Claims-Aware Middleware for Securing IoT Services

by

Vathalloor Merin George

A Thesis submitted to
University of Ontario Institute of Technology

in partial fulfillment of requirements
for the degree of
Master of Applied Science
in
Electrical and Computer Engineering

Oshawa, Ontario, Canada

ABSTRACT

Claims-Aware Middleware for Secure IoT Services

Vathalloor Merin George
University of Ontario Institute of Technology, 2016

Advisor:
Professor Qusay H. Mahmoud

Take a look at the world around us. There has been tremendous change in the way of living. With the world around us getting smart, *Internet of Things* is gaining ground in our life. Applications like smart home and eHealth are so user friendly that any person with zero programming background is able to use it. But for the developer, due to the ubiquitous nature and distributed architecture of IoT which includes devices, applications and humans, it presents a complex structure. Also, the incorporation of thousands of heterogeneous *things* with different configurations into a single network creates the risk of threat against security and privacy. These challenges make the significance of a middleware important. Middleware is a software layer that provides the platform for various devices with different protocols to communicate with ease and provides all the functions intended for a particular task. Hosting these tasks as microservices simplifies the job of an application developer. In this thesis work, we introduce a claims-aware middleware to address one of the major challenges in IoT which is security. A proof of concept has been developed by implementing a prototype of our framework. The evaluation results of the prototype show the feasibility and the stability of the security framework.

Dedication

To my father for he has been my backbone since the day I was born.

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my supervisor, Dr. Qusay Mahmoud, for his continuous support and encouragement right from the beginning of this research work. Without his guidance, I would have never been able to complete this work. He has been extremely patient and supported me when I was struggling with the work and directed me towards the right track. I will always be thankful for the times that he had let me postpone the due dates in several phases of my research and let me work at my slow pace. I will always remember the generous funding he provided and the fact that he always had the faith in my research ability.

I would like to thank my friend, Orane Cole, for his boundless help throughout this research period. The motivation he gave me when I felt helpless is commendable. I wish him and his family all the very best and may God bless you. Also, I thank my friend, Jortin Mathew, for his support and help.

I cannot go further without expressing ma sincere gratitude to my friend Subin Ben for his moral support and motivation. His words were really inspirational and kept me going forward.

I would like to thank my family members, especially my father, as without the continuous encouragement from him I might never have completed this work. He is my role model as his continuous effort and hard work to complete any given task, has always inspired me. I thank all my friends for all the love and prayers.

Table of Contents

.

ACRONYMS AND ABBREVIATIONS

IoT	Internet of Things
IERC	European Research Cluster on the Internet of Things
WSN	Wireless Sensor Networks
SQL	Structured Query Language
NoSQL	Not Only SQL
M2M	Machine-to-Machine
WCF	Windows Communication Foundation
API	Application Programming Interface
HTTP	HyperText Transfer Protocol
DBMS	Database Management System
REST	REpresentational State Transfer
RPC	Remote Procedure Call
OS	Operating System
MQTT	Message Queuing telemetry Transport
URI	Uniform Resource Identifier
XMPP	Extensible Messaging and Presence Protocol
SASL	Simple Authentication and Security Layer
TLS	Transport Layer Security
SIoT	Social driven IoT
STS	Security Token Service
QoS	Quality of Service
RPi	Raspberry Pi
JDK	Java Development Kit
JRE	Java Runtime Environment
JSON	Java Script Object Notation

SAML	Security Assertion Markup Language
IdP	Identity Provider
TCP	Transport Control Protocol
UPnP	Universal Plug and Play
IP	Internet Protocol
OTP	One Time Password
BLE	Bluetooth
PoE	Power over Ethernet
LAN	Local Area Network
AES	Advanced Encryption Standards
CBC	Cipher Block Chaining

LIST OF FIGURES

LIST OF TABLES

Chapter 1

Introduction

This chapter gives a brief introduction about the topics that are present in our research work. An overview of Internet of Things and its security challenges are also introduced. A short detailing about our motivation to select this research area along with the research statement has been presented. The main contributions of our thesis have been pointed out in this chapter.

1.1. Internet of Things

Most of the devices in our daily life perform some sort of operations or services. It was found out to be so productive if those devices such as mobile phones, televisions or even a sensor could communicate with each other. Such a system where objects (things) could be recognized uniquely and obtain intelligence by analysis of an event and reacting accordingly to produce a smart reaction, could be called Internet of Things. Each such object may access information from other entities and may contribute a small part of service within the whole system. The strong pillars of IoT are cloud computing and the transition of Internet towards IPv6 with almost unlimited addressing capacity [1].

IERC (European Research Cluster on the Internet of Things) along with ITU-T (International Telecommunication Union) Study Group 13 has been working together on standards for future generation networks. They formulated a definition [2] for Internet of Things which explains the system in a highly technical manner with additional information on border perspective of IoT applications and services. But the definition given by IERC [1] is more simple and it states that IoT is "*A dynamic global network infrastructure with self-configuring capabilities based on standard and interoperable communication protocols where physical and virtual "things" have identities, physical attributes, and virtual personalities and use intelligent interfaces, and are seamlessly integrated into the information network.*".

Healthcare, automotive, smart grid and home automation are just few applications which use IoT. With the help of wireless sensor networks (WSN), control and monitoring of dumb nodes are made much more efficient [3]. Unlike the machine-to-machine (M2M) communication, IoT includes man in the loop, either to control the things or to react with the result produced from the things. There are various areas like transportation, logistics, field service and so on, in which coupling of *information* and *things* may create highly efficient services [1]. The things not only obtain information from the surrounding but also interact with each other to either produce a command or to control a situation [4]. The applications that use the idea of IoT range from entertainment programs to security applications. With the increased dependency of people on the Internet, keeping every single data secure is also important.

One of the major challenges in Internet of Things is *security*. The attacks against security and privacy are considered challenging because things are low power objects with low memory capacity. Thus complex cryptographic algorithms may not be supported and the fact that most of them will be wirelessly connected makes it more vulnerable to attacks. In an application like wireless sensor networks for healthcare, security and privacy threats could affect the person in a very bad manner-if the data is leaked as it is very sensitive and any information regarding the patient can be used by the adversary which put patient’s life at risk. Another scenario could be home automation in which the energy is managed efficiently with the help of monitoring sensors. The data collected from those devices could be used by an attacker to retrieve the information whether the home is empty and utilize it for compromising the security system. Thus, security at both the edge (device/things) level and network level is required in IoT [5].

1.2. Motivation

Protection of data has always been a problem that existed in communication between two parties. So in a network of things where millions of nodes will be communicating, securing the data is a nontrivial task. Also, the heterogeneous nature of the different devices, sensors or applications that have to communicate with each other to produce meaningful result or reaction according to an event, also demands for a mechanism to support interoperability between different protocols.

In the past few years, the demand of IoT applications has been increased and the world is becoming smart. An interesting yet serious question that has to be answered is that whether the IoT services are secure. The catastrophe that may happen if the personal data of the user falls into a wrong hand would be irrevocable. Thus the basic problem is to provide secure IoT services without causing any degradation in the performance of the IoT applications and services.

1.3. Research Statement

To achieve a secure middleware for Internet of Things, first of all, we need to study the existing frameworks for IoT and learn the basic functionalities of the middleware. Also, it is important for us to learn the drawbacks of the existing solutions so that we can come up with a better framework that is secure by design. There is a need to analyze the security attacks in IoT so as to come up with a secure and reliable architecture to enable the services to interact with each other in a more secure manner. The heterogeneity in things is another problem and interoperability between these devices or applications should be made smooth enough to achieve good performance. So our solution is to come up with a framework for Internet of Things that is secure and sound. Implementing the IoT services as Microservices is another task to achieve. The security achieved in our framework is then evaluated to verify its performance and see that our goal has been achieved.

1.4. Thesis Contributions

In this thesis, we introduce a framework for securing services in Internet of Things. The security issues associated with the Internet of Things and the existing solutions are presented along with an overview of our solution. In this research work, we built a prototype implementation of our solution as a proof of concept and the challenges that we came across in our work are also presented in detail. The system is evaluated to find out the overhead of using claims and the results obtained are presented along with the performance level.

The main contributions of this research are:

- **A secure IoT middleware:** A framework for IoT is proposed in this thesis and a sound solution to the security threat associated with the Internet of Things is established based on the Claims- identity [6].
- **Prototype implementation**: The prototype of the claims-aware framework is implemented using Windows Communication Foundation (WCF) [7]. It provides security tokens that are used to provide the claims identity to the network. Thus the IoT devices are authenticated by the services before actually performing the services.
- **Implementation of IoT services as Microservices**: In this system of large number of connected components, decentralization is very important to ensure reliable services. Such a way of programming architecture is Microservices architecture that totally eliminates the need of centralized management of system. In short, these are independent services that communicate using REST (Representational State Transfer) APIs.
- **Cloud-based evaluation of the prototype:** To test the feasibility of the framework, we have evaluated the prototype implementation in the cloud.

1.5. Thesis Outline

The rest of the thesis is structured as follows. Chapter 2 provides an overview of the related works about the existing middleware for IoT and its challenges. In Chapter 3, detailed description about our proposed solution is given which explains the claims-aware security model and Microservices. The implementation details are provided in Chapter 4 in which the comparisons of tools are done. Evaluation results of the framework are presented in Chapter 5 together with the obtained results. A brief explanation on the case study of smart home is given in Chapter 6. Chapter 7 concludes the thesis with an emphasis on the future works.

Chapter 2

Background and Related Work

This chapter presents an overview of the related topics and relevant works done in this research area which include, middleware for Internet of Things, challenges related to IoT, security threats and existing solutions and microservices. The background work helps in gaining basic knowledge about the IoT middleware and its functionalities.

2.1. Middleware

Middleware is a software layer that sits between the operating system and the applications, to simplify the integration of heterogeneous applications. The different types of middleware are [4]:

- *Message oriented middleware*: This type of middleware enables the disparate processes to communicate using message passing. This is a simple and easy way of achieving less complicated distributed applications.
- *Object oriented middleware*: Unlike the message passing in the above type, this middleware infrastructure supports the communication between applications in the form of objects and follows an object oriented architecture.
- *Remote Procedure Call (RPC) middleware*: As the name indicates, it calls the remote procedures between systems for interaction.
- *Database middleware:* There are different types of software or drivers that let us connect to various databases. This kind of middleware provides connectivity and accessibility to databases.

- *Transaction middleware:* This is mainly for web-application servers dealing with the transactional procedures.
- *Portals*: The enterprise portal servers are regarded as middleware because it serves as a piece of software that enables the integration between the front-end user interface and the back-end server.
- *Embedded middleware*: It stands as a medium of integration between the real-time OS and the user application of the embedded systems or it can be integrated along with the operating system.
- *Enterprise service bus* [8]: It is a software architecture model used to achieve communication between services that follow Service Oriented Architecture (SOA).

2.1.1. Middleware for Internet of Things

IoT middleware is a software layer that holds all the heterogeneous devices, applications and users together. Middleware provides the platform for various devices with different protocols to communicate with ease and provides all the functions intended for a particular task. There are various services that have to be provided by IoT middleware. The simplified diagram of general IoT middleware is shown in Figure 2.1 and a detailed version is presented in [9]. These are the basic services that have to be delivered by the IoT middleware and each of them is explained in detail.

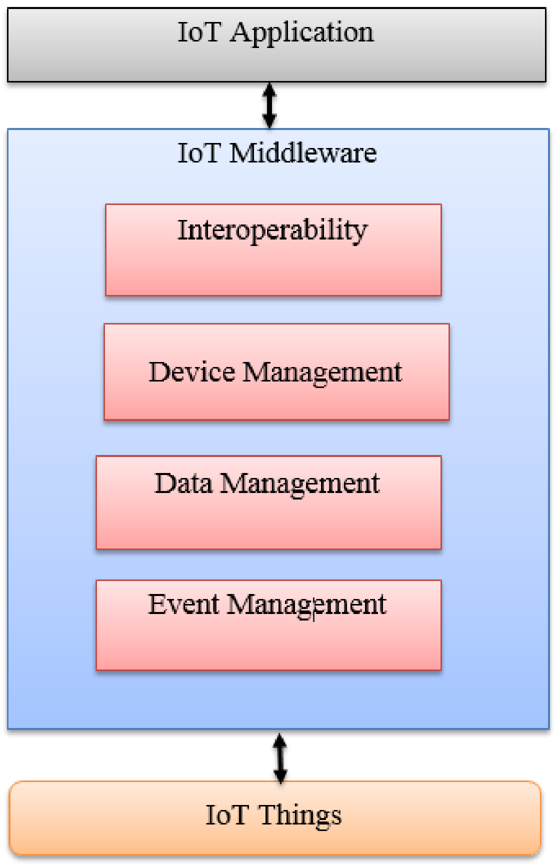


Figure 2.1: Architecture of IoT middleware (adapted from [9])

The major functional components of IoT are:

- **Interoperability:** Interoperations are performed by API gateways. The main functional operation done by API Gateway is protocol translation. There are mainly three kinds of interoperations like, network, semantics and syntactic [9]. Embedded control gateways provide intelligence along with *interoperability*. Transferring the tasks to gateways from nodes will reduce the complexity and cost of end nodes [10]. It converts the different protocols into a common one and enables the exchange of information among different networks. Things could be any kind of devices or applications with different protocols like Bluetooth, ZigBee, Wi-Fi etc. For example, in home automation system, things could be smart TV, light sensors, temperature sensors, security cameras, smart phone and so on. The user could either control it from smart phone or the sensors detect context and perform the services. Thus it is the role of middleware to perform the protocol translation in to a single protocol for further processing.

Multiple devices can be connected to a single gateway depending upon the load of the network and capacity of the Gateway device. Thus there could be more than one Gateway for large network with large number of nodes. The two of the most commonly used protocols in Internet of Things are REST and MQTT (Message Queuing Telemetry Transport). While Web Sockets encapsulate the identity between the client and the server, REST comes as a powerful way of providing Unique Resource Identifier (URI) for each resource. REST style consists of any HTTP methods along with a particular resource for manipulating that resource. For example, a GET request at /resource will provide the data stored in the resource. Message Queuing Telemetry Transport (MQTT) is a lightweight binary protocol [11] which is power efficient. It works on the principle of publish/subscribe (pub/sub) in which each client publishes the updates or subscribe for updates from others regarding a particular topic with a central broker. The subscribers connect using a CONNECT message and the publishers needs to acknowledge it. The clients can either use PUBLISH or SUBSCRIBE message as per the situation. It differs from REST as MQTT is a stateless protocol and also it does not have any discovery mechanism [12].

- **Device Abstraction:** IoT middleware should enable any device to communicate to the neighboring devices without any hindrance. Not only should they be discovered automatically, but also they should be registered to the network with no trouble. The other devices should be notified about the new device being added. It is not always essential that each device should communicate to every other device. Access control may be exercised to restrict the invalid interference of applications. Thus we can maintain a systematic order or hierarchy of communication within the network.

- **Big Data Control:** The fact that the network consists of not just one but many (hundreds or thousands or even trillions) devices points to need of managing enormous volume of data being produced every second. Various challenges that have to be addressed by the middleware in this component are: querying, processing and modelling [9]. Thus it is essential to control the huge amount of data being produced, exchanged between devices or application and being stored.

- **Context Detection:** This functional component collects the data being generated by the IoT devices and analyzes the data to derive a meaningful situation from it. It is the responsibility of the middleware to make a decision based on the context detection and analysis and produce the result. Data mining techniques may be required to get the valid information out of the raw data.

The high level architecture is shown in Figure 2.2. In an embedded system, the middleware either sits on the device drivers or on top of the operating system. In general, the middleware acts as an abstraction layer that mediates different application software that intend to provide scalability, security, flexibility and intercommunication between applications [24].

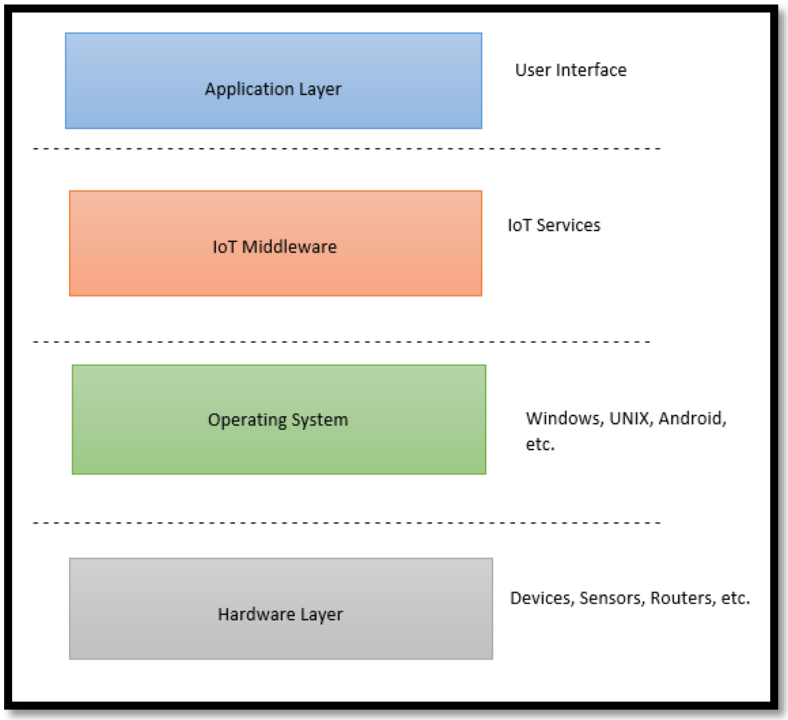


Figure 2.2: High level architecture of IoT

2.2 Related Work

Various kinds of middleware with different frameworks have been developed so far and noticeable work has been done in the field of Internet of Things. An overall view of the significance of middleware in IoT is given in [9] and an analysis of existing IoT-middleware is made. The authors point out the various reasons for the need for a middleware such as to enforce a common standard between heterogeneous devices and the ability to provide APIs. The various functional components and basic services that have to be offered by an IoT middleware are detailed in brief. Proper context detection and data management are considered as an open issue in IoT.

Some of the existing middleware include *HYDRA* [13] which was the project to develop middleware for networked embedded systems. It is based on Service Oriented Architecture in which devices are loosely coupled for scalability and supports all the functionalities mentioned in [9]. It follows the semantics way of designing the structure and the security is taken care of in the application layer.

The authors of [14] have achieved the feature of self-management of devices for interoperability in the middleware named *UBIWARE* with the help of Semantic technologies and Agents are used to manage the complex system. Every agent in this system is self-aware and self-manageable entity which allows extension at operation time.

Though [14] does not include any security features, the authors of [15] have come up with a secure framework for embedded peer-to-peer network which provides security at group level using challenge-answer protocol and at device level using cryptographic symmetric encryption techniques. The authors of [15] enabled the IoT paradigm to the e-health with the help of *VIRTUS* middleware. Rather than taking the normal approach of SOA, the authors employed publish-subscribe paradigm using dynamic XMPP accounts. Security is achieved using SASL (Simple Authentication and Security Layer) and TLS

(Transport Layer Security) [15]. Secure event- driven communications are obtained with the help of the cryptographical functions mentioned above.

The authors of [16] have done a great job in analyzing the research areas associated with IoT and they strongly believed that IoT will change the face of the world in the coming future. Various challenges include *Big Data management, Openness* of the system and thus the *Security and Privacy* threats associated with it, *Robustness* and the role of *Humans in the loop*.

While a detailed description of each case is explained by the authors of [16], the authors of [17] classified the challenges into IoT specific and generic. They mainly focus on *Interoperability* and *People Centric Design* which also takes into account the Security threats. The authors have provided a brief summary of how M2M solutions have migrated to open IoT solutions.

The authors of [18] examined the effect of associating IoT with the social networks to improve the human-to-device [18] interactions. They also did some research to find out the challenges in *Social driven IoT (SIoT).* Two of such challenges were social aware services and location aware services in IoT. The scalability factor associated with the IoT is explained well by the author of paper [19] in which IoT is depicted as a tree-structure and as the branches grow, so does the devices and services.

There are various commercial platforms available today in the market that are built for Internet of Things. One of those industrial platforms is *Blackberry IoT Platform* [20] which is a flexible cloud based framework allowing users to connect devices and applications. The design principles were documented in a systematic manner. Another good platform is the *Amrita* [21] middleware for IoT in which the key feature is the secured REST APIs to create user dashboards. It supports protocols such as ZigBee, Wifi, Bluetooth, UPnP, IPv6 and 6loWPAN. Another remarkable platform is *MuleSoft* [22] with the support of *Anypoint Platform* [22]. It provides a complete integration platform for connecting any kind of applications, data source and APIs with the edge devices at user end. API designing and management is the main highlight of this platform.

The existing solutions mostly present frameworks to support the semantics of the IoT middleware and only a few have considered the security of the middleware for Internet of Things. One of the architectures that support security incorporate SASL abstraction layer to achieve authentication and channel encryption [15]. There are also solutions put forward to have the security features to be added in the application layer [13]. Also, many of them achieve security by opting secure communication protocol like SSL/TLS [15]. The fundamental problem that we are trying to solve is to prevent the intrusion of a third party into the network and gain access to the IoT services along with ensuring the privacy of data. As a solution, we introduce a Claims-aware middleware for securing IoT services in which trust between servers are protected using Claims. Thus it identifies the server trying to access the IoT services by verifying the Claims. Also, the existing solutions followed either Service Oriented Architecture [13] or Publish-

Subscribe [15] model in their framework and in our solution we followed a new architecture of modelling services, that is, Microservices architecture.

2.3 Summary

This chapter presented an overview of different types of Middleware, especially, Middleware for Internet of Things and discussed about the components present in it. A detailed study on the related works is also presented along with its limitations. In next chapter, we present our proposed framework to address the limitations of the existing framework.

Chapter 3

Proposed Solution

In this chapter, we present our proposed framework and discuss about the main components present in our middleware. This chapter provides a brief overview about the middleware for Internet of Things and also a detailed description about our proposed framework. The functional components present in our framework are explained briefly. The key components of thesis such as Claims-identity as well as Microservices architecture have been presented here.

3.1. Overview

Middleware is now a part of every embedded device. It bridges the gap between the low-level entities and the high level applications. The things constituting the IoT differ in their nature and working environments. All the devices operate upon different set of sensors and produce a wide variety of outputs. It is the middleware layer that stands as a platform for all the disparate devices to communicate in the most reliable manner.

IoT middleware is the glue that holds together the operating system, the hardware devices, the software application as well as the back-end databases. In fact, it could be said that the whole software part which is in between the edge-level entities and the user level application forms the middleware. A fully functional system of middleware should have a runtime environment [23] and supports multiple applications and handles various services offered in Internet of Things.

