



Enhanced Security by using AWS MQTT Broker as Middleware Architecture for IoT Environment

1. (Author & Corresponding author) **Prof. Nimavat Dhaval M.**, Research Scholar, Computer Engg. Dept., RK University, Rajkot. Email: dhaval.nimavat@gmail.com 9428075424
2. (Co-Author) **Dr. Raiyani Ashwin G.**, Nirma University, Ahmedabad. Email: ashwin.rkcet@gmail.com

Financial/Non-Financial Conflicts of Interest

There are no competing interests to declare, here we are declaring that the article confirms that there are no relevant financial or non-financial competing interests to report.

Abstract

The act of preserving security on Internet-connected devices and the networks to which they are connected as well as locating, tracking, and repairing issues from various devices that potentially pose security threats is known as IoT security. The issue of securing devices that are connected to the internet has always been difficult. The aim of the given research work is to enhance security in an IoT environment by leveraging AWS IoT Core concepts and the AWS MQTT broker by implementing X.509 digital certificates, using private keys and applying AWS IoT Core device policies. AWS IoT Core is a managed cloud service provided by Amazon Web Services (AWS) that enables the secure and scalable connection of IoT devices to the cloud. It provides various features and functionalities to manage and interact with IoT devices effectively. By utilizing X.509 digital certificates, we enhance the security of the communication between IoT devices and AWS IoT Core. To implement enhanced security, we generate our own private key, which will be used to sign and encrypt your digital certificates. The private key should be kept confidential and securely stored. We would then configure AWS IoT Core to use X.509 certificates for device authentication and secure communication. This involves registering your devices with AWS IoT Core and associating the corresponding digital certificates with each device. The devices will present their digital certificates during the connection process to establish a secure connection with the AWS MQTT broker.

Keywords: *IoT, Security, MQTT Broker, AWS IoT Core, Node-Red*

1. Introduction to IoT Security

IoT security refers to securing internet-connected gadgets including smart devices, sensors, and digital machinery. Each IoT component is described by a "Thing," which has unique identifiers that can send and receive data using various protocols that are supported such as CoAP, DDS, LoRa, LoRaWAN, LWM2M, and MQTT. The protocol is in charge of both the security and transmission of data because, if not properly safeguarded, the internet exposes data to a number of major dangers[1].

It is crucial to make sure that all networks with connected Internet of Things devices are secure. IoT security always focuses on a wide range of techniques, tactics, protocols and actions with the objective of minimising the growing IoT vulnerabilities over internet-connected devices. IoT security measures like API (Application Programme Interface) security and Public Key Infrastructure (PKI) Authentication are frequently used to protect against hacker attacks and cybercrime[2].

1.1. Challenges within IoT Security

Remote Access allows unknown entities to interact with devices remotely, taking advantage of the prevalence of HTTP (Hypertext Transfer Protocol) and APIs in internet-connected devices. This presents security risks, particularly when it comes to network targets that hackers often exploit for phishing attacks. As a result, it has become crucial to employ Cloud Security measures to protect against such threats.

Industries such as automotive and healthcare are expanding the range of IoT devices to increase productivity and efficiency as part of their digital transformation. However, increased reliance on technology also makes it more vulnerable to the consequences of a successful data breach. Of particular concern is the vulnerability of IoT devices. Also, many Medicare and auto manufacturers are not ready to devote the necessary resources and sufficient financial resources to secure these devices. Therefore, the lack of consensus in the industry intends many institutions and companies to the cyber security threat.

Another key challenge to IoT security is the resource limits of various IoT devices. In contrast to traditional computing systems, not all IoT devices have enough processing capacity to handle complex security measures such as strong firewalls or antivirus software. Some gadgets even have difficulty with basic connectivity. As a result, IoT devices that use Bluetooth and other technologies have become targets for data breaches. These security flaws have had a substantial impact on the automotive industry in particular[3]–[5].

1.2. IoT Security Parameter

Digital Certificates and Personal Key Infrastructure (PKI) are crucial elements of contemporary cryptographic systems used to safeguard communications and confirm the legitimacy of digital entities. The management, distribution, and use of cryptographic keys are made possible by the PKI framework. To build trust between participants in online transactions and communications, it offers a secure and trustworthy means to create, store, distribute, and revoke digital certificates.

