Authentication and Authorization schemes for Internet of Things

Submitted By

Saurabh Patel

16MCEN14

DEPARTMENT OF INFORMATION TECHNOLOGY

INSTITUTE OF TECHNOLOGY

NIRMA UNIVERSITY

AHMEDABAD-382481

MAY 2018

Authentication and Authorization schemes for Internet of Things

Major Project

Submitted in partial fulfillment of the requirements

for the degree of

Master of Technology in Computer Science and Engineering (Networking Technologies)

Submitted By

Saurabh Patel

(16MCEN14)

Guided By

Prof. Gaurang Raval

DEPARTMENT OF INFORMATION TECHNOLOGY

INSTITUTE OF TECHNOLOGY

NIRMA UNIVERSITY

AHMEDABAD-382481

MAY 2018

Certificate

This is to certify that the major project entitled **"Authentication and Authorization schemes for Internet of Things"** submitted by **Saurabh Patel (16MCEN14)**, towards the partial fulfillment of the requirements for the award of degree of Master of Technology in Computer Science and Engineering (Networking Technologies) of Nirma University, Ahmedabad, is the record of work carried out by him under my supervision and guidance. In my opinion, the submitted work has reached a level required for being accepted for examination. The results embodied in this major project part-I, to the best of my knowledge, haven't been submitted to any other university or institution for award of any degree or diploma.

Dr. Gaurang Raval
Guide & Associate Professor,
Coordinator M.Tech - CSE (Networking Technologies)
Institute of Technology,
Nirma University, Ahmedabad

Dr. Madhuri Bhavsar
Professor and Head,
IT Department,
Institute of Technology,
Nirma University, Ahmedabad.

Dr Alka Mahajan
Director,
Institute of Technology,
Nirma University, Ahmedabad

Statement of Originality

I, **Saurabh Patel** give undertaking that the Major Project entitled "**Authentication and Authorization schemes for Internet of Things**" submitted by me, towards the partial fulfillment of the requirements for the degree of Master of Technology in **Computer Science & Engineering (Networking Technologies)** of Institute of Technology, Nirma University, Ahmedabad, contains no material that has been awarded for any degree or diploma in any university or school in any territory to the best of my knowledge. It is the original work carried out by me and I give assurance that no attempt of plagiarism has been made.It contains no material that is previously published or written, except where reference has been made. I understand that in the event of any similarity found subsequently with any published work or any dissertation work elsewhere; it will result in severe disciplinary action.

_____

Signature of Student

Date:

Place:

Endorsed by

Guide Name

(Signature of Guide)

Acknowledgements

It gives me immense pleasure in expressing thanks and profound gratitude to **Prof. Gaurang Raval**, Associate Professor, Computer Engineering Department, Institute of Technology, Nirma University, Ahmedabad for his valuable guidance and continual encouragement throughout this work. The appreciation and continual support he has imparted has been a great motivation to me in reaching a higher goal. His guidance has triggered and nourished my intellectual maturity that I will benefit from, for a long time to come.

It gives me an immense pleasure to thank **Dr. Madhuri Bhavsar**, Hon'ble Head of Information Technology Department, Institute of Technology, Nirma University, Ahmedabad for his kind support and providing basic infrastructure and healthy research environment.

A special thank you is expressed wholeheartedly to **Dr. Alka Mahajan**, Hon'ble Director, Institute of Technology, Nirma University, Ahmedabad for the unmentionable motivation he has extended throughout course of this work.

I would also thank the Institution, all faculty members of Computer Engineering Department, Nirma University, Ahmedabad for their special attention and suggestions towards the project work.

- Saurabh Patel

16MCEN14

Abstract

Nowadays the Internet of Things is very much in use in the real world which is the network of constrained devices like the sensor. The IoT devices have very limited resources in terms of computational power, memory, energy etc. The IETF (Internet Engineering Task Force) has defined several protocols especially for the resource which have constrained environment like CoAP, RPL, MQTT etc. at different layers. Authentication and Authorization are essential to handle secure communication in IoT environment. OAuth 2.0 Protocol is an open standard protocol which is used to grant access to the restricted resource via third-party authentication for REST Web architecture. There are some challenges which are faced when we deploy Oauth 2.0 Protocol in IoT environment because of constrained resources used in IoT devices. To analyze these challenges we experimented with various OAuth servers and messaging standards which packs and unpacks messages by serialization and deserialization. Different messaging approaches for authorization and authentication for the constrained environment have been analyzed in the context of latency and energy consumption.

Abbreviations

Acronyms	**Full forms**
IoT	Internet of Things
AS	Authentication Server
6LowPAN	IPv6 over Low-power Wireless Personal Area Networks
DLTS	Datagram Transport Layer Security
CoAP	Constrained Application Protocol
MQTT	Message Queuing Telemetry Transport
IETF	Internet Engineering Task Force

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Overview of the Internet of Things (IoT) and OAuth Protocol

OAuth protocol is open standard authorization framework which is allowed to access services by the user via a third-party authentication server.it provides secure delegate access to the resource server on behalf of the Resource Owner. Internet of Things is the network of the smart object and system which is the interconnected real-world sensor and the actuators to the internet.

1.2 Objective of Study

Our objective for fast processing of authentication and authorization related message exchanges. We propose a method to reduce the latency involved and thereby reducing the energy consumption during the message exchange process while user wants to access the restricted data which is generated by the smart IoT devices. We deploy the OAuth Protocol on IoT environment so the user can to access the IoT devices via access token which is generated by the Authentication server.

1.3 Purpose

The main purpose of this research is to provide security to the IoT environment by the authentication framework. There are so many challenges while we deploy the OAuth protocol on the IoT devices because the IoT device has constrained resources.

Internet of things is providing remote access for the IoT devices. Now a day millions of IoT devices (sensor) is connected to IoT devices. Different users want to access the restricted data which is generated by the some of the IoT devices among the whole IoT network. So we need to develop the lightweight authentication mechanism (OAuth protocol) for the IoT environment.

Chapter 2

Literature Survey

2.1 OAuth-IoT

2.1.1 OAuth Protocol

Traditionally, application stores the authentication and authorization information for all users. But there is some problem faced in a traditional approach like security issue regarding multiple credentials with a different application. To solve this problem OAuth Protocol is introduced. OAuth protocol is scalable delegation protocol. OAuth allows to client application restricted access to your data at resource server via access token which is generated by the OAuth server. There are four actor resource owner (user), an authentication server (OAuth), and resource servers and client (Application) act in this protocol. Resource server has store the protected resource (services) which is access by resource owner via access token. Most of the web application use web server flow to obtain a token on behalf of the end user. Fig 3 show sequence diagram of the OAuth Protocol [12].

The user wants to access the services which are resided on resource server from the client application (web browser). Client application wants to access data from the resource server. If the user has not authenticated then Resource server delegate the authentication to the authorization server (Facebook, Google, Twitter). Client application redirects a user to the Authentication server. The user gives the user credential to the authentication server. After successful authentication, the Authorization server sends an authentication code to the user. The user sends an authorization code to the client application which is also sent to the Authentication server to generating the access token. The

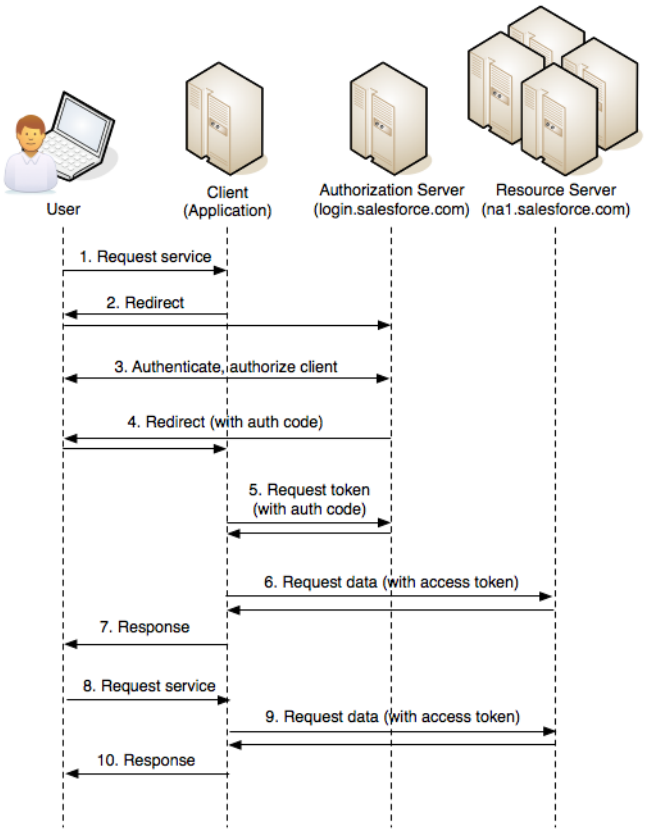

Figure 2.1: Sequence diagram of OAuth Protocol [7]

client application sends a request for access the protected services of the resource server with the access token. Client application stores the access token because whenever a user wants to access the services form resource server in future it can use these access token directly so every time user doesnt give the user credential to the Authentication server[12]. The authorization code is a short-lived token which is generated by authentication server which is sent to the client application via the browser.

The access token is then used by the client application which is generated by the authentication server. Access token defines that the user is valid to allow the restricted services which are on resource server. The access token has some limited lifetime after it automatically expires

The refresh token is the type of access token which has the infinite lifetime. Access token and refresh token store by the client application. When Refresh token is expired within a time which depends on the OAuth server, the Authentication server gives another token to the client application without any authentication [12]. The access token has a parameter like code, grant type, client id, client secret and redirects URL. There are five

different grant types which are different according to the access policy. We set the grant type value as authentication code. The code parameter defines the value which is return by the authentication server in the previous step [12].

2.1.2 IoT-OAS: An OAuth-Based Authorization Service Architecture for Secure Services in IoT

In this paper, Author proposes an architecture which is targeting HTTP/CoAP services to provide the authentication and authorization for access data which is the send by constraining IoT devices. There are four ways they represent the IoT-OAS architecture regarding network coordinator Sink node.

A. Network Broker Communication

Network Broker is just passing the request and response to/from external network to/from IoT Network through a secure channel based on HTTPS or CoAP. Network Broker could be implemented using an embedded device or a device with limited computational and storage capability. In this case, there is extra overhead on IoT devices.

B. Gateway-based Communication

Gateway is introduced in this architecture because of computational overhead of Oauth Protocol on constraints on devices is not significance way. Gateway is translating the incoming requests from the external networks to available Smart Objects inside its own network.

C. End to End CoAP Communication

These architecture proposed with one consideration which is the IoT devices can reachable at an IPv6 address in a sensor network so it provides directly a remote CoAP service. There is one protocol which is defined by IETF is 6LowPAN to deploy IPv6address scheme for IoT devices. The network Gateway acts only as a router without the need to translate incoming and outgoing messages between the external world and the sensor network.

D. hybrid Gateway based CoAP Communication

In this case, the external client uses an HTTP protocol. On another side, the IoT devices use CoAP protocol. The Gateway is translating incoming HTTP request to CoAP for smart devices. The OAuth service is used CoAP channel for the authentication process [8].

