler is not well optimized yet compared to C++.

• Memory management, with Java, is a little expensive.

• The lack of templates can limit the ability of Java to create high quality data structures.

• One can find some bugs in browsers and example programs.[52]

Specifications of aforementioned programming languages summarized in table one.

2.3.2 C

Unsurprisingly one of the top programming languages for IoT. An intensive and powerful language with good resources and support, libraries, and community of developers. With the presence of Microsoft Azure, C# will be one of the key players in the IoT domain. C# is primarily object-oriented, but it also supports some features typically found in functional languages such as lambdas, delegates and anonymous classes [56]. C# runs on top of the .NET framework, which provides many libraries containing classes used for common tasks such as connecting to the Internet, displaying a window or editing files. Unlike many other languages, you don’t have to pick between a handful of libraries for every small task you want to do.

2.3.3 Lua

Lua is a powerful, efficient, lightweight, embeddable scripting language [55]. Lua has been used in many industrial applications, with an emphasis on embedded systems, and games. It is a good practice to use Lua to glue pieces of code in other programming languages such as C, C++, C#, Java, etc. Lua is a fast language engine with a small footprint that you can embed easily into your application. Lua has a simple and well-documented API that allows strong integration with code written in other languages [55]. It is so small that source code and documentation of Lua 5.3.4, takes 297K compressed and 1.1M uncompressed. eLua (Embedded Lua) project offers the full implementation of the Lua Programming Language to the embedded world, extending it with specific features for efficient and portable software embedded development [54]. However, Lua like all other products have some problems. Here we can see a number of technical drawbacks to Lua:

• No Unicode support; there is a binding to ICU library that implements full Unicode support.

• Limited pattern-matching support, although the included one is still quite powerful. can use LPeG and its re module.

• No ternary operator
Figure 2.1: Programming Languages Comparison Summary

- No class/object finalizers. Lua provides finalizer functionality through the gc metamethod, but it is available only for user data types (and not tables).

- No built-in bit operations in Lua 5.1. This is addressed in LuaJIT (BitOp) and Lua 5.2 (bit32), which both include bit libraries. LuaJIT is no longer maintained by the creator.

- Number of elements in a table is not easy to get and the result depends on how you do this (or what you mean by "length").

- return statement can’t be used if it’s not the last statement in a block;

- Only one value is returned from a function if it’s not the last one in a list.

2.3.4 Node.js

Node.js is a runtime environment and library for running JavaScript applications outside the browser. It mostly used to run real-time server applications, chat applications, game servers, Ad servers, Streaming servers [53]. It has its flexibility, scalability, compatibility, and web-oriented from JavaScript.

In table 2, we have a brief comparison between different languages with respect to IoT domain.

2.4 Application Layer Protocols

Select the right communication protocol is important… A brief intro about IoT protocols… The Object Management Group’s (OMG) Data Distribution Service for Real-Time Systems (DDS) is an open middle-
<table>
<thead>
<tr>
<th>Security</th>
<th>MQTT</th>
<th>XMPP</th>
<th>DDS</th>
<th>CoAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low, or use TLS/SSL</td>
<td>High, SSL</td>
<td>High, SSL/DTLS</td>
<td>CoAP, DTLS</td>
<td></td>
</tr>
<tr>
<td>Pay. Form.</td>
<td>Flexible, byte</td>
<td>Low</td>
<td>Flexible</td>
<td>Flexible</td>
</tr>
<tr>
<td>Compatibility</td>
<td>Low</td>
<td>High(XML)</td>
<td>Low</td>
<td>Low</td>
</tr>
<tr>
<td>QoS</td>
<td>High, 3 levels</td>
<td>High</td>
<td>Very High</td>
<td>Medium, con-non-Ack-retrans</td>
</tr>
<tr>
<td>Performance</td>
<td>High</td>
<td>Low</td>
<td>Low</td>
<td>High</td>
</tr>
<tr>
<td>Deployment Scenario</td>
<td>Telemetry, Large Area Network</td>
<td>H2M Interaction</td>
<td>High dependable sys, V2V, military</td>
<td>Web connection, HTTP gateway, multicast, Device Control</td>
</tr>
<tr>
<td>Architecture</td>
<td>Publish-Subscribe</td>
<td>Publish-Subscribe, Request-Response</td>
<td>Publish-Subscribe</td>
<td>Request-Response, Resource observe/Publish-Subscribe</td>
</tr>
<tr>
<td>Transport Layer</td>
<td>TCP</td>
<td>TCP</td>
<td>UDP, TCP</td>
<td>UDP</td>
</tr>
<tr>
<td>Note</td>
<td>Lightweight Single layer</td>
<td>Heavy</td>
<td>Delivery guaranty</td>
<td>Easy HTTP-CoAP, Single layer &gt; 2 conceptual layers, RESTful</td>
</tr>
</tbody>
</table>

Figure 2.2: Application Layer Protocols Comparison Summary

A software standard that enables scalable, real-time, dependable, high performance and interoperability data exchanges between publishers and subscribers. Authors of [16] performed a comparison between three application layer protocols in terms of efficiency and Round Trip Time on CoAP, WebSocket, and MQTT. The result shows that CoAP achieves the highest protocol efficiency and the lowest average RTT, closely followed by WebSocket. The performance of MQTT protocol strongly depends on the Quality of Service (QoS) profile[16]. In table 3 we can see a brief specification of four different protocols useful in IoT domain.

**CoAP Implementations in Java: Californium**

Californium (Cf) is an open source implementation of the Constrained Application Protocol (CoAP). It is written in Java and targets unconstrained environments such as back-end service infrastructures (e.g., proxies, resource directories, or cloud services) and less constrained environments such as embedded devices running Linux (e.g., smart home/factory controllers or cellular gateways). Californium (Cf) has been running code for the IETF standardization of CoAP and was recently re-implemented from scratch having all the experience. In particular, Cf focuses now on service scalability for large-scale Internet of Things applications. The new implementation was successfully tested at the ETSI CoAP and OMA LWM2M Plug tests in November 2013 and March 2014. It complies with all mandatory and optional test cases. Californium is a significant Java-based implementation of CoAP. It is known as most popular implementation of CoAP provided by Eclipse foundation[18]. Total size of library is about 6.5 MB. Full implementation of RFC 7252. It has a big community and since it used during the first implementation of the protocol by IETF many professional
projects has been done using californium in a commercial and enterprise level. A reasonable help and support is behind Californium mainly by Eclipse community. It became a facto CoAP framework from eclipse community. The only problem is its big size which is more suits for back end or more powerful devices such as gateways, not for small resource constrained IoT devices. I believe it will be a negative point limits broad use of this library. The project is divided into four sub-projects. The Californium (Cf) core provides the central framework with the protocol implementation to build your Internet of Things applications. This repository also includes example projects to get you started. All Californium sources are hosted on GitHub, so you can easily contribute through pull requests. Scandium (Sc) is a sub-project of Californium that implements security aspects of CoAP. Actinium (Ac) is the app-server for Californium to realize IoT mashups. JavaScript apps become available as RESTful resources and can directly talk to IoT devices using CoapRequest object API. Copper (Cu) is Firefox add-on that makes it able to communicate through CoAP by easily enter CoAP URI in address bar (“caop://10.1.2.3:5683/resource”). Californium (Cf) is dual-licensed under EPL and EDL. The latter is a BSD-like license, which means the Cf CoAP framework can be used together with proprietary code to implement your IoT product! Cf is available in Maven Central and very easy to consume in your Java project. We have got your back covered from OSGi wrapper to HTTP-CoAP cross-proxy to advanced test suites that have been used at the ETSI Plugtests.