3.2. Middleware Architecture

The IoT framework is not just between machines but it includes humans too. The roles of humans in the loop either could be to control the system or to get monitored by the system so as to take necessary control measures. This part of human interaction makes IoT different than M2M (Machine-to-Machine) communication.

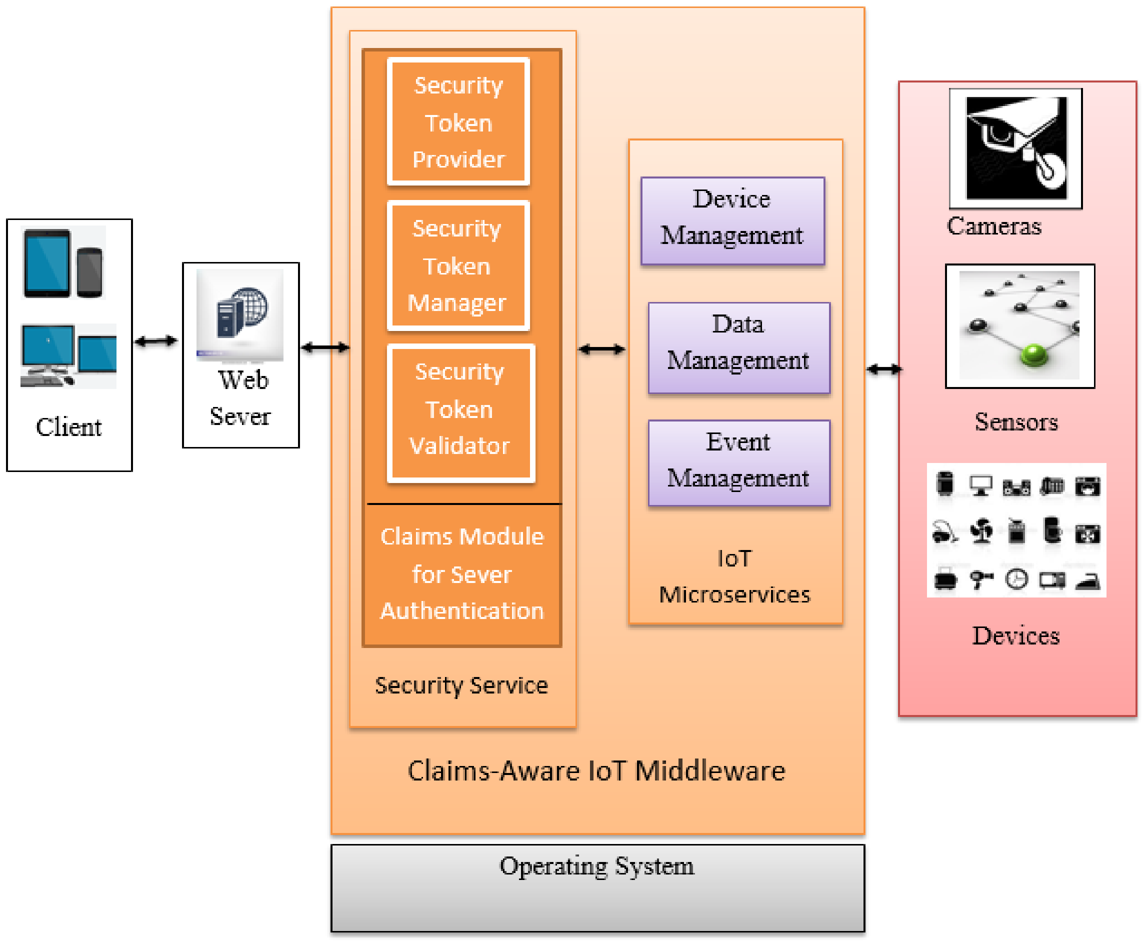


Figure 3.1: Proposed middleware architecture for securing IoT services

In general, IoT middleware is a service that lies between the applications and the platforms [25]. The middleware is usually a general purpose service that in other words can be called as platform service. In our framework as shown in Figure 3.1, the IoT middleware sits on top of the operating system. The claims-aware IoT middleware has four main services. The security service is the module responsible for the authentication of the servers. The other three IoT services such as device management, data management and event management services are implemented as microservices. The call to the IoT services are allowed based on the authenticity of the server generating the request. Thus every call to the IoT services goes through the security service which actually checks the claims associated with the intermediary server.

3.3. Middleware Components

The functional services offered by our solution are:

- **Device Management:** This service layer includes two main categories namely, Device Discovery and Registration processes. Self- discovery of services is one of the main highlights of IoT middleware. Self-management is an essential feature that has to be present in a fully functional middleware. Self-configuration can be maintained by modelling the framework as generic without having to be specific for a particular home or region [26] in which relationships between objects are defined using properties and follows hierarchy. Device registration would add the devices to the network and provides all the details of the devices present in the network. This service handles the CURD (create, update, read, delete) function of all the devices within the network. This layer provides the details of all the devices connected within the network to the user interacting through the application.

- **Data Management:** The enormous volume of real time data that is being produced from the devices that might be used in different ways in different applications calls for an efficient way of managing this Big Data. Cloud is an intelligent choice for storing the huge volume of data. It has changed the landscape of storage, communication infrastructure, computing and other services. The choice of database is crucial and should be done in such a way so as to provide the maximum efficiency and utility with minimum speed of querying. Also, to eliminate the huge chunk of raw data being processed and stored every moment, a standard approach would be to pre-process the data and store only those which are necessary. API gateways could be useful in such intelligent decision making techniques.

- **Event Management:** This service layer performs the context detection and analyzes during the occurrence of an event. The situation of an entity has to be processed in which the entity can be devices, applications or even users. The fundamental blocks associated with this service are Context Detection and Context Processing [9]. It is the function of the IoT system to respond to a particular situation with best possible intelligent solution to that event. This task is achieved by collecting data from the sensors or devices and performing analyzes techniques to produce meaningful information out of the event which results in sensible response. The context model may need to detect the context, process the context and break it down in to meaningful data [27]. Such information has to be stored in Databases for processing needs.

- **Security Service:** One of the fundamental problems that has always been present in the internet is security attacks [16]. Security and privacy of the things and data ensures confidentiality, availability and reliability of the whole system. Security can be achieved in two ways; a) securing topology management by proper authentication and authorization of devices and user; b) securing the communication within and out of the network, by the implementing secure transfer protocols there by ensuring the protection of sensitive data [9]. In situations in which an adversary is trying to take control of the device or tries to manipulate the data, the system must come up with a quick response for Self-healing in the real time environment. Such responses have to be transmitted securely to the appropriate nodes and the measures may include encryption, authentication and authorization which could be computationally complex in nature.

Though the cloud enables resource pooling and cost-effective management, outsourcing the data or product to a third party can cause security and privacy issues. The fact that there is a virtual entity representing the actual real world entities in IoT such as services, user, devices etc., also add to the threat to the security of the IoT network [28].The security in our framework ensures that no service or user without proper authentication and authorization can be added in the network. Once the things are properly identified and registered, every request for a particular service must produce the claims assigned to it and only valid request from a valid user will be sanctioned and others will be denied of the services. Thus no adversary can spoof in to the network and play with the IoT services or impersonate a valid user. This level of protection is possible with the help of claims for identifying each and every entity in the IoT network.

3.4. Claims-Aware Identity for Securing IoT Services

A claim is a statement which basically tells what the object is or is not. The trust between the disparate entities in IoT can be established by enabling the trusted exchange of arbitrary claims with random values [6]. The claims contain the identity about the entities which could be either user or device or application that are being connected together. The main advantage of claims aware objects is that it simplifies the authentication where multiple signing processes are avoided.

In our framework, we use the SAML [29] tokens as claims. Security Assertion Markup Language (SAML) is an XML based, open standard data format for exchanging authentication and authorization data between entities. It contains mainly three components [30]:

- **Assertion**: This component reveals the identity of the entity and holds the important information about the user. It is based on these assertions that the system decides what the entity is authorized to do.
- **Protocol:** It defines how the assertions are being passed. The different protocols supported by SAML are Hypertext Transfer Protocol (HTTP), Simple Mail Transfer Protocol (SMTP), and File Transfer Protocol (FTP) etc.
- **Binding:** It defines how the SAML (request-response) message exchanges are mapped into Simple Object Access Protocol (SOAP) exchanges.

The main components [31] in the claims middleware in our architecture are as follows:

- **SAML Token Provider**: This component implements a custom SAML token provider which actually creates and returns a custom claim as per the SAML assertion given. A sample claim is provided in Appendix A.6.
- **SAML Token Manager:** This component returns the token provider if the client token requirements matches the SAML token.
- **SAML Token Validator**: This component checks and validates the claims provided by the intermediary server.

A sequence diagram of the interactions between an intermediary server and our middleware is shown in Figure 3.2:

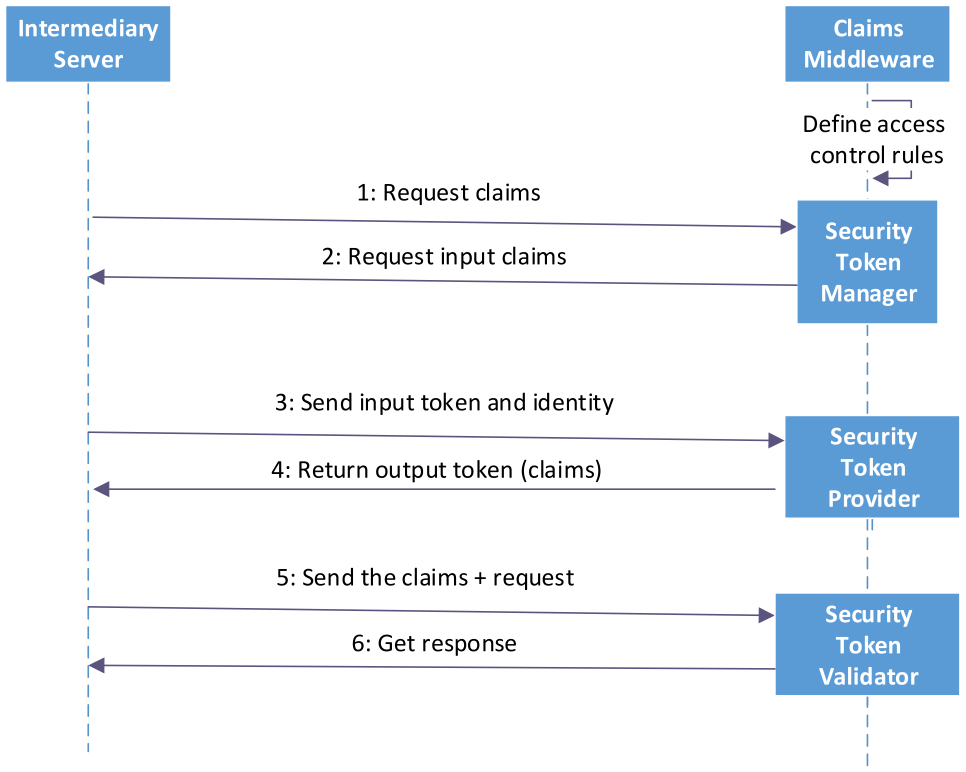


Figure 3.2: Generation and processing of claims

A complete scenario of how claims are issued by the middleware is demonstrated in Figure 3.3. The main steps involved can be pointed out as:

1- Client communicates with the web server

2- Web server then requests for the claims to the middleware

3- The web server is then authenticated by the middleware

4- Once authenticated, it can pass the message to the *things*

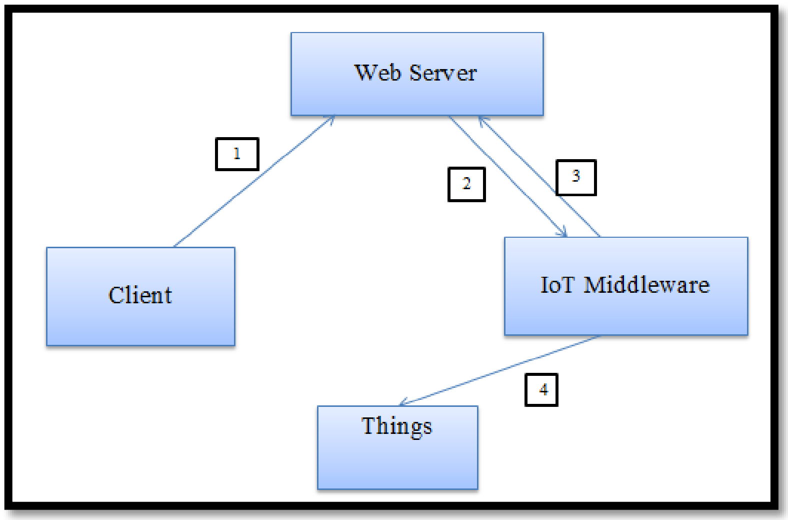


Figure 3.3: Claims issuing scenario

Our framework authenticates the web servers to prevent any clients from directly using the middleware to access the device end. The security token provider issues the appropriate credentials to the web server so that they can transmit the message securely.

The most prevailing problem associated with the security token service is the single point of failure. In order to make the system more efficient, the system could have distributed servers as issuers. The claims or Tokens issued by one is agreed by others and all mutual trust is established between them. Thus a complete decentralized system improves the Quality of Service (QoS) of the entire system.

The QoS for each object can be achieved with minimal possible overhead even if the needs of each object would be different, for example, one entity may need confidentiality and the other may not really need secrecy as the data is public. So our approach enables the entity to register their security policies with the issuer at the time of registration and it can also reveal the behavior of the object, i.e., what type of objects can use its data, from which type of devices can it accept the data, under what conditions. A more concrete discussion on these security policies is presented in the implementation chapter.

3.5. Microservices Architecture

Microservices Architecture has been a buzzword around recent years in the world of software architecture. The main intention of such an approach is to eliminate the need of a centralized management of applications. Microservices architectural style [32] is a method of developing a software application as a suite of small independent services running in its own process. These services can be deployed independently and they communicate with others using HTTP resource API like lightweight mechanism [32].

The advantages of this approach rather than traditional monolithic means of development in which the whole application runs in a single process are [33, 34]:

- Easier to develop and understand as each microservice is comparatively small.
- Easier to deploy new versions as each service is independently deployed.

- Easier to debug errors and faults are more isolated within each service

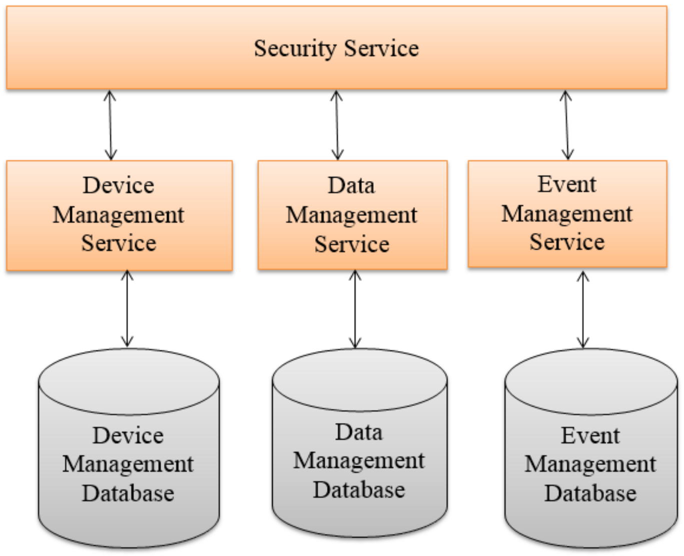


Figure 3.4: Microservices architecture based IoT services

Each service offered by IoT middleware is implemented as microservices as shown in Figure 3.4, which is fully independent of each other. Decentralization [32] is the main feature associated with such architecture and decentralized governance divides the burden of managing applications and services into multiple streams. The decentralization of data management too demands the need of persistent database storage. This was one of the reasons why the enterprises used monolithic approach with single database for a particular application. In microservices oriented approach, we could either use completely different database systems or different instances of a single database system [32]. Maintaining consistency between the data storages is the key task of decentralized databases and we selected the database that supports maximum consistency.

3.6. Summary

In this chapter, we presented the proposed solution to address one of the main challenges in the area of Internet of Things, Security. A detailed description of each components present in the middleware is given. We also discussed claims and how they are issued and used in our system. The merits of implementing the architecture as microservices are also discussed in this chapter.

The implementation details of our prototype are explained in the following chapter. The work is explained in the order of our different implementation phases. We also will see the challenges that we encountered in the implementation stage of our prototype.

Chapter 4

Prototype Implementation

In this chapter we discuss the implementation details of our prototype for securing the IoT services. The implementation of IoT services as microservices has been explained along with a brief discussion on the tools and the database used. The details of the implementation of the security service based on claims authentication are also presented in this chapter.

4.1. Implementation of Microservices

Implementing the IoT services as Microservices really helped us especially in the phases of updating of code or error debugging and it outweighed the extra efforts in the initial setup of different independent projects. Our initial research on tools led us to use Java as the programing language. Microservices are single responsibility services which can be easily understood as it does only one thing in each service [35].

There were a number of tools in Java supporting microservices architecture. Spring Boot from Spring Tool Suite and RestExpress [36, 37] are open source resources for building microservices based REST APIs. Both are very powerful frameworks for Java and provide stand-alone REST APIs. Though both provides Maven archetype, Spring Boot allows us to choose the parts of the framework that are only needed rather than adding the whole run time dependencies. Additionally, Spring Boot comes with an embedded Tomcat container which could be bootstrapped easily. This prevents the programmer from configuring the server explicitly. From all the above mentioned facts, we chose Spring Boot as the framework for Microservices based RESTful APIs. The decomposition of monolithic components into individual unit of independent components is done efficiently by Spring Boot. Thus testing of the services is made much easier by streaming the documentation in to small deployable components. In other words, each of these services holds a single responsibility and to test a particular service, we only need to check that service module rather than going through the entire project which has all the services in it. Another feature of Spring Boot that highly supports the microservices is incorporation of all the necessary resources to build it into a readily runnable deployable. It enables the use of Maven and Gradle for dependency management. But one thing to point out is that in the POM file of each project we have to add the same dependencies for each of the services and they are assigned with different port number to run locally in the machine.

4.2. Database Selection

SQL databases [38] are very persistent and consistent as compared to NoSQL databases. SQL databases are Relational Databases (RDMS) which is table based storage while NoSQL databases are mainly distributed ones without any particular schema which are based on documents, key-value pairs, etc. The best choice for the distributed nature of our heterogeneous system is NoSQL databases.

There are several NoSQL databases available in which MongoDB and CouchDB [39] are the most popular and easy to use data storages. The time taken to query is much less in MongoDB as compared to the MySQL database [40]. MongoDB is mainly concerned with consistency of data while availability is mostly favored by CouchDB. The former follows *Strict Consistency* while the latter supports *Eventual Consistency.* In IoT applications, the consistency is a non-trivial need that has to be met and for our middleware, MongoDB will place the updates better as compared to CouchDB for constantly changing real time data and so we use it for our implementation. MongoDB is document based database which is highly scalable and highly available. It supports JSON (JavaScript Object Notation) document format and also has lots of features that are available in traditional RDMS [41]. The high speed access to mass data using MongoDB makes it highly preferred NoSQL database for many applications [42]. The scaling is achieved by adding more machines or servers.

4.3. Implementation of Claims Identity

The implementation of claims-based identity resolution is very well supported in .NET compared to Java with the help of windows identity foundation or ADFS. Microsoft provides claims-aware applications [6] in which the claims associated with a security token is validated before allowing access to various web applications. In our framework, the request to a particular service is allowed or denied as per the validity of the claim provided by the server.

The username and password once verified, the client communicates with the web server. Once the client is authenticated on the web server, the server requests the middleware to provide it with claims. The token provider after examining the target infrastructure, issues the appropriate claims to the server. The web server then uses the claims to pass the message securely to the device end so, our system ensures that the server is authorized to communicate with other servers in the system. The access control policies limit the interference of a service as per the defined rules. Thus the claims-aware authentication and authorization ensure that the whole system is secure because a third party service or an invalid user will have to authorize to the server first before accessing the middleware. The communication channel is secured using TLS/SSL thus making the system more safe and sound.

4.3.1. SAML Token Provider

In our prototype we used the SAML Token provider supported by Windows Communication Foundation (WCF). In Security Assertion Markup Language (SAML), Claims for authorization and authentication is in XML format. The basic entities identified by SAML consist of a) user or principal entity, b) the Identity Provider (IdP) and, c) the service provider (SP) [43]. Briefly, how the scenario works is that, the principal requests for a service from the service provider and it then requests the Claims or assertion of that principal from the identity provider. If the Claims are valid, then the service provider responds to the user entity request as per the access control decision specified within the Claims.

In our framework, the claims middleware implements a custom SAML token provider and it merges the two components such as identity provider and the service provider into one. Thus our middleware itself issues claims and validates them.

The WCF SAML tokens have cryptographic operations performed on them so that the identity of the issuer and the integrity of the client entity are verified at the server level. The procedures can be briefly explained as the following steps [31]:

1. Client requests the SAML claims from the Security Token Service (STS) using a valid username and password (Windows credentials).
2. The token provider then issues the claim and encrypts the certificate with a proof key.
3. A copy of the poof key is received by the Client and the token along with the message encrypted using the proof key is sent to the service provider.
4. The server can validate the issuer using the signature on the SAML tokens and the principal entity as message is signed using the same key.

In our framework, we used custom token provider to create our own token and then transformed it to a WCF supported client format. The above mentioned procedures are implemented using our custom token provider which include configuration of client using the provider, passing the claims over to the client as well as the WCF client framework and finally the authentication of server using its X.509 certificate [44]. In our service configuration, we have two endpoints for communication. Each endpoint lets the client present the claims and authenticate using symmetric and asymmetric proof keys. An endpoint is what the service exposes and is the portal for the process or service to communicate to outside. The WCF endpoint is composed of an Address that specifies the address (URI) where it can be located, a Binding that specifies how the client can pass a message to the server using this endpoint including protocol (HTTP), message security (SSL) and the data format.