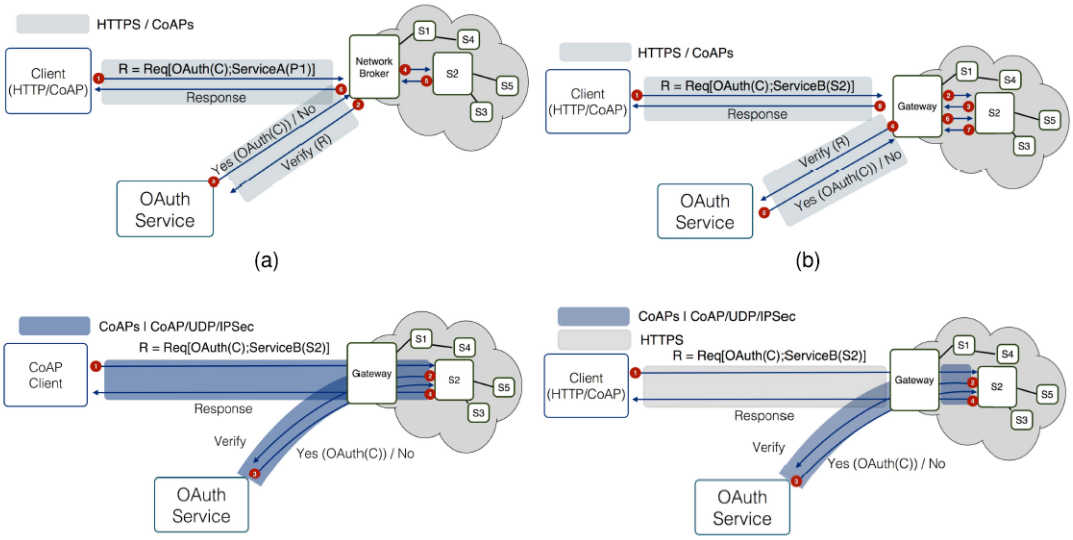

Figure 2.2: IoT-OAS Model [8]

2.1.3 An Implementation of Open Authentication Protocol for IoT Based Application

In this paper, the proposed architecture of Oauth is deploying for fire alarm a system (Device). Whenever fire is sensing the devices it posts to services provider which store authenticate data. The authenticating data is accessible by the service consumer through the access token which is generated by the services provider according to OAuth Protocol. The client application sends the request to the service provider for the authentication code. Service provider gives the valid authentication code after successfully authentication is done with user's credential. The token is generated by the service provider when the authentication code is sent to the service provider. Now the client application accesses the restricted data which reside on services provider. This whole process will show the Oauth service where delegate authority to the application through the access token from the Service Provider. The Author implements these proposed work and measures the latency with fix hardware configuration. It analyzes that the fire alarm system takes about 20 to 30 ms for generating the alarm. On the other hand, the proposed Oauth based notification system takes about 20 to 50 ms time for triggering the alarm [13].

2.1.4 OAuth-IoT: an access control framework for the IoT based on the open standard

In this paper, the author proposed authentication and authorization framework for internet of Things. It modifies the standard OAuth protocol according IoT environment like access token format and some other protocol. It defines three different token formats, first Bearer token which is simply define information according users credential. Second is the JSON Web Token (JWT) which is carrying some other field like time validity, revocation time, issuer, owner and authentication filed and third is the Proof-of-Possession (POP) token, it used when addition security protection is required. Therefor the client sends the token pulse MAC/sing forwarding to Gateway.

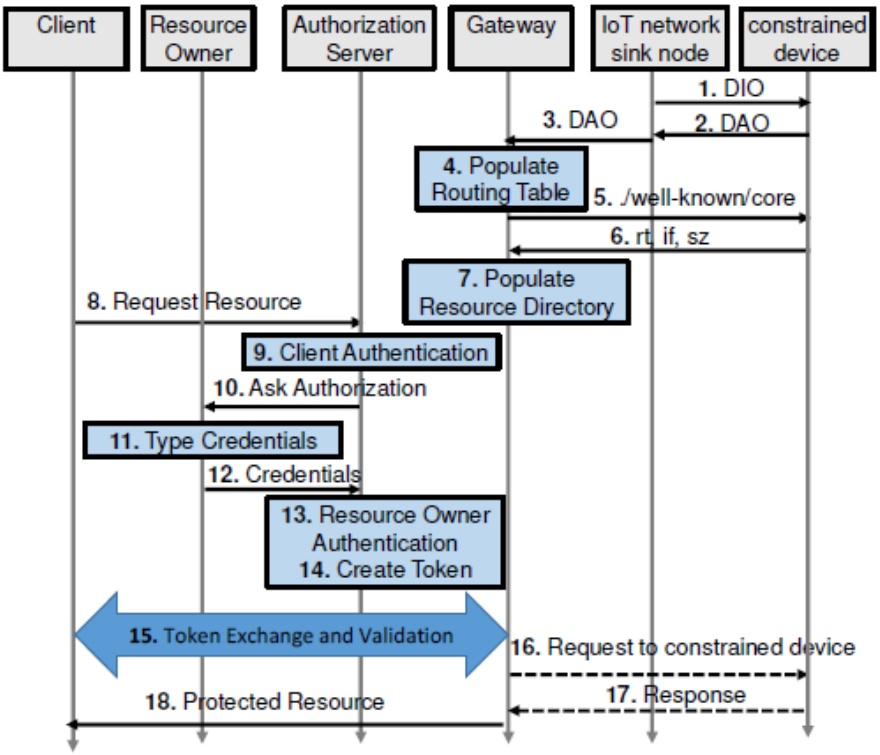

Figure 2.3: Sequence diagram of OAuth-IoT Model [9]

There are six actors define which is shown in figure 2.3. The Client wants to access the data of the IoT devices. There are so many constrained IoT devices (sensor) which sense the environment. All constraint devices to send the data to the IoT network sink node which is continuously broadcast the DIO request and forward the data to the gateway. Gateway is the unconstrained device which translates the request/response between

HTTP protocol and CoAP protocol. The IoT device uses the CoAP protocol to communicate with Network Coordinate sink node. Authorization Server which is manages authorization mechanisms for the grant the permission to a client for accessing the IoT devices. When the gateway receives a new request from the network coordinate sink node for the newly connected constraint devices, it updates the resource discovery table and register constrain IoT device with device ID. The Constrain device sends the data to the gateway for update data table with a time-stamp. Time-stamp filed is used to check the data freshness of the IoT device. When the client wants to access the data of the IoT device, it will send the users credential to the Authentication server. The authentication checks the users credential and creates the access token send to the Gateway and Client if the client is valid. The client accesses the protected IoT data with help of access token.

Gateway has three data structure; one is routing table which give the information about currently active node in IoT environment. Second one is Data table which store the data value and time-stamp according smart IoT devices. Third one is Resource directory which contain information of smart device whenever it connect with IoT environment. When IoT devices wants to connect with IoT environment it will send response to the broadcast signal of the network coordinate sink node. The network coordinate node send request to getaway to update resource directory and routing table. The Gateway update data table by sending request to the IoT devices. On Other side, the client send the users credential to the Authentication server for access the restricted data of the IoT device. If the authentication is done successfully then the access token is generated by the authentication server which is send to the client as well as the Gateway. After this process the client send request to the gateway for access the data for IoT devices using the access token. The above model is implemented with some fix configuration hardware. It measures the response time (latency) to access the IoT device data. The beauty of the modal is the caching of the data which reduce the 85 percentage latency [9].

2.1.5 OAuthing: Privacy-enhancing Federation for the IoT

This paper proposed modal for IoT that grant the access to the user and smart devices to be federated. It also introduces some new actor in the model like IGNTE(Intelligent Gateway for Network IoT Environments), PCM(Personal Cloud Middleware), MQTT Broker which is mapping the OAuth token API to the MQTT Protocol. It also creates a

simple web-based cloud service which is connected to the OAuthing DIdP using standard OAuth2 HTTP flow to request access to IGNITE. The user logs in with the UIdP which is register with the devices. After the authentication, third-party authentication grants the access to the IGNITE that the simple application is loaded. This modal uses MQTT over Web Sockets which provide a User Interface for the user to interact with the device [14].

The above model is implemented by the author with some fix configuration hardware. It measures latency (connection time) for three different scenarios. The Mosquitto broker connection time is 24.5 ms. The IGNITE connection time is 1.3 second when the user has not previously connected. In this case, the system needs to introspect the access token. It also measures the latency when the server is in the loaded condition. It also drew the number of user at a time send the request to the response time graph. It supports the Dynamic Client Register API to support each device having unique OAuth client Identifier which is manually deployed on the smart devices [14].

The OAuth model is connecting the smart devices to the third party without the users credential. The user can bring the pre-existing identity to the system to the system rather than being required to the new credential [14].

2.1.6 An Implementation of AccessControl Protocol for IoT Home Scenario

This paper proposed the access control protocol which is the grant the authentic user to access the protected data. The Protocol is deployed on the Contiki OS and measure constrains resource of the smart device by the Power trace tool. The CoAP protocol is communication protocol for constraint devices which is not handling the authentication mechanism. So the new option enables in the CoAP protocol for the Authentication via the third party. It wants to reduce the burden of the authentication on the smart devices. Therefore the burden of handling the Authentication process has done by the external authentication server (Gateway). When the device boots it will send its device id to the gateway and the Owner. It will generate a unique key by HMAC-SHA1 cryptography which is based on the device ID. The Gateway and OWNER have used these key to authenticate the user to access the resources. It defines the authenticate information for the authentication process [15].

The above modal is implemented on the Contiki OS and Cooja simulator measures the energy consumption and the latency for the smart devices. The main advantage of the modal is that it can support a large amount of validating user at the same time with the help of counter-based token expire mechanism. [15].

2.2 Message Standard

2.2.1 JavaScript Object Notation (JSON)

JavaScript Object Notation (JSON) is a lightweight data formate. It defines a small set of formatting rules for the structured data. it can represent four primitive types like strings, numbers, boolean and null and two structured types like objects and arrays. JSON string is a sequence of zero or more Unicode characters. An object is consist of zero or more key/value.[10]

A JSON text is a set of tokens which includes six structural characters, strings, numbers, and three literal names. The JSON text is a serialized object or array object structure which represented as a key-value pair within curly brackets the key and values separated by the single colon. the key within one object should be unique. In JSON the array structure is represented as square brackets with zero or more values which are separated by the commas. A number contains an integer component and fraction part. the minus sign and fraction part are optional. The octal and hexadecimal format is not allowed for JSON. The representation of string within quotation marks. The default encoding scheme for JSON is UTF-8, where the first two characters of a JSON text will be ASCII characters.[10] A JSON parser transforms a JSON text into another representation which accepts all JSON texts according to JSON grammar.

```
{
   "Image": {
       "Width": 800,
       "Height": 600,
       "Title": "View from 15th Floor",
       "Thumbnail": {
           "Url": "http://www.example.com/image/481989943",
           "Height": 125,
           "Width": "100"
        },
        "IDs": [116, 943, 234, 38793]

    }

}
```

Figure 2.4: Example of JSON Object [10]

2.2.2 JWT

JSON Web Token (JWT) is a string which represents a set of claims to be exchanged between two parties. it encoded as a JSON object that is used as the payload of a JSON Web Signature (JWS) structure or JSON Web Encryption (JWE) structure which represents the digitally signed claims with a Message Authentication Code (MAC). JWT intended for space environments like HTTP Authorization headers and URI query parameters. JSON web token is an open standard for passing the security information (claim).we can send the JWT in different forms like part of URL, in body parameter and HTTP Header (access token). The structure of JWT divide into three different part. A JWT represente a sequence of URL parts which is separated by (.) characters. Each part contains a base64url-encoded value.[1]

Header:

There is two parameter in these part one of the fields is 'typ' which have the default value is 'JWT' second is the 'alg' field which represents the hashing algorithm for creating signature according to the payload.

Payload:

The payloads represent the information in key/value pair which we need to be transmitted. it is encrypted by base64 encoded format. There is two type of claim we used in the payload one is custom claim and another is registered claim which describes in JSON formate.

Signature:

This part creates the signature using algorithms which are described in the header part. the header and payload part is encoded using the base64 algorithm. The signature is created using the secret key which is not publicly available. At another side, the secret key available so it verifies the signatures using the secret key.

The members of the JSON object represented by the JOSE Header which describes the cryptographic operations applied to the JWT but it is optionally and additional properties of the JWT. it is depending upon whether the JWT is a JWS or JWE.