**Known limitations of Californium**

When trying the examples, there are some limitations and behavior that should be taken into account: -in early versions : when sending the first request from the nRF5x DK, the DTLS handshake phase might take up to 3 minutes to complete, which now it is already fixed. Since the example has no way of knowing if a DTLS session is closed, it cannot free its resources until it is reset. If you invoke the client multiple times, the dev kit will not be able to establish a new session. From the client side, you will appear to get a timeout. If you modify the client to use an unsupported cipher suite, you will get a timeout when trying to connect.

**Leshan, LwM2M Implementation in Java**

A solution for overcome limitations of working with Californium API directly, is employ Leshan. Leshan is a LwM2M implementation in Java. LightweightM2M is principally a device management protocol, but it should be designed to be able to extend to meet the requirements of applications. LightweightM2M is not restricted to device management, it should be able transfer service / application data[19]. LightweightM2M implements the interface between M2M device and M2M Server. It provides a choice for the M2M Service Provider to deploy a M2M system to provide service to the M2M user.
ARM Mbed

ARM Mbed is another implementation of Coap protocol based on ARM architecture. It is supported by ARM and only can be used for ARM processors.

nCoAP

nCoAP is a light weight implementation of CoAP protocol which is designed and developed by a small community of developers. Unfortunately, nCoAP can be used only for experimental use cases not for a commercial use because it does not support DTLS, because it uses Nitty framework which is not supports DTLS! As a consequence it turned nCoAP to a experimental library rather than a practical one.
Chapter 3

Related Work

As it mentioned in first chapter, one of the biggest obstacles for this work was lack of practical related work especially in forensics domain. unsurprisingly, in development and evolution of any new technology first trend is to setup the most fundamental blocks to enable the basic functionalities of the technology up and running then it comes to concerns about more advanced features such as forensics concerns. In this aspect, edge computing is not an exception and current trend is mostly targets standardization, define protocols, and procedures. In this chapter we mentioned the latest researches and literature and also limited accomplished works about forensics aspect in edge computing.

3.1 Edge Security

For several reasons, the cloud computing model is unable to fulfill certain requirements such as; low latency and jitter, mobility support, and context awareness that are critical for many applications. Many technologies developed to compensate for these shortcomings (E.g. mobile cloud computing, mobile edge computing, and fog computing). While these technologies are emerging by different communities and partially big enterprises they have many things in common but there is no synergy among them. The lack of cooperation is more visible when it comes to security. Where in most cases the analyze targets only one edge paradigm while ignoring the others. R. Roman et al. surveyed security, threats and challenges of edge computing in [35].

Also in [36] surveys security threads by attackers to edge computing listed as following:

- Forgery, not only forge their identities and profiles, but also generate fake information to mislead other entities.

- Tampering, drop, delay or modify transmitting data
• Spam, unwanted content

• Sybil: Sybil attackers either manipulate fake identities or abuse pseudonyms in order to compromise or control the effectiveness of fog computing.

• Jamming, jam communication channels or computing resources by generating huge amount of bogus messages

• Eavesdropping, listen on communication channels to capture transmitting packets and read the content.

• Denial-of-Service, disrupt the services provided by fog nodes.

• Collusion, in edge computing any two or more parties can collude to increase their attack capability, such as several fog nodes, IoT devices, IoT devices with the cloud, or fog nodes with IoT devices.

• Man-in-the-Middle,

• Impersonating, the attacker can impersonates a legitimate fog node to offer fake or Phishing services to users.

3.2 Real-time Challenges

Edge computing provides many low-latency services to IoT devices. due to the security risks and threats mentioned in section 3.1.2, an IoT user cannot unconditionally trust fog nodes and enjoy the excellent edge services. As a consequence, providing a secure real-time service is extremely critical. In this work we investigate the following challenges.

3.2.1 Identity Authentication

Cloud service providers at the edge and users need different trust domains and it leads to various security threats against user data and IoT services [37]. Hence, many different Authentication mechanisms have been developed to support IoT services security [36]. C. Li et al. study the opportunities SDN remote programming in IoT-Fog domain and present an efficient authentication solution to address the security of the OpenFlow channel between the controller and its switches. they specifically investigate the potential threats of Man-in-the-Middle attacks on the OpenFlow control channel. Finally, they propose a lightweight countermeasure using Bloom filters which consumes a few amount of resources. How ever main focus of this study is on SDN networks and securing authentication in SDN network, the common issue [38] deals with with our work is authenticating remote host in order to transfer program rather than configuration or data.
The biggest problem with all of the proposed solutions is they do not consider the mobility of IoT devices. In edge computing, edge and also fog nodes serving to IoT devices which a large portion of them are mobile devices such as cars, smart phones, drowns, and wearable devices which can travel from a fog node's area to another and if they authenticate each time they reach to new node the latency can unacceptably increase.

### 3.2.2 Access Control

Another challenge for edge computing is "Access Control" because in an IoT scenario users and IoT devices have distinctive rights to access particular services. As we mentioned in 3.1.2 by [4] it is impossible to have no authentication mechanism. currently, role-based and attribute-based access control based on attribute encryption are very popular. the problem of mobility is still an issue here as well as multiple devices owned by on user. So far, Several mobile device management protocols [21] have been implemented to support access control for multiple mobile devices in bring your own device (BYOD) environment, but seldom [22] can achieve device and key management for users. Latter in chapter five, we will see that our proposed authentication method solves the problem with multiple devices own by one user or entity. The goal is make it possible for multiple devices owned by a user access the real-time services without any configuration for new joining devices, and the revocation of old devices should be consistent for all fog nodes [22].

### 3.2.3 Lightweight Protocol Design

Since IoT devices communicate with edge devices and fog nodes up to one or two hopes, it is suitable for real-time communications scenarios, though, low latency is not only depends on short-range communication but also the computation delay at edge devices. As we pointed out edge computing definitions and basic idea in chapter two, most of the local computing operation perform at edge device regard to IoT devices to recount security operations we can name identity authenticated key agreement, data encryption, digital signature, and spam detection [5]. All in all, it is obvious that lightweight protocols are necessary here to avoid add over-load to the edge devices and reduce their response time. Keep this fact in consideration, leads us to choose the most IoT-compatible and lightweight protocol to implement the idea of this research. Which is nothing but Constrained Application Protocol A.K.A CoAP.

### 3.2.4 Intrusion Detection

It is necessary to protect whole network architecture by an intrusion detection mechanism either Host-based Intrusion Detection System HIIDS or Network-based Intrusion Detection System NIDS. Houmansadr et al. [6] designed a cloud-based intrusion detection and response system for mobile phones.
They developed an active intrusion detection system which continuously performs and in-depth forensics task to detect malicious activity and reacts as soon as found the intrusion activity. the problem with their work is since they could not relay on computing resources on the smart phone they compensate this shortcoming by moving the rest of computations to the central cloud IDS. Our project is basically opposite of their work in that aspect as we want to avoid centralized computing in cloud neither on the end IoT device but in edge and fog nodes. The idea can be adopt to edge and fog nodes so that the intrusion detection system can be distributed among cloud and fog nodes with the central IDS engine placed in cloud. Houmansadr et al. [6] summarized IDS implementation in fog nodes beneficial but some challenges due to heterogeneous, decentralized and distributed architecture of fog computing are still exist.