The data integrity transmitted or received from IoT devices to pub/sub systems must be protected, and only authorised devices, developers, and apps should be able to interface with APIs. Due to a certain API, the mobile massive exposed the personal details of millions of consumers, including those customers' phone numbers along with other personal information.

The IoT security process includes the following steps[3]:

1. Key Generation - This field generates a pair of encryption keys - a private (secret) key and a corresponding public key.
2. Request Certificate - The entity provides a Certificate Authority (CA) with a certificate signing request (CSR), together with its public key and identification data.
3. Certificate Issuance: The CA verifies the entity's identification and then issues a digital certificate following successful validation. The CA creates a digital signature on the certificate by signing it with its private key.
4. Certificate Distribution: The entity receives the digital certificate, allowing it to use it to build trust and validate its identity.
5. Certificate Validation: Relying parties can confirm the digital certificate by validating the CA's digital signature and checking its revocation status.

2. Working with MQTT Broker

Message Queuing Telemetry Transport (MQTT) is a lightweight message protocol that is widely used in machine-to-machine (M2M) and Internet of Things (IoT) applications as shown in figure 1. An MQTT broker is a central intermediate server that allows MQTT clients to exchange messages[2].

Following are the important parameters for MQTT Broker

1. MQTT brokers accept messages broadcast by MQTT clients and route them to their intended receivers. The broker serves as a messaging hub, allowing various clients to communicate with one another.
2. Topic-based Publish/Subscribe communications structure: MQTT employs a publish/subscribe communications structure. Messages can be published on specified topics by clients, and other clients can subscribe to those topics to receive the messages. The broker handles subscriptions and guarantees that published messages are delivered to subscribers who are interested.

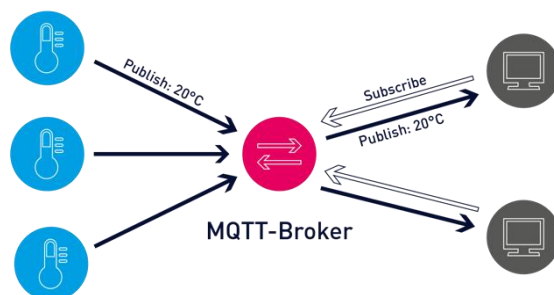


Fig 1. MQTT Broker

3. MQTT provides many Quality of Service (QoS) levels for message delivery. The broker guarantees that messages are delivered and acknowledged in accordance with the QoS level established by the publisher.
4. Persistent Messaging: Because MQTT brokers may store and keep messages, new subscribers who subscribe to a subject can get previously published messages. This function makes certain that subscribers do not miss out on important information.
5. Scalability: MQTT brokers are intended to efficiently manage a high number of clients and messages. They may be horizontally scaled to meet additional traffic while maintaining excellent availability.
6. Security and Authentication: To ensure secure communication between clients and the broker, MQTT brokers frequently provide security measures such as Transport Layer Security (TLS) encryption and authentication.
7. MQTT brokers are frequently integrated with IoT platforms and frameworks, providing smooth connectivity and communication between IoT devices and applications.

MQTT brokers are critical in enabling effective and dependable messaging in MQTT-based IoT systems. They control message routing, QoS, and scalability and security, making them an essential component of MQTT-based communication infrastructures[6], [7].

3. Proposed Algorithm

There are several methods available to meet the various security needs of an IoT System. The first and most important need is mutual authentication between the IoT device and the gateway inside the IoT System's resource-constrained environment. To improve security mechanisms, we presented Amazon Web Services (AWS) Core Concepts and MQTT Broker as IoT Middleware, coupled with custom digital certification, private key, and custom policy to enable safe communication in an IoT environment in this research paper[8], [9].

Step 1: Connect AWS IoT Core from Amazon Web Services

To define a customised policy for secured communication, it's essential to establish a connection between clients or things or Publisher Subscriber "RKU1" to the AWS IoT Core environment.