Example of JWT claim:

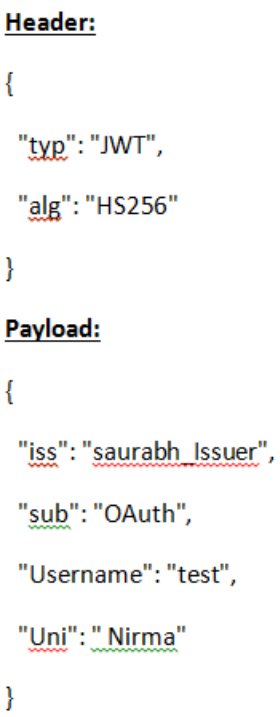

Figure 2.5: Example of JWT Claim [1]

Encode Header JWT claim:

eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9

Encoded Payload JWT claim:

eyJpc3MiOiJzYXVyYWJoX0lzc3VlciIsInN1YiI6Ik9BdXRoIiwiVXNlcm5hbWUiOiJ0ZXN0IiwiVW5pIj

Encoded Signature JWT claim:

vXxOM2m1cg7OvFmvKvYS$_Y$c3$Lyn4SMTCn637PPdOXQA$

2.2.3 Concise Binary Object Representation (CBOR)

There are so many standardized formats for binary representation of structured data which is known as binary serialization formats like BSON,ASN.1, CBOR, and MessagePack. The Concise Binary Object Representation (CBOR) is a standard data format for encoding or decoding Serialize object. It is able to be compact in order to support systems with very limited memory, processor power, and instruction sets like Constrain(IoT) Device. It generates extremely small code after encoding the message. we can decode serialize object without a schema description of the message because the serialized object is self-described. CBOR provide extensibility so the extended data must be decodable by earlier decoders. we can easily convert JSON standard formate into CBOR standard formate.[1]

An encoder is a process that generates the representation in the hexadecimal or binary format of a CBOR data item from application information according to some predefined syntax rules. A Decoder is a process that decodes a CBOR data item. it contains a parser which breaks the input string using predefined syntax rules which available to an application.[1]

we take one example which is convert an indefinite-length map key/value pairs)to CBOR formats."0x" notation represent the hexadecimal format.

Input String: "Fun": true, "Amt": -2

Encoded Format: 0xbf6346756ef563416d7421ff

CBOR Code	Description
BF	Start indefinite-length map
63	First key, UTF-8 string length 3
46756e	”Fun”
F5	First value, true
63	Second key, UTF-8 string length 3
416d74	”Amt”
21	-2
FF	”break”

Table 2.1: Description of CBOR Encoder code [1]

2.2.4 CBOR Object Signing and Encryption (COSE)

As we mentioned above, CBOR is standard data format which is designed for small code size and small message size. we need to provide message security services for IoT which is given by CBOR Object Signing and Encryption (COSE). it defines process signature, authentication codes (MAC), and encryption using CBOR for serializing objects [11].

The serialized objects have only one of the map key is in string format for JSON data formate but we need to more compact message size after encoding so we use strings as well as negative integers and unsigned integers as map keys in COSE. The integers are used for compactness of encoding and easy comparison.The message structure of COSE object is built on CBOR array type. COSE encrypted messages are consisted using two layers which separate the different types of cryptographic concepts[11].

1. Content layer

In these layer, the plain-text is encrypted and information about the encrypted message is placed.

2. Recipient layer

In these layer, the content encryption key (CEK) is encrypted and information about how it is encrypted for each recipient is placed.

Step to compute the signature:

- Create a Sign-structure according the fields.

- Create the value ToBeSigned by encoding the Sign-structure to a byte string according encoding rules.
- Call the signature creation algorithm by passing key(K) with the appropriate algorithm which describes in 'alg' field, the value of sign which is in 'ToBeSigned' field.
- Put the value of the resulting signature in the 'signature' field of the array.

Steps to verify the signature

- Create a Sign-structure according the fields.
- Create the value ToBeSigned by encoding the Sign-structure to a byte string according encoding rules.
- Call the signature verification algorithm by passing key(K) with the appropriate algorithm which describes in 'alg' field, the value of sign which is in 'ToBeSigned' field and signature values which are described in 'sign' field for verification.

There is one example for Single Signer which use Single ECDSA signature.The size of a binary file after encoding the message by ECDSA algorithm is 98 Bytes.
Structure of the COSE data formate:

```
18(
  [
    / protected / h'a10126' / {
        \ alg \ 1:-7 \ ECDSA 256 \
      } / ,
    / unprotected / {
      / kid / 4:'11'
    },
    / payload / 'This is the content.',
    / signature / h'eae868ecc176883766c5dc5ba5b8dca25dab3c2e56a551ce
5705b793914348e19f43d6c6ba654472da301b645b293c9ba939295b97c4bdb84778
2bff384c5794'
  ]
)
```

Figure 2.6: Structure of the COSE for Single Signer [11]

2.2.5 CBOR Web Token (CWT)

CBOR Web Token (CWT) is a compact web token which representing claims for message exchanging between two parties. The CWT claim is a piece of information about the subject which encoded in the Concise Binary Object Representation (CBOR) and CBOR Object Signing and Encryption (COSE). it represented as a key/value pair. there are two type of claim key used in CWT one is registered claim key which has fixed value and another is the non register claim key. The registered claim Keys are represented using integers or text strings which is shown in below table.[2]

Claim Name	**Key**	**Values**
iss	1	text string
sub	2	text string
aud	3	text string
exp	4	integer or floating-point number
nbf	5	integer or floating-point number
iat	6	integer or floating-point number
cti	7	byte string

Table 2.2: Description of CBOR Encoder code [2]

The below example shows a signed CWT with a single recipient and a full CWT Claims Set.

```
18(
  [
    / protected / << {
      / alg / 1: -7 / ECDSA 256 /
    } >>,
    / unprotected / {
      / kid / 4: h'4173796d6d6574726963454344534134
                   23536' / 'AsymmetricECDSA256' /
    },
    / payload / << {
      / iss / 1: "coap://as.example.com",
      / sub / 2: "erikw",
      / aud / 3: "coap://light.example.com",
      / exp / 4: 1444064944,
      / nbf / 5: 1443944944,
      / iat / 6: 1443944944,
      / cti / 7: h'0b71'
    } >>,
    / signature / h'5427c1ff28d23fbad1f29c4c7c6a555e601d6fa29f
                    9179bc3d7438bacaca5acd08c8d4d4f96131680c42
                    9a01f85951ecee743a52b9b63632c57209120e1c9e
                    30'
  ]
)
```

coap://light.example.com

Figure 2.7: Signed CWT with a single recipient and a full CWT Claims Set[2]

Chapter 3

OAuth standard and Internet of Things

3.1 Development

3.1.1 Kaa-IoT framework

Kaa is an open IoT framework which provides the platform for the Internet of Things. It provides many facilities to customize all the functionality of layer in IoT environment. We can deploy mechanism on the IoT environment so we easily evaluate our performance according to our mechanism. There are so many good features for the Kaa-IoT framework. It manages an unlimited number of connected devices at a time. You can set any schema version for IoT framework. It also Collects and analyze sensor data according to user behavior and delivers theme by the notifications using REST API. It provides Admin UI for server-side profile management It also provides the structure of IoT features and extensions for different types of IoT applications. kaa provide plug-and-play modules for the IoT application developers. Kaa enables data management for smart objects using back-end architecture which is provided by Kaa server and endpoint SDK components which are deployed on the smart object. The SDKs transfer the data in a bi-direction way to the Kaa server.

The Kaa server provides all the back-end functionality which is very much required for large and mission-critical IoT environment. It handles the data consistency and communication across connected the smart object. The Kaa-IoT framework provides the customize IoT environment with the help of Kaa middleware. We want to deploy the

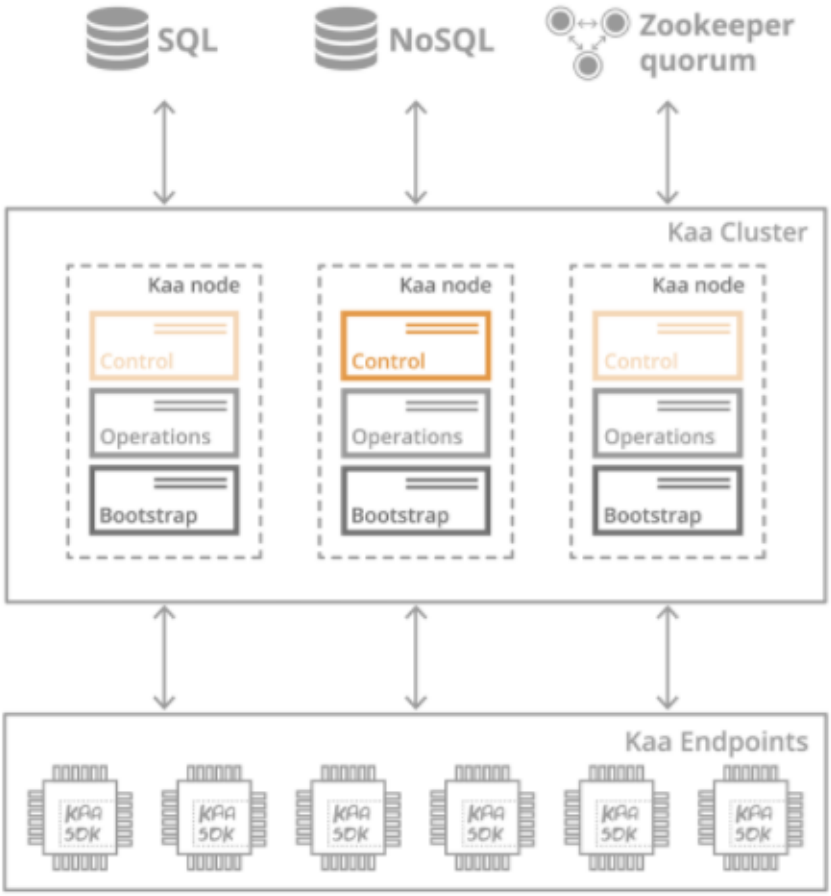

Figure 3.1: Kaa-IoT Framework

OAuth protocol on Kaa-IoT simulator because we can customize all protocol at different layer so we want to modify the authentication mechanism and deploy on the Kaa-IoT framework. We also evaluate the performance of the OAuth protocol in terms of response time after deploying on Kaa-IoT framework.

There are many Kaa-application which created the Kaa-IOT framework that requires attaching the same user to for service Kaa- SDK (endpoint). the endpoint needs to register it's identity to the server when it attached or detached so the Kaa provides necessary API for attaching or detach endpoints to/from users in two different way. User access token flow In these type, the user authenticates himself in an external authentication system like Facebook, Google, Twitter to obtains the access token. The user performs this authentication from the endpoint which is due to be registered with him in the Kaa instance. Then, Kaa SDK transfers this token to the Kaa cluster over a secure channel. The Kaa cluster verifies the access token and attaches the endpoint to the user.