3.2.5 Trust Management

Despite identity management it is not easy to trust all of the fog nodes. A variety of trust mechanisms has been proposed which mostly are either evidence-based or monitoring-based. In evidence-based trust models trust is based on a kind of evidence to proof an entity’s actual identity such as; public key, address, identity etc. On the other hand, monitoring-based trust model establishes trust among users through observing the behavior of past interactions between them. One study [39] shows that proposed a trust management scheme based on the direct evidence to estimate the trustworthiness of a host from the positive and negative feedback about this entity. When direct evidence is not available, indirect evidence such as third-party testimonies should also be complemented for estimating an entity’s trustworthiness. [7] realized due to the distributed nature of fog it is hard to collecting evidences in order to trust all of the fog nodes is challenging. With the current research and our proposed edge computing method the problem to trust the nodes will solve and a legitimate cloud entity can collect evidences from different heterogeneous nodes in fog. Then it is possible to build a trust evaluation model for the entire fog network. Second condition for an acceptable trust model is being situation-specific, indicating that different trust metrics should be considered to reflect the unique properties of trust in different services and applications. Which it is one of the requirements of our project so that a unique trust measure exists in the edge computing terminal module witch is used by all other services. Finally, a trust mechanism should support scalability and consistency when changing network conditions, including dynamics of fog nodes, mobility of IoT devices, and traffic patterns.

3.3 Edge Computation Verification

Attackers not only can compromise a fog node to disclosure of information but also can control the result of computations [36]. To verify that computation is safe and verify the computation source, various methods
proposed which we survey them below.

Gemmaro et al. [8] introduced the notion of verifiable computation and designed a non-interactive verifiable computation scheme based on garble circuit. They introduce and formalize the notion of Verifiable Computation, which enables a computationally weak client to "outsource" the computation of a function $F$ on various dynamically-chosen inputs $x_1, ..., x_k$ to one or more workers. The workers return the result of the function evaluation, e.g., $y_i = F(x_i)$, as well as a proof that the computation of $F$ was carried out correctly on the given value $x_i$. The primary constraint is that the verification of the proof should require substantially less computational effort than computing $F(x_i)$ from scratch. They present a protocol that allows the worker to return a computationally sound, non-interactive proof that can be verified in $O(m\text{poly}(\lambda))$ time, where $m$ is the bit-length of the output of $F$, and Lambda is a security parameter. The protocol requires a one-time preprocessing stage by the client which takes $O(C\text{-poly}(\lambda))$ time, where $C$ is the smallest known Boolean circuit computing $F$. Unlike previous work in this area, this scheme also provides (at no additional cost) input and output privacy for the client, meaning that the workers do not learn any information about the $x_i$ or $y_i$ values. The method [23] insures the trust between two hosts without knowing one another and have no negotiation on trust but one can trust the result of computation by the other party. This notion is useful in edge computing when we cannot trust all the fog nodes but we can verify the result of computations[24].

Another study [25] shows we can verify the result of computation performed by a worker utilizing full homomorphic encryption schemes to construct a non-interactive verifiable computation scheme with a small size of the public key. The significant point of this study is possibility to verify the encrypted data without decrypting it. They designed a method in which provide any result other than $F(x)$ is infeasible that is acceptable for the delegator.

Consequently, Parno et al. [26] designed a publicly verifiable computation scheme based on CP-ABE. This research extends verifiable computation in not only in public delegation aspect but also in public verifiability, which have important applications in many practical delegation scenarios such as edge computing. Utilizing such methods in edge computing solves problems in trust management of fog. They establish a connection between verifiable computation and attribute-based encryption (ABE), a primitive that has been widely studied.

### 3.4 Chapter Summary

In this chapter we survey fog and edge technologies and platforms and the security threats and risks for an edge computing scenario and some of their consequences such as disclosure of user information and compromising fog nodes. In this section we investigate some of real-time threats which is one of the most
important and advantages of edge computing; such as identity authentication, access control, light-weight protocol, IDS, and trust management.

Finally, we investigate latest proposed methods to safely outsource computations which is absolutely necessary to consider in an edge computing design. Although, in our work we did not use computation verification because we found it unnecessary in an evidence collection task. In an evidence collection task the aim is to make sure of the identity of the host not the origin of computation. However, the notion of verifiable computation has many uses which we covered in future work section in chapter five.
Chapter 4

Implementation

The proposed method will be implemented in three steps:

1. Establish a simple connection between Gateway and cloud. Doing a simple echo server running on the gateway. It can response to incoming messages through CoAP protocol. In this step we do not use encrypted communication and the aim is to use CoAP protocol to communicate between devices. More details about the program will be added later under “Simple GW-Cloud Communication” section.

2. Implement Edge computing. Move business logic to edge. “How to retrieve log files from gateways belong to private owners and Network provider companies in different regional locations?” is an open research problem which is challenging because the big number of IoT devices makes it almost impossible to retrieve all of those log files from gateways. On the other hand device owners doesn’t have will to share log files from their devices do to security and commercial competitions. A solution could be processing log files in the gateway resulting reduce the size of log files significantly and then transfer the smaller and preprocessed files to central IDS application in a proper time. Moreover, we can send latest patterns and signatures to gateway as a piece of code to update mal-traffic detection method in the gateway.

3. Secure Communication using DTLS protocol and ECE security suite. All of the traffic will exchange through a secured encrypted channel using DTLS protocol that is a variant of TLS for UDP transport paradigm. In this work we use Elliptic curve cryptography method which is lighter in processing aspect and more secure.[27]

4.1 Choose the Edge Computing Task

Logs are one of the most important types of data in software development life cycle, more specifically if we assume software development life cycle use-case surface as following order:
• Debugging and Forensics

• Fault monitoring

• Troubleshooting

• Feature usage

• Performance monitoring

• Security / incident detection

• Regulatory and standards compliance

We can see the essential role of logs in many of those. Logs are absolutely crucial in some and at least helps to solve problems in the rest. [4]. Logs are usually the last and sometimes the only hope in forensics investigation, both live or post incident, to find evidences.

Considering the threads in 3.1 and notice that many of those left traces in logs, encouraged us to research more about logs and the state-of-the-art researches and probably proposed solutions in edge computing paradigm.

According to [20], one of the challenges in edge computing security and forensics is collecting data from log files. As far as our knowledge, no developed or even proposed practical scientific work found in the community. All in all, we found the topic interesting and beneficial as a connection point between edge computing and a needed forensics use-case that is fits the research initial aims.

4.2 Implementation Design

In this section we describe the overall design goal and architecture. we will see the software engineering method and standard used in development process of the idea as well as the software program. Later on we implement a test prototype, first and finally the cloud and edge pairs of the software. For establish such a connection and then communicate gateway with cloud client (or the server application resident in cloud). We have to deploy two pieces of software; one installed on the edge gateway and the other one on cloud. The gateway part preliminarily has the ability to communicate with cloud (outer network), and then it perform some computing tasks which reduces overheads of communication and computation in the cloud.

4.2.1 Implementation Software Engineering

One of the most accepted software development methods is spiral architecture. It’s most distinguishing feature is that it creates a risk-driven approach to the software process, rather than a strictly specification-
driven or prototype-driven process[29]. It is simple and efficient in a way that can be used by an individual developer as well as enterprise developer teams. In this well known method we pass the stages of software development in an spiral shape as in each pass after evaluation we can eliminate errors and bugs from the last pass and this process will continues until no bugs and flows left in the product. Spiral model has four major steps:

1. Determine Objectives
2. Identify and Resolve Risks
3. Development and Test
4. Plan the Next Iteration

### 4.2.2 Simple Cloud-Edge Communication

As a first step in implementation and in order to get familiar with the environment of cloud and a gateway at the edge of the IoT network, we decide to develop a test software that can establish a connection between the edge gateway and cloud. To fulfill this simple communication, we developed a simple echo server in the cloud which uses CoAP as communication protocol without any encryption of data to make it easier to study the whole communication and read the raw data packets in plain text. This way we learned quite reasonable knowledge of CoAP communication and some limitations. For instance, since CoAP uses UDP ports 5683 and 5684, its traffic blocked by many firewalls, especially enterprise firewall configurations. This is actually one of the limitations when using UDP ports which can be overcome by utilizing a kind of tunnel or load the CoAP packet as payload into another TCP packet.