Syntax

```
"Version": "2023-05-17",  
"Statement": [  
  {  
    "Effect": "Allow",  
    "Action": "iot:Connect",  
    "Resource": [  
      "arn:aws:iot:us-east-1:123456789012:RKU1/${iot:Connection.Thing.ThingName}"  
    ]  
  }  
]
```

Step 2: Create a Topic for IoT Core Device

IoT Services need a topic to recognize messages transmitted or received from publishers. Using the following command, we can create single or multiple topics as per the need of communication as shown in figures 2 and 3 with a general and specific topic subscription.

Syntax

```
arn:aws:iot:aws-region:dhaval.nimavat:topic/Topic
```

Example,

arn:aws:iot:us-east-1:topic/application/topic/device/sensor/temperature/room1 is Amazon Resource Names for the topic application/topic/device/sensor.

The syntax for Topic Filter

```
arn:aws:iot:aws-region:dhaval.nimavat:topicfilter/TopicFilter
```

For example,

arn:aws:iot:us-east-1:topicfilter/application/topic/+sensor is an Amazon Resource Name for the topic filter application/topic/+sensor.

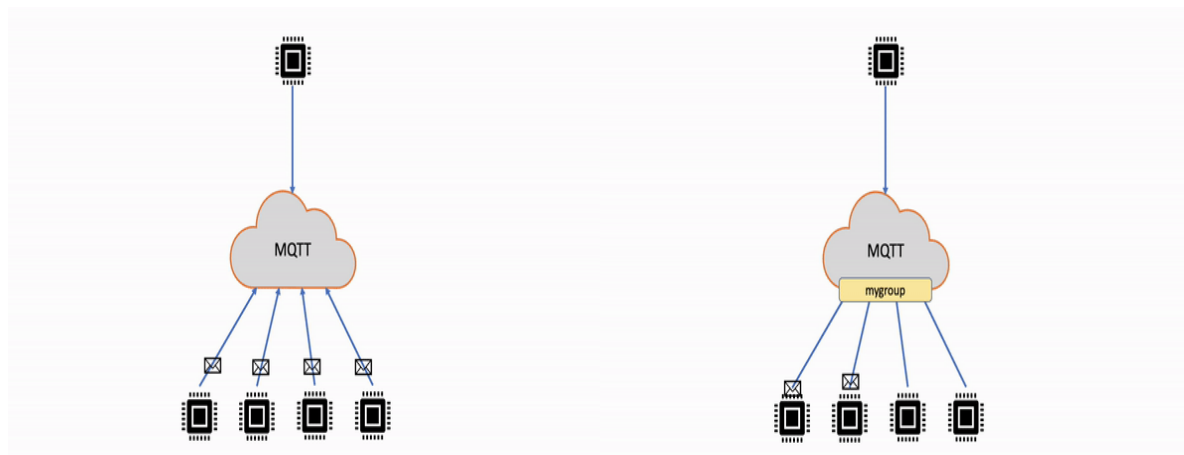


Fig 2. General Topic Subscription

Fig 3. Specific Topic Subscription

Step 3: Attaching custom digital certificate (.CSR)

Custom authentication empowers to establish their own method for client authentication and authorization through the utilization of authorizer resources. Each authorizer encompasses a connection to a Lambda function functioned by the customer, a public key for verifying the credentials of devices, and supplementary configuration details[10].