4.3.2 SAML SOAP Message

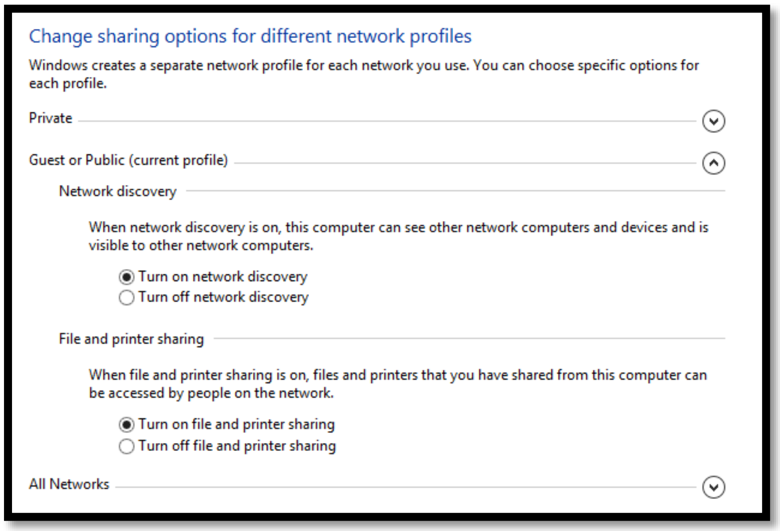
In SAML security token [45], the assertions that defines the identity and access rights are signed using an X.509 certificate. This signature acts as an evelop signature that binds the assertions to the issuing SAML authority. In our implementation the X.509 certificate used for this purpose is named “localhost”. This certificate is used to verify the signature and then decides if the assertions are from a trusted SAML authority. Next, the entire SOAP message is signed using the private key and the signature has a KeyInfo element. This element tells the receiver of this message to decrypt it using the public key mentined in the assertion. This certificate used in our implementation is named “Alice”. This signature binds the body of the message to its subject (entity sending the token). Thus the integrity of the message and the sender’s authenticity is verified.

The WCF security policies [46] can be used to secure a web service. The WSHttpBinding are used in our implementation to secure the services. The behaviors also provide information about the service. Theses are defined in the Web. Config file of the service (Appendix A.1). A sample SAML token used in our implementation as claims, is shown in Appendix A.6.

To sum up, we have the Claims middleware that lies between the IoT services and the intermediary server and the calls from the client are passed to the microservices only if the server is verified to be a trusted party by presenting valid Claims.

4.4. Implementation of Device Management

The framework was completed by adding a device to the network. Because of the fact that every device will be assigned with an IP address over the internet, we tried to get the IP address of the smart device using the router so as to assign it a static one for communication. After hours of trial, we came to know that it does not use TCP (Transmission Control Protocol) but UPnP (Universal Plug and Play) networking protocol. It enables the networked devices to discover each other and communicate over the network [47]. But the device was not being detected in the network. The popular solution was to *Turn on Network Discovery* in the Network and Sharing option in the control panel as in Figure 4.2.


Change sharing options for different network profiles

Windows creates a separate network profile for each network you use. You can choose specific options for each profile.

Private

Guest or Public (current profile)

Network discovery

When network discovery is on, this computer can see other network computers and devices and is visible to other network computers.

● Turn on network discovery
○ Turn off network discovery

File and printer sharing

When file and printer sharing is on, files and printers that you have shared from this computer can be accessed by people on the network.

● Turn on file and printer sharing
○ Turn off file and printer sharing

All Networks

Figure 4.2: Screenshot of network discovery setup

But even after that, there was no such device visible and took some time to find the solution. The smart device named WeMo was visible under the *View network computers and devices* options in Windows operating system.

Figure 4.3 shows all the computers and the devices that are visible and our smart device WeMo is detected under other devices option in that window.

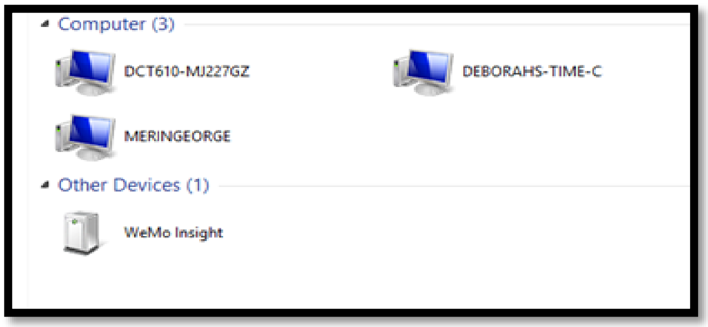


Figure 4.3: Screenshot of device being detected in our system

Finally the device was visible and the IP address and the port number are obtained from its webpage as shown in Figure 4.4. Thus using this IP address and the port number, which is basically the end points for the smart device, we could now communicate to the device programmatically.

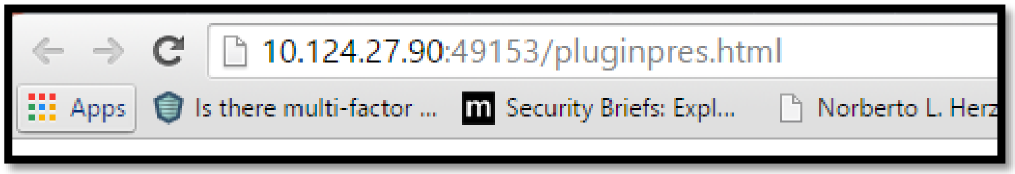


Figure 4.4: Screenshot of device webpage

4.5. Interoperability between Platforms

The fact that the IoT middleware should be independent of the platform used, we wanted to do the interoperability between at least two programming platforms. So, the security part of the middleware is implemented using .NET and the rest of the IoT microservices in Java. To ensure that security is maintained in Java services, we used the idea of one time password (OTP). It serves as a token of security to the services which are finally verified at the server. The background research works on this topic showed us a method to do the object passing; Remote Procedure Calls (RPC) between .NET and Java using TCP [19]. This suits our case where our client application is in Java and our service application is in .NET. Another method that was much more efficient and simple was to use the REST communication in which data is passed in the form of JSON objects. The usage of MongoDB database which stores the document data in JSON made our work much easier.

4.6. Summary

This chapter presents the implementation details of our prototype and the order in which we have done our implementation explaining each step. It includes the initial trial with Raspberry Pi and then implementing the Microservices. A detailed description of how we achieved the implementation of Claims-aware middleware is also presented. In the next chapter, we will see the performance evaluation done on our system along with the results obtained out of the experiment.

Chapter 5

Evaluation and Results

In this chapter, we evaluate our system based on certain criteria that will be discussed in one of the following sections and present the produced result of the performance of our framework. The experimental setup for each evaluation scenario is explained along with the reflection on how the results support our claims-aware middleware.

5.1. Background

As we have explained in Chapter 4, implementation of the prototype, we currently have the provision to complete the prototype using a smart device. The case study using one smart device is presented in Chapter 6. The performance of our framework cannot be evaluated with just one device. In real world applications, there would be many devices connected to the network and thus we need to do the evaluation with more devices and services. We decided to simulate multiple devices that use our IoT services and thus validate the Claims on every service call.

The current setup of our framework makes sure that the IoT device authenticates to the local network using the wireless authentication standards. Once the device is registered to the network, we can manually register the device to our middleware. It also requires authentication to take place between the intermediary server and the middleware before the client request is forwarded to the IoT device by the middleware.

5.2. Simulation Scenario

To simulate multiple smart devices, we created multiple IoT endpoints. The endpoints are the URLs that we would connect to communicate with the smart device via the device's API. We can track the calls being made to these endpoints, which represent smart devices, to track the performance of our framework. This basically shows the number of calls that are sent to the endpoints. The fact that each smart device in the IoT network has an IP address makes it easier to track the API calls made within a specific period. Therefore in our framework, we enable the clients to make calls using the web client, we need to track the time taken for each call that is made within a specified period from the web client end. This gives the time based performance evaluation of our framework.

5.3. Evaluation Criteria

The criteria upon which the framework would be tested are as follows:

- **Time based performance evaluation of IoT Services with and without Claims**

  The IoT services should provide good performance to large number of users. Thus we measure the time taken to deliver the services by simulating sets of different number of devices. First we will do the evaluation of the services without adding the security factor, that is, the claims and we will note the time taken to finish the tasks. Next, we will find out the overhead of our claims-based security model by measuring the time taken to process the service requests after enabling the claims.

- **Performance evaluation of the Security Model**

  To evaluate our security model of the middleware, we will make calls to the middleware with and without claims enabled at the client. The calls that are made without claims should be denied of the IoT services. The calls that are made from the claims enabled client should have access to the IoT services.

5.4. Experimental Methodology

The applications of Internet of Things can vary from simple smart home to smart grid. The number of devices and services in each domain may be different in each scenario. Thus it is necessary to test the prototype in scenario which only requires LAN and the other scenario in which they are accessed or controlled remotely over the internet. In our evaluation method, we have two sets of scenarios in which the claims middleware is tested: 1**) LAN** network in which everything is running in local machine; 2**) Cloud** network in which the client and the intermediary server are deployed. We tested the performance of our framework in both cases and found out the overhead of using the claims.

To log the time taken to perform the entire task, we created two log files to track the time taken to perform end to end service. We have basically two stages of calls which are as follows:

- *Web service* – The service that comes in the client part where the call from the web client is forwarded to the middleware for authentication. The time taken to make calls right from the web end side until it reached the middleware part is present in this log file.

- *Claims service* – This is the part of the framework where the authentication using claims are being done. In this log file we get the timestamp of each iteration which is then used to find out the time taken in the middleware part to place the calls to the IoT services. This would give us the actual overhead of using claims for authenticating the intermediary web servers.

5.5. Experimental Setup

In this section, we present the details of experimental setup used to do the performance evaluation of our framework by simulating multiple smart devices. Each components of this experiment is explained in brief.

5.5.1. IoT Device Simulation

This was the first step to setup since it is the main part of the process. To simulate the different endpoints of the IoT smart devices, we built a microservice that can generate different URLs to simulate the device API URL. Thus we can generate thousands of independent endpoints that can be accessed by the client.

5.5.2. Device Registration

The different IoT URLs that simulated IoT devices is then registered in the database as in the similar manner of manually registering the devices. The device is registered to the network with the help of the device management IoT service. The device details are stored in the database which is then retrieved at the service end.

5.5.3. Claims-Aware IoT Middleware

The middleware performs the functional services that are present in the framework. It will validate the caller (server) and if verified to be authenticated, routes the requests. If it is validated to be unauthenticated, then it is denied of the services and the requests are not processed.

5.5.4. Client Website

In the client side, we automated the code to make calls to each registered IoT endpoint. For example, if we have simulated 500 device endpoints using URL simulation and are registered to the particular client, we will make the calls to all the endpoints. The code is modified to make calls to different device endpoints.

5.5.5. Tracking Metrics

In order to come up with results, we needed to log each data at each node so that we can track all the activity so as to summarize the data and present the findings of our experiment. The two different log files used in our setup are web service and the claims service log files. The total time taken by a request to be processed by the middleware is calculated from these two log files.

5.5.6. JMeter Tool

Apache JMeter is a Java application [48] that is mainly designed to measure the performance of a system and load test the application. It is efficient software to generate multiple calls to a web application and produce the performance results of that http request. Thus we have used it to generate multiple calls for different sets of requests to our framework.

5.6. Evaluation in LAN

In this section, we present the evaluation done on our local system based on the criteria mentioned in Section 5.3. In this setup, we have every service and the clients being presented in the local area network (LAN). We also explain the experimental setup done for each criterion and also present the architectural diagrams. The results obtained from each evaluation are presented after the brief description of architecture.

5.6.1 Performance Evaluation of IoT Services in LAN

To measure the overhead of using the claims in the IoT framework we did the time based performance evaluation of our system. The overhead can be found out by measuring the time taken to deliver the calls or requests to the IoT services with and without the claims. The time difference is calculated and that time period gives the overhead of using claims identity. The feasibility of our claims-aware prototype is tested with a different number of requests.

5.6.1.1. Performance without Claims

In this experimental setup, we invoked the IoT services without the claims, that is, without enabling any security feature in it. The calls to the IoT services are made for each set of requests. We changed the setup of JMeter for each set of requests and recorded the time taken by each server to process the requests. Essentially, the total time to process the requests right from the client side till it is forwarded to the IoT services is measured.

5.6.1.2. Architecture of LAN Setup without Claims

The structure of the framework remains the same as in Figure 3.1 except that the security feature of claims is not activated and we have simulated devices rather than a real smart device. We varied the number of calls being made to the device end and measured the time taken to place the call from the web client to the device end. The total time taken to perform the action is calculated by adding the time delays in the two log files as mentioned in the methodology.

5.6.1.3. Experiment Results in LAN without Claims

We tested the performance of requests generated using the event management web client in LAN without claims. We used the JMeter to make different sets of calls to the event management service which are directed to the device endpoints. To generate 1 request, the number of threads (users) was set to one. To generate 10 calls we used five users/threads generating two requests in sequence. Similarly, for 50, 100, 500 and 1000 requests we have 25, 50, 250 and 500 users generating 2 requests respectively. Each request was directed to different device endpoints. For instance, for the request set of 100, we had 100 microservices simulation 100 device endpoints.

We tested the performance of the middleware for six different sets of user requests which are 1, 10, 50, 100, 500 and 1000. Each test case is tested for ten times and then we took the average value to best suit the more accurate value. The results obtained are presented in Table 5.1 for the request sets of 1, 10, 50, 100, 500 and 1000. The minimum and the maximum value obtained in the ten iterations are also presented.

Table 5.1 Performance result without claims in LAN

Without Claims in LAN				
No. of Requests	**Average Time per Request (seconds)**	**Minimum Value**	**Maximum Value**	**Variance**
1	0.11	0.01	0.33	0.01
10	0.33	0.24	0.48	0.01
50	0.22	0.13	0.31	0.003
100	0.19	0.16	0.20	0.0003
500	0.20	0.16	0.22	0.0006
1000	0.17	0.15	0.19	0.0002

A request in our setup is a roundtrip command to control the device. For instance, the request goes to the device end when generated from the client end, then the response is returned to the client. In Table 5.1, the average time per request means the time taken to process one request and it is found out by calculating the mean time of the total set. The average value is obtained by calculating the mean of each iteration. Each test case is tested for 10 times and the minimum and the maximum value obtained in that iteration is also given in the table.

In the test case of 1 request, the lowest value obtained in that iteration is 0.01 second whereas, the highest value is 0.33. In this setup, all the IoT services and the client were on the local machine. There server may get busy processing any pending tasks and may affect the response time. In all the six request sets, the time taken to process them are different for some of the iterations out of 10 iterations. The local machine performance and the tasks running on the system could also affect the processing time of each iterations.

The variance, which is the measurement of the spread between the data in a particular request set is also presented in Table 5.1. The variance is comparatively large for the small number of request such as 1 and 10. This implies that the data obtained in cases with small variance does not vary much in range for the 10 iterations.

There is a sudden increase in the average time to process a request in the test cases of 10, 50, 100, 500 and 1000 requests compared to the case of 1 request. The response time can be delayed as the number of requests increase. Also, the performance of the system in which the services are running, will also affect the performance. In essence, the fluctuation in the processing time is because of the increased load to be processed by the server.

5.6.1.4 Performance with Claims

In this experimental setup, we did the performance evaluation of the framework by activating the security feature of claims-based authentication in our middleware setup in LAN. This setup produces the results that reveal the overhead of adding claims identity. It was expected that there will be overhead due to the time taken for the cryptography computations like encryption, decryption etc. The certificates that are used to provide claims are installed in our local machine. How the communication flow goes is that the client makes a call from the client side which is forwarded to the middleware and verifies the authenticity of the intermediary server by validating the claims. If the server is verified to be trusted by presenting the SAML tokens to the security service, the calls are routed to the device end to process the request.

5.6.1.5. Architecture of LAN Setup with Claims

The components of the architecture are almost same as that of the proposed framework except that instead of actual devices, we have simulated devices. The end point URLs represent different device APIs that are called in the program from the client side. The procedure to find the total delays is the same as that of without claims. The difference in the setup is the presence of the Claims middleware which authenticates the web server that the clients use to call the IoT services.

5.6.1.6. Experimental Results in LAN with Claims

The performance for the claims middleware was conducted with the same set of requests as in the setup for without claims using the event management service. We did the test for 1, 10, 50, 100, 500 and 1000 requests and used JMeter to change the number of requests. The average time to process one request is shown in Table 5.2 for the different sets of requests.

Table 5.2 Performance results with claims in LAN

With Claims in LAN				
No. of Requests	**Average Time per Request (seconds)**	**Minimum Value**	**Maximum Value**	**Variance**
1	0.26	0.02	1.45	0.33
10	0.43	0.37	0.50	0.002
50	0.39	0.30	0.54	0.010
100	0.37	0.28	0.49	0.006
500	0.31	0.22	0.38	0.0002
1000	0.36	0.28	0.72	0.03

Each of the test case is repeated 10 times and the mean of the 10 values are taken as the average time per request for that particular test case. The minimum and the maximum value obtained in that iterations are also presented in Table 5.2. These measurements show the worst case and the best cases in each data sets. The spread of each data from the mean value (average) is represented by variance.

The average time taken per request is calculated and presented in Table 5.2 for the same test cases of 1, 10, 50, 100, 500 and 1000 requests. Evidently, there is an increase in the processing time for all the test cases of 10, 50, 100, 500 and 1000 requests. The increase in the time consumed is because of the increase in the load to be processed by the server due to the increased number of requests. Network latency limits the maximum speed with which the requests are transmitted.

5.6.2 Comparison Results in LAN

The time taken to process the different sets of requests without the claims and with the claims will give us the actual overhead of adding the security factor of claims in the system. The comparison is represented in the form of graph in Figure 5.1. The minimum and the maximum value obtained in the 10 iterations of each request set is also presented in the performance graph.

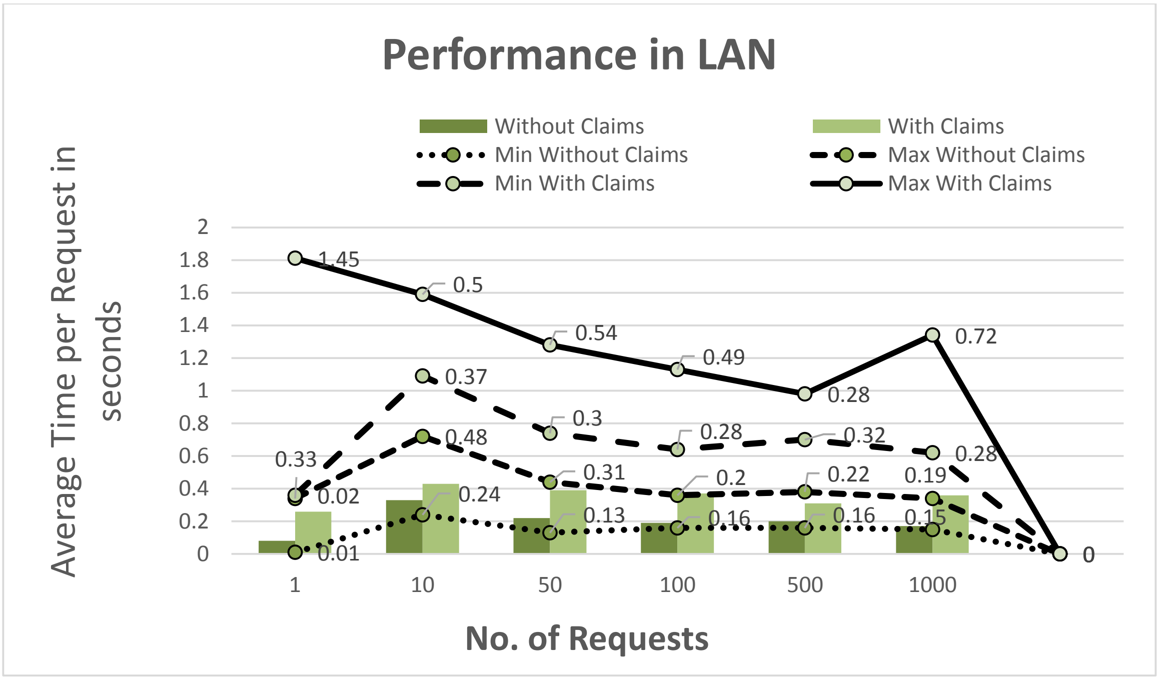


Figure 5.1: Graph showing the performance in LAN

From the above graph, the overhead of using claims to achieve secure communication from the client end to the device end is only few milli seconds. The time taken to process a number of 1000 requests is only 368 seconds which gives the average time taken to process single request as 0.36 seconds.

There is an increase in the processing time of request set of 10 as compared to the rest of the values. The IIS hosted WCF service uses managed I/O threads and as the time to execute a request increase as the load increases, it spans for new threads using Thread Pool. That is why the processing time for large set of requests are less since the calls are executed in parallel by the WCF framework. Thus the overall processing of the requests is done parallel by the WCF service.

Though the calls are made sequentially by the client, it cannot be confirmed that the average time per request will always be the correct time taken to process a single request. For example, in the test case of 1000 requests with claims, the total average time taken to process 1000 requests is 360 seconds which is 6 minutes. The assumption that if 1000 requests take 360 seconds, time per request will be the average value, 0.36 may not be the exact case in different situation.

- Worst case scenario: The worst case scenario could be the case in which the first request is processed after processing all other requests. In that case, the time per request would be the total time taken for the entire set of requests. For example, in the test of 1000 requests, the worst case would be the situation in which the first request is processed after the rest of 999 request. In that case, the worst case average time per request is 360 seconds (6 minutes).
- Best case scenario: The best case scenario would be the case that in which the first request has been processed first and the rest of the requests, as per the order they are called. This is the case that favors our assumption that if 1000 requests take

360 seconds, then a single request would take the average time of 0.36 seconds to be processed.

The processing time is a bit higher for all the sets except in which number of request is one. The processing time for requests is higher for all the request set except for the test case of 1 request. The performance of the server can be affected as the load increases and also if the system is processing some other tasks, the response time can be delayed too. The longer response time for 10 requests is because the WCF service use asynchronous thread pooling as the concurrent calls to the service is increased. Thus for a large set of requests like 50, 100, 500 and 1000, the execution time is less.

5.7. Evaluation in the Cloud

The Cloud provides an environment to share the resources that are provided by the service providers. The main features of the cloud are resource sharing, scalability, pay per use, faster access. They include infrastructures, software, storage locations, applications etc. The services provided by the cloud can be of three types mainly [49]:

- *Infrastructure resource*: It includes the services such as storage, computing power, and machine provisioning. For example, Amazon EC2 and Windows Azure provides web service interface to configure capacity online. Microsoft Skydrive is used for online storage and provides free storage services to users. In terms of computing power, grid computing uses clustering and parallel computing technologies

- *Software resources*: It includes middleware and development resources. The middleware resources consist of cloud centric operating systems, application servers and databases, whereas the development resources include design platforms, development, testing and deployment tools.