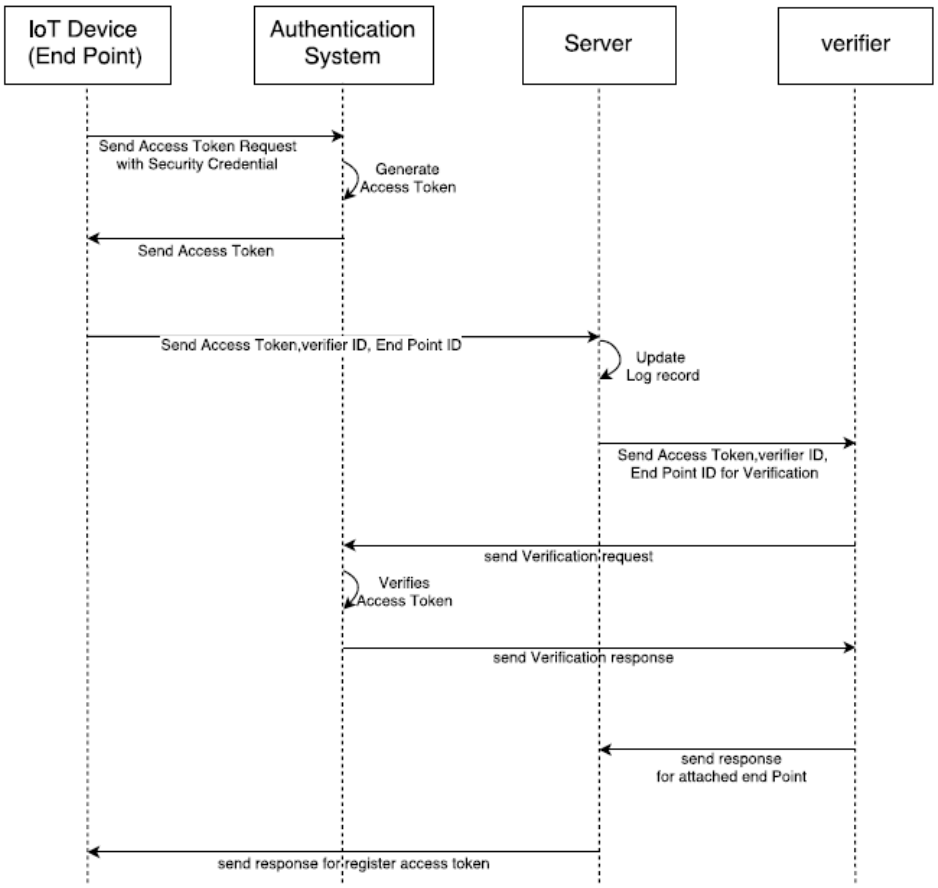

Figure 3.2: Sequence diagram for Authentication System in Kaa framework

There is four actor use for an authentication system which is shown in the figure. the endpoint sends the access token request to the authentication system with the security credentials of the endpoint. the authentication system verifies the endpoint's credentials and sends the access token to the endpoint. after getting access token the endpoint sends attach request to the Kaa server with userid, access token, verifier ID parameter. The Kaa server sends the user id and access token to the user verifier for verification. The role of user verifier is to test and debug the user id and access token for authenticating the user. The user verification is done by user verifiers agent which is shown in the figure. There is default user verifier like facebook, google, and twitter which have already implemented by Kaa sandbox. we can also create our plugin custom verifier for Kaa application. we can use multi user-verifier for single Kaa application. we can create our customer user verifier or default user verifier by Admin UI or REST API. the user verifies send the acknowledge to the authentication server after verification. the authentication server registers the endpoint id and access tokens in its log record and sends the registration response to the endpoint.

Chapter 4

Proposed model

We implement OAuth protocol in three different way of evaluating the server response time (latency) at different network traffic level in the real world. The OAuth architecture is shown in figure 4.1 for three different scenarios. In the first scenario, we install the XAMPP server and create a WordPress blog. The OAuth protocol is deploying on WordPress blog as the client application which wants to access the resource server as a WordPress blog admin on behalf of the Authentication server like Facebook, Google, Twitter.

In the second scenario, we only change the client application which is the host on the external server so the request/response sends between the Authentication server and external server. In the third scenario, we create local authentication server instead of using a well-known server.

We proposed a model which is shown in the figure 4.1. There is serval method to serialize and deserialize object for a message while authentication is done so we take three different methods like JWT, CBOR, CWT. When the user wants to access the protected resource for the first time, the user needs to send a request for getting the access token. We consider a scenario for IoT environment so sometimes the Application and user both agents are combined in the IoT device so among the four messages, two messages are the upload from the IoT device and another two message are download to the IoT device. when the first IoT device sends the request for the getting the services, the IoT device serializes the object using the different method like CBOR, CWT, JWT. another side it deserializes the object and gets the original message.

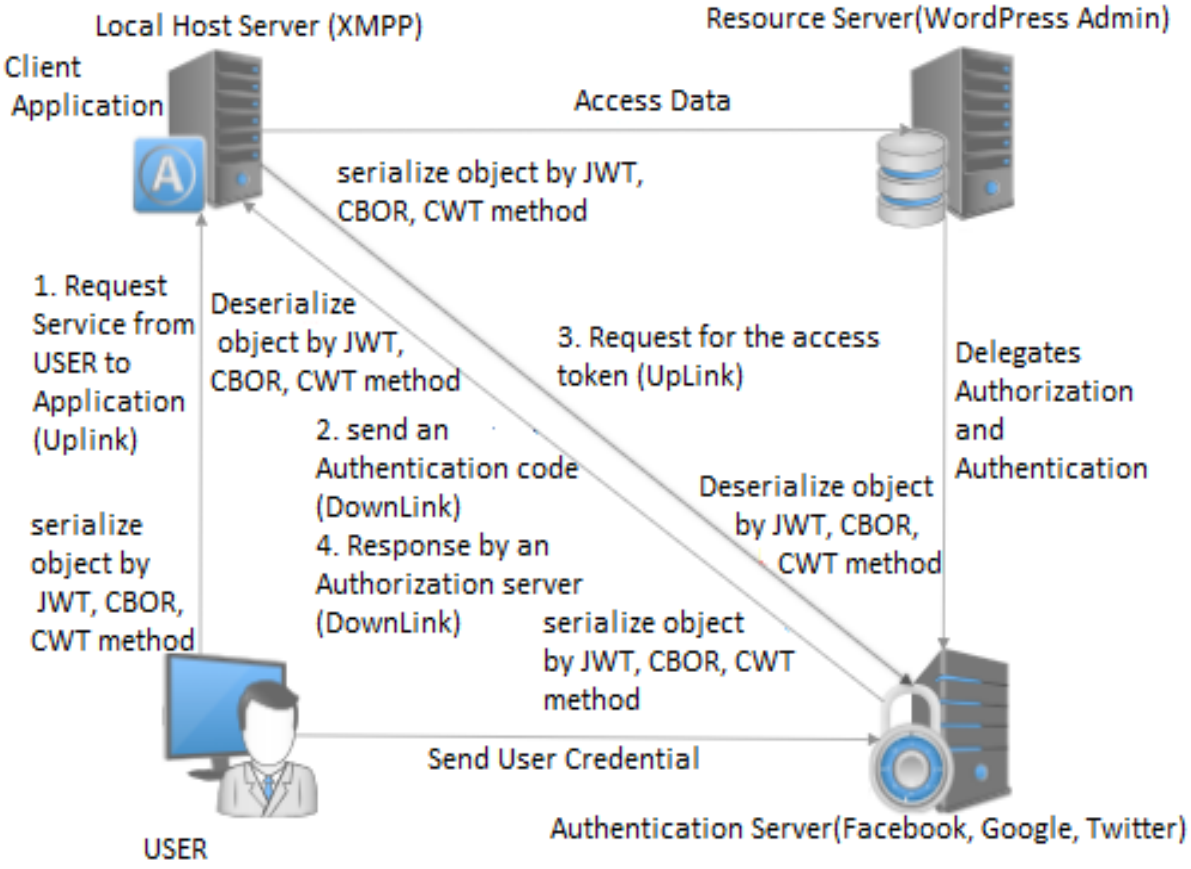

Figure 4.1: OAuth Protocol architecture while message Exchange

An access token is very important for access the IoT data after authentication is done by the authentication server. We want to modify the access token and the authorization mechanism so that we can reduce the burden on the IoT device and also reduce the response time (latency) when the client sends the request to the IoT devices. Authentication for the end to end Communication for IoT device is done by the DTLS protocol via Elliptic Curve Cryptography(ECC). Datagram Transport Layer Security (DTLS) is the communications protocol which is used to security for datagram based applications. IPv6 address over Low-power Wireless Personal Area Networks (6LoWPAN) is used to deploy the IPv6 address scheme over the constraint environment to identify the IoT device. When we deploy the OAuth protocol on IoT device, the authentication code and access token are exchanged among the IoT device with the help of DTLS, 6LoWPAN, RPL, CoRE protocol which is constructed for the constrained environment. We want to modify the size of the access token packet which is the exchange constraint (IoT) envi-

ronment. Figure 4.2 shows the packet information while access token and authentication

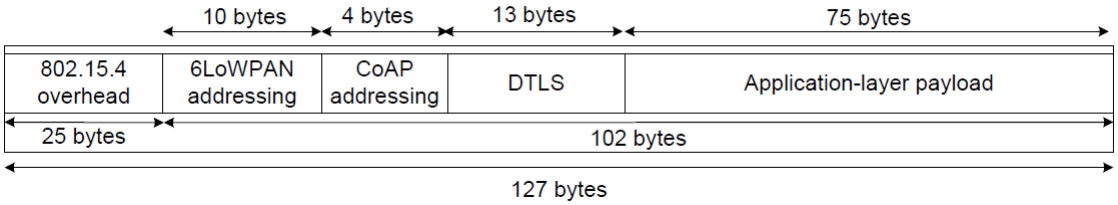

Figure 4.2: Payload space usage for 6LoWPAN and CoAP

code in the IoT environment. The payload takes as the access token and authentication code.

There is some implication to reduce the size of access token packet.

- We can reduce the size of DLTS header by some modification do in the information which passed for the message security.
- Authentication code and access token is unique identification string which used for the authorization purpose. We can reduce the size of the authentication code and the access token with modifying some parameter in the unique string.
- We can reduce the header size of 6LowPAN protocol because when the access token and the authorization code is an exchange in the IoT environment. The header of the 6LoWPAN is always attached with the payload which resides access token and the authentication code.

Chapter 5

Experimental Result and Analysis

As we mention above, we implement OAuth Protocol in three different scenarios and measure the server response time (latency). In the first scenario, we establish local server (XAMPP) on our system. The user can logs in WordPress blog via three different third-party authentications (OAuth server) like Facebook, Google, and Twitter. We measure the server response time while authentication is done by the Authentication server. In these scenario first request send to Authentication server from local-host server which is installed in my system and response form Authentication server to the local-host server.

Sr. No.	**Authentication server**	**3:40 pm**	**9:40 am**	**9:40 pm**
1	Facebook	0.48 sec	0.9 sec	1.01 sec
2	Google	0.5 sec	0.55 sec	0.57 sec
3	Twitter	0.5 sec	0.50 sec	0.54 sec

Table 5.1: latency for local-host to authentication server

In the second scenario, we use the externally hosted server (byethost) which is a free hosting server. We create WordPress blog on the externally hosted server. The OAuth Protocol deploy on the WordPress blog. We measure the response time when the request/response sends between the external server to Authentication servers like Facebook, Google, and Twitter.

Sr. No.	**Authentication server**	**3:40 pm**	**9:40 am**	**9:40 pm**
1	Facebook	3.5 sec	4.0 sec	4.23 sec
2	Google	1.9 sec	2.0 sec	2.1 sec
3	Twitter	2.01 sec	2.3 sec	2.56 sec

Table 5.2: latency for externally hosed server to authentication server

In the third scenario, we establish local server (XAMPP) on our system. We also create local authentication server in our system. We deploy the OAuth protocol on the WordPress blog. So the request/response for the authentication mechanism sends between the local Authentication server and localhost server which reside on our system. We measure the server response time (latency) while authentication is done by the local authentication server which is 0.3 second.

Sr. No.	**Authentication server**	**3:40 pm**	**9:40 am**	**9:40 pm**
1	Local Authentication Server	0.3 sec	0.3 sec	0.3 sec

Table 5.3: latency for local-host to local authentication server

There are different methods exist to create Access Token for authentication process but we want to implement OAuth Protocol for IoT constrained Environment so we deploy the different method like CBOR, JWT, CWT in IoT and Windows environments and analysis the results according to Power consumption, the Execution time to encode and decode the Access token and number of Bytes to create Access token. To implement above method we take Raspberry PI 3 Processor as IoT device. we deploy three different methods CBOR, JWT, CWT on Raspberry PI 3 processor. We modified the code of CBOR [16], JWT[17], and CWT[18] for the Raspberry PI 3 processor which is already present on GitHub. On implementing the CBOR, JWT and CWT we measured the power consumption, the execution time and a number of bytes.