### 4.2.3 Edge Application

The edge application is the main focus of this project and the part that fulfills the edge computing task. The edge application is consist of many different parts which interact with each other as illustrated in figure three. The very first part to develop is the interface which acts as a terminal and connection manager. It handles all of the communications with the network at on bound and provide connection services to other processes in the local host (here the smart edge gateway). Main features of this piece of software are guaranty the safety of the connection and other security measures such as avoiding unwanted access to the internal processes. We utilize ECE security suite to encrypt the information which described in 4.2.5 section in details.

Another important part is a logger program which can work side-by-side the basic local logger programs to log some specific details from edge computing process to use in later stages of log collection. For instance
it logs details of log collections and caches the latest results in case of further interest in the local gateway from the cloud party. It also provide each log an access permission tag which helps easier categorization of logs by permission of the requester.

Log explorer is another process which explore through logs using tags, fast search methods, and ELK compatible. It can read ELK queries and response with the proper selected logs. Log explorer has a layer-driven architecture which excludes the confidential logs in last level before sending the response. This way we prevent possible injection attacks over other security.

### 4.2.4 Cloud Application

The cloud application has a great scalability potential, both in tasks it performs and communication with other cloud services. In this topology we have developed a communication terminal resident in cloud which communicates with ELK stack in one end and our edge gateway on the other end. However, there is no limitation for functionalities one can add to a cloud application but the available cloud technologies and infrastructure.

Our cloud application in its final version also uses DTLS protocol to encrypt all of the communications with the edge gateway and TLS in cloud to cloud communications with ELK stack installed in another cloud.

### 4.2.5 Secure Communication using DTLS protocol

In the early prototype we used CoAP protocol which transfers the data in plain text. It is obvious due to security risks it cannot be the case in a real life industrial use-case. Hence, in the practical version we need to encrypt the data before sending on the transform medium. Fortunately, CoAP has a secure version which basically is CoAP over TLS, called CoAPS. To have a better idea we can compare it with correlation of HTTP and HTTPS. CoAPS uses the same principle as HTTPS, the only difference is HTTP uses TCP transform layer while CoAPS uses UDP.

The DTLS protocol provides communications privacy for datagram protocols. The protocol allows client/server applications to communicate in a way that is designed to prevent eavesdropping, tampering, or message forgery. The DTLS protocol is based on the Transport Layer Security (TLS) protocol and provides equivalent security guarantees. Datagram semantics of the underlying transport are preserved by the DTLS protocol [30].

TLS is the most widely deployed protocol for securing network traffic. It is widely used for protecting Web traffic and for e-mail protocols such as IMAP and POP. The primary advantage of TLS is that it provides a transparent connection-oriented channel. Thus, it is easy to secure an application protocol by
inserting TLS between the application layer and the transport layer. However, TLS must run over a reliable transport channel – typically TCP. Therefore, it cannot be used to secure unreliable datagram traffic. The basic design philosophy of DTLS is to construct "TLS over datagram transport". To the greatest extent possible, DTLS is identical to TLS [30].

4.3 Client Authentication

Even-though, we are going to do edge processing and pre-filtering logs before sending them to the client, but to have a level of trust we need to authenticate the client. At the same time we want to have a generic way of authentication in a fashion which a legitimate client must be able to collect logs from different devices belong to different owners and under different regulations. Hence, the proposed solution is to have a hash pair for each device with a particular generation method. A scheme of the aforementioned method illustrated in figure 4.1.

Where "128 Bytes Random Number" is a random number generated during the production of each device in different stages and is unique for each device. First we select arbitrary amount of bytes out of the random number. The place and number of bits which are selected is a global constant which we can use the same algorithm to gain the same bits (if we have the complete random number). Later, each "n bytes" will combine
<table>
<thead>
<tr>
<th>Random number</th>
<th>123456789012345678911234567892…1212345678</th>
</tr>
</thead>
<tbody>
<tr>
<td>Current date</td>
<td>20180615</td>
</tr>
<tr>
<td>Concatenate number</td>
<td>56789012320180615</td>
</tr>
<tr>
<td>Hash</td>
<td>qx194Qk193r/Kz5xWraFpybbF14fS2FSsMTGQQQQtlyw=</td>
</tr>
</tbody>
</table>

Figure 4.2: Hash Generation Example

with current date as an eight-digit integer. For instance the combination can be a simple concatenation of two numbers or more complex combinations. Finally, a hash generates out of the combination result. It is shown in the figure 4.2.

Generated hash is unique hash which can obtain only today, and in another date we will have a completely different hash. Same steps took for the other pair of hash. One of these hashes will be public and can send to any requesting host, the other part will keep private. Only those hosts are able to communicate with the edge gateway which have the corresponding pair (the private hash). A third party host belong to a partner or a kind of authority who wants and allowed to collect logs from edge gateway receives a table containing hash pairs. This client can sent initial request to an edge gateway and receive its public hash. Then the client find and sends the corresponding hash for that particular edge gateway (if it has in its hash table). The point here is each hash-pair table is valid for a particular slice of time and after that point of time calculated hashes in the gateway will be different, thus the client is not recognized as a legitimate one and will not be authenticated.

4.4 Experimental Environment

To actually implement the idea we needed a platform ready to deploy cloud and edge applications and establish real connection between those. To do so we setup an experimental network environment in a physical network which its topology is illustrated in figure 4.3.
Figure 4.3: Experimental Network Environment Topology

- HmS Netbiter Gateway
- Performing Edge Computing
- Log Storage
- Log Processing
- Secure Connection
- Reduce Network traffic
- Distributed Log Collection
Chapter 5

Results

In this chapter, we summarize the final results of the research. We separately review test results, conclusion, suggestions, and future opportunities, and we discuss our idea and hypothesis in more details in the discussion section. We will evaluate how much successful and efficient our research was.

5.1 Test Results

Here we survey the initial expectations and the final results achieved in this work. Initial goals expected were:

1. Preserve Privacy

2. Low and Predictable Latency

3. Increase Robustness

4. Reduce Data Transmission

5.1.1 Preserve Privacy

One of the most important aims of the project was preserving privacy especially when we work in an industrial IoT environment. In this project we consider different security measures in different layers. As a lower level solid security, we have encryption of all transmitted information with the latest ECE method which unlike some of older encryption methods (i.e. RSA j1024) is not compromised yet and is the safest known wide used encryption method. Moreover, we used logical layers of security like selective log processing at the edge gateway. Finally, selected logs will scan for any confidential information based on generated access control
tags customizable by the owner of the gateway. The final check insures prevent any kind of injection which could pass other security barriers.

5.1.2 Lower Latency

Compare to collecting all of the logs from many devices it has less latency on the cloud part to send queries to many devices and collect the responses generated by gateways. Conversely, we had to design a mechanism to prevent a flood against the cloud part by many gateways prepared answers and want to send the selected logs back to the cloud client. We designed a queue system in which each edge gateway by preparing the result and before sending it back, first registers itself in a queue with cloud client and then cloud client requests gateways in the queue to send their responses one by one.

5.1.3 Increase Robustness

Since the process of initial logging and log retrieve takes place locally we don’t need to deal with different formats of logs. The application receives a query and then locally explores through logs which generated and stored locally in the same format. It guarantees the robustness of log retrieval.

5.1.4 Reduce Data Transmission

No need to mention compare transferring full-size log files and a selection subset of logs depend on the investigation scenario, less bandwidth is needed as well as less data will be transformed on the wire.