- *Application resources*: Information industry is moving application data to the Internet. For example: Google uses Cloud computing platform for web applications needed for communication and collaboration.

- *Business processes:* It is business driven application supporting reuse, composition and provisioning.

In a cloud computing environment, there are multi-domains in which each domain can have different security, privacy and requirements to run various mechanisms, interfaces and semantics [50]. Most start-up companies, small, medium and large enterprises are now interested in the cloud computing. Therefore these users should be highly interested in the cloud computing security. It is important to identify the current risks that exist within the cloud computing to formulate solutions. Some specific threats to security includes: Flops in Provider security, attacks by other customers, convenience and consistency issues and legal and regulatory issues. These days the trust between the user and the service provider is a matter of security that is concerned with the end user [51].

5.7.1. Performance Evaluation of IoT Services in the Cloud

In this experimental setup, we evaluated the performance of our framework by deploying the client and the intermediary server in to the cloud. This case holds the scenario in which the user or the person accessing the service is not the local network, but somewhere out in the internet. In a nutshell, we have the middleware and the IoT services in the LAN and the client and the intermediary web server deployed in the cloud.

5.7.1.1. Performance without Claims in the Cloud

The procedure of evaluation remains the same in which requests are sent using JMeter to the client hosted in the cloud and the log files give the time taken to process the requests. The claims verification part is disabled in this setup and the calls are directed straight towards the IoT services without going through the security process.

5.7.1.2. Architecture of the Cloud Setup without Claims

The architecture mainly consists of two parts, the LAN and the Cloud. We have the web client and the web server in the cloud and the IoT middleware and the services in the LAN.

One of the challenges that we faced in this setup is to expose the localhost to the internet. Our research leads to setting up secure tunnels to the localhost for exposing a local server which is behind the firewall. The tool that helped us achieve this connection is ngrok [52] that gave us a public address to access the localhost.

5.7.1.3. Evaluation Results without Claims in the Cloud

The cloud computing platform that we chose was Windows Azure [53] and it provides a wide range of resources that are very helpful. The Java client was deployed using the Eclipse plugin for Azure and it was a straightforward and simple procedure. One of the biggest challenges in deploying the intermediary web server was to setup the certificates in the windows virtual machine which will act as the claims of the server. In our setup we had the intermediary server as IIS (Internet Information Services) hosted service in the local network setup and we deployed that service from the visual studio .NET framework into the cloud.

We tested the performance on the same set of requests which are 1, 10, 50, 100, 500 and 1000 requests using JMeter. The request was to communicate to the device end using the event management service. The request sets of 10, 50, 100, 500 and 1000 are generated by using 5, 25, 50, 250 and 500 users/threads with 2 requests respectively. In this setup, we disabled the claims and directed the request straight to the IoT Services without checking the authenticity of the server that is accessing the services. The average time to process one request for each set of requests is presented in the Table 5.3. The time taken by the request is the round trip time to reach the device end and get the response back to the client.

Table 5.3: Performance results without claims in the cloud

Without Claims in the Cloud				
No. of Requests	Average time per request (seconds)	Minimum Value	Maximum Value	Variance
1	0.36	0.18	0.39	0.02
10	0.42	0.83	1.43	0.015
50	0.31	0.72	1.28	0.003
100	0.25	0.61	1.38	0.0005
500	0.20	0.59	1.26	0.003
1000	0.25	0.4	1.17	0.018

The minimum and the maximum value of the average time obtained in the 10 iterations of each request set is presented in Table 5.3. These values show the largest (Maximum value) and the lowest (Minimum value) time taken to process the requests. The measure by which the data are distributed in each set is represented by the variance. It gives the spread of the values in that set. The difference in the processing time could be due to the changes in the network performance. If other users access the Internet to perform other tasks, the network may become slower which can may increase the execution time.

As seen in the Table 5.3, there is a slight increase in the average time per request in the test case for 1 request as compared to the test of 10 requests due the latency of the network. Evidently, the average time consumed per request is decreased for the rest of the samples. This is because the service uses thread pooling to serve the large number of concurrent requests. Thus the execution time per request is feasible for large number of requests.

The requests are made by the client sequentially. But the requests are asynchronously processed by the WCF framework. Thus the overall execution is in parallel threads. Thus the system makes sure that if the number of requests are increased, the performance of the system is improved with the help of thread pooling.

5.7.1.4. Performance with Claims in the Cloud

In this experimental setup, we make the calls to the IoT services from the client which is hosted in the cloud. The client then communicates to the web server which is deployed in the cloud. The call is then directed to the claims middleware which is running in the local machine. So in short, this setup represents the case if the IoT services are accessed over the internet from outside the local network.

5.7.1.5. Architecture of the Cloud Setup with Claims

The architecture of this setup consists of the same two main parts as that explained in the previous setup using the Cloud. In this evaluation setup we have the claims enabled in the LAN which actually checks whether the server using the IoT services is actually authenticated. Thus the client request is passed to the web server and then it is forwarded to the Claims middleware. If it is verified to be a valid request, then the call to the IoT service is processed.

5.7.1.6. Evaluation Results with Claims in the Cloud

By calculating the total time taken from the two timestamps in the two log files we obtained the time to transfer the call from the client ends to the IoT service end. The JMeter is configured for each set of requests like 1, 10, 50, 100, 500 and 1000 requests. The experimental results are presented in Table 5.4.

Table 5.4: Performance results with claims in the cloud

With Claims in the Cloud				
No. of Requests	**Average time per request (seconds)**	**Minimum Value**	**Maximum Value**	**Variance**
1	0.90	0.81	1.61	0.04
10	1.98	1.23	2.34	0.003
50	1.35	0.69	1.93	0.001
100	1.66	0.94	1.81	0.0003
500	1.36	0.73	1.52	0.0006
1000	1.76	0.99	1.89	0.003

Evidently, the increase in the processing time per request is visible here as we go from test case of 1 to 10 requests. Also, there is a decrease in the latency for 50 requests when compared to 10 requests. This is because the service uses thread pooling in the large set of concurrent calls. Thus the overall execution time is less in case of 50, 100, 500 and 1000 requests.

5.7.2. Comparison Results in the Cloud

The overhead of using our claims enabled middleware is found out by comparing the time taken to deliver the requests in the setup with and without claims. The obtained results are presented in the form of graph as shown in Figure 5.2.

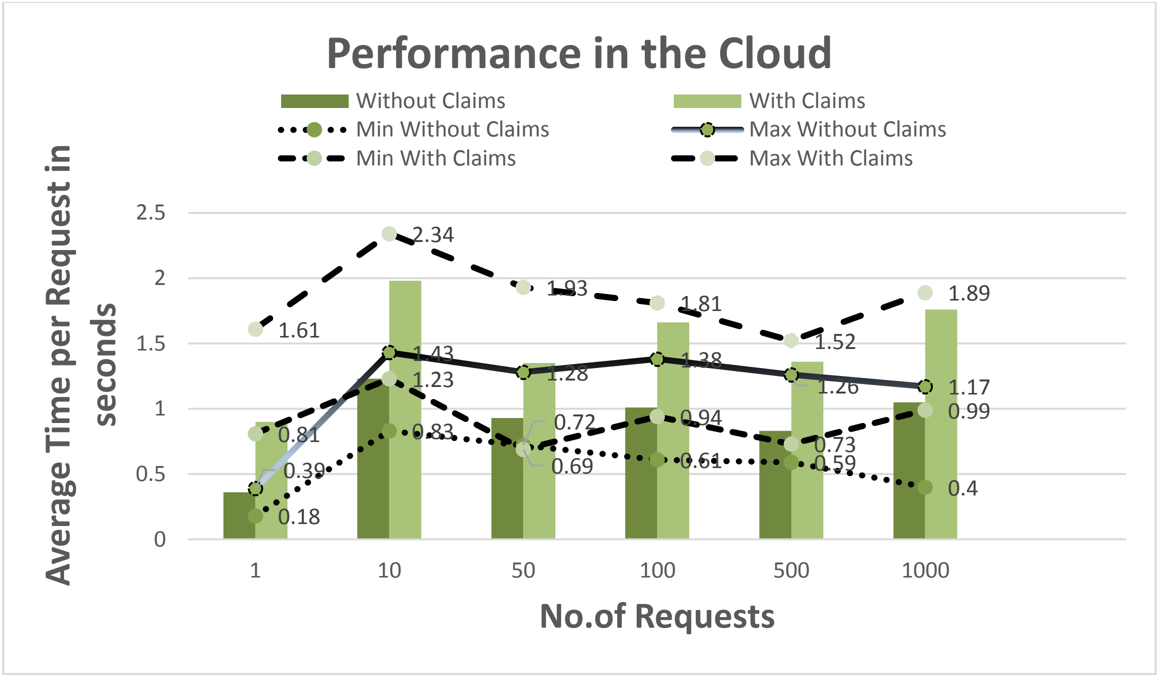


Figure 5.2: Graph showing the performance in the cloud

The average time per request in the Figure 5.6 is obtained using the assumption that the first request is executed first and the rest of the requests are being executed in the order as the calls are made. But this may not be the case every time.

- Worst case scenario**:** the worst case scenario would be in which the first request is getting processed at the last. For example, in the test case of 1000 requests, the total average time to process 1000 requests with claims is 1760 seconds (28 minutes approximately). The worst case would be the situation where the first request is executed after the rest of 999 requests. Thus in such a scenario, the worst case time per request would be 1760 seconds.

- Best case scenario: The best case scenario would be the situation as per our assumption. For example, the test case of 1000 requests take 1760 seconds to finish the whole set of requests. If the requests are processed in the same order as they are made, then the first request will be processed first and the last request would be process last. Thus the best case average time per request, for the test case of 1000 requests would be 0.17 seconds.

The increase in the response time as the load increases can be due to many factors such as the network latency, the performance of the server, the condition of the network, etc. the overall execution of the requests are in parallel and the requests are asynchrously processed by the service.

5.8. Comparison of LAN vs Cloud

After obtaining the set of results in both LAN and the Cloud, we compared the performance of our Claims enabled middleware in local area network and using the Cloud services. The scenario in which the calls are made over the internet is found to have more overhead than the LAN. This overhead is expected as the requests are made remotely over the internet rather than using local network in which the IoT middleware and services are hosted. The results obtained are presented in the graph format in Figure 5.3.

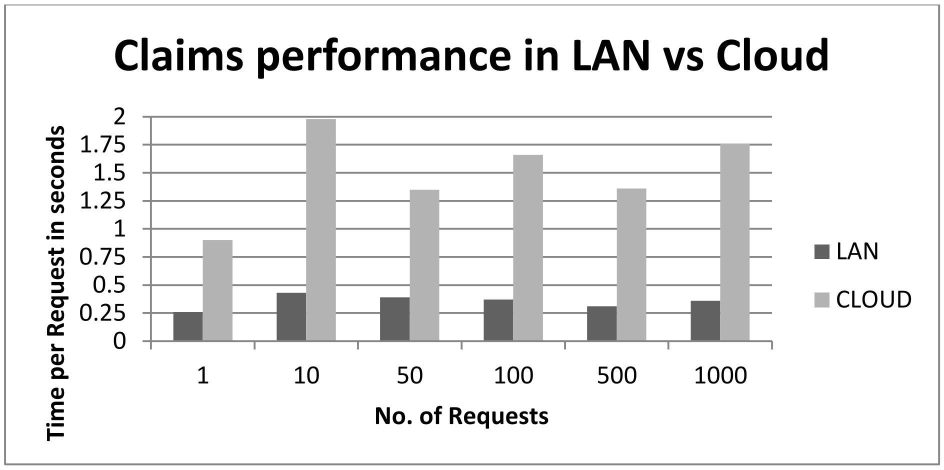
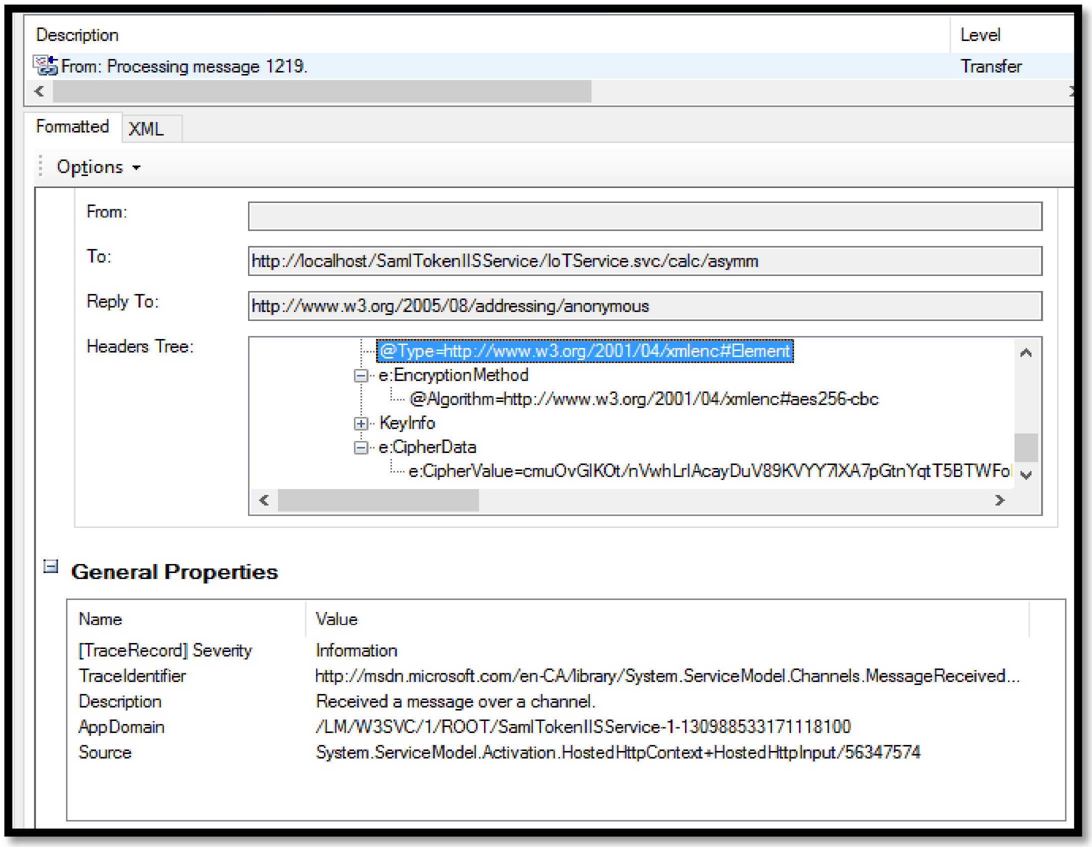


Figure 5.3: Graph showing the performance of claims in LAN vs the cloud

The increase in the average time to process a single request is slightly large for the request sets of 10 and above. It is because the network is loaded with multiple calls and the server is busy processing the queue of requests. The performance of our system in both LAN and in the cloud is tested to be feasible in processing large number of requests.

5.9. Performance Evaluation of Claims Security Model

The framework is tested to see if the intended function of the claims middleware is performed for the requests or calls being made. Since the middleware service is hosted as a local windows service using WCF and IIS, we were able to look at the messages being passed and view every details of the security features of that message.



Description | Level
From: Processing message 1219. | Transfer

From:
To: http://localhost/SamlTokenIISService/IoTService.svc/calc/asymm
Reply To: http://www.w3.org/2005/08/addressing/anonymous
Headers Tree:

```
@Type=http://www.w3.org/2001/04/xmlenc#Element
e:EncryptionMethod
    @Algorithm=http://www.w3.org/2001/04/xmlenc#aes256-cbc
KeyInfo
e:CipherData
    e:CipherValue=cmuOvGIKOt/nVwhLrIAcayDuV89KVYY7lXA7pGtnYqtT5BTWFo
```

General Properties

Name	Value
[TraceRecord] Severity	Information
TraceIdentifier	http://msdn.microsoft.com/en-CA/library/System.ServiceModel.Channels.MessageReceived...
Description	Received a message over a channel.
AppDomain	/LM/W3SVC/1/ROOT/SamlTokenIISService-1-130988533171118100
Source	System.ServiceModel.Activation.HostedHttpContext+HostedHttpInput/56347574

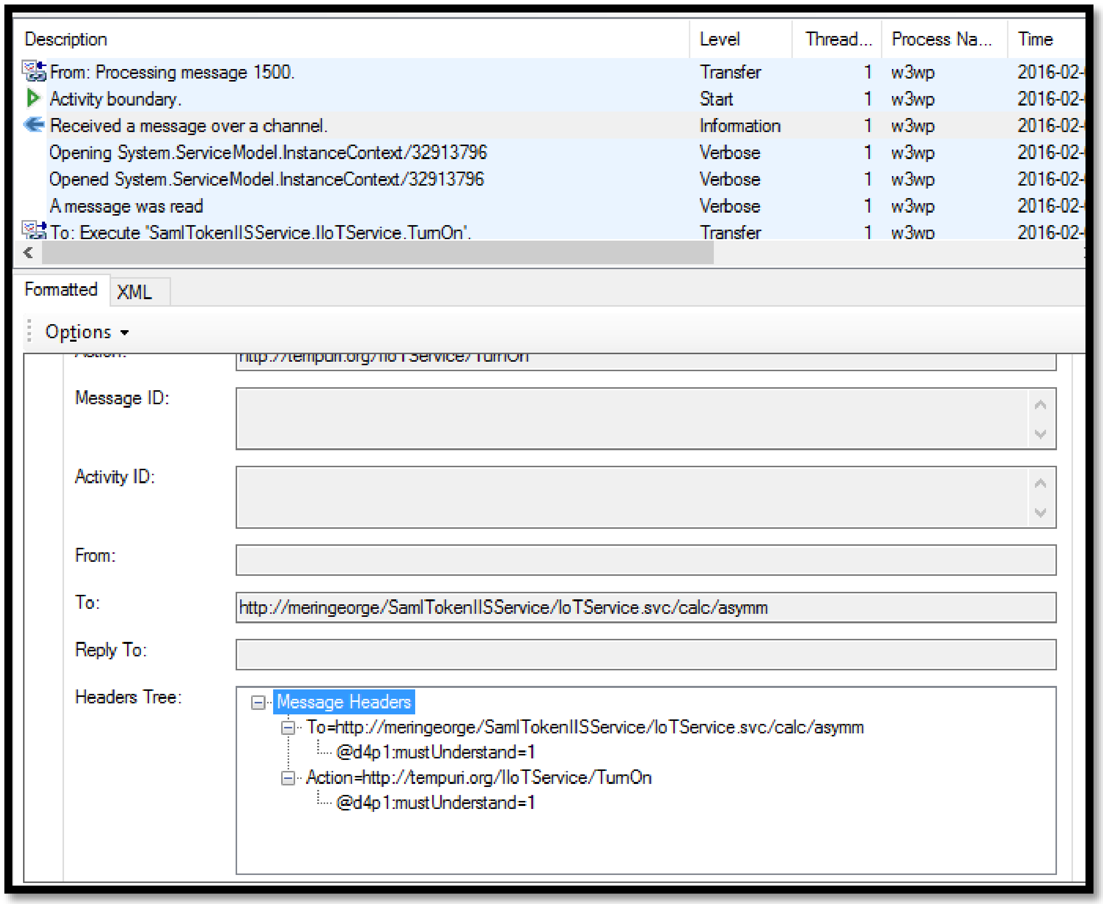
Figure 5.4: Security features of using claims

The screen shots of the Microsoft Service Trace Viewer are shown in Figure 5.4 shows one of the commands being processed by the Claims service. This screen was obtained when we ran the command with Claims enabled. It shows the encryption algorithm used for the message, the cipher text and the endpoint URL.

In the message presented in Figure 5.4, the encryption technique used is AES 256-CBC. AES stands for Advanced Encryption Standards and CBC stands for Cipher Block Chaining. It is a mode of operation of encryption and the cipher text after the encryption technique can also be seen in Figure 5.4.

We also tested the framework without enabling the claims. As shown in Figure 5.5, there were no cryptographical operations being done on the command being passed. The screenshot of the WCF trace viewer for no claims is shown in Figure 5.5.

In Figure 5.5, the message is just passed without any encryption by the middleware. The encryption technique or the cipher text would have been visible of it was encrypted. Thus it is evident that the security model using claims is delivering the intended security function if enabled. If the system could not find the intended claims in the intermediary web server, it does not call the IoT services.



Description	Level	Thread...	Process Na...	Time
From: Processing message 1500.	Transfer	1	w3wp	2016-02-
Activity boundary.	Start	1	w3wp	2016-02-
Received a message over a channel.	Information	1	w3wp	2016-02-
Opening System.ServiceModel.InstanceContext/32913796	Verbose	1	w3wp	2016-02-
Opened System.ServiceModel.InstanceContext/32913796	Verbose	1	w3wp	2016-02-
A message was read	Verbose	1	w3wp	2016-02-
To: Execute 'SamlTokenIISService.IIoTService.TurnOn'.	Transfer	1	w3wp	2016-02-

Formatted XML

Options

Action: http://tempuri.org/IIoTService/TurnOn

Message ID:

Activity ID:

From:

To: http://meringeorge/SamlTokenIISService/IoTService.svc/calc/asymm

Reply To:

Headers Tree:

```
Message Headers
  To=http://meringeorge/SamlTokenIISService/IoTService.svc/calc/asymm
    @d4p1:mustUnderstand=1
  Action=http://tempuri.org/IIoTService/TurnOn
    @d4p1:mustUnderstand=1
```

Figure 5.5: Security features of not using claims

5.10. Summary

The evaluation of our framework is made and the overhead of using the claims are obtained in two scenarios, one in which everything is running locally and the other where the calls are made remotely over the internet. Our framework produced satisfactory results as expected and thus it reveals the efficiency of our system.

We also implemented a case study and its evaluation using a smart device, which is presented in the next chapter. The implementation details and the performance results are presented in brief.

Chapter 6

Case Study: Smart Home Automation

In this chapter we will talk about the case study in which we applied the prototype into a home automation system. A detailed description of how a third party device is integrated into the prototype is also presented in this chapter. The architecture of the case study using an IoT smart switch and the sequence in which the communication flows among the user and the services are also explained.

6.1. Overview

Internet of Things has various applications such as eHealth smart system, smart grid, smart city, smart home and so on. The reason behind choosing home automation system as case system is that it is cost effective and can be even applied to normal users. Security is one of the biggest factors not only to the corporate societies but also to everyday society. The proof is the increasing consumption of smart security cameras and appliances. People want to go out without worrying about the safety of their houses. There were days when people lived in the fear of someone breaking into their houses stealing documents and personal identity. It is because home is the place to look at if we want to know about a person. But with the enhancement of technologies, this fear has been eliminated by smart devices and applications. Securing the private data is also important because the knowledge of the location of a user could be used to decide that no one is at home. Also, the leakage of personal data like bank details and health related data can put the person’s life at risk. Thus securing the services or the applications being used is also important.