There are four types of message exchange in standard OAuth protocol while authentica-

tion is done. The parameters for the message exchange are taken from the OAuth RFC standard.

Type 1. Request Service form USER to Application (UP-LINK)

When the first time user wants to take Access Token for accessing the restricted resources (temperature, humidity, light), three parameters are passed to the Application which is shown below. Authorization Grant has five types which represent that how the Owner's authorization give to access to IoT resources. we take authorization code type as an Authorization Grant so by default set the value of code as a response type parameter. At the time of registration of the application, the Authorization server creates client id and send to the Application. the authentication server redirects the user to the redirect URL which is one of the parameters used in this type of message exchange.

- response type=code
- client id=188617085284773 (Facebook)
- redirect URL=https://www.applicationdemo.com

Type 2. Send Authentication Code (DOWN-LINK)

After validating user credentials the Authentication Server creates Auth Code which is the temporary Access token send to Application. In these message type only one parameter exchange.

- code=SplxlOBeZQQYbYS6WxSbIA (RFC)

Type 3. Request for Access Token by Application (UP-LINK)

After getting Authentication Code the Application sends the request for getting access token. In these message type five parameter sent by the application to the Authorization server for generating the Access token which is shown in below. we take values of client-id and client-secret parameters by the Facebook authorization server for experiment purpose.

- code=SplxlOBeZQQYbYS6WxSbIA (RFC)
- grant type=authorization code
- client id=188617085284773 (Facebook)

- client secret=8581722fdf457b53790a7cc621258cdf (Facebook)
- redirect URI=https://www.applicationdemo.com

Type 4.Response Token by Authorization Server (DOWN-LINK)

After verifying the Authentication code which is sent in the request message, the Authorization server generates the Access token. In these message type, the Authorization server sends seven parameters in the response message to the Application which is shown in below. There are different types of token used in OAuth protocol. we take Bearer token values as a token type parameter. Authorization server also generates the signature for the access token. The life of access token describe by the expires parameter in Millisecond. we take the 3600 values for the expire parameter.

- id=https://www.applicationdemo.com
- expires in=3600
- scope=full API permission
- token type= Bearer
- Signature=rQGz10jkGpNLAzMOGZiSw4xggHSdYVtIAUIsPaY1iO0 (JOSE)
- Access token="2YotnFZFEjr1zCsicMWpAA" (RFC)

for implementation, we deploy the different method which generates the Access token on the raspberry PI 3 (IoT devices) and Windows platform. we measured time to encode and decode the object for different message types. There are four different graph which is shown in figure 5.1, 5.2, 5.3, 5.4. It represents the execution time for the different method on Windows Platform.
Table 5.4, 5.5, 5.6, 5.7 shows the execution time (ms) and the number of bytes to the different method for different types. When we compare only CBOR and JWT method, the CBOR methods takes low execution time for the different method and it also generates the low number of Bytes. CBOR generate encoded String in a hexadecimal format which is predefined by the CBOR Grammar. There is no any signature algorithm used in CBOR method but in the JWT method, the execution time and the number of bytes is highest because it used the complex signature algorithm to generate the signature.

Method	**Execution Time(ms)**	**Number of Bytes**
CBOR	120	93
JWT	811	212
Encrypted single recipient (CWT)	176	103
MACed multiple recipients (CWT)	155	122
MACed single recepaint (CWT)	144	89
Signed multiple recipients (CWT)	260	154
Signed single recipient (CWT)	230	146
Encrypted multiple recipients (CWT)	47.89	111

Table 5.4: Execution Time and Number of Bytes for Message Type 1 on Raspberry PI (IoT) Platform

Method	**Execution Time(ms)**	**Number of Bytes**
CBOR	120	29
JWT	812	128
Encrypted single recipient (CWT)	174	57
MACed multiple recipients (CWT)	156	76
MACed single recepaint (CWT)	143	43
Signed multiple recipients (CWT)	258	108
Signed single recipient (CWT)	238	100
Encrypted multiple recipients (CWT)	47.87	65

Table 5.5: Execution Time and Number of Bytes for Message Type 2 on Raspberry PI (IoT) Platform

Method	**Execution Time(ms)**	**Number of Bytes**
CBOR	120	180
JWT	810	327
Encrypted single recipient (CWT)	279	158
MACed multiple recipients (CWT)	158	177
MACed single recepaint (CWT)	147	144
Signed multiple recipients (CWT)	265	209
Signed single recipient (CWT)	241	201
Encrypted multiple recipients (CWT)	49.9	166

Table 5.6: Execution Time and Number of Bytes for Message Type 3 on Raspberry PI (IoT) Platform

Method	**Execution Time(ms)**	**Number of Bytes**
CBOR	120	188
JWT	809	338
Encrypted single recipient (CWT)	176	200
MACed multiple recipients (CWT)	156	219
MACed single recepaint (CWT)	146	186
Signed multiple recipients (CWT)	262	251
Signed single recipient (CWT)	239	243
Encrypted multiple recipients (CWT)	49.9	208

Table 5.7: Execution Time and Number of Bytes for Message Type 4 on Raspberry PI (IoT) Platform

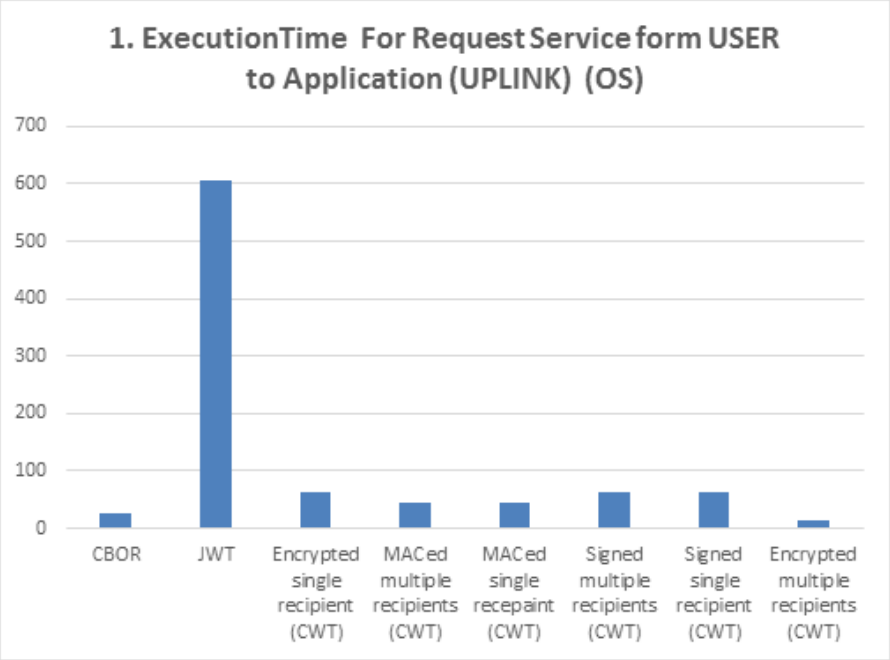

Figure 5.1: Execution Time vs Different Method on Windows Platform for Message Type 1

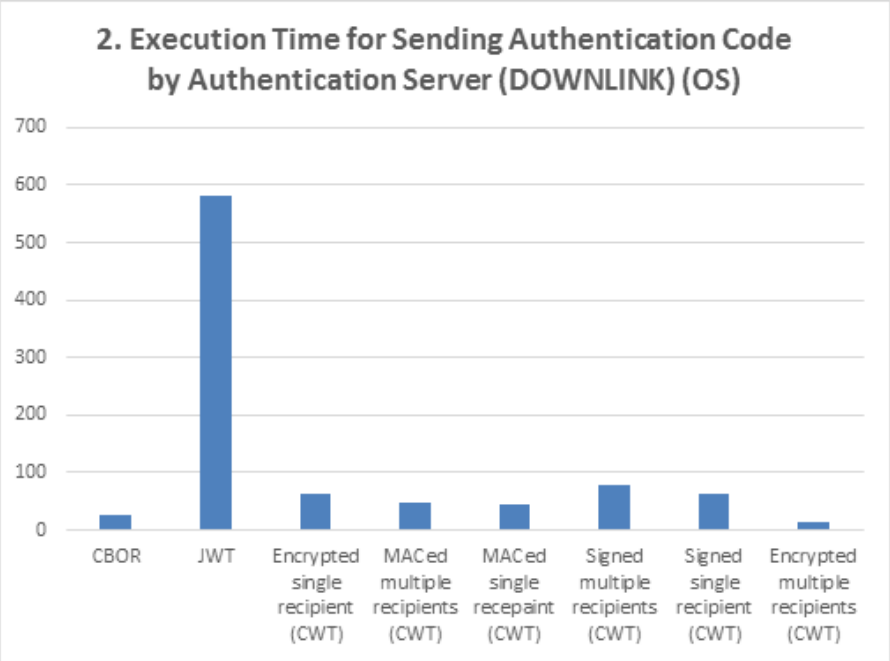

Figure 5.2: Execution Time vs Different Method on Windows Platform for Message Type 2

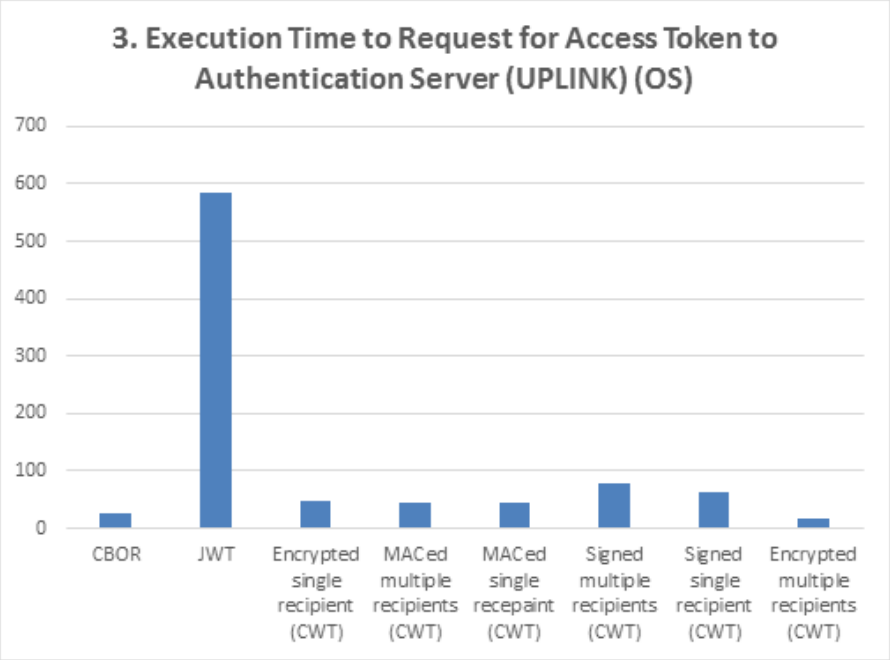

Figure 5.3: Execution Time vs Different Method on Windows Platform for Message Type 3

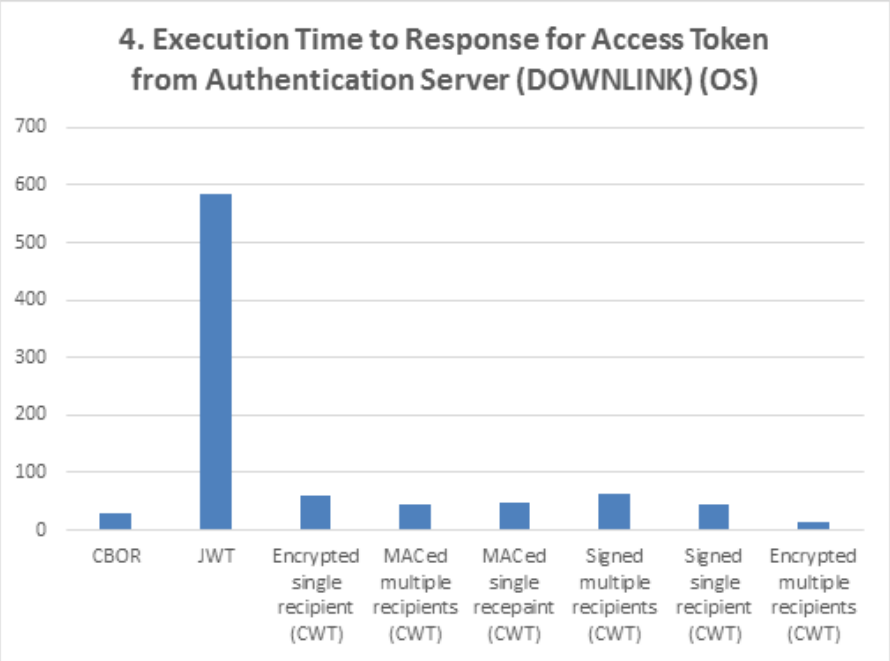

Figure 5.4: Execution Time vs Different Method on Windows Platform for Message Type 4

There are four different graph which is shown in figure 5.5 5.6 5.7 5.8. It represents the execution time for the different method on Raspberry PI Platform.