5.2 Discussion

-Is it possible to collect logs in an efficient way from edge devices belong to heterogeneous owners without compromising the confidentiality of data?

Refer to the initial scientific question we attempt to examine the possibility to collect logs from an edge device (gateway in this work) with several qualities that; first does the log collection selectively and only the logs that contain desired information retrieved from the gateway, second, not to compromise edge-device’s private and confidential data to an extent that heterogeneous gateways belong to different owners can trust the cloud investigator entity.

In this way, we need to investigate and find the best available technologies and methods. We found the most suitable protocols, programming languages, and platforms to implement our proposed idea. Finally,
after logical planning and software development rounds we successfully implement the logic to an actual application. The application executed, tested, and results collected.

For example, we state one query and evaluate its results with and without edge computing. The results are original, for example, size of transmitted data presented even plus cryptographic padding and overhead.

Query: logs between 2018-06-01 to 2018-07-01 with severity level more than 3 that include these patterns (regex) "field" or ".service" but not "systemd"?

Translated Query to JSON:

```
{
    "date_from": "2018-06-01",
    "date_to": "2018-07-01",
    "severity_level": "3",
    "interface": "eth*",
    "include": ["field", ".service"],
    "exclude": ["systemd"]
}
```

In table 5.1 we summarized the results for the example query.
<table>
<thead>
<tr>
<th>Aspect</th>
<th>Without Edge Computing</th>
<th>With Edge Computing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low Latency</td>
<td>All the logs within the time period must send back to cloud which imposes latency in many-device log collection. moreover, sending results back to the cloud as a stream over UDP channel increases data loss and latency.</td>
<td>As the query receives, all other operations perform locally</td>
</tr>
<tr>
<td>Response Size</td>
<td>&gt;5 MB</td>
<td>&lt;1024 bytes (one UDP packet payload)</td>
</tr>
<tr>
<td>Access Control</td>
<td>remote host must have access privilege to run &quot;journalctl&quot; command with limited query options.</td>
<td>No access permission needed for the remote host.</td>
</tr>
<tr>
<td>Context Awareness</td>
<td>Remote host have no idea of the log format, stored logs addresses and providing these information is a security breach.</td>
<td>without disclosure of any local information query is fully compatible because the log exploration performs locally and by our edge computing solution. log format, storage, confidential information, permissions.</td>
</tr>
<tr>
<td>Privacy</td>
<td>Either a gateway let a remote device to access its logs or not. (no levels of trust)</td>
<td>-Gateway could successfully exclude confidential information from the generated results.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-Remote hosts have no access to gateway in system level.</td>
</tr>
</tbody>
</table>

Table 5.1: Edge Computing Results Comparison

Although, we have done just a proof of edge computing concept and its significant effect on the performance of the network. Results presented in table 5.1 clearly shows that our initial expectations are addressed by this research. Moreover, we could successfully implement a solution that pointed out as an open research area by many researchers in the community. Our work was a successful attempt to fill the existing gap in forensics domain in the edge of IoT paradigm. However, the nature of fog requires distributed and interconnected solutions. Thus a stand-alone software barely can address the ultimate promises of fog and edge computing. From this point of view, our project is not an exception and needs other parts of a full stack system to collaborate with and form a meaningful and global solution that can provide an intelligence over the low-level data.
5.3 Conclusion

In this work, we study the Fog/Edge computing and found differences, overlapping areas, available technologies, protocols, and standards available in this domain. We realized some notions adopted like DTLS and some particularly designed for IoT and edge computing such as CoAP. We investigate technologies, platforms, and dominant cloud providers as well as programming languages and application layer protocols. Later in chapter three, we survey related works and open research topics.

After studying and discover the state-of-the-art in the topic and the trend in the community we planned for our practical solution and algorithms. We found the importance of logs and its crucial role in forensics ecosystem at the same time the need of research on logs and especially "log collection challenge" which pointed out by different academic surveys and researches. We implement our ultimate software considering robust security measures and methods.

Finally, we test our edge computing application in an actual cloud-edge network and summarize the results in the last chapter.

5.4 Suggestions

The main idea of edge computing is to provide faster and more reliable services in a more efficient way by devices which are generally have constrained hardware resources. As we reviewed, those services over the network are performed in a secure way using encryption for all communications. To Encrypt and decrypt all of the information at each gateway consume relatively a big amount of processing resource and when it comes to do all of cryptographic tasks in software program the efficiency is less. In compare cryptographic madules are available in the market which significantly reduce processing over-head from the main processing core. For instance the gateway we work with in this project (HMS EC350 Netbiter) benefits Cortex-A53 ARM processor without a cryptographic extra module. as a result all of the cryptographic calculations perform side-by-side the higher level edge computing and business logic tasks.

However, Cortex-A53 processor Cryptography Extension exists that supports the ARMv8 Cryptography Extensions. The Cryptography Extensions add new A64, A32, and T32 instructions to Advanced SIMD that accelerate Advanced Encryption Standard (AES) encryption and decryption, and the Secure Hash Algorithm (SHA) functions SHA-1, SHA-224, and SHA-256 [57]. It makes cryptographic tasks significantly faster at the same time free main processor to handle other processes with more resources.

As a suggestion for future generations of gateways, We found it useful and worthy to choose the type of processors with build-in hardware implemented cryptographic modules.
5.5 Future Works

Edge computing is a new technology and needs initial development. Furthermore, its critical and unique topological position offers a great potential to employ in many interesting sections to it, such as; security and forensics, AI, data mining, quality of service, and many more use-cases by the time. This work was just a good start to determine possibilities and features of edge computing and a "hello world" application in a set of interconnected applications that can perform a full stack of a real-life edge service.

As a future work in this aspect, it is recommended to also research on edge computing opportunities in live forensics. Study protocols such as NetFlow and NetFlow Security Event Logging with an edge computing point of view can be very interesting.
Appendix A

Source Codes

A.1 Codes

A.1.1 Cloud Application

The Cloud application consist of one main class and other programs which provide services such as file
operations, rules manager, editor, log file reader, and log selector. The following code is the main class
which handles the main communication and is the central node for edge computing.

/***********************************************************************************/

* Copyright (c) 2018 Shooresh Sufiye,

* 

* Halmstad University

* 

* HMS Industrial Networks

* 

***********************************************************************************/

package org.eclipse.californium.examples;

import java.io.BufferedReader;
import java.io.DataOutputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
//import java.net.Inet4Address;
import java.net.InetAddress;
import java.net.InetSocketAddress;
//import java.net.MalformedURLException;
//import java.net.ProtocolException;
import java.net.URI;
import java.net.URISyntaxException;
import java.net.URL;
import java.security.GeneralSecurityException;
import java.security.KeyStore;
import java.security.PrivateKey;
import java.security.cert.Certificate;
import java.util.HashMap;
import java.util.LinkedList;
import java.util.Queue;
import java.util.Scanner;
import java.util.Set;
import java.util.logging.Level;