In our system, the servers that are trying to access the IoT services are verified by the claims-aware middleware before granting the permission. Therefore, in this case study, the server that tries to access the IoT services to control the smart device is verified by the claims-aware middleware. Once the claims produced by the intermediary server is validated by the security service, the request is forwarded to the device end.

The advantages of using our system in home automation are:

1) It restricts the access of home automation services to invalid servers as they will be denied of the service if they could not produce valid claims.
2) User can add more devices into the network using our system that will be registered into our system.
3) Our system also shows the list of devices that are present in the network and gives the user the authority to add, update, search and remove a device.
4) User can control the device from anywhere if they have access to internet.
5) It logs all the activities of a particular device into the database and could be used by the client to see the activities or to determine the energy consumption by the device.

6.2. Physical Components of Home Automation

To do the case study of home automation system, we used one smart device and performed the IoT services presented in the functional components of IoT middleware as discussed in Chapter 3.

6.2.1. Belkin WeMo Switch

Belkin WeMo is a family of many smart appliances that could be controlled using the mobile network [54]. It is very cost effective and lets the users to control the device connected to the switch anywhere over Wi-Fi, 3G or 4G networks. WeMo has an application that is supported in Android as well as in Apple operating systems and this app is used to normally control the device.

Figure 6.1: Belkin WeMo smart switch

It also works with IFTT which stands for “if this then that” [55]. IFTT is a simple web-oriented service which is used to create *‘recipes’* that connect several other web applications. What it basically does is that, it creates conditional statements that trigger our services on the basis of an event output from other applications such as e-mail, text messages, etc.

6.2.2. Lamp

The simplest device in case of home automation is a lamp that has the functions to be turned on/off. The lamp is made smart by connecting it with the WeMo smart switch that can be set to turn on/off as per the need of the user

6.2.3. D-LINK Wireless Router

The smart switch is connected using the wireless router named D-LINK (dlink) DIR-505 router [56]. It is very handy and multipurpose router too. The key features of this router are as follows:

- It is a portable plug-in router
- It also acts as a repeater
- It can be used as a Wi-Fi hotspot
- It functions as a USB with the help of SharePort Mobile App and can access our files wirelessly from tablets or phones.
- It also has a provision to function as Mobile charger.

The specifications of the lamp and the D-LINK device can be found in [56, 57].

6.3. Home Automation Architecture

The architecture of our case study scenario is presented in this section. The structure of architecture is formed out of the implementation flow of the system. The communication and the configuration details of each component will be explained in the following part of the section. Figure 6.2 shows the architecture of the home automation setup using WeMo smart switch. The case study is conducted in local area network and each of the services is running in the local machine.

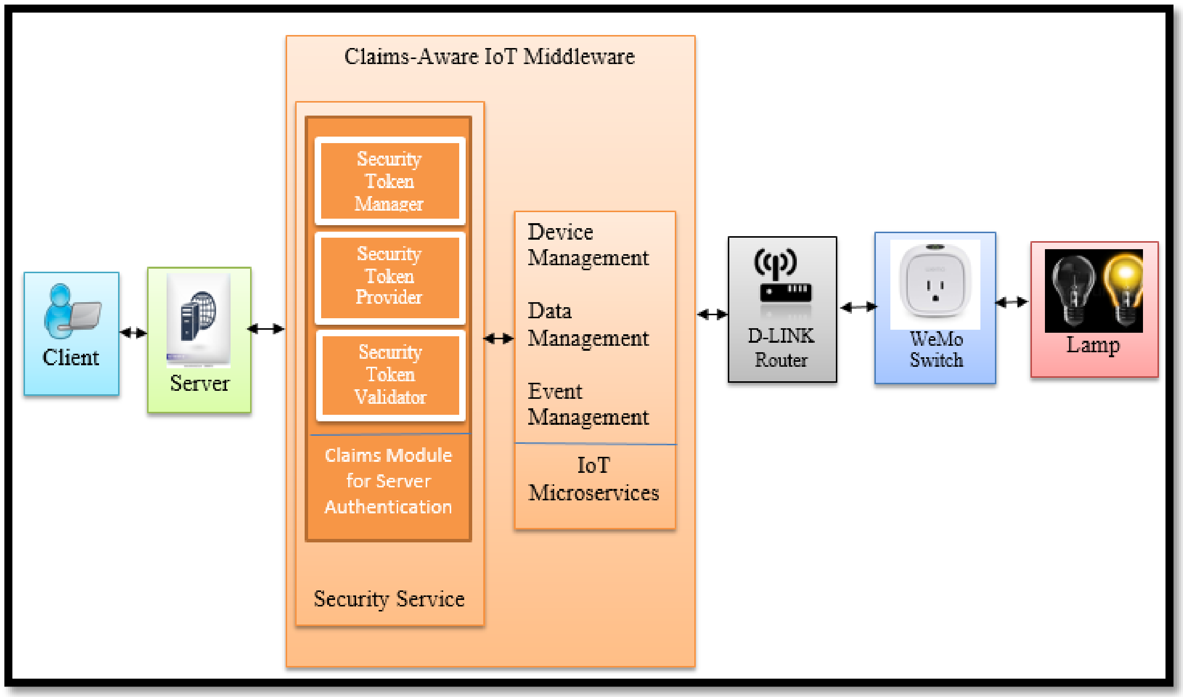


Figure 6.2: Architecture for the home automation

The main components of the architecture as shown in Figure 6.2 are:

- **Client:** It represents the valid user who accesses his network devices. Our framework provides a user friendly interface to the client and the services are made simple to the client so that a person even without any programming background will be able to use it. The user is asked to authenticate themselves by providing valid claims such as Username and Password. Once the user is authenticated, the web client passes the requests from the user to the intermediary server. The server then forwards the requests along with its claims to the claims-middleware.

- **IoT Middleware:** IoT middleware consists of four functional components such as security service, device management service, data management service and event management service. The security service (claims-aware middleware) stands as a gateway between the intermediary server and the IoT services. The key feature that distinguishes our solution is the claims-identity based security service. The rest of the services are also secured using one time password (OTP). Each time when the user logs in, independent OTP is created and used for the session. Each of these microservices, as shown in Figure 6.3, is an independent service.

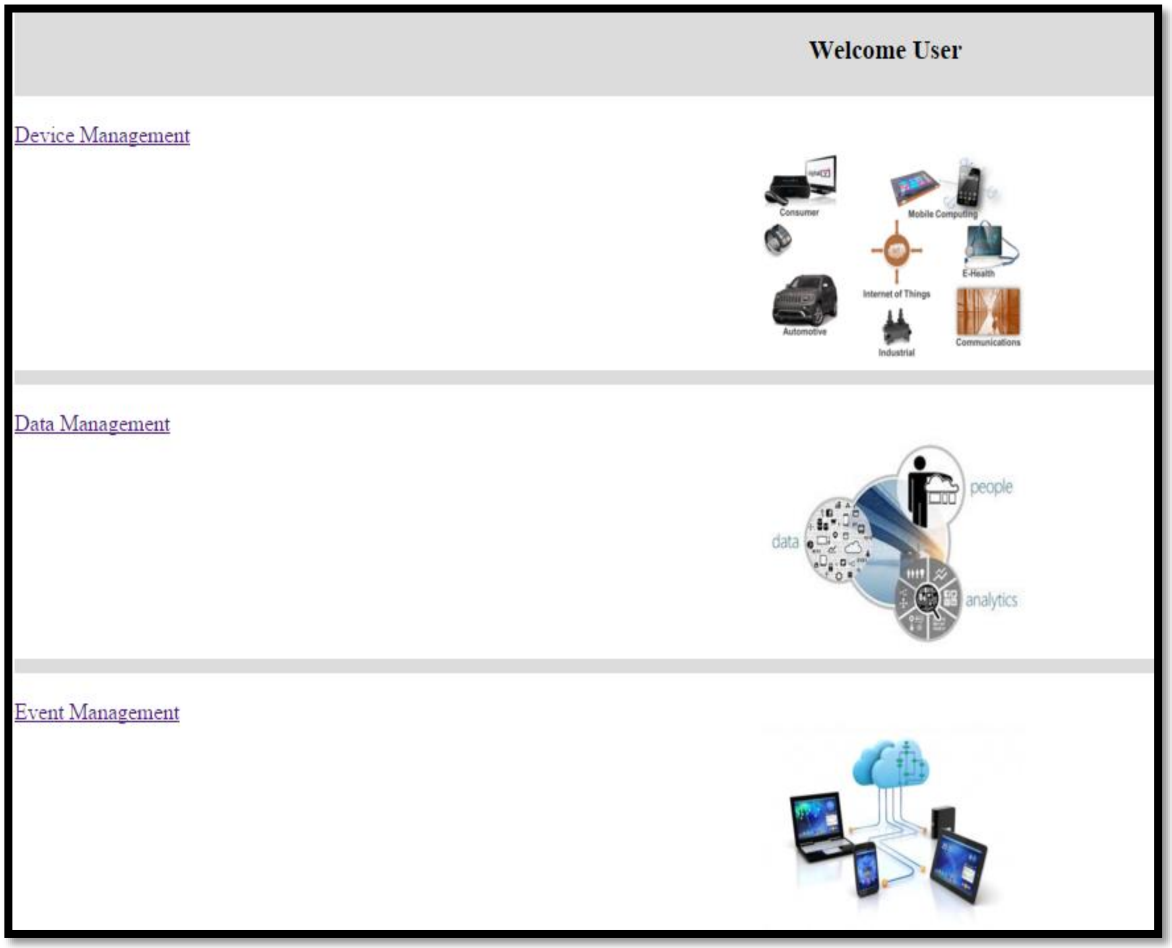


Figure 6.3: Web client showing IoT services in home automation application

When a command is passed to control the lamp using event management service, the system checks for valid claims from the server that produce the request. If it is verified to be correct, the access is granted and in the other case, we assume that an invalid third- party is trying to access the services to gain access to the user's IoT device. In such a case, the request is invalid and the security service deny the access and throws an exception. In our system, we have used two certificates which are stored in our local machine and are used to encrypt and sign the claims as explained in the implementation chapter.

- **Router, Switch and Lamp**: The smart switch (WeMo) uses UPnP network protocol and the communication was achieved using a wireless router. In order to do the initial setup of the WeMo switch, we had to download the WeMo app and connect to WeMo Wi-Fi network. Since we wanted to create our own interface rather than using the app, the network connection is changed to the D-LINK network in the WeMo app.

  To integrate the smart WeMo smart switch into our prototype, we used an API [58] that could be used to control the switch and incorporated that API into our framework. To make the communication possible, the local machine was also connected to the D-LINK router and the WeMo device was visible under the connected devices. Thus we were able to control the lamp connected to the smart switch that communicates via the router. Thus a third party service has been integrated into our framework which also indicates the scalability of our middleware.

6.4. The Process Flow

The sequence in which the communication flows in the smart home setup is illustrated in Figure 6.4. The request generated from the web client by the user is directed to the web server. The server passes the request along with its claims to the claims middleware. The middleware verify the validity of the server and forwards the requests to the different IoT services such as device management, event management and data management.

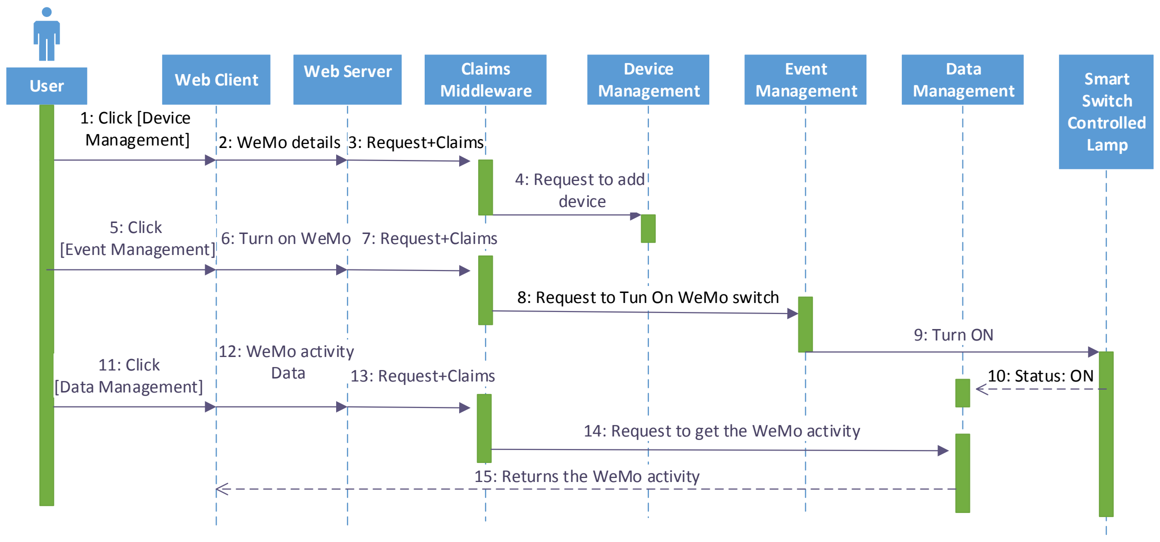


Figure 6.4: UML sequence diagram of communication process in smart home

At first, the user registers the WeMo smart device in the network using the IP address contained in the WeMo details using the device management service. Once the device is added, it can be controlled using event management service to turn the lamp on / off. Whenever the status of the switch is changed, the data is logged using the data management service and the activities of the device can be obtained using this IoT service.

6.5. Performance Evaluation

This section presents the evaluation done on the case study of home automation system using our framework. After setting up the components as shown in the architecture, we tried to access the end device using the web client.

As a user, an account was created and logged into the user account. Once the client is authenticated, he/she has the access to our IoT services specially designed for this case, Home Automation. For other applications, all components except the event management will be the same. Since the situation occurring at each event is different, to provide a proof of concept, we worked only on the scenario of an event in home automation. The results of such an event may not work for another kind of situation like eHealth system.

As a client, after getting access to the services, we evaluated the performance of each of the IoT services and see if the services are being delivered correctly. When a request is passed from the client via an intermediary web server, the claims-middleware checks for the claims that are presented by the intermediary web server. If the claims are verified to be valid, the call to the end device is sanctioned. In case the claims are invalid or if it does not have any claims, the system throws an exception and are not allowed to make the call to the device end. The functionalities of each of the IoT services are tried as follows:

- **Device Management**: This service lets the user register a device to his network. Using the user interface, we clicked on the link to add a new device. There we had the options to enter the details of a particular device. In our case, the device name was WeMo smart lamp. The IP address of the smart switch was assigned dynamically by the network initially. So we had to configure the IP address of the WeMo switch in the web page of the router and assigned a static address, and we entered the device ID, IP address and location of the device. There were provisions to view all the devices in the network, search for a device using the ID, update and remove the device using the same ID.

- **Data Management**: This service deals with the data output from the device. In our case, we retrieve the status data of the device and stored it in the database in case of future analysis by the user. Also it presents the current status of the device to the end user.

- **Event Management**: Using this service, we could control the device remotely using the interface provided. As it is a lamp, we can either turn that on or off only after validating the claims.

So we tried all the functional components of the IoT and tested if they are delivering the services efficiently. Using these IoT services, the WeMo smart switch has been added to the network, status of each event has been logged to see its activities and the lamp can be controlled (turn on/ turn off) remotely.

The performance of our system in this case study is evaluated on the basis of time taken by the middleware to process the requests with and without Claims. To find out the average overhead of claims, we used the event management service to generate the requests to control the lamp connected to the WeMo switch. The evaluation results obtained are presented in Figure 6.5.

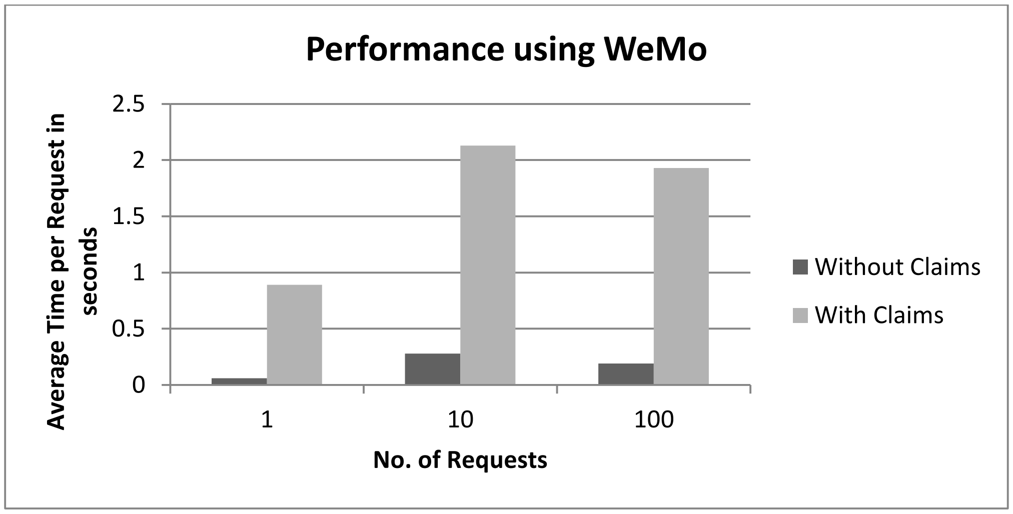


Figure 6.5: Graph showing the performance of case study

The multiple requests are generated using the JMeter tool. There are three sets of requests such as 1, 10 and 100. The average overhead of using claims to validate the server in the test case of one request to turn on the WeMo switch was 0.06 seconds. The test case of 100 requests consists of 5 turn on requests and 5 turn off requests being processed alternatively. The test case of 100 requests consists of 50 turn on and 50 turn off requests being processed alternatively. These requests are processed in parallel by the middleware.

The average overhead of using claims in the real world scenario of smart home using the WeMo switch is 1.47 seconds. The time taken to process the 100 requests are less compared to 10 requests. This is because of the fact that the Thread Pool will span new threads in order to serve the incoming requests in case of an increased load. That makes the performance of the system feasible even with the increased number of requests.

6.6. Summary

In this chapter, we discussed the implementation details about setting up a home automation system which is used in our case study. The way in which the communication is achieved from the user end to the device end is explained in brief. Also, the evaluation setup and the results obtained in our case study are presented. We will conclude our Thesis in the next section along with the future works.

Chapter 7

Conclusions and Future Work

In this thesis, we introduced a middleware for *Internet of Things* services. The need of such a software layer and the basic functions of an IoT middleware are presented. One of the major challenges associated with IoT, s*ecurity,* is analyzed in detail and C*laims-aware* approach for securing IoT services is introduced. Implementing these services as *microservices* makes the system more scalable and distributed in nature. A brief discussion about the existing IoT middleware and the challenges faced by each framework has been presented. All those related works were very helpful in understanding the functionalities of IoT middleware layer and paved our way to the proposed solution.

7.1. Contributions

The contributions of this thesis are:

- A claims-aware framework for IoT services has been introduced to add security to the IoT services. Claims-based identity provides the valid claims for each service in the network and maintains the secrecy of the network efficiently.
- The IoT services are implemented as Microservices architecture. Rather than following the monolithic approach in which every services comes under one server, implementing IoT services as Microservices eliminated the threat of single point of failure. Splitting up of a large task into smaller modules has always been

the best practice for error debugging. This approach enables new services to be added to the network without affecting the existing services.

- A prototype of the proposed solution has been developed as the proof of concept.
- The prototype is used to evaluate the performance of our framework and obtained satisfactory results that support the feasibility of our prototype. The evaluation is done in LAN as well as in the Cloud.
- A home automation system is implemented and used as the case study for our prototype of IoT middleware.

7.2. Future work

The research area in the field of IoT middleware is very vast. It is not that easy to have answered all the challenges answered in this thesis. Some of the remaining research topics are:

- Security is just one of the major challenges among many other threats like Big Data, openness, etc., and a complete system will be formed only when all the challenges are answered. Since it is not possible within the scope, we would like to go through other challenges in the future.
- Based on the background works done, we believed that the implementation of IoT services as Microservices is an efficient method that totally supports decentralization and independency. But we have not made any evaluation of Microservices architecture and that is another area of interest that can be done in future work.

- Since our main focus is on security, we implemented the IoT services that are essential to support our framework rather than adding all the services. In our prototype, we have not done the functional component interoperability which does the protocol conversion. Interoperability itself is a wide area that could be studied and researched on.
- The fact that Raspberry Pi (RPi) does not support the Windows Communication Foundation made us shift the gateway into the local machine. In the future, we could host the gateway in the RPi itself by finding some other ways to do the Claims by researching on it.
- Data Mining is a vast area that should be given special care in case of a large volume of data. In the future, we could implement some sort of data analysis and mining techniques.

Appendix A

Source Code

Selected snippets of the source code that are relavant to the implementation and evaluation of the prototype are presented in this section.