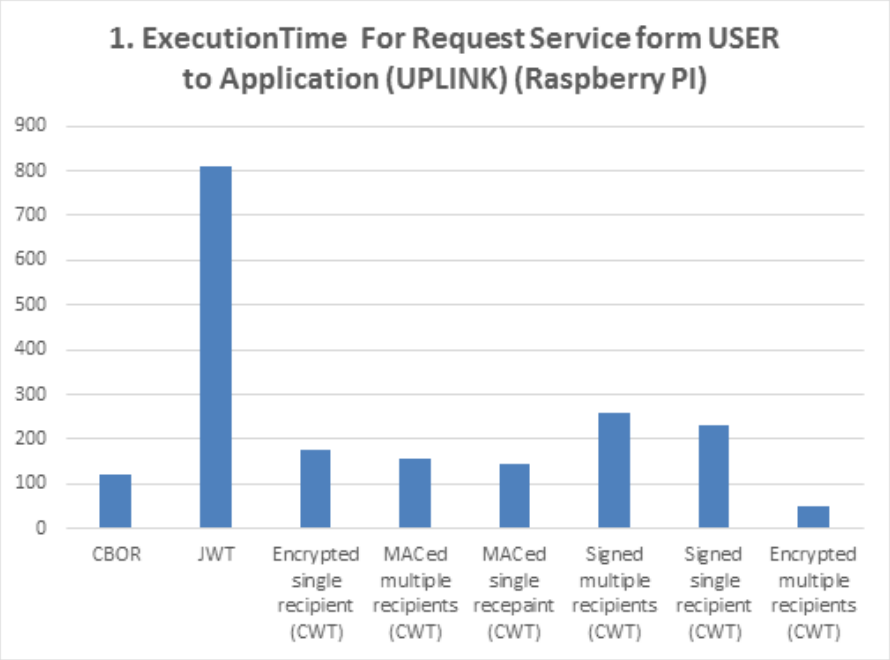

Figure 5.5: Execution Time vs Different Method on Raspberry PI 3 (IoT Device) for Message Type 1

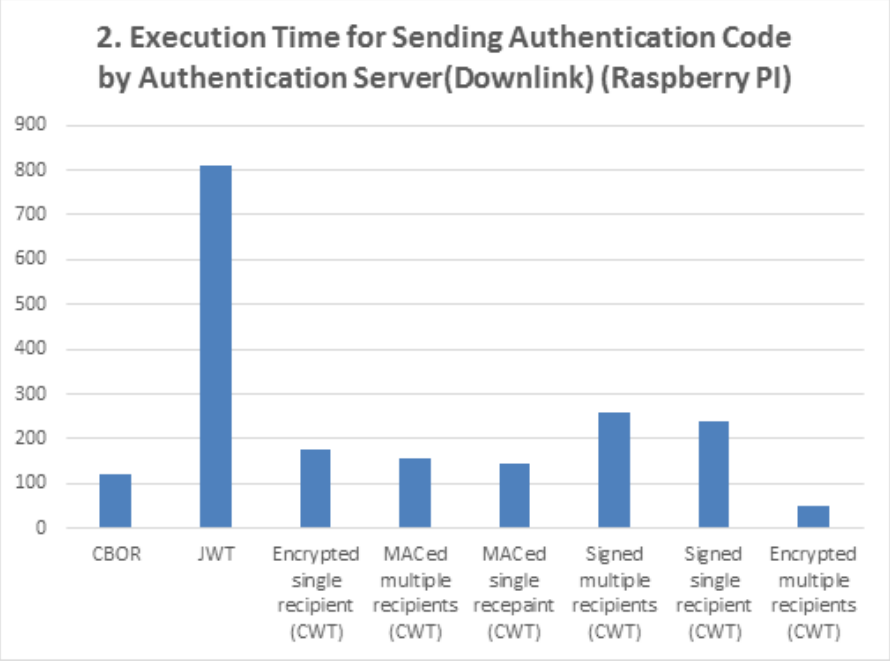

Figure 5.6: Execution Time vs Different Method on Raspberry PI 3 (IoT Device) for Message Type 2

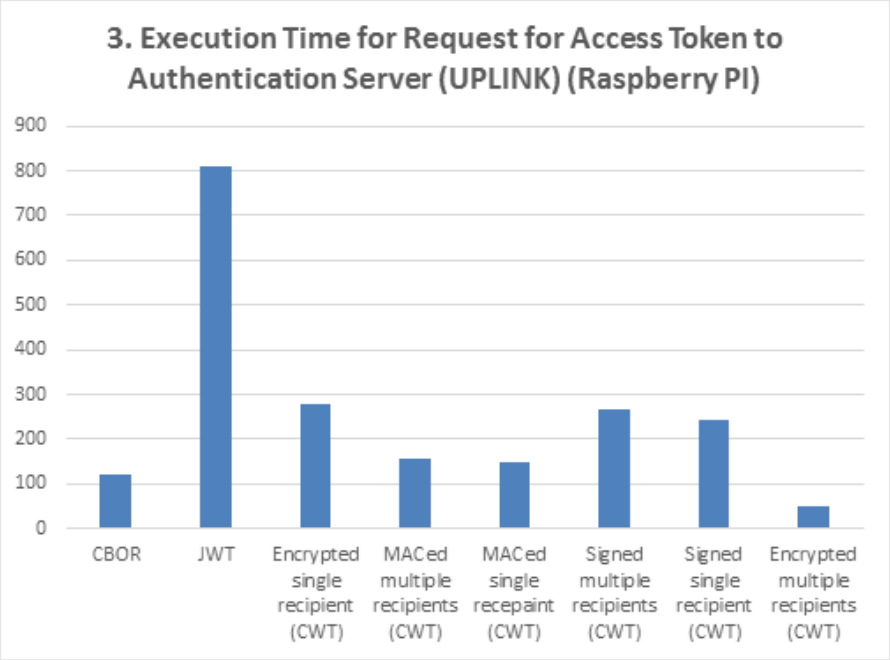

Figure 5.7: Execution Time vs Different Method on Raspberry PI 3 (IoT Device) for Message Type 3

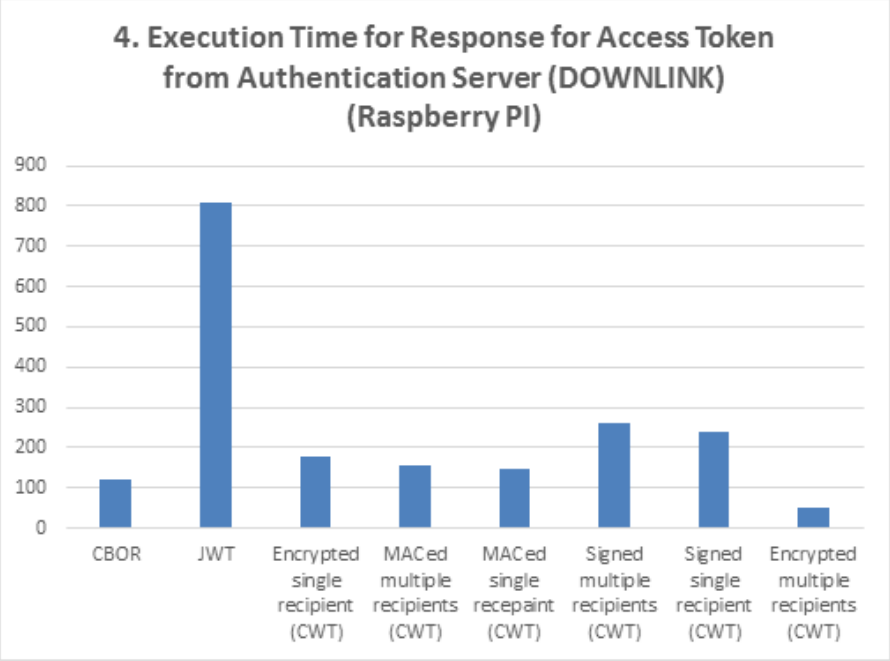

Figure 5.8: Execution Time vs Different Method on Raspberry PI 3 (IoT Device) for Message Type 4

we also find the number of Bytes generates when the different message type is encoded. There are four different graphs which are shown in figure 5.9, 5.10, 5.11, 5.12. It represents

the Number of Bytes for the different methods. the least number of Bytes generate for CBOR methods.