//import org.json.simple.*;
import org.json.simple.parser.ParseException;
import org.json.JSONObject;
import org.eclipse.californium.core.CoapClient;
import org.eclipse.californium.core.CoapHandler;
import org.eclipse.californium.core.CoapResource;
import org.eclipse.californium.core.CoapResponse;
import org.eclipse.californium.core.CoapServer;
import org.eclipse.californium.core.Utils;
import org.eclipse.californium.core.WebLink;

import org.eclipse.californium.core.coap.CoAP.ResponseCode;
import org.eclipse.californium.core.network.CoapEndpoint;
import org.eclipse.californium.core.network.Endpoint;
import org.eclipse.californium.core.network.EndpointManager;
import org.eclipse.californium.core.network.config.NetworkConfig;
import org.eclipse.californium.core.network.interceptors.MessageTracer;
import org.eclipse.californium.core.server.resources.CoapExchange;
import org.eclipse.californium.scandium.DTLSConnector;
import org.eclipse.californium.scandium.ScandiumLogger;
import org.eclipse.californium.scandium.config.DtlsConnectorConfig;
import org.eclipse.californium.scandium.dtls.cipher.CipherSuite;
import org.eclipse.californium.scandium.dtls.pskstore.InMemoryPskStore;
import org.eclipse.californium.scandium.dtls.pskstore.StaticPskStore;

public class SCCI {

    static {
        ScandiumLogger.initialize();
        ScandiumLogger.setLevel(Level.WARN);
    }

    public static final int DTLS_PORT = 5684;

    private static final String TRUST_STORE_PASSWORD = "rootPass";
    private static final String KEY_STORE_PASSWORD = "endPass";
    private static final String KEY_STORE_LOCATION = "certs/keyStore.jks"
    private static final String TRUST_STORE_LOCATION = "certs/trustStore.jks"
    private static final String SERVER_URI = "coaps://localhost/secure"

    private static String server_res = "";
    private static Queue<String> ready_queue = new LinkedList<String>();
private static String toELK = "";
private static CoapClient client = null;
private static CoapServer fetchServer = null;
private DTLSConnector dtlsConnector;
static Scanner inp = new Scanner(System.in);

public SCCI() {
try {
// load key store
KeyStore keyStore = KeyStore.getInstance("JKS");
InputStream in = getClass().getClassLoader().getResourceAsStream(KEY_STORE_LOCATION);
keyStore.load(in, KEY_STORE_PASSWORD.toCharArray());
in.close();

// load trust store
KeyStore trustStore = KeyStore.getInstance("JKS");
in = getClass().getClassLoader().getResourceAsStream(TRUST_STORE_LOCATION);
trustStore.load(in, TRUST_STORE_PASSWORD.toCharArray());
in.close();

// You can load multiple certificates if needed
Certificate[] trustedCertificates = new Certificate[1];
trustedCertificates[0] = trustStore.getCertificate("root");

DtlsConnectorConfig.Builder builder = new DtlsConnectorConfig.Builder(new InetSocketAddress(0));
builder.setPskStore(new StaticPskStore("Client_identity", "secretPSK".getBytes()));
builder.setIdentity((PrivateKey)keyStore.getKey("client", KEY_STORE_PASSWORD.toCharArray()),
keyStore.getCertificateChain("client"), true);
builder.setTrustStore(trustedCertificates);
dtlsConnector = new DTLSConnector(builder.build());

} catch (GeneralSecurityException | IOException e) {
System.err.println("Could not load the keystore");
e.printStackTrace();

// the method to establish a connection to the end-point server program
public void connect() {
    URI uri = null;
    CoapResponse response = null;
    try {
        System.out.println("[SCC] Enter edge device address? ");
        String uri_str = inp.next();
        uri = new URI(uri_str);
        client = new CoapClient(uri);

        // set the end-point which this client is going to connect to
        client.setEndpoint(new CoapEndpoint(dtlsConnector, NetworkConfig.getStandard()));

        // observe resources of the end-point (edge gateway)
        Set<WebLink> res = client.discover();
        System.out.println("[SCC] Discovered Resources:");
        for (WebLink w : res) {
            System.out.println("resource: "+ w);
        }

        // get request to the main coaps uri not any of resources
        response = client.get();
    } catch (URISyntaxException e) {
        System.out.println("[SCC] It is not a valid address. localhost selected");
        System.err.println("Invalid URI: " + e.getMessage());
        System.exit(-1);
    }

    if (response != null) {
        System.out.println(response.getCode());
        System.out.println(response.getOptions());
    }
System.out.println(response.getResponseText());
System.out.println("\nADVANCED\n");
System.out.println(Utils.prettyPrint(response));

} else {
System.out.println("No response received.");
}

// the method to create and add a server to this host in order to receive data from the gateway
public static String addserver() {
// discover all available NICs on this host
String endpoint ="";
if(fetchServer==null) {
int i=1;
InetAddress[] addresses = new InetAddress[32];
for(InetAddress addre : EndpointManager.getEndpointManager().getNetworkInterfaces()) {
if (addre instanceof InetAddress) {
addresses[i] = addre;
System.out.println("address "+i++ +"\t"+addre);
}
}
System.out.println("[SCC] Choose endpoint to receive queue? ");
i = inp.nextInt();
endpoint = "coaps://"+ addresses[i].toString()+"":5684/queue";

fetchServer = new CoapServer();

fetchServer.add(new CoapResource("queue") {
public String reg_cli = "";
@Override
public void handlePOST(CoapExchange exc) {
reg_cli = exc.getRequestText();
ready_queue.add(reg_cli.substring(reg_cli.indexOf(">")+1));
System.out.println("\n[SCC] A host is ready:\n\t->"+reg_cli);
exc.respond("you are registered");
}
});

// prepare DTLS connection
try {
    // Pre-shared secrets
    InMemoryPskStore pskStore = new InMemoryPskStore();
    pskStore.setKey("password", "sesame".getBytes()); // from ETSI Plugtest test spec

    // load the trust store
    KeyStore trustStore = KeyStore.getInstance("JKS");
    InputStream inTrust = SecureServer.class.getClassLoader().getResourceAsStream(TRUST_STORE_LOCATION);
    trustStore.load(inTrust, TRUST_STORE_PASSWORD.toCharArray());

    // You can load multiple certificates if needed
    Certificate[] trustedCertificates = new Certificate[1];
    trustedCertificates[0] = trustStore.getCertificate("root");

    // load the key store
    KeyStore keyStore = KeyStore.getInstance("JKS");
    InputStream in = SecureServer.class.getClassLoader().getResourceAsStream(KEY_STORE_LOCATION);
    keyStore.load(in, KEY_STORE_PASSWORD.toCharArray());

    DtlsConnectorConfig.Builder config = new DtlsConnectorConfig.Builder(new InetSocketAddress(addresses[i]));
    config.setSupportedCipherSuites(new CipherSuite[]{CipherSuite.TLS_PSK_WITH_AES_128_CCM_8,
            CipherSuite.TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8});
    config.setPskStore(pskStore);
    config.setIdentity((PrivateKey)keyStore.getKey("server", KEY_STORE_PASSWORD.toCharArray()),
            keyStore.getCertificateChain("server"), true);
    config.setTrustStore(trustedCertificates);

    DTLSConnector connector = new DTLSConnector(config.build());
fetchServer.addEndpoint(new CoapEndpoint(connector, NetworkConfig.getStandard()));
for (Endpoint ep : fetchServer.getEndpoints()) {
    System.out.println("endpoint<. " + ep.getAddress());
}

fetchServer.start();
}

} catch (GeneralSecurityException | IOException e) {
    System.err.println("Could not load the keystore");
    e.printStackTrace();
}

// add special interceptor for message traces
for (Endpoint ep : fetchServer.getEndpoints()) {
    ep.addInterceptor(new MessageTracer());
}

System.out.println("Secure CoAP server powered by Scandium (Sc) \nis listening on port " + DTLS_PORT);

fetchServer.addEndpoint(new CoapEndpoint(new InetSocketAddress(addresses[i], 5684)));