A.1 Intermediary Server Web.config

The web bindings that configures the SAML tokens are presented in here. The bindings for the two endpoints for the symmetric and the assymetric call is shown along with the token type issued, which is SAML.

```
<bindings>
      <wsFederationHttpBinding>
        <binding name="Symmetric" closeTimeout="00:2:00"
            openTimeout="00:2:00"
          receiveTimeout="00:2:00" sendTimeout="00:2:00"
            maxReceivedMessageSize="2147483647">
          <security mode="Message">
            <message issuedTokenType=
                  "http://docs.oasis-open.org/wss/
                  oasis-wss-saml-token-profile-1.1#SAMLV1.1"
              negotiateServiceCredential="false" />
          </security>
        </binding>
        <binding name="Assymetric" closeTimeout="00:2:00"
            openTimeout="00:2:00"
          receiveTimeout="00:2:00" sendTimeout="00:2:00"
            maxReceivedMessageSize="2147483647">
          <security>
            <message issuedKeyType="AsymmetricKey"
                  issuedTokenType="http://docs.oasis-open.org/wss/
                  oasis-wss-saml-token-profile-1.1#SAMLV1.1"
              negotiateServiceCredential="false" />
          </security>
        </binding>
      </wsFederationHttpBinding>
    </bindings>
```

The client endpoint of Intermediary server points towards the Claims middleware running in the local machine using the localhost URL. The encoded value of the certificates used for symmetric as well as assymetric binding is shown in the web.config file.

```
<client>
      <endpoint

address="http://localhost/SamlTokenIISService/IoTService.svc/calc/symm"

        binding="wsFederationHttpBinding" bindingConfiguration="Symmetric"
        contract="DeviceControllerClient.IIoTService" name="Symmetric">
        <identity>
          <certificate

encodedValue="AwAAAAEAAAAUAAAApDyI78FhbVWR+OxltL+Uqg04hCogAAAAAQAAADkCAAAwggI1MII
B46ADAgECAhA3K2wdfTBTt0/ctdFGvAGJMAkGBSsOAwIdBQAwFjEUMBIGA1UEAxMLUm9vdCBBZ2VuY3kw
HhcNMTUwOTA0MjM1MTU5WhcNMzkxMjMxMjM1OTU5WjAUMRIwEAYDVQQDEwlsb2NhbGhvc3QwggEiMA0GC
SqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQCe2Q9ODtVPjFed1ttf6zFzr7tYSJZHuw63Fde2rkwA/+/qQc
A80QTjpPKb5tzgQLW27uOG/8tQysk9B6pUcggXgcaEQonWmJFPzsjs70n4NL8Qiuuv6VylbwAtvWXlt+y
tyZLGxmRX0hKLwkVwwck7HM1qw/L8aISpKASK8pqIqUlsDZI1KrfVgfgqqXRnxt47InGVIk4aIXXoMjId
rlyAIzmx/H3fd8Q1m23/jtdUYFKSZPFoVLg/+YvELlFQdulZRCSyN8csaFQBTa8nnUVLKKIw36TijTl9s
uszl5LQ3oSHhdyqbJG0SkCEgoTj1hwbeUBFsK4+aYqlGcETP01HAgMBAAGjSzBJMEcGA1UdAQRAMD6AEB
LkCS0GHR1PAI1hIdwWZGOhGDAWMRQwEgYDVQQDEwtSb290IEFnZW5jeYIQBjdsAKoAZIoRz7jUqlw19DA
JBgUrDgMCHQUAA0EAFlF00X7mPsbhOrRCb3oj03MW2zZXVUnbkITWztH1akSvHcLRI0AQ/6kwod9CF1QP
C3CATq61fcDePX/GuwLWTA==" />

        </identity>
      </endpoint>
      <endpoint

address="http://localhost/SamlTokenIISService/IoTService.svc/calc/asymm"

        binding="wsFederationHttpBinding" bindingConfiguration="Assymetric"
        contract="DeviceControllerClient.IIoTService" name="Assymetric">
        <identity>
          <certificate

encodedValue="AwAAAAEAAAAUAAAApDyI78FhbVWR+OxltL+Uqg04hCogAAAAAQAAADkCAAAwggI1MII
B46ADAgECAhA3K2wdfTBTt0/ctdFGvAGJMAkGBSsOAwIdBQAwFjEUMBIGA1UEAxMLUm9vdCBBZ2VuY3kw
HhcNMTUwOTA0MjM1MTU5WhcNMzkxMjMxMjM1OTU5WjAUMRIwEAYDVQQDEwlsb2NhbGhvc3QwggEiMA0GC
SqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQCe2Q9ODtVPjFed1ttf6zFzr7tYSJZHuw63Fde2rkwA/+/qQc
A80QTjpPKb5tzgQLW27uOG/8tQysk9B6pUcggXgcaEQonWmJFPzsjs70n4NL8Qiuuv6VylbwAtvWXlt+y
tyZLGxmRX0hKLwkVwwck7HM1qw/L8aISpKASK8pqIqUlsDZI1KrfVgfgqqXRnxt47InGVIk4aIXXoMjId
rlyAIzmx/H3fd8Q1m23/jtdUYFKSZPFoVLg/+YvELlFQdulZRCSyN8csaFQBTa8nnUVLKKIw36TijTl9s
uszl5LQ3oSHhdyqbJG0SkCEgoTj1hwbeUBFsK4+aYqlGcETP01HAgMBAAGjSzBJMEcGA1UdAQRAMD6AEB
LkCS0GHR1PAI1hIdwWZGOhGDAWMRQwEgYDVQQDEwtSb290IEFnZW5jeYIQBjdsAKoAZIoRz7jUqlw19DA
JBgUrDgMCHQUAA0EAFlF00X7mPsbhOrRCb3oj03MW2zZXVUnbkITWztH1akSvHcLRI0AQ/6kwod9CF1QP
C3CATq61fcDePX/GuwLWTA==" />

        </identity>
      </endpoint>
    </client>

```

The endpoint URL is different from the one in LAN when the Claims middleware is accessed from the Cloud over the internet. The tool named ngrok [47] produce a secure tunnel into the localhost and provided a public URL to access our local machine from the Cloud.

```
<!--client>
  <endpoint

address="http://03ae8de2.ngrok.io/SamlTokenIISServicev2/IoTService.svc/calc/sym
m"

    binding="wsFederationHttpBinding" bindingConfiguration="Symmetric"
    contract="DeviceControllerClient.IIoTService" name="Symmetric">
    <identity>
      <certificate

encodedValue="AwAAAAEAAAAUAAAApDyI78FhbVWR+OxltL+Uqg04hCogAAAAAQAAADkCAAAwggI1M
IIB46ADAgECAhA3K2wdfTBTt0/ctdFGvAGJMAkGBSsOAwIdBQAwFjEUMBIGA1UEAxMLUm9vdCBBZ2Vu
Y3kwHhcNMTUwOTA0MjM1MTU5WhcNMzkxMjMxMjM1OTU5WjAUMRIwEAYDVQQDEwlsb2NhbGhvc3QwggE
iMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQCe2Q9ODtVPjFed1ttf6zFzr7tYSJZHuw63Fde2rk
wA/+/qQcA80QTjpPKb5tzgQLW27uOG/8tQysk9B6pUcggXgcaEQonWmJFPzsjs70n4NL8Qiuuv6Vylb
wAtvWXlt+ytyZLGxmRX0hKLwkVwwck7HM1qw/L8aISpKASK8pqIqUlsDZI1KrfVgfgqqXRnxt47InGV
Ik4aIXXoMjIdrlyAIzmx/H3fd8Q1m23/jtdUYFKSZPFoVLg/+YvELlFQdulZRCSyN8csaFQBTa8nnUV
LKKIw36TijTl9suszl5LQ3oSHhdyqbJG0SkCEgoTj1hwbeUBFsK4+aYqlGcETP01HAgMBAAGjSzBJME
cGA1UdAQRAMD6AEBLkCS0GHR1PAI1hIdwWZGOhGDAWMRQwEgYDVQQDEwtSb290IEFnZW5jeYIQBjdsA
KoAZIoRz7jUqlw19DAJBgUrDgMCHQUAA0EAFlF00X7mPsbhOrRCb3oj03MW2zZXVUnbkITWztH1akSv
HcLRI0AQ/6kwod9CF1QPC3CATq61fcDePX/GuwLWTA==" />

    </identity>
  </endpoint>

  <endpoint

address="http://03ae8de2.ngrok.io/SamlTokenIISServicev2/IoTService.svc/calc/asy
mm"

    binding="wsFederationHttpBinding" bindingConfiguration="Assymetric"
    contract="DeviceControllerClient.IIoTService" name="Assymetric">
    <identity>
      <certificate

encodedValue="AwAAAAEAAAAUAAAApDyI78FhbVWR+OxltL+Uqg04hCogAAAAAQAAADkCAAAwggI1M
IIB46ADAgECAhA3K2wdfTBTt0/ctdFGvAGJMAkGBSsOAwIdBQAwFjEUMBIGA1UEAxMLUm9vdCBBZ2Vu
Y3kwHhcNMTUwOTA0MjM1MTU5WhcNMzkxMjMxMjM1OTU5WjAUMRIwEAYDVQQDEwlsb2NhbGhvc3QwggE
iMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQCe2Q9ODtVPjFed1ttf6zFzr7tYSJZHuw63Fde2rk
wA/+/qQcA80QTjpPKb5tzgQLW27uOG/8tQysk9B6pUcggXgcaEQonWmJFPzsjs70n4NL8Qiuuv6Vylb
wAtvWXlt+ytyZLGxmRX0hKLwkVwwck7HM1qw/L8aISpKASK8pqIqUlsDZI1KrfVgfgqqXRnxt47InGV
Ik4aIXXoMjIdrlyAIzmx/H3fd8Q1m23/jtdUYFKSZPFoVLg/+YvELlFQdulZRCSyN8csaFQBTa8nnUV
LKKIw36TijTl9suszl5LQ3oSHhdyqbJG0SkCEgoTj1hwbeUBFsK4+aYqlGcETP01HAgMBAAGjSzBJME
cGA1UdAQRAMD6AEBLkCS0GHR1PAI1hIdwWZGOhGDAWMRQwEgYDVQQDEwtSb290IEFnZW5jeYIQBjdsA
KoAZIoRz7jUqlw19DAJBgUrDgMCHQUAA0EAFlF00X7mPsbhOrRCb3oj03MW2zZXVUnbkITWztH1akSv
HcLRI0AQ/6kwod9CF1QPC3CATq61fcDePX/GuwLWTA==" />

    </identity>
  </endpoint>
```

A.2. Service Contract of Intermediary Server

The services are basically the operations that has to be performed and the contract determines how the operation is performed. Usually in service contract there can be muliple operations with different sets of arguments. The contract that is shown here represents the WebGet attribute that can be used to get the information from the service pointed out by the URI.

```
namespace Event
{
    [ServiceContract]
    public interface IIotRestMethods
    {
        [OperationContract]
       // [WebGet(UriTemplate =
"/Invoke?deviceID={deviceID}&action={action}&userID={userID}&otp={otp}")]
        [WebInvoke(UriTemplate = "/Invoke", Method = "POST", ResponseFormat =
WebMessageFormat.Json
            , RequestFormat = WebMessageFormat.Json)]
         bool ControlDevice(IoTParamsparam);
   }
}
```

A.3 Intermediary Server calling Claims middleware [32]

The code snippet shows the function used to call the Claims middleware and pass on the arguments which contains the data to communicate to the smart device. It also shows how to create a SAMl client and use the certificates for Claims.

```
void CreateSAMLAndCallModule(string action, string ID, string token, string IoTIP)
        {
            bool success = false;
            DeviceControllerClient.IoTServiceClient client = null;
            try
            {
                // Create a client with given client endpoint configuration
                client =new DeviceControllerClient.IoTServiceClient("Assymetric");
                client.ClientCredentials.SupportInteractive = false;
               // client.ChannelFactory.Credentials.UseIdentityConfiguration=true;
                // Create new credentials class
                SamlClientCredentialssamlCC = new SamlClientCredentials();
            // Set the client certificate. This is the cert that will be used to sign the SAML token in the symmetric proof key case0c c1 11 4c c7 24 04 8c 56 65 1f 80 f6 9c 8d 9d e8 61 b4 ed
            samlCC.ClientCertificate.SetCertificate(StoreLocation.CurrentUser, StoreName.TrustedPeople, X509FindType.FindByThumbprint, "0cc1114cc724048c56651f80f69c8d9de861b4ed"); //Alice
            // Set the service certificate. This is the cert that will be used to encrypt the proof key in the symmetric proof key case
            samlCC.ServiceCertificate.SetDefaultCertificate(StoreLocation.LocalMachine, StoreName.My, X509FindType.FindByThumbprint, "a43c88efc1616d5591f8ec65b4bf94aa0d38842a"); //localhost
            // Create some claims to put in the SAML assertion
            IList<Claim> claims = new List<Claim>();
            var s = samlCC.ClientCertificate.Certificate.Subject;
            claims.Add(Claim.CreateNameClaim(samlCC.ClientCertificate.Certificate.Subject));
            ClaimSetclaimset = new DefaultClaimSet(claims);
            samlCC.Claims = claimset;
            // set new credentials
            client.ChannelFactory.Endpoint.Behaviors.Remove(typeof(ClientCredentials));
            client.ChannelFactory.Endpoint.Behaviors.Add(samlCC);
            if (action.Equals("on"))
            {

                varuserAuth = client.TurnOn(ID, token, IoTIP); //Claims logs here
                if (userAuth) //If call to claims service validated then proceed to turn on device
                {
                    _logger.Info("Wemo device: {0} turned ON.", ID); //WebServer logs here
                }
                else
                {
                    _logger.Info("Error validating user for device: {0}", ID);
                }
            }
            if (action.Equals("off"))
            {
                varuserAuth = client.TurnOff(ID, token, IoTIP);

                if (userAuth)
                {
                    _logger.Info("Wemo device: {0} turned OFF.", ID);
                }
                else
                {
                    _logger.Info("Error validating user for device: {0}", ID);
                }
            }
        }
       }
```

A.4. Claims Middleware Web.config

The web.config file of our Claims middleware shows the service behaviors and their corresponding endpoints. The two service behavior configurations set to do the evaluation with and without claims can also be seen. They are NoSecurityServiceBehavior and CalculatorServiceBehavior.

```
<services>
      <!--NO CLAIMS: ENABLED BELOW, once uncommented,
       comment the other section out, then Publish to IIS-->

      <service behaviorConfiguration="NoSecurityServiceBehavior"
       name="SamlTokenIISService.IoTService">
         <endpoint address="mex" binding="mexHttpBinding" name="Metadata"
           contract="IMetadataExchange" />
         <endpoint address="calc/symm" binding="basicHttpBinding"
           bindingConfiguration="NewBinding0" name="Symmetric"
               contract="SamlTokenIISService.IIoTService" />
         <endpoint address="calc/asymm" binding="basicHttpBinding"
           bindingConfiguration="NewBinding0" name="Assymetric"
               contract="SamlTokenIISService.IIoTService" />
      </service>


      <!--CLAIMS: ENABLED BELOW, once uncommented, comment
       the other section out, then Publish to IIS-->

      <service behaviorConfiguration="CalculatorServiceBehavior"
       name="SamlTokenIISService.IoTService">
         <endpoint address="mex" binding="mexHttpBinding" name="Metadata"
           contract="IMetadataExchange" />
         <endpoint address="calc/symm" binding="wsFederationHttpBinding"
           bindingConfiguration="Binding1" name="Symmetric.OLD"
               contract="SamlTokenIISService.IIoTService" />
         <endpoint address="calc/asymm" binding="wsFederationHttpBinding"
           bindingConfiguration="Binding2" name="Assymetric.OLD"
               contract="SamlTokenIISService.IIoTService" />
      </service>
</services>
```

```
<bindings>
      <basicHttpBinding>

        <binding name="NewBinding0" closeTimeout="00:2:00" openTimeout="00:2:00"
receiveTimeout="00:2:00" sendTimeout="00:2:00" maxReceivedMessageSize="2147483647"
/>
        <binding name="BasicServiceBinding" />
      </basicHttpBinding>
      <wsFederationHttpBinding>
        <binding name="Binding1" closeTimeout="00:2:00" openTimeout="00:2:00"
receiveTimeout="00:2:00" sendTimeout="00:2:00"
maxReceivedMessageSize="2147483647">
          <security mode="Message">
            <message issuedKeyType="SymmetricKey"
issuedTokenType="http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-
1.1#SAMLV1.1"
              negotiateServiceCredential="false" />
          </security>
        </binding>

        <binding name="Binding2"  closeTimeout="00:2:00" openTimeout="00:2:00"
receiveTimeout="00:2:00" sendTimeout="00:2:00"
maxReceivedMessageSize="2147483647">
          <security mode="Message">
            <message issuedKeyType="AsymmetricKey"
issuedTokenType="http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-
1.1#SAMLV1.1"
              negotiateServiceCredential="false" />
          </security>
        </binding>
      </wsFederationHttpBinding>
</bindings>

<behaviors>
      <serviceBehaviors>
        <behavior name="NoSecurityServiceBehavior">
          <dataContractSerializermaxItemsInObjectGraph="6553500"/>
          <serviceMetadatahttpGetEnabled="true"/>
          <serviceDebugincludeExceptionDetailInFaults="true"/>
           </behavior>

```

```
<behavior name="CalculatorServiceBehavior">
          <dataContractSerializermaxItemsInObjectGraph="6553500"/>
          <serviceMetadatahttpGetEnabled="true"/>
          <serviceDebugincludeExceptionDetailInFaults="true"/>
          <!--
          The serviceCredentials behavior allows one to define a
             service certificate.
          A service certificate is used by a client to
             authenticate the service and provide message protection.
          This configuration references the "localhost"
             certificate installed during the setup instructions.
          -->

          <serviceCredentials>

            <!-- Set allowUntrustedRsaIssuers to true to allow
                  self-signed, asymmetric key based SAML tokens -->

            <issuedTokenAuthenticationcertificateValidationMode="PeerTrust"
                  audienceUriMode="Never" allowUntrustedRsaIssuers="true">
              <allowedAudienceUris>
                <add allowedAudienceUri="IoTService.svc"/>
              </allowedAudienceUris>
              <!-- Add Alice to the list of certs trusted to issue SAML tokens
-->

              <knownCertificates>
                <add storeLocation="LocalMachine" storeName="TrustedPeople"
                         x509FindType="FindBySubjectName" findValue="Alice"/>

              </knownCertificates>
            </issuedTokenAuthentication>
            <serviceCertificatestoreLocation="LocalMachine" storeName="My"
                  x509FindType="FindByThumbprint"
                  findValue="a43c88efc1616d5591f8ec65b4bf94aa0d38842a"/>
          </serviceCredentials>

</behavior>
```

A.5. Calling the IIS Web Server from the Java

To have the interoperability between Java and the .NET servers, we directed the call from the Tomcat server to the IIS server. the communication is achieved using JSON objects. The end URL will be different for LAN and the Cloud configurations.

```

public class NetworkService {
    public static void CallIotService(String action, int simulation){
        //http://localhost:55037/IoTRestService/
            {DEVICEID}/{ACTION}/{USERID}/{OTP}
                 try{
            String actionUpdate="on";
            for(inti=1;i<=simulation;i++){
              HttpClient client = HttpClients.createDefault();
             HttpPost post = new HttpPost("http://iotintservice.
                   cloudapp.net/IoTEvent/IoTRestService/Invoke");
             //HttpPost post = new HttpPost("http://localhost/
                   Event/IoTRestService/Invoke");

              String deviceID="1";
              String userID="1";
              String otp="01234567-89ab-cdef-0123-456789abcdef";
              String iotIP="127.0.0.1:8080";
              StringBuildersb = new StringBuilder();
              sb.append("{\"deviceID\":\"").append(iotIP).append("\",")
              .append("\"userID\":\"").append(userID).append("\"}")
              .append("\"action\":\"").append(actionUpdate).append("\"}")
              .append("\"IoTIP\":\"").append(iotIP).append("\"}")
              .append("\"otp\":\"").append(otp).append("\"}");


              StringEntity input = new StringEntity(sb.toString());
              input.setContentType("application/json");
              post.setHeader("Content-Type", "application/json");
              post.setHeader("Accept", "application/json");
              post.setEntity(input);
              HttpResponse response = client.execute(post);
              BufferedReaderrd = new BufferedReader(new InputStreamReader
                    (response.getEntity().getContent()));
              String line = "";
              while ((line = rd.readLine()) != null) {
               System.out.println(line);
              }
                  if(actionUpdate=="on")
                  {
                      actionUpdate="off";
                  }else{
                      actionUpdate="on";
                  }
            }
        }
```

The following code snippet shows a part of Device Management IoT service in which the new device in added to the network. The details of the device is stored in to the MongoDB.

```
public void AddDevice(Device device)
     {
            try
            {
     MongoDatabasedb = null;
            MongoClientmongoClient = new MongoClient("localhost", 27017);
     db = mongoClient.getDatabase("deviceiot");
     MongoCollection<Document>coll = db.getCollection("mydevices");

            //Code to simulate the multiple calls :UNCOMMENT THIS SECTION FOR
SIMULATION
            /* for(int port=8080; port<8090;port++){
              String simulateDevice="127.0.0.1:"+port;
            coll.insertOne(new Document().append("name", device.getName()).
     append("id", port).
                       //append("type", device.getType()).

     append("ip",simulateDevice ).

                       //append("status", device.getStatus()));
     append("location", "HOME"));
     System.out.println("Wemo Simulated endpoint created IP-"+ simulateDevice
+"/ConnectWemo");
                        } */


            //Code to add a device   :UNCOMMENT THIS SECTION FOR ACTUAL APPLICATION

            coll.insertOne(new Document().append("name", device.getName()).
            append("id", device.getId()).
                append("ip", device.getIp()).
               append("location", device.getLocation()));

            System.out.println("new added device-"+ device.getName()  +
device.getId() +device.getIp() +device.getLocation());



         mongoClient.close();
            }
            catch(Exception e)
            {
                   e.printStackTrace();
            }
```