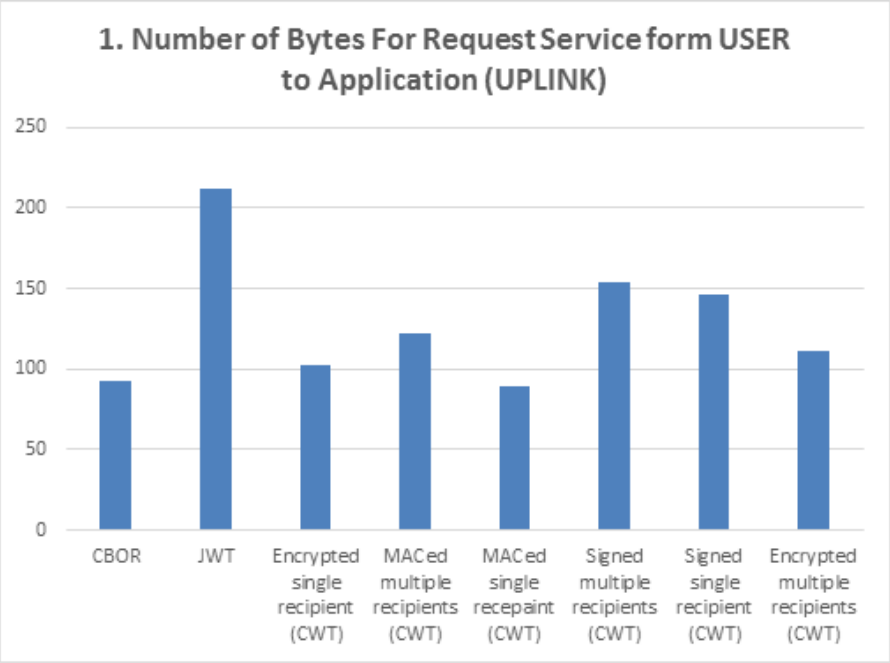

Figure 5.9: Number Bytes vs Different Method for Message Type 1

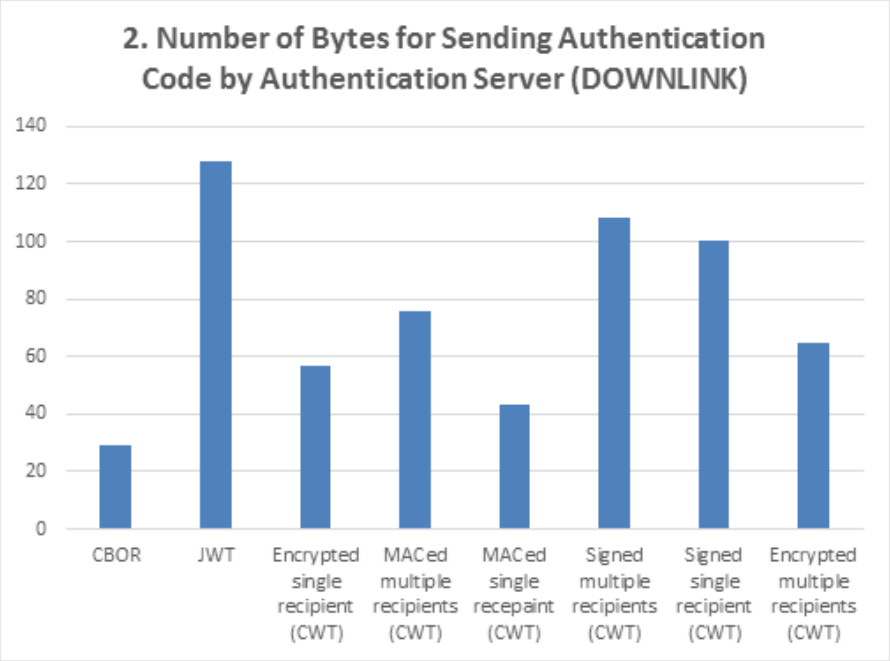

Figure 5.10: Number Bytes vs Different Method for Message Type 2

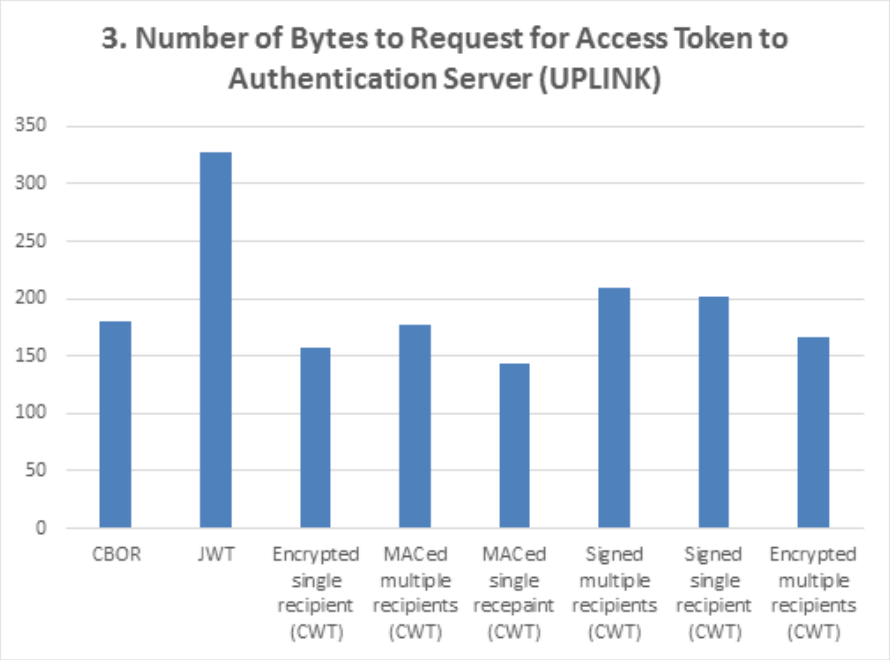

Figure 5.11: Number Bytes vs Different Method for Message Type 3

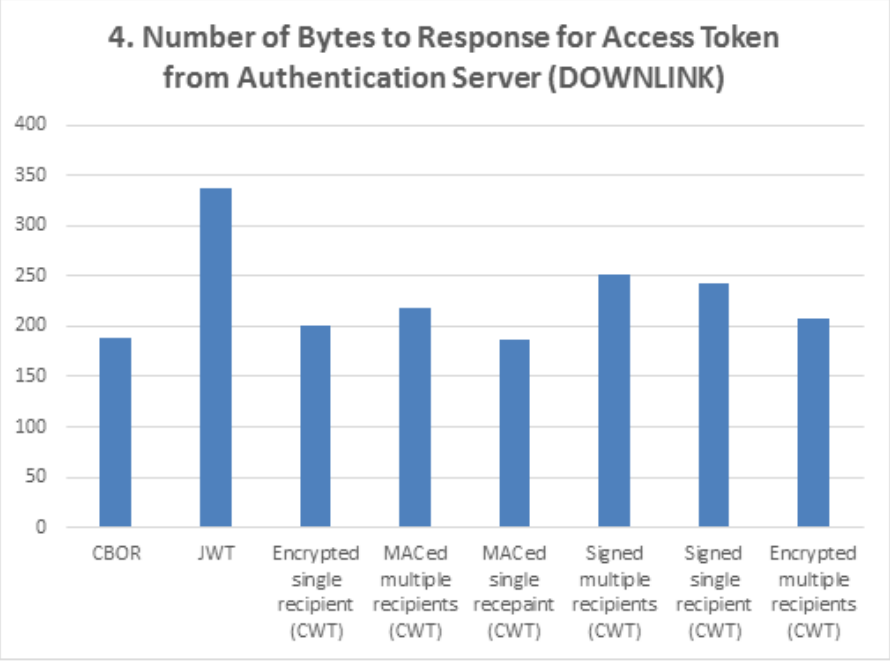

Figure 5.12: Number Bytes vs Different Method for Message Type 4

we also find the power consumption while uploading and downloading for the different massage type. There is standard power consumption model which is described in below equation in 5.1 [3] [4].

$$P_{Pi} = P_{idle} + P_{CPU}(u) + \sum_{if} (P_{if,idle} + P_{if,UP}(r) + P_{if,DW}(r)) \tag{5.1}$$

P_{Pi} = Total Power Consumption for Raspberry PI 3

P_{idle} = Power Consumption for Raspberry PI 3 in IDLE Mode

$P_{CPU}(u)$ = Power Consumption of Raspberry PI 3 platform which depending on the CPU utilization.

u = The CPU utilization in the range 0 to 1 which is depend on message size

$P_{if,idle}$ = Power Consumption during while Interface is IDLE

$P_{if,UP}$ = Power Consumption during while Uploading Data on the Interface

$P_{if,DW}$ = Power Consumption during while Downloading Data on the Interface

r = The transferred data rate in Mb/s (for WI-FI Interface value of r is 3 Mb/s and for Ethernet Interface value of r is 40 Mb/s)

We implement different method likes CBOR, JWT, CWT on Raspberry PI 3. we find total power consumption from above equation. for that, we take some values for specific Raspberry PI (IoT Device)model 3 which is shown below in table 5.8.

Function	**values**	**Range[Mb/s]**
P_{idle}	1.488	-
$P_{Ethernet,idle}$	-0.1176	-
$P_{WI-FI,idle}$	0.7645	-
$P_{CPU}(u)$	0.6191(u)	[0,1]
$P_{Ethernet,UP-Link}(r)$	$(26.2)10^{-06}(r)^2 + (0.357)10^{-03}r + 0.007$	$[0, 80]$
$P_{Ethernet,Down-Link}(r)$	$(-4.33)10^{-06}(r)^2 + (0.485)10^{-03}(r) - 0.007$	$[0, 80]$
$P_{WI-FI,UP-Link}(r)$	$(-0.25)10^{-06}(r)^2 + (1.99)10^{-03}r - 0.072$	$[0, 6]$
$P_{WI-FI,Down-Link}(r)$	$(1.85)10^{-03}(r)2 + (-13.5)10^{-03}(r) + 0.072$	$[0, 6]$

Table 5.8: Power Consumption for Raspberry PI Model 3 [3] [4]

we compute the power consumption from the equation by using table 5.4. now we have only the power consumption of Raspberry PI processor so we need to add the power consumption while the bit is up-link and down-link. we take value for receiving and transaction bit from Raspberry PI form the below table 5.9.

Modulation	**values(Watt)**
$E_b(T_x)$	0.00000013
$E_b(R_x)$	0.000000049

Table 5.9: The average energy cost per bit for Raspberry PI Model 3 [5] [6]

E_b = The average energy cost per bit
T_x = for Transmitting the bit
R_x = for Receiving the bit

Finally, we get the total power consumption equation for transmitting and receiving bit for Raspberry PI 3 with the different interfaces like Ethernet and WI-FI which is shown in equation 5.2.

$$P = P_{Pi} + E_b \tag{5.2}$$

P = Total power consumption equation for transmitting and receiving bit for Raspberry PI 3 with the different interfaces like Ethernet and WI-FI
P_{Pi} = Total Power Consumption for Raspberry PI 3 Processor only
E_b = The average energy cost per bit for Transmitter(T_x) or Receiver(R_x)

Method	Power Consumption (%)			
	Type1	Type2	Type3	Type4
CBOR	18.22	0.0	47.28	49.23
JWT	57.96	29.90	96.36	100
Encrypted single recipient (CWT)	21.56	6.20	39.93	53.93
MACed multiple recipients (CWT)	27.91	12.54	46.27	60.28
MACed single recepaint (CWT)	16.89	1.52	35.25	49.26

Signed multiple recipients (CWT)	38.59	23.22	56.96	70.96
Signed single recipient (CWT)	35.92	20.55	54.29	68.29
Encrypted multiple recipients (CWT)	24.23	8.87	42.60	56.60

There are four different graphs which are shown in figure 5.13, 5.14, 5.15, 5.16. It represents the Total Power Consumption(P) for the different method on Raspberry platform.

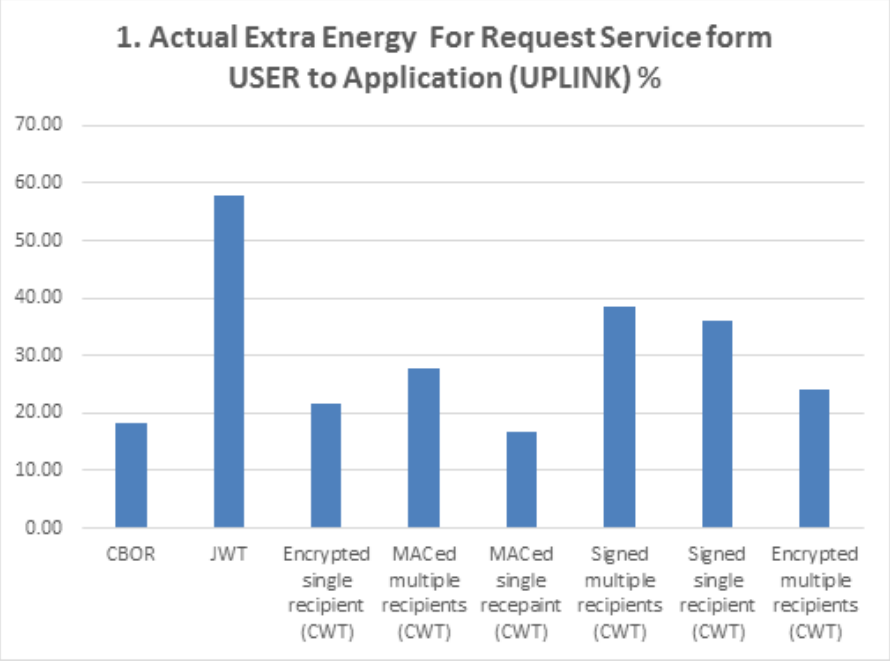

Figure 5.13: Power Consumption vs Different Method on Raspberry PI 3 (IoT Device) platform for Message Type 1