} else {
    System.out.println("[SCC] A server already exist at coap://" + fetchServer.getEndpoints().get(0).getAddress() + ");
    System.out.println("[SCC] Fetch server started");
    return endpoint;
}

public static void main(String[] args) throws InterruptedException, IOException, JSONException, ParseException {
System.out.println("+-------[ Secure Cloud Client ]-------+" );
System.out.println("|
  Shooseh Sufiye
|");
System.out.println("|
  Halmstad University
|");
System.out.println("|
  HMS Industrial Networks
|");
System.out.println("|
  (c) 2018
|");
System.out.println("+---------------------------------------+" );

String endpoint = addserver();
SCCI sclient = new SCCI();
sclient.connect();

HashMap<String,String> hashes = new HashMap<String,String>();
hashes.put("public id", "default");
// daily hash pairs must be obtain for the current date
hashes.put("KjcgpsVMX0Po4pU3CFKzxyvpxmgZutnZniZ5xyb1h/c=", "6W4+iOEWIGeb8a6kCcEeqAuGuXaE0Ewktzit9pI1Jn/");

String addr = null;
CoapResponse response = null;
String ch = null;
String pay ="";
String res_name = "";
String pubid = "public id";

do {
  System.out.println("+-------[ Client ]-------+");
  System.out.println("|
  1.GET resource
|");
  System.out.println("|
  2.POST rules
|");
  System.out.println("|
  3.Fetch logs
|");
  System.out.println("|
  4.Forward to ELK
|");
  System.out.println("|
  5.Rules on Smpl log
|");
  System.out.println("|
  0.EXIT
|");
  System.out.println("+---------------------------------------+" );
  System.out.print ("[ CHOOSE ]\n[?] ");
}
ch = inp.nextLine();
switch (ch) {
    case "1":
        System.out.println("[SCC] Enter resource name? ");
        res_name = inp.next();
        addr = client.getURI();
        System.out.println("[SCC] " + addr+res_name);
        // if is address
        try {
            client.setURI(addr+res_name);
        } catch (Exception e) {
            System.out.println("[SCC] The address you entered is not correct\n"+e);
        }
        response = client.get();
        client.setURI(addr);
        System.out.println("[SCC] server response:\n" + Util.prettyPrint(response));  //response.getResponseText()

        JSONObject js = new JSONObject(response.getResponseText());
        if(js.has("pubid")) {
            pubid = js.getString("pubid");
            System.out.println("[SCC] [" + addr+"]’s public hash id is " + pubid);
        }

        JSONParser jp = new JSONParser();
        // type structure
        Object receivedObj = jp.parse(response.getResponseText());

        // recognize end-point device
        break;
    case "2":
// GET the end-point/resource
//System.out.print("End-point resource name? ");
//res_name = inp.nextLine();
res_name = "new_rules";
addr = client.getURI();
System.out.println("[SCC]" + addr+res_name);
try {
    client.setURI(addr+res_name);
} catch (Exception e) {
    System.out.println("[SCC] The address you entered is not correct\n"+e);
}
// read rules and add proper hash to it
//System.out.print("File name? ");
//String filename = inp.nextLine();
File f = new File("new_rules.txt"); //filename);
FileInputStream fs = new FileInputStream(f);
byte[] b = new byte[(int) f.length()];
fs.read(b);
StringBuilder sb = new StringBuilder();
for(byte i : b) {
    sb.append((char) i);
}
pay = sb.toString();
//System.out.println(pay);
// to json add private hash and send as JSON or text
JSONObject json_post = new JSONObject(pay);
if(hashes.get(pubid)!="null") {
    json_post.put("private_hash", hashes.get(pubid));
} else {
    json_post.put("private_hash", "not found");
}
json_post.put("reg_server", endpoint);
pay = json_post.toString();
// add a server to receive logs
CoapHandler post_handler = new CoapHandler() {
    String s = "";
    @Override
    public void onLoad(CoapResponse response) {
        s = response.advanced().getPayloadString();
        // hard coded member to queue
        System.out.println("\n[SCC] Server response:\n +"+\n");
    }
    @Override
    public void onError() {
        System.out.println("[SCC] Error accured during POST!");
    }
};

client.post(post_handler, pay, 50); // 0: plain text, 50: JSON, 40: link, 41: XML

client.setURI(addr);
break;
case "3":

// later on read in turn
String next = "";
if(ready_queue.isEmpty()) {
    System.out.println("[SCC] Queue is empty");
} else {
    while(!ready_queue.isEmpty()) {
        next = ready_queue.poll();
        System.out.println("[SCC] Next device in fetch queue is "+next+" ");
        client.setURI(next);
    }
    System.out.println("[SCC] Fetching from:"+next);
CoapResponse toelk_res = client.get();
toELK = toelk_res.getResponseText();
System.out.println("[SCC] Ready to ELK:\n" + new JSONObject(toELK).toString(4)); //Utils.prettyPrint(toELK)
}

break;
case "4" :
  // http or coap forward toELK to Logstash URI
  //toELK = toELK.substring(toELK.indexOf("\"text\": \""\")+"\"text\": \"".length(), toELK.length()-2);
  StringBuffer sbuf = new StringBuffer();
  sbuf.append(toELK+"\n");
  String elk_url = "http://httpbin.org/post";
  URL url1 = new URL(elk_url);
  HttpURLConnection http_con = (HttpURLConnection) url1.openConnection();
  http_con.setRequestMethod("POST");
  http_con.setRequestProperty("User-Agent", "Mozilla/5.0");
  http_con.setRequestProperty("Accept-Language", "en-US,en;q=0.5");
  String url_parameters = sbuf.toString();
  http_con.setDoOutput(true);
  DataOutputStream dout = new DataOutputStream(http_con.getOutputStream());
  dout.writeBytes(url_parameters);
  dout.flush();
  dout.close();

  int http_res_code = http_con.getResponseCode();
  System.out.println("[SCC] HTTP server response:");
  BufferedReader br = new BufferedReader(new InputStreamReader(http_con.getInputStream()));
  String line = "";
  StringBuffer sb_in = new StringBuffer();
  while((line = br.readLine()) != null) {
    sb_in.append(line);
  }
  String reply = sb_in.toString();
  http_con.disconnect();
  http_con.getConnection().disconnect();

}
A.1.2 Edge Application

The main edge computing engine which glue all other program modules is SECI (Secure Edge Computing Instance). The following code is SECI.java file's content that performs edge computing tasks as well as outside-world communications.

```java
package org.eclipse.californium.examples;
```
/***************************************************************
 * Copyright (c) 2018 Shooresh Sufiye,
 * 
 * Halmstad University
 * 
 * HMS Industrial Networks
 * 
 ***************************************************************

import java.io.IOException;
import java.io.InputStream;
import java.net.Inet4Address;
import java.net.InetAddress;
import java.net.InetSocketAddress;
import java.nio.charset.StandardCharsets;
import java.security.GeneralSecurityException;
import java.security.KeyStore;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.security.PrivateKey;
import java.security.cert.Certificate;
import java.text.SimpleDateFormat;
import org.eclipse.californium.scandium.util.Base64;
import java.util.Date;
//import java.util.List;
import java.util.Scanner;
//import java.util.concurrent.ExecutorService;
import java.util.logging.Level;

import org.eclipse.californium.core.CaliforniumLogger;
import org.eclipse.californium.core.CoapResource;
import org.eclipse.californium.core.CoapServer;
//import org.eclipse.californium.core.coap.EmptyMessage;
//import org.eclipse.californium.core.coap.Request;
//import org.eclipse.californium.core.coap.Response;
import org.eclipse.californium.core.coap.CoAP.ResponseCode;
import org.eclipse.californium.core.network.CoapEndpoint;
import org.eclipse.californium.core.network.Endpoint;
import org.eclipse.californium.core.network.EndpointManager;
//import org.eclipse.californium.core.network.EndpointObserver;
//import org.eclipse.californium.core.network.Exchange;
import org.eclipse.californium.core.network.config.NetworkConfig;
//import org.eclipse.californium.core.network.interceptors.MessageInterceptor;
import org.eclipse.californium.core.network.interceptors.MessageTracer;
//import org.eclipse.californium.core.server.MessageDeliverer;
import org.eclipse.californium.core.server.resources.CoapExchange;
import org.eclipse.californium.scandium.DTLSConnector;
import org.eclipse.californium.scandium.ScandiumLogger;
import org.eclipse.californium.scandium.config.DtlsConnectorConfig;
import org.eclipse.californium.scandium.dtls.cipher.CipherSuite;
import org.eclipse.californium.scandium.dtls.pskstore.InMemoryPskStore;