A.6. Sample Claim in XML Format

The SAML token are in XML formats, used for authentication and authorization purposes. It consists of a SOAP message body which is encrypted and signed to secure the integrity of the sender. The token also has assertions, protocols and bindings to define its security features. The whole message is contained in an outer envelop which has to be decrypted by the reciever to retrive the message.

```
<s:Envelope xmlns:s="http://www.w3.org/2003/05/soap-envelope">
<s:Header>
<a:Action s:mustUnderstand="1" u:Id="_5" xmlns:u="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
xmlns:a="http://www.w3.org/2005/08/addressing">http://schemas.xmlsoap.org/ws/2005/02/trust
/RST/SCT</a:Action>
<a:MessageID u:Id="_6" xmlns:u="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-utility-1.0.xsd"
xmlns:a="http://www.w3.org/2005/08/addressing">urn:uuid:2fa96b4c-3f5a-4364-960e-
0158b4e837f0</a:MessageID>
<ActivityId CorrelationId="49de3ccc-0504-4664-b539-67b1cdc5f9f7"
xmlns="http://schemas.microsoft.com/2004/09/ServiceModel/Diagnostics">2c44dfdb-b4c1-41f1-
a639-7bf72df95af6</ActivityId>
<a:ReplyTo u:Id="_7" xmlns:u="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-utility-1.0.xsd" xmlns:a="http://www.w3.org/2005/08/addressing">
<a:Address>http://www.w3.org/2005/08/addressing/anonymous</a:Address>
</a:ReplyTo>
<a:To s:mustUnderstand="1" u:Id="_8" xmlns:u="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
xmlns:a="http://www.w3.org/2005/08/addressing">http://localhost/SamlTokenIISService/IoTSer
vice.svc/calc/symm</a:To>
<o:Security s:mustUnderstand="1" xmlns:o="http://docs.oasis-open.org/wss/2004/01/oasis-
200401-wss-wssecurity-secext-1.0.xsd">
<u:Timestamp u:Id="uuid-08bacf29-2b6b-488f-b870-82497cc9a05d-3"
xmlns:u="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-
1.0.xsd">
<u:Created>2016-02-23T21:32:58.559Z</u:Created>
<u:Expires>2016-02-23T21:37:58.559Z</u:Expires>
</u:Timestamp>
<e:EncryptedKey Id="uuid-08bacf29-2b6b-488f-b870-82497cc9a05d-2"
xmlns:e="http://www.w3.org/2001/04/xmlenc#">
<e:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p">
<DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"
xmlns="http://www.w3.org/2000/09/xmldsig#"></DigestMethod>
</e:EncryptionMethod>
<KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
<o:SecurityTokenReference>
<o:KeyIdentifier ValueType="http://docs.oasis-open.org/wss/oasis-wss-soap-message-
security-1.1#ThumbprintSHA1" EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-
200401-wss-soap-message-security-
1.0#Base64Binary">pDyI78FhbVWR+Ox1tL+Uqg04hCo=</o:KeyIdentifier>
</o:SecurityTokenReference>
</KeyInfo>
<e:CipherData>
<e:CipherValue>gqPrbwsnPCrRvw2THppiKDuVOUbq9iQoi4TZRTliqd+FcNUOTzB/NuKdj92cP6DVYg+b0bjIMmo
q7pyqwhsF4OKZnVCqA8RQOZFR4XMIaenfIaT2xd4t1a2kF9djcvt/FZwDOJKQxCFoGMh7bQ36Hwk73pRw5xxPqAbiF
E++E0XVk4UNAw/lEnB7OJq4GTcuSvfb5Og5N6r8c4pGCotDBs6Bb+Bn8G2OK6GAc3ymIQ17Us/05pkRmJP7JfLVy3B
U+E4sXpHKDhf1NSLhsdqa+BWhU9knaPGZca9j6SVxLhLF4qqNcBYyN9NMxV8T0JUThiA72UysOj3Bdetj1WSG+Q==<
/e:CipherValue>
</e:CipherData>
```

```
</e:EncryptedKey>
<c:DerivedKeyToken u:Id="_0" xmlns:c="http://schemas.xmlsoap.org/ws/2005/02/sc"
xmlns:u="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-
1.0.xsd">
<o:SecurityTokenReference k:TokenType="http://docs.oasis-open.org/wss/oasis-wss-soap-
message-security-1.1#EncryptedKey" xmlns:k="http://docs.oasis-open.org/wss/oasis-wss-
wssecurity-secext-1.1.xsd">
<o:Reference ValueType="http://docs.oasis-open.org/wss/oasis-wss-soap-message-security-
1.1#EncryptedKey" URI="#uuid-08bacf29-2b6b-488f-b870-82497cc9a05d-2"></o:Reference>
</o:SecurityTokenReference>
<c:Offset>0</c:Offset>
<c:Length>24</c:Length>
<c:Nonce>
<!-- Removed-->
</c:Nonce>
</c:DerivedKeyToken>
<c:DerivedKeyToken u:Id="_2" xmlns:c="http://schemas.xmlsoap.org/ws/2005/02/sc"
xmlns:u="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-
1.0.xsd">
<o:SecurityTokenReference k:TokenType="http://docs.oasis-open.org/wss/oasis-wss-soap-
message-security-1.1#EncryptedKey" xmlns:k="http://docs.oasis-open.org/wss/oasis-wss-
wssecurity-secext-1.1.xsd">
<o:Reference ValueType="http://docs.oasis-open.org/wss/oasis-wss-soap-message-security-
1.1#EncryptedKey" URI="#uuid-08bacf29-2b6b-488f-b870-82497cc9a05d-2"></o:Reference>
</o:SecurityTokenReference>
<c:Nonce>
<!-- Removed-->
</c:Nonce>
</c:DerivedKeyToken>
<e:ReferenceList xmlns:e="http://www.w3.org/2001/04/xmlenc#">
<e:DataReference URI="#_4"></e:DataReference>
<e:DataReference URI="#_10"></e:DataReference>
<e:DataReference URI="#_11"></e:DataReference>
</e:ReferenceList>
<saml:Assertion MajorVersion="1" MinorVersion="1" AssertionID="_3c69ff08-3284-44e0-aa2d-
b91c282a2ae0" Issuer="Self" IssueInstant="2016-02-23T21:32:56.660Z"
xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion">
<saml:Conditions NotBefore="2016-02-23T21:32:56.660Z" NotOnOrAfter="2016-02-
24T07:32:56.660Z">
<saml:AudienceRestrictionCondition>
<saml:Audience>http://localhost:8000/servicemodelsamples/service/calc/symm</saml:Audience>
<saml:Audience>http://localhost:8000/servicemodelsamples/service/calc/asymm</saml:Audience
>
</saml:AudienceRestrictionCondition>
</saml:Conditions>
<saml:Advice></saml:Advice>
<saml:AttributeStatement>
<saml:Subject>
<saml:NameIdentifier>
<!-- Removed-->
</saml:NameIdentifier>
<saml:SubjectConfirmation>
<saml:ConfirmationMethod>urn:oasis:names:tc:SAML:1.0:cm:holder-of-
key</saml:ConfirmationMethod>
<KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
<e:EncryptedKey xmlns:e="http://www.w3.org/2001/04/xmlenc#">
<e:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p">
<DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"></DigestMethod>
</e:EncryptionMethod>
<KeyInfo>
<o:SecurityTokenReference>
<o:KeyIdentifier ValueType="http://docs.oasis-open.org/wss/oasis-wss-soap-message-
security-1.1#ThumbprintSHA1">pDyI78FhbVWR+OxltL+Uqg04hCo=</o:KeyIdentifier>
</o:SecurityTokenReference>
</KeyInfo>
<e:CipherData>
<e:CipherValue>PQEv8mhGFhtbhRvmLzPlj1M0IAsSEf+/i/wZmFAc9W4L5XIJiFu8HZBhhS+i8H0EKCshbzSZaTG
S03h7dnGi9g+2zSdbH/g8PiE+3wDTsC7xzhWo/3OiLhvPi7EqaDvtUDnRphQRTuX/ohRCl44dzEB4nxH948vD0e1vx
ApJhoxARPdKLHsxwlJAlfdlfFumFDE8p5qU/Cgba2T7MnkLNEd5finXS07GUaczcYCTjE/nOysJLSgOwR3OtzrtUjm
PINLblOAZJEe5tLA+11MvcJ+t9N9n3ElwCu/jnsEuQyTq8kn5R47aC/zAz21AMmMUbz1EXOWtTauNbsIzRz0shw==<
/e:CipherValue>
```

```
</e:CipherData>
</e:EncryptedKey>
</KeyInfo>
</saml:SubjectConfirmation>
</saml:Subject>
<saml:Attribute AttributeName="name"
AttributeNamespace="http://schemas.xmlsoap.org/ws/2005/05/identity/claims">
<saml:AttributeValue>
<!-- Removed-->
</saml:AttributeValue>
</saml:Attribute>
</saml:AttributeStatement>
<Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
<SignedInfo>
<CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-
c14n#"></CanonicalizationMethod>
<SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"></SignatureMethod>
<Reference URI="#_3c69ff08-3284-44e0-aa2d-b91c282a2ae0">
<Transforms>
<Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"></Transform>
<Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"></Transform>
</Transforms>
<DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"></DigestMethod>
<DigestValue>583Y/8cTnrnFE17g1iAVFn17F9g=</DigestValue>
</Reference>
</SignedInfo>
<SignatureValue>GQfghQe14FQT7hplHlRMWfO5SdplnqOlBHSZs2VaPGX3MIEOCb9vZiJkfcCn04Lz8w2qonayqu
NOtGs7xYc0/uFYEcPJjktGyA9hteuejyIhqcj5UcNXu27d6a9Fa0Dojf9ZiqYbLGdV7ru1wqr9lgYHObza5n7nG0Is
5ZUgjVPRwWJZJZb+WvwyD9+YOWAuzq9HsvPaeQ4NjqiSF3udUK/uWzWFiTxpy9jYBkyc4T3jnOAuvmrvaHj8eo8yDC
UuVy/iA3IC5/itiVr4UZ/BwmhBei+j5XHNhLZxy1NnT5TO4WrgnavnHx8seJ94cxu0p8+aoBSxzU1W9aGxsZ3u4Q==
</SignatureValue>
<KeyInfo>
<o:SecurityTokenReference>
<o:KeyIdentifier ValueType="http://docs.oasis-open.org/wss/oasis-wss-soap-message-
security-1.1#ThumbprintSHA1">DMERTMckBIxWZR+A9pyNnehhtO0=</o:KeyIdentifier>
</o:SecurityTokenReference>
</KeyInfo>
</Signature>
</saml:Assertion>
<c:DerivedKeyToken u:Id="_9" xmlns:c="http://schemas.xmlsoap.org/ws/2005/02/sc"
xmlns:u="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-
1.0.xsd">
<o:SecurityTokenReference k:TokenType="http://docs.oasis-open.org/wss/oasis-wss-saml-
token-profile-1.1#SAMLV1.1" xmlns:k="http://docs.oasis-open.org/wss/oasis-wss-wssecurity-
secext-1.1.xsd">
<o:KeyIdentifier ValueType="http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-
1.0#SAMLAssertionID">_3c69ff08-3284-44e0-aa2d-b91c282a2ae0</o:KeyIdentifier>
</o:SecurityTokenReference>
<c:Offset>0</c:Offset>
<c:Length>24</c:Length>
<c:Nonce>
<!-- Removed-->
</c:Nonce>
</c:DerivedKeyToken>
<e:EncryptedData Id="_10" Type="http://www.w3.org/2001/04/xmlenc#Element"
xmlns:e="http://www.w3.org/2001/04/xmlenc#">
<e:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#aes256-
cbc"></e:EncryptionMethod>
<KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
<o:SecurityTokenReference>
<o:Reference ValueType="http://schemas.xmlsoap.org/ws/2005/02/sc/dk"
URI="#_2"></o:Reference>
</o:SecurityTokenReference>
</KeyInfo>
<e:CipherData>
<e:CipherValue>7UgU45U+uNzyf8o5xUR9m0iBb4aoJv9L9MRUgjknbz3m33bx9k8Ite6LPUFfPfHnsFJWjAihObp
iuBCSYxiMDw2h7BKMnWs8/NthfZhG4Trm5S0ewK4SEBQuQtG+GNsFBLigg1eJ8mm+M2YJ4ZVO7FrSLhAmNgrHZ4iHE
1Yc2vlvZfHuTFu8nX5wzBKSDWve/r++176BX8ryUmOQCJ6orP0jTMg/4v/PgMdDPrFX91Vkj14IdsUwzulmmxtYAvS
Y8t8uAKEA05XlAOo7OGaNlhBqd7Mdg1kb/12YnHsg+7G59W2+ehBUel1TMKpcfsI1+wNL7b2sO6XkKrx3eww4Ex5GG
JKiWlPrKE94IuTOmpWwodOaH6assg+t2NcN7ZBZjvPcVFYGYCvV3nnyQi+nIaa0Mq/P9BU+8nRuYhirFF2wNeE1NjZ
k1gA1IfrkUWc7zouYmYpDLKdySA+YVDvLgkaeDdN0+LZVN/SN9pEiSJ2wCq2h6cZTMO7jSjJokmUpxOTdYykq2jcEC
```

```
hnSu37WW9X85xGvsB/PDFR4a2QDNWJpoUMOzKtAHCn1TBy+o/qsCejErq0g6liC4BQgpXVd6ZhfEhGxkxaBnzCyHph
PgHRn2H6A6/lyah7+3F5jaRHarc5jrJGZaTvUci6CO4m4iaPn165CqprFIIwz6+QZcNqMOkdUTV2A5ybetaEslvkyN
0pEuko/HaoEfUUgsEMfPu/h0iXCjs/QRu2bufHzo5WdSCYucVS5Ox32v3ZZUwTksnr2q81J7x9v2e9PhP2N2x0zr7f
iayiPacczl7iT0i/Q6c1sDofwo3P4iaR5CmHHpMP6l/UIuJBiZ37QdOR6uh4Ic1gzMqLqjWIRvT5SppiUcff1a6J6Z
pHwBGZDK+FrTc9Mm+9p7ugoinGmRC8Zdrvgdl16jQWIABpHwXHTcAynSJLSbZLwS1WyIEqnSAdKGxwfMplPDtp9vjt
srVKwV6gvH2sBKz89pLPwsHot1okyECKczA+9U9Ax2Rs03duwnJvPj5K4AaNOsdG+J36onZq5T+L6+Nmktg15e30ip
TSJJ9R6SKvq1aJHQYueFJ3+VAiDS149DmhE2veuAdBLoJ8xj2aaVqFxAnJaaQ+Tv/ssw6PrS51+/EaJ/eYyXKmxoQd
D2Ox7uIL7t08zwgzK8s8ZegNDbr+LA+yPK1acazuD7k5auoZOyCDyc8MCYJ2CQFhKxgjmYCNC2RKIjHJiMaCb+pcmb
IkbU4C/8xM4ixZADSx1n+2TrUY2ZlPX7aouCZkJoOzVCcIgcv12llXy95m3fdlt47GLJcoZuCLscKq6RHANK8dbXKh
QWco0Ezos1vdpaFuwaYrpD5bqVOeWIiZGQBRjiwxtZapybQcdZAMhR+k0us7pGGN9H1iyZXNW8Die6wOAEQGdpnY0m
KVWrU8u8VMnV47HkwIhRsGOlfRqgXuJdn6JxJ3oB25U9H9CeskvSOBJeUEP3iXoEhPvRsXnONGEuPGm4wfHpfO6IW5
R9btKxvEAz3ssC1yiGcF3ZOQILQoIkQk912evmB9W4zHCP4AUsaxO9Qifj3gLbS6XNfC1RIZvKW0uRuTCf/XUE/ah/
HBJBOzcvKFd+hEwX7oEP4XcsSM/HHDqhNNU2mzJ2Us5w7lNf6Y0SNF+PoGeCOhihIxYkaqKvARUCLR8pIs8lvL1J8n
/Amn7nH77iVjatMZE2+fr0nwY2Nx6N8z0fIWZ6j8yeTHbosaEB7fwDXUz6KxwV2Pqubh8PRQ4Nk7hgfv+FhO3FOWl2
hAuU85aCRtRbbjtkpDfTuL65ddeVJDWwx5yQvceD9IS6TiFb0eM8MokBRNaOdcSCSltcXl57jtdGxwRmnfR/eZOBM2
3dO9OJjTcF9jNwWILC30JCZa98dnkNZJdemrQ2BD4Hb7jxVZj12vovYnQxGxDvL7GmGcSTepmb/ZFxkCOyula0Ih+/
gtWQnPeuVwIccpYCmX/+scBk8Y9z/Y3UuqsdRclN+bYYop2/keTJUZs96SgVaJrlol5fpoy5urwyMxJYQbD86U3TkZ
YEndMasTypi7iysWhdRrT5zKhM4yjSuisxJlB10+AbXwHRJ/EwBGVDRsoK+zHVD4OKTbpf2FPC0zEyJLBfim6TD7fk
LVqNhhK4wB04qfPwSD7Wg8NHX+Kp/U1/YfwnvimglbBADGXKM1pTegtmxd9mvE9xfdIfMfnPPFagz+id89Hy/WAsHQ
11wAvccsyXbhnwZnMm6IfbgeDPcNh+8Eg11nOrDY8E+y1v2h7XAuxIn2AOC+KbCa1alnSs1KREquB2MI7uhNNZPRB7
yd7Ph1XWIjuGrr3x3vBoexgd/UBK/XlnZDPHBwJkaQ9CYkZfbLbhj+DoU5TmHKIiXaQBw8ICSODi/q0uknmzobgcGE
em58oTxD4EspJ4cQtUWBDP+dn/P2Wv2Qy/Ch25yFjp/lSFuWr3bH04QS1JSm0mpIkPi2Asa+/ZH/xIia8aTTFMKJRh
yax/EhXhYguBhBoRwO6DhpfAp70fyN/Kg96yxx8+8jHioMbQCW4YJSfQb3ZMTo7oMYCvF6VPseTiOT8u2kIN1Cixp8
3m476OdirvAbUCZ+Ouw2Kb17dB2StYLwGSygzshgi7EnbAocYUfbCiJjVVsLOR2N4Be4Kwk7RyCyZILvEsX3Wurgdz
gl+mE9O6VjL227fg3VBF78ebSveXCWsnRJR9QtoPywhvq3D5WDI/lDBHDCiTcMGX7sPQ19bXVHj7s3q5tyn1ZQitw=
=</e:CipherValue>
</e:CipherData>
</e:EncryptedData>
<e:EncryptedData Id="_11" Type="http://www.w3.org/2001/04/xmlenc#Element"
xmlns:e="http://www.w3.org/2001/04/xmlenc#">
<e:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#aes256-
cbc"></e:EncryptionMethod>
<KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
<o:SecurityTokenReference>
<o:Reference ValueType="http://schemas.xmlsoap.org/ws/2005/02/sc/dk"
URI="#_2"></o:Reference>
</o:SecurityTokenReference>
</KeyInfo>
<e:CipherData>
<e:CipherValue>aQskguyi2G2vIF/pgWeW/CE4LzaRhylULmf47cfq/BrLSA3JMaAbEO9UB6JImyTTV1px9K6m0Fi
h4aAZNpY/cc3s2Fcekg3FADxYUs4jpP5jK3EMpY9S5jqd5EsxdFFv/nniZJI2IsyMYpjfJuccXyUj5nuYShFwFSNfi
EsNoGFAUZsF+BwXmM10UyENmtCRN5nWCw/iq7gLjR+MiVVnzs286NdAsOf1MypWRQwkfbuCZt7lohMgD/pYLVUhGm7
TOxB2b4yWc6F5338Fg00gzpqGlISb9az6w4AsqEl8Xa4hkLAuN6H621IwjA0Na7/vBuymIpC1KP3AcIuYtosNcxJJ2
Z9MbGDyPy15aDOkXfdDokgJp83NUjubLSL3jIImbXeM1d42Ormg4sf8mpkBZJi/JkoK2KuJcBdghz6rmMqONK18KdT
mZllNzoelmJ4RYVlW8nFdEnziv5B+ffhVcLVR4UQYDfgeMyJX8qsNuxQHct5WUvxE0+zPc9eWhFMnibbRofcZo7JbI
2+58M/BBZjM4skMRotSIMUufnNIHH5hpRmiYTdK8jlag1j/uJNJBkc+fMWRFWw+yXeJYIrzjiRHBOLJmJP+AEjnZip
gqwCgMhxTuxjT9BfEajqwOCWC6Wi+CLJh6+EppqGeiJeGgcf08y72qtJNMguMr+cWHDDE7Ew8+ekr8GwJr0AkUEFny
1MdIyQKvqjqloAUfFyG2ctNlSojGY2gYYhiyGKGNr0T8L82otmsLVXRd1KeG84jY/wOWEFKjZxIxFxselxHik2rKW1
7VukI7BZO6fnbGtmOiqyf4MFWKQzh5+QzzrKN63nRRTOXKbTRQuwkdWbCNLv860WEW9VsyMcF3COFT1TajMh0BGwsP
u6y9ETmknY+Df+bd7Tkqg6di0Vb+fEB1a8C+bfXx/ykMYCMCbXVBEMyyx1rWApRc2CX4ZsZe2hNun7fsXun5TZdYwt
vZ8gqtaJ0MWuSu5Yf81wfYOiIj6Ind3QkOwyNy9Vog8jaPcZgOJBBQX+VmYazVifCmHHIuMDCusRpIjuxaYHumKgSn
lqGduXV96fMybvP212ZENBz</e:CipherValue>
</e:CipherData>
</e:EncryptedData>
</o:Security>
</s:Header>
</s:Envelope>
```

Appendix B

Log Files

B1. Sample Log files

The log files that contains the timestamp of each iterations performed in the evaluation phase is given below. The log files shows the timestamp values of the action for 10 requets when we used the WeMo device for case study in LAN.