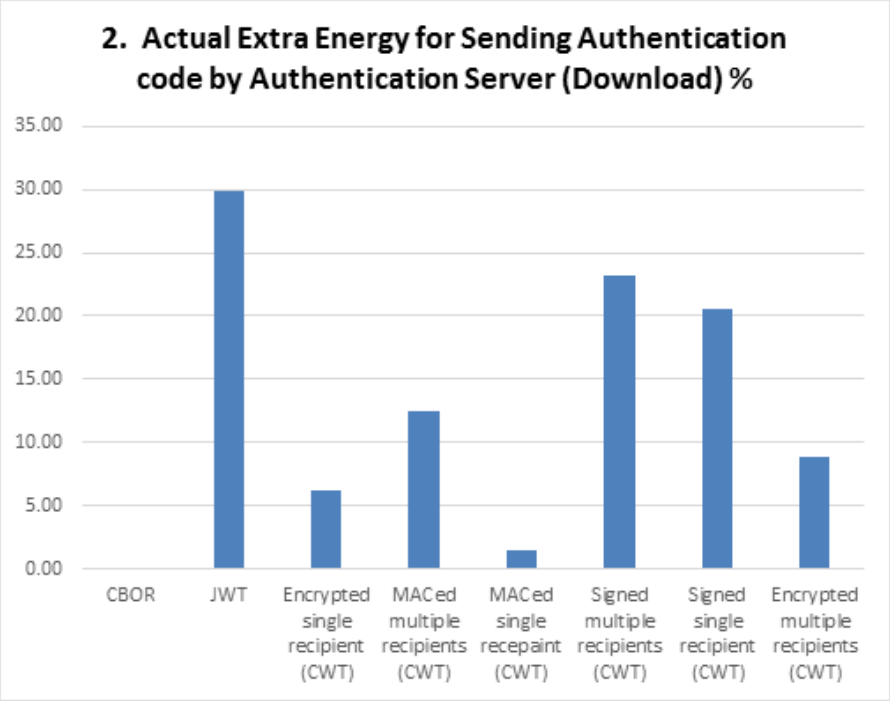

Figure 5.14: Power Consumption vs Different Method on Raspberry PI 3 (IoT Device) platform for Message Type 2

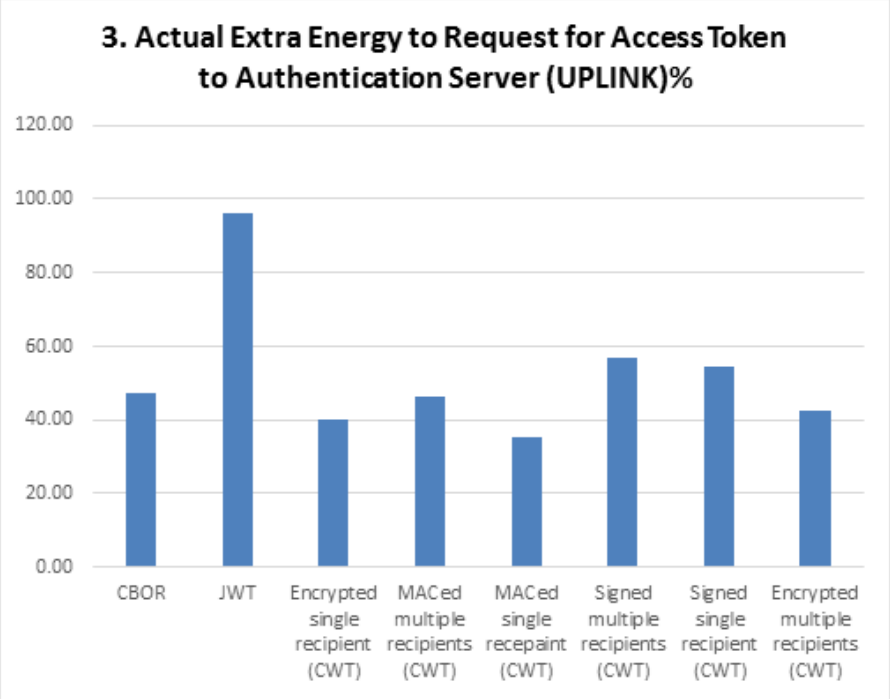

Figure 5.15: Power Consumption vs Different Method on Raspberry PI 3 (IoT Device) platform for Message Type 3

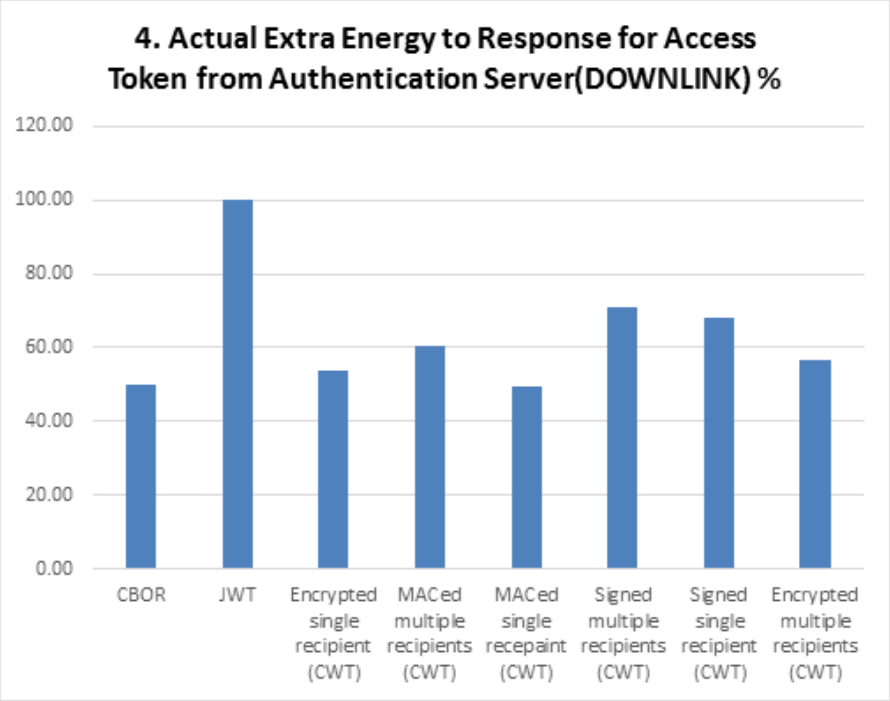

Figure 5.16: Power Consumption vs Different Method on Raspberry PI 3 (IoT Device) platform for Message Type 4

Chapter 6

Conclusion & Future Scope

Various messaging standards like CBOR, CWT, JWT have been experimented with for sending and receiving OAuth related messages on the Raspberry PI 3 (IoT Device) as well as Laptop category device. These methods have been analyzed according to Power consumption and Execution time taken during serialization and deserialization of the exchange object for authorization process for IoT environment. The CBOR and CWT based approach consumes less power and execution time as compared to JWT method. In CWT, we create CBOR based web token(CWT) using CBOR data format with hashing and signing as well as encrypting the message. The power consumption and execution time is minimum in MACed CWT as compared to Signed CWT and Encrypted CWT. It also uses less time in single recipient approach as compared to multiple recipients approach while generating the signature. The CBOR standard data format are suitable among the all methods because it takes less energy while message exchange in IoT environments because the IoT device has very limited resource according to power consumption and memory.

In future, the messaging methods can be integrated with Kaa-IoT framework. Kaa-IoT frame-work generates the different IoT applications as Kaa-SDK. Raspberry PI 3 support only C, C++ Kaa SDK for IoT Application so these messaging standards can be extended to the SDKs of open source IoT framework like Kaa and others. Kaa-IoT application may then perform faster processing of authentication and authorization related message exchanges.

Bibliography

[1] M. Jones, J. Bradley, and N. Sakimura, "Json web token (jwt)," tech. rep., 2015.

[2] M. Jones, E. Wahlstroem, S. Erdtman, and H. Tschofenig, "Cbor web token (cwt)," tech. rep., 2018.

[3] F. Kaup, P. Gottschling, and D. Hausheer, "Powerpi: Measuring and modeling the power consumption of the raspberry pi," in *Local Computer Networks (LCN), 2014 IEEE 39th Conference on*, pp. 236–243, IEEE, 2014.

[4] D.-I. F. Kaup *et al.*, "Energy-efficiency and performance in communication networks,"

[5] K. M. Gomez Chavez, *Energy Efficiency in Wireless Access Networks: Measurements, Models and Algorithms*. PhD thesis, University of Trento, 2013.

[6] K. Gomez, R. Riggio, T. Rasheed, D. Miorandi, and F. Granelli, "Energino: A hardware and software solution for energy consumption monitoring," in *Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks (WiOpt), 2012 10th International Symposium on*, pp. 311–317, IEEE, 2012.

[7] "OAuth 2.0 Protocol." https://www.google.com/search?q=sequence+diagram+oauth+2.0+authentication&source=lnms&tbm=isch&sa=X&ved=2ahUKEwiBmfeWwYXbAhUH6Y8KHWM9CS0Q_AUoAXoECAAQAw&biw=1094&bih=486#imgrc=L_t7LycrUp0gYM:, 2016.

[8] P. G. L. V. Simone Cirani, Marco Picone and G. Ferrari, "Iot-oas: An oauth-based authorization service architecture for secure services in iot scenarios," 2015.

[9] D. C. G. B. Savio Sciancalepore, Giuseppe Piro and G. Bianchi, "Oauth-iot: an access control framework for the internet of things based on open standards," *Symposium on Computers and Communications (ISCC)*, 2017.

[10] D. Crockford, "The application/json media type for javascript object notation (json)," 2006.

[11] J. Schaad, "Cbor object signing and encryption (cose)," tech. rep., 2017.

[12] D. Hardt, "The oauth 2.0 authorization framework," 2012.

[13] D. S. T. Swati Kinikar, "Implementation of open authentication protocol for iot based application," 2017.

[14] B. A. Paul Fremantle, "Oauthing: Privacy-enhancing federation for the internet of things," 2016.

[15] J. L. C. R. Xiaoyang Wu, Ron Steinfeld, "An implementation of access-control protocol for iot home scenario," 2017.

[16] P. Occil, "A Java implementation of Concise Binary Object Representation (RFC 7049)." https://github.com/peteroupc/CBOR-Java, 2017.

[17] T. Gesellchen, "Nimbus-JOSE-JWT." https://github.com/gesellix/Nimbus-JOSE-JWT, 2012.

[18] LudwigSeitz, "A java library for handling CBOR Web Tokens." https://github.com/LudwigSeitz/CWT-Java, 2017.

[19] J. K. y. Paul Fremantley, Benjamin Aziz and P. Scott, "Federated identity and access management for the internet of things," *International Workshop on Secure Internet of Things*, 2014.

[20] A. J. J. A. F. S. Jos L. Hernndez-Ramos, Marcin Piotr Pawlowski and L. Ladid, "Toward a lightweight authentication and authorization framework for smart objects," *SELECTED AREAS IN COMMUNICATIONS*, 2015.

[21] L. T. K. V. Renzo E. Navas, Manuel Lagos, "Nonce-based authenticated key establishment over oauth 2.0 iot proof-of-possession architecture," 2016.

[22] L. M. J. S. Federico Fernandez, A lvaro Alonso, "A model to enable application-scoped access control as a service for iot using oauth 2.0," 2017.

[23] D.-Y. H. K.-S. K. Shamini Emerson, Young-Kyu Choi and K.-H. Kim, "An oauth based authentication mechanism for iot networks," 2016.

[24] J. S. S. Jorge Granjal, Edmundo Monteiro, "End-to-end transport-layer security for internet-integrated sensing applications with mutual and delegated ecc public-key authentication," 2013.

[25] P. P. K. M. P. A. A. P. Aimaschana Niruntasukrat, Chavee Issariyapat, "Authorization mechanism for mqtt-based internet of things," *W07-Workshop on Convergent Internet of Things*, 2016.

[26] V. L. A. O. A. S. M. R. Riccardo Bonetto, Nicola Bui, "Secure communication for smart iot objects: Protocol stacks, use cases and practical examples," 2012.

[27] S. S. K. Sye Loong Keoh and H. Tschofenig, "Securing the internet of things: A standardization perspective," *INTERNET OF THINGS*, 2014.

[28] P. Solapurkar, "Building secure healthcare services using oauth2.0 and json web token in iot cloud scenario," 2016.

[29] C. Bormann and P. Hoffman, "Concise binary object representation (cbor)," 2013.