public class SECI extends CoapServer {

    private static Scanner inp = new Scanner(System.in);
    public static final int COAP_PORT = 5684 ; // NetworkConfig.getStandard().getInt(NetworkConfig.Keys.COAP_PORT);
    public static String end_p;
    public static SECI server = null;

    static {
        CaliforniumLogger.initialize();
        CaliforniumLogger.setLevel(Level.CONFIG);
        ScandiumLogger.initialize();
        ScandiumLogger.setLevel(Level.FINER);
    }
}
// allows configuration via Californium.properties

public static final int DTLS_PORT = 5684; // NetworkConfig.standard().getInt(NetworkConfig.Keys.COA_LARGE_DTLS_PORT)

private static final String TRUST_STORE_PASSWORD = "rootPass'';
private final static String KEY_STORE_PASSWORD = "endPass'';
private static final String KEY_STORE_LOCATION ="certs/keyStore.jks";
private static final String TRUST_STORE_LOCATION ="certs/trustStore.jks";

private String addEndpoints() {
    int i=1;
    // up to 32 NICs can be store in addresses array
    InetAddress[] addresses = new InetAddress[32];
    // iterates through all net interfaces and store IPs into addr
    for (InetAddress addr : EndpointManager.endpointManager().getNetworkInterfaces()) {
        // only binds to IPv4 addresses and localhost
        if (addr instanceof Inet4Address || addr.isLoopbackAddress()) {
            addresses[i] = addr;
            System.out.println("address "+ i++ +"\t"+ addr);
        }
    }

    System.out.print("Choose? ");
    int c = inp.nextInt();
    // reading raw end point IP address, then construct a full address
    end_p = "coaps://" + addresses[c].toString() + ":" + COAP_PORT + "/";
    System.out.println("end_p "+ end_p);
    //addEndpoint(new CoapEndpoint(new InetSocketAddress(addresses[c], COAP_PORT)));
    return addresses[c].toString().substring(1);
}

// the method to generate SHA-2 hash out of a string
private byte[] toHash(String Res_id) {
    MessageDigest digest = null;
    try {
digest = MessageDigest.getInstance("SHA-256");
} catch (NoSuchAlgorithmException e) {
    e.printStackTrace();
}

byte[] hash = digest.digest(Res_id.getBytes(StandardCharsets.UTF_8));

System.out.println("id hash >"+ hash.length);

return hash;
}

// The method to convert a hash byte array to string representation. using base64 coding
private String hashtoString(byte[] hash) {
    // convert using base64"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/"
    //System.out.println(hash+ " string "+ hash.toString());
    String s = Base64.encodeBytes(hash); // (java.util base64 method => ) .getEncoder().encodeToSe
    //System.out.println("base64 representation of hash " + s);
    return s;
}

public static void main(String[] args) {
    System.out.println("+-------[ Secure Edge Gateway ]-------+");
    System.out.println("|
    Shooresh Sufiya
    "); System.out.println("|
    Halmstad University
    "); System.out.println("|
    HMS Industrial Networks
    "); System.out.println("|
    (c) 2018
    "); System.out.println("+-----------------------------------------------+");
    server = new SECI();

    server.add(new CoapResource("secure") {
        @Override
        public void handleGET(CoapExchange exchange) {
            exchange.respond(ResponseCode.CONTENT, "hello security");
        }
    });
System.out.println("from handle GET
+exchange.toString());
}
});

try {
    // Pre-shared secrets
    InMemoryPskStore pskStore = new InMemoryPskStore();
    pskStore.setKey("password", "sesame".getBytes()); // from ETSI Plugtest test spec

    // load the trust store
    KeyStore trustStore = KeyStore.getInstance("JKS");
    InputStream inTrust = SecureServer.class.getClassLoader().getResourceAsStream(TRUST_STORE_LOCATION);
    trustStore.load(inTrust, TRUST_STORE_PASSWORD.toCharArray());

    // You can load multiple certificates if needed
    Certificate[] trustedCertificates = new Certificate[1];
    trustedCertificates[0] = trustStore.getCertificate("root");

    // load the key store
    KeyStore keyStore = KeyStore.getInstance("JKS");
    InputStream in = SecureServer.class.getClassLoader().getResourceAsStream(KEY_STORE_LOCATION);
    keyStore.load(in, KEY_STORE_PASSWORD.toCharArray());

    DtlsConnectorConfig.Builder config = new DtlsConnectorConfig.Builder(new InetSocketAddress(server.address(), port));
    config.setSupportedCipherSuites(new CipherSuite[]{CipherSuite.TLS_PSK_WITH_AES_128_CCM_8,
                                           CipherSuite.TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8});
    config.setPskStore(pskStore);
    config.setIdentity((PrivateKey)keyStore.getKey("server", KEY_STORE_PASSWORD.toCharArray()), keyStore.getStore("JKS", KEY_STORE_PASSWORD.toCharArray()));
    config.setTrustStore(trustedCertificates);

    DTLSConnector connector = new DTLSConnector(config.build());

    server.addEndpoint(new CoapEndpoint(connector, NetworkConfig.getStandard()));
for (Endpoint ep : server.getEndpoints()){
    System.out.println("[SEC] endpoint<.> " + ep.getAddress());
}

Date today = new Date(System.currentTimeMillis());
SimpleDateFormat formatter = new SimpleDateFormat("yyyy-MM-dd");
String today_str = formatter.format(today).replaceAll("-", "");

System.out.println("[SEC] today " + today_str);
EciResource res1 = new EciResource("identity");
res1.IDpub = server.hashToString(server.toHash("1234" + today_str)); //daily hash. serial number of
res1.set_priv_id(server.hashToString(server.toHash("4567" + today_str))); //
res1.json = "{\"a\":\"b\"}";
res1.text_f = "old rules";
server.add(res1);

System.out.printf("\npub id\t%s\npriv id\t%s\npub\t%s\npriv\t%s\n","1234" + today_str, "4567" + today_str);

EciResource res2 = new EciResource("new_rules");
res2.ID = "111000112";
res2.json = "{\"c\":\"d\"}";
res2.text_f = "rules come here";
server.add(res2);

EciResource res3 = new EciResource("filtered");
res3.ID = "111000113";
res3.json = "{\"c\":\"d\"}";
res3.text_f = "logs come here";
server.add(res3);

server.start();

} catch (GeneralSecurityException | IOException e) {

```
System.err.println("[SECI] Could not load the keystore");
e.printStackTrace();
}

// add special interceptor for message traces
for (Endpoint ep : server.getEndpoints()) {
    ep.addInterceptor(new MessageTracer());
}

System.out.println("[SECI] Secure CoAP server powered by Scandium (Sc) \nis listening on port " + DTLS_server_port + "]

}
Bibliography


59


[41] https://cloud.google.com/iot-edge/, last access 2018 Aug.


Other References


Shoresh Sufiye received his B.Sc in Computer Science from Tehran Technical and Vocational University in 2016. His research interests are IoT edge, distributed computing, and AI.