Claims Middleware Log for 10 requests:

```
2016-02-24 08:56:04.5473|INFO|Device ON: User authenticated to
contact Wemo device 192.168.100.101:49153
2016-02-24 08:56:05.3067|INFO|Device OFF: SUCCESS. User
authenticated for Wemo device 192.168.100.101:49153
2016-02-24 08:56:06.1409|INFO|Device ON: User authenticated to
contact Wemo device 192.168.100.101:49153
2016-02-24 08:56:06.9276|INFO|Device OFF: SUCCESS. User
authenticated for Wemo device 192.168.100.101:49153
2016-02-24 08:56:07.7432|INFO|Device ON: User authenticated to
contact Wemo device 192.168.100.101:49153
2016-02-24 08:56:08.5525|INFO|Device OFF: SUCCESS. User
authenticated for Wemo device 192.168.100.101:49153
2016-02-24 08:56:09.3916|INFO|Device ON: User authenticated to
contact Wemo device 192.168.100.101:49153
2016-02-24 08:56:10.1547|INFO|Device OFF: SUCCESS. User
authenticated for Wemo device 192.168.100.101:49153
2016-02-24 08:56:11.0039|INFO|Device ON: User authenticated to
contact Wemo device 192.168.100.101:49153
2016-02-24 08:56:11.8147|INFO|Device OFF: SUCCESS. User
authenticated for Wemo device 192.168.100.101:49153
```

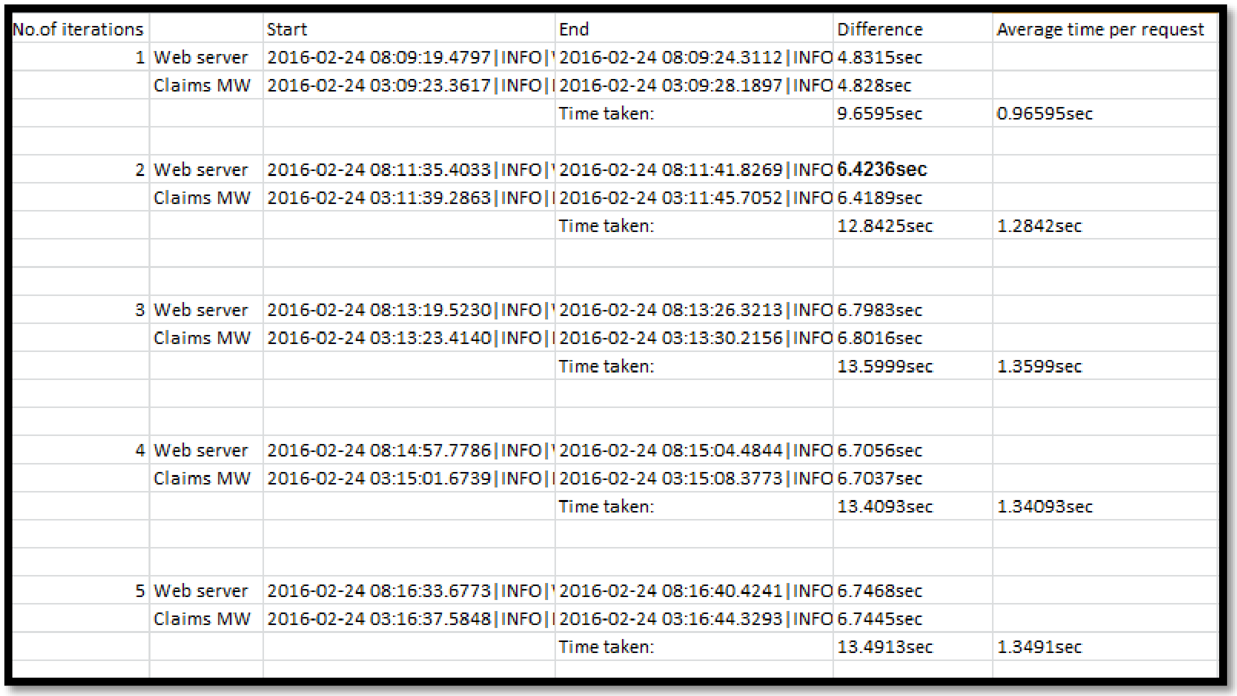
Intermediary Web Service Log for 10 requests:

```
2016-02-24 09:55:59.7721|INFO|Wemo device: 192.168.100.101:49153
turned ON.
2016-02-24 09:56:00.5405|INFO|Wemo device: 192.168.100.101:49153
turned OFF.
2016-02-24 09:56:01.3781|INFO|Wemo device: 192.168.100.101:49153
turned ON.
2016-02-24 09:56:02.1548|INFO|Wemo device: 192.168.100.101:49153
turned OFF.
```

```
2016-02-24 09:56:02.9740|INFO|Wemo device: 192.168.100.101:49153
turned ON.
2016-02-24 09:56:03.8083|INFO|Wemo device: 192.168.100.101:49153
turned OFF.
2016-02-24 09:56:04.6199|INFO|Wemo device: 192.168.100.101:49153
turned ON.
2016-02-24 09:56:05.3835|INFO|Wemo device: 192.168.100.101:49153
turned OFF.
2016-02-24 09:56:06.2382|INFO|Wemo device: 192.168.100.101:49153
turned ON.
2016-02-24 09:56:07.0392|INFO|Wemo device: 192.168.100.101:49153
turned OFF.
```

B.2. Calculation of Performance Time

The calculation details of the time taken for both the intermediary server and the Claims middleware is given below.



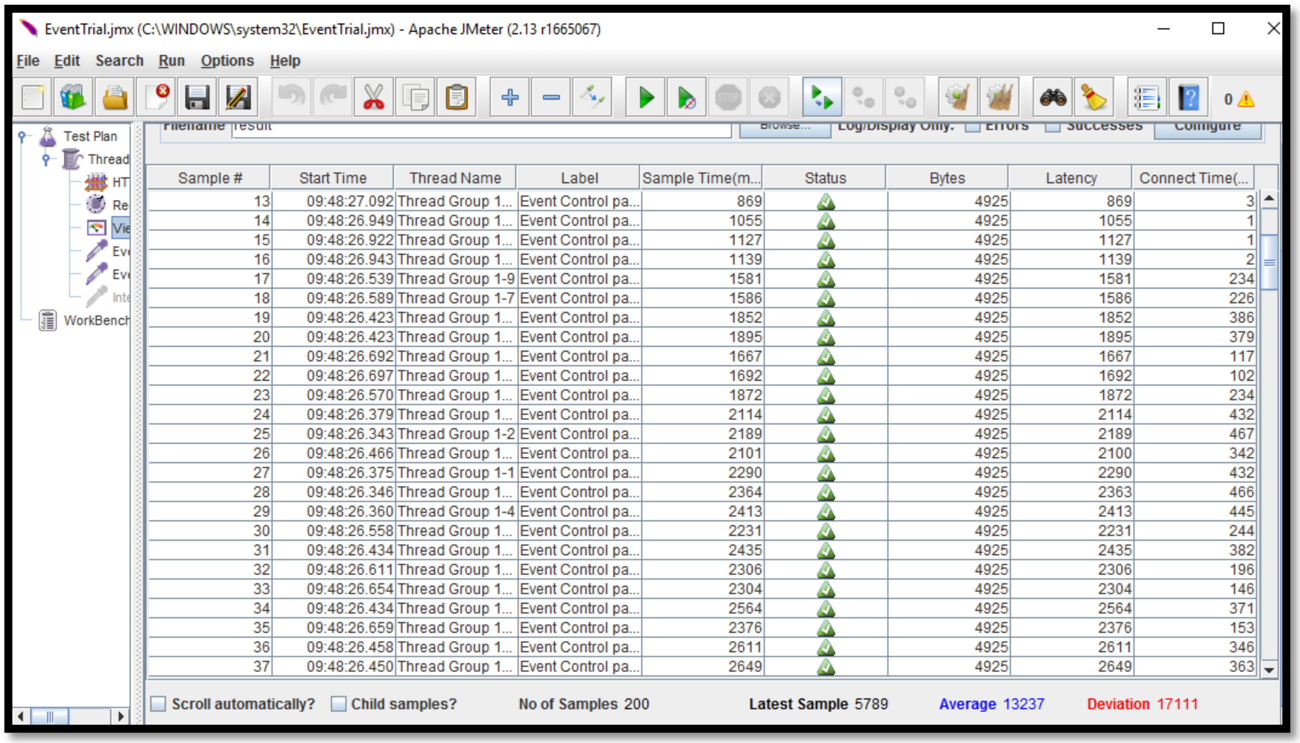
No.of iterations		Start	End	Difference	Average time per request
1	Web server	2016-02-24 08:09:19.4797|INFO|'	2016-02-24 08:09:24.3112|INFO	4.8315sec	
	Claims MW	2016-02-24 03:09:23.3617|INFO|I	2016-02-24 03:09:28.1897|INFO	4.828sec	
			Time taken:	9.6595sec	0.96595sec
2	Web server	2016-02-24 08:11:35.4033|INFO|'	2016-02-24 08:11:41.8269|INFO	**6.4236sec**	
	Claims MW	2016-02-24 03:11:39.2863|INFO|I	2016-02-24 03:11:45.7052|INFO	6.4189sec	
			Time taken:	12.8425sec	1.2842sec
3	Web server	2016-02-24 08:13:19.5230|INFO|'	2016-02-24 08:13:26.3213|INFO	6.7983sec	
	Claims MW	2016-02-24 03:13:23.4140|INFO|I	2016-02-24 03:13:30.2156|INFO	6.8016sec	
			Time taken:	13.5999sec	1.3599sec
4	Web server	2016-02-24 08:14:57.7786|INFO|'	2016-02-24 08:15:04.4844|INFO	6.7056sec	
	Claims MW	2016-02-24 03:15:01.6739|INFO|I	2016-02-24 03:15:08.3773|INFO	6.7037sec	
			Time taken:	13.4093sec	1.34093sec
5	Web server	2016-02-24 08:16:33.6773|INFO|'	2016-02-24 08:16:40.4241|INFO	6.7468sec	
	Claims MW	2016-02-24 03:16:37.5848|INFO|I	2016-02-24 03:16:44.3293|INFO	6.7445sec	
			Time taken:	13.4913sec	1.3491sec

Figure B.1: Excel sheet containing the calculation details

The total time for each service is calculated by substracting the start time from the end time. The total time to process the whole requests is calculated by adding each server’s processing time.

B.3. JMeter generating Requests

The requests are made to the client using the JMeter tool. It has the provison to change the number of users and the requests per user. This tool helped us make multiple calls to the server.



EventTrial.jmx (C:\WINDOWS\system32\EventTrial.jmx) - Apache JMeter (2.13 r1665067)

Sample #	Start Time	Thread Name	Label	Sample Time(m...	Status	Bytes	Latency	Connect Time(...
13	09:48:27.092	Thread Group 1...	Event Control pa...	869	✓	4925	869	3
14	09:48:26.949	Thread Group 1...	Event Control pa...	1055	✓	4925	1055	1
15	09:48:26.922	Thread Group 1...	Event Control pa...	1127	✓	4925	1127	1
16	09:48:26.943	Thread Group 1...	Event Control pa...	1139	✓	4925	1139	2
17	09:48:26.539	Thread Group 1-9	Event Control pa...	1581	✓	4925	1581	234
18	09:48:26.589	Thread Group 1-7	Event Control pa...	1586	✓	4925	1586	226
19	09:48:26.423	Thread Group 1...	Event Control pa...	1852	✓	4925	1852	386
20	09:48:26.423	Thread Group 1...	Event Control pa...	1895	✓	4925	1895	379
21	09:48:26.692	Thread Group 1...	Event Control pa...	1667	✓	4925	1667	117
22	09:48:26.697	Thread Group 1...	Event Control pa...	1692	✓	4925	1692	102
23	09:48:26.570	Thread Group 1...	Event Control pa...	1872	✓	4925	1872	234
24	09:48:26.379	Thread Group 1...	Event Control pa...	2114	✓	4925	2114	432
25	09:48:26.343	Thread Group 1-2	Event Control pa...	2189	✓	4925	2189	467
26	09:48:26.466	Thread Group 1...	Event Control pa...	2101	✓	4925	2100	342
27	09:48:26.375	Thread Group 1-1	Event Control pa...	2290	✓	4925	2290	432
28	09:48:26.346	Thread Group 1...	Event Control pa...	2364	✓	4925	2363	466
29	09:48:26.360	Thread Group 1-4	Event Control pa...	2413	✓	4925	2413	445
30	09:48:26.558	Thread Group 1...	Event Control pa...	2231	✓	4925	2231	244
31	09:48:26.434	Thread Group 1...	Event Control pa...	2435	✓	4925	2435	382
32	09:48:26.611	Thread Group 1...	Event Control pa...	2306	✓	4925	2306	196
33	09:48:26.654	Thread Group 1...	Event Control pa...	2304	✓	4925	2304	146
34	09:48:26.434	Thread Group 1...	Event Control pa...	2564	✓	4925	2564	371
35	09:48:26.659	Thread Group 1...	Event Control pa...	2376	✓	4925	2376	153
36	09:48:26.458	Thread Group 1...	Event Control pa...	2611	✓	4925	2611	346
37	09:48:26.450	Thread Group 1...	Event Control pa...	2649	✓	4925	2649	363

Scroll automatically? Child samples? No of Samples 200 Latest Sample 5789 Average 13237 Deviation 17111

Figure B.2. JMeter tool used for simulation

Bibliography

[1] Vermesan, O., & Friess, P. (Eds.). (2013). *Internet of things: converging technologies for smart environments and integrated ecosystems*. River Publishers.

[2] Vermesan, O., & Friess, P. (Eds.). (2014). *Internet of Things-From Research and Innovation to Market Deployment* (pp. 74-75). River Publishers.

[3] Lefort, L., Henson, C., Taylor, K., Barnaghi, P., Compton, M., Corcho, O., ... & Page, K. (2011). Semantic sensor network xg final report. *W3C Incubator Group Report*, *28*.

[4] Gubbi, J., Buyya, R., Marusic, S., & Palaniswami, M. (2013). Internet of Things (IoT): A vision, architectural elements, and future directions. *Future Generation Computer Systems, 29*(7), (pp. 1645-1660).

[5] Shipley, A. J. (2013). Security in the internet of things, lessons from the past for the connected future. *Security Solutions, Wind River, White Paper*, Retrieved April 13, 2016, from http://www.intel.com/content/dam/www/public/us/en/documents/white-papers/wind-river-security-in-the-internet-of-things.pdf.

[6] Claims-Aware Applications. Retrieved September 27, 2015, from https://msdn.microsoft.com/en us/library/windows/desktop/bb736227(v=vs.85).aspx.

[7] What Is Windows Communication Foundation. Retrieved July 15, 2015, from https://msdn.microsoft.com/en-us/library/ms731082(v=vs.110).aspx.

[8] Enterprise service bus. (2016, March 7). In Wikipedia, The Free Encyclopedia. Retrieved April 13, 2016, from https://en.wikipedia.org/w/index.php?title=Enterprise_service_bus&oldid=708708094.

[9] Bandyopadhyay, S., Sengupta, M., Maiti, S., & Dutta, S. (2011). Role of middleware for internet of things: A study. *International Journal of Computer Science & Engineering Survey (IJCSES)*, *2*(3), (pp. 94-105).

[10] Joe Folkens, (2014, December). Building a gateway to the Internet of Things. Texas Instruments, White Paper, Retrieved October 18, 2015, from http://www.ti.com/lit/wp/spmy013/spmy013.pdf.

[11] Collina, M., Corazza, G. E., & Vanelli-Coralli, A. (2012, September). Introducing the QEST broker: Scaling the IoT by bridging MQTT and REST. In *Personal Indoor and Mobile Radio Communications (PIMRC), 2012 IEEE 23rd International Symposium on* (pp. 36-41). IEEE.

[12] De Caro, N., Colitti, W., Steenhaut, K., Mangino, G., & Reali, G. (2013, November). Comparison of two lightweight protocols for smartphone-based sensing. In *Communications and Vehicular Technology in the Benelux (SCVT), 2013 IEEE 20th Symposium on* (pp. 1-6). IEEE.

[13] Eisenhauer, M., Rosengren, P., & Antolin, P. (2010). Hydra: A development platform for integrating wireless devices and sensors into ambient intelligence systems. In *The Internet of Things* (pp. 367-373). Springer New York.

[14] Katasonov, A., Kaykova, O., Khriyenko, O., Nikitin, S., & Terziyan, V. Y. (2008). Smart Semantic Middleware for the Internet of Things. *ICINCO-ICSO,8*, (pp. 169-178).

[15] Bazzani, M., Conzon, D., Scalera, A., Spirito, M., & Trainito, C. I. (2012, June). Enabling the IoT paradigm in e-health solutions through the VIRTUS middleware. In *Trust, Security and Privacy in Computing and Communications (TrustCom), 2012*

IEEE 11th International Conference on (pp. 1954-1959). IEEE. Stankovic, J. (2014). Research directions for the internet of things. *Internet of Things Journal*, *IEEE*, *1*(1), (pp. 3-9).

[16] Chen, Y. K. (2012, January). Challenges and opportunities of internet of things. In *Design Automation Conference (ASP-DAC), 2012 17th Asia and South Pacific*, *IEEE*, (pp. 383-388).

[17] Ortiz, A. M., Hussein, D., Park, S., Han, S. N., & Crespi, N. (2014). The cluster between internet of things and social networks: Review and research challenges. *Internet of Things Journal*, *IEEE*, *1*(3), (pp. 206-215).

[18] DaCosta, F. (2013). *Rethinking the Internet of Things: a scalable approach to connecting everything*. Apress, Retrieved November 08, 2015, from http://dl.acm.org/citation.cfm?id=2578967, ACM Digital Library

[19] Blackberry IoT Platform. Retrieved September 10, 2015, from http://ca.blackberry.com/internet-of-things.html.

[20] Amrita Internet of Things Middleware. Retrieved September 10, 2015, from http://www.aiotm.in/atotmmiddleware.html.

[21] MuleSoft AnyPoint Platform. Retrieved November 18, 2015, from https://www.mulesoft.com/integration-solutions/api/iot.

[22] Salem Hadim and Nader Mohamed, "Middleware Challenges and Approaches for Wireless Sensor Networks," *IEEE Distributed Systems Online*, vol. 7, no. 3, 2006, art. no. 0603-o3001.

[23] Noergaard, T. (2012). *Embedded systems architecture: a comprehensive guide for engineers and programmers*. Newnes. Retrieved April 10, 2016 from http://www.eetimes.com/document.asp?doc_id=1276764.

[24] Bernstein, P. A. (1996). Middleware: a model for distributed system services. *Communications of the ACM*, *39*(2), (pp. 86-98).

[25] Privat, G., Zhao, M., & Lemke, L. (2014, April). Towards a Shared Software Infrastructure for Smart Homes, Smart Buildings and Smart Cities. In*International Workshop on Emerging Trends in the Engineering of Cyber-Physical Systems, Berlin*.

[26] Gu, T., Pung, H. K., & Zhang, D. Q. (2005). A service-oriented middleware for building context-aware services. *Journal of Network and computer applications*, *28*(1), (pp. 1-18).

[27] Roman, R., Najera, P., & Lopez, J. (2011). Securing the Internet of Things.*Computer*, IEEE Computer Society, *44*(9), (pp. 51-58). IEEE.

[28] Security Assertion Markup Language. (2016, March 23). In *Wikipedia, The Free Encyclopedia*. Retrieved 13:32, April 13, 2016, from https://en.wikipedia.org/w/index.php?title=Security_Assertion_Markup_Language&oldid=711590214.

[29] SAML (Security Assertion Markup Language).(2008, January) Retrieved April 10, 2016, from http://searchfinancialsecurity.techtarget.com/definition/SAML.

[30] SAML Token Provider. Retrieved December 5, 2015, from https://msdn.microsoft.com/en-us/library/aa355062(v=vs.110).aspx.

[31] Lewis J and Fowler M. (2014, March 25). Microservices. Retrieved October 2, 2015, from http://martinfowler.com/articles/microservices.html.

[32] Richardson C. (2014). Microservice Architecture. Retrived Decmber 15, 2015, from http://microservices.io.

[33] Li, F., Vögler, M., Claeßens, M., & Dustdar, S. (2013, June). Efficient and scalable IoT service delivery on Cloud. In *Cloud Computing (CLOUD), 2013 IEEE Sixth International Conference on* (pp. 740-747). IEEE.

[34] Thones, J. (2015). Microservices. *Software, IEEE*, *32*(1), (pp. 116-116).

[35] Stafford G. A. (2015, May 18). Building a Microservices based REST API with RestExpress, JavaEE and MongoDB: Part 1. Retrieved October 13, 2015, from https://programmaticponderings.wordpress.com/2015/05/18/building-a-microservices-based-rest-api-with-restexpress-java-and-mongodb-part-1.

[36] Woods D. (2015, February 10). Building Microservices with Spring Boot. Retrieved October 13, 2015, from http://www.infoq.com/articles/boot-microservices.

[37] Issac L. P. (2014, January 14). SQL vs NoSQL Database Differences Explained with few Example DB. Retrieved October 24, 2015, from http://www.thegeekstuff.com/2014/01/sql-vs-nosql-db.

[38] Gorst D. (2014, August 4). MongoDb vs CouchDB. RetrievedOctober12, 2015, from http://blog.scottlogic.com/2014/08/04/mongodb-vs-couchdb.html.

[39] Wei-ping, Z., Ming-Xin, L., & Huan, C. (2011, May). Using MongoDB to implement textbook management system instead of MySQL. In*Communication Software and Networks (ICCSN), 2011 IEEE 3rd International Conference on* (pp. 303-305). IEEE.

[40] MongoDB. (2013). Retrieved November 25, 2015, from https://www.mongodb.org/about/introduction.

[41] Han, J., Haihong, E., Le, G., & Du, J. (2011, October). Survey on NoSQL database. In *Pervasive computing and applications (ICPCA), 2011 6th international conference on* (pp. 363-366). IEEE.

[42] SAML Tokens and Claims. Retrieved December 5, 2015, from https://msdn.microsoft.com/en-us/library/ms733083(v=vs.110).aspx.

[43] Kim, D. S., Lee, J. M., Kwon, W. H., & Yuh, I. K. (2002). Design and implementation of home network systems using UPnP middleware for networked appliances. *IEEE Transactions on Consumer Electronics*, *48*(4), (pp. 963-972).

[44] Anatomy of a SAML-Secured SOAP Message. Retrievd April 10, 2016, from https://blogs.oracle.com/superpat/entry/anatomy_of_a_saml_secured.

[45] Fundamentals of WCF Security. Retrieved April 11, 2016, from http://www.codemag.com/article/0611051.

[46] Types of Middleware. Retrieved January 12, 2016, from http://apprenda.com/library/architecture/types-of-middleware/.

[47] Jing, Y., Lan, Z., Hongyuan, W., Yuqiang, S., & Guizhen, C. (2010, August). JMeter-based aging simulation of computing system. In *Computer, Mechatronics, Control and Electronic Engineering (CMCE), 2010 International Conference on* (Vol. 5, pp. 282-285). IEEE.

[48] Zhang, L. J., & Zhou, Q. (2009, July). CCOA: Cloud computing open architecture. In *Web Services, 2009. ICWS 2009. IEEE International Conference on* (pp. 607-616). IEEE.

[49] Lar, S. U., Liao, X., & Abbas, S. A. (2011, August). Cloud computing privacy & security global issues, challenges, & mechanisms. In *Communications and*

Networking in China (CHINACOM), 2011 6th International ICST Conference on (pp. 1240-1245). IEEE.

[50] Kaufman, L. M. (2009). Data security in the world of cloud computing.*Security & Privacy, IEEE, 7*(4), (pp. 61-64).

[51] Bupe, P., Haddad, R., & Rios-Gutierrez, F. (2015, April). Relief and emergency communication network based on an autonomous decentralized UAV clustering network. In *SoutheastCon 2015* (pp. 1-8). IEEE.

[52] Copeland, M., Soh, J., Puca, A., Manning, M., & Gollob, D. (2015). Microsoft Azure and Cloud Computing. In *Microsoft Azure* (pp. 3-26). Apress.

[53] Belkin WeMo HOME AUTOMATION. Retrieved, January 24,2016, from http://www.belkin.com/us/Products/home-automation/c/wemo-homeautomation/#whyWemo.

[54] Ur, B., McManus, E., Pak Yong Ho, M., & Littman, M. L. (2014, April). Practical trigger-action programming in the smart home. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 803-812). ACM.

[55] Best Seller Online Store. Retrieved January 25, 2016, from http://compareuyesn.bl.ee/reviews-for-bendable-desk-lamp-gunmetal-with-adjustable-stem-table-lamp/.

[56] D-LINK. SharePort^{TM} Mobile Companion DIR-505. Retrieved January 24, 2016, from http://ca.dlink.com/products/access-points-range-extenders-and-bridges/shareport-mobile-companion-2/.

[57] GitHub. WeMoWsdl. Retrieved December 2, 2015, from https://github.com/sklose/WeMoWsdl.