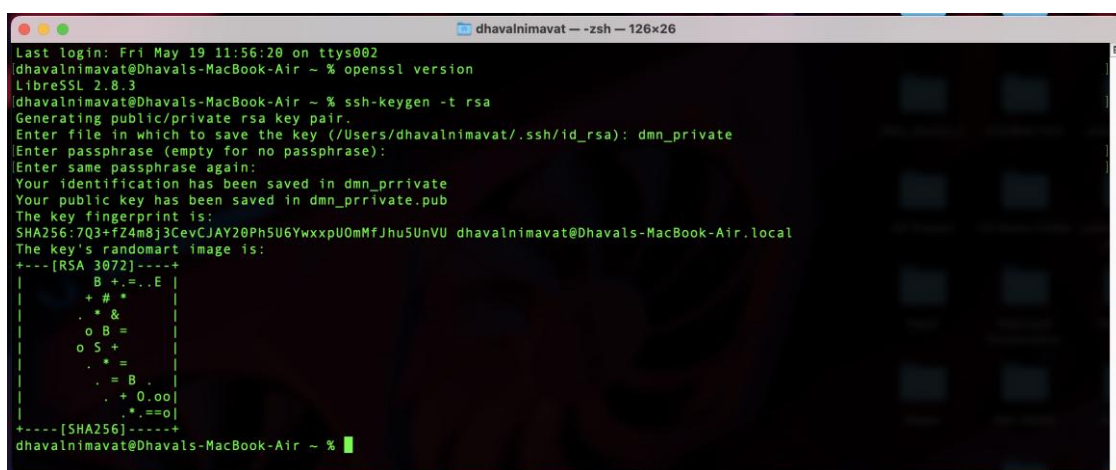
A digital signature provides the following benefits,

Authentication: The recipient can verify the identity of the sender by confirming that the signature matches the sender's public key[11].

Integrity: Any alteration to the signed message or document, no matter how small, will result in a different hash value and thus fail the verification process[11].

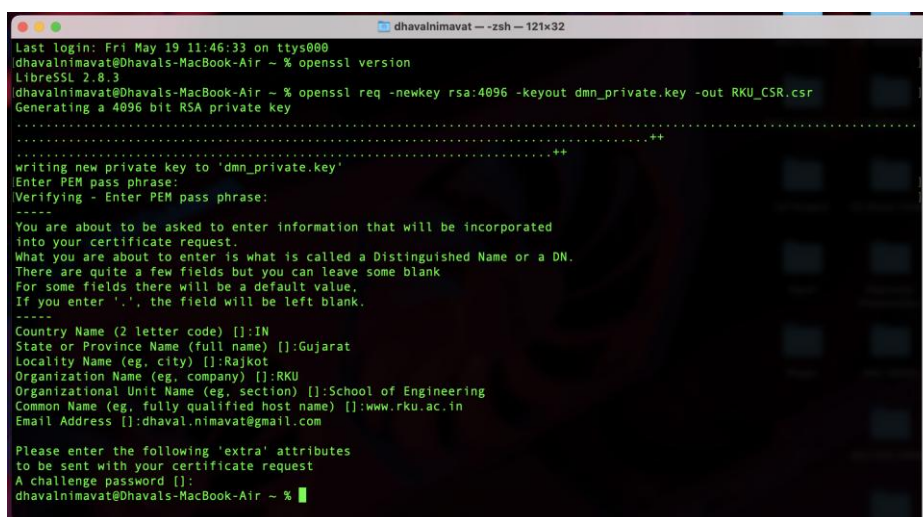
Non-repudiation: The sender cannot deny having signed the message or document since their private key is required to produce a valid signature[12].

Using OpenSSL, It's required to generate a private key "dmn_private.key" in order to embed the same into the digital certificate "RKU_CSR.csr" as shown in figures 4 and 5 respectively.



```
dhavalnimavat ~ % openssl version
dhavalnimavat@Dhavalns-MacBook-Air ~ % ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/Users/dhavalnimavat/.ssh/id_rsa): dmn_private
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in dmn_private
Your public key has been saved in dmn_private.pub
The key fingerprint is:
SHA256:7Q3+fZ4m0j3CevJAY20Ph5U6YwxpU0mMfJhu5UnVU dhavalnimavat@Dhavalns-MacBook-Air.local
The key's randomart image is:
+---[RSA 3072]---+
| B + . . E |
| + # * |
| . * & |
| o B = |
| o S + |
| . * = |
| . = B . |
| . + 0.00 |
| . * . = 0 |
+----[SHA256]-----+
dhavalnimavat@Dhavalns-MacBook-Air ~ %
```

Fig 4. Generating private key using LibreSSL 2.8.3



```
dhavalnimavat ~ % openssl req -newkey rsa:4096 -keyout dmn_private.key -out RKU_CSR.csr
Generating a 4096 bit RSA private key
-----+-----+
writing new private key to 'dmn_private.key'
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value.
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) []:IN
State or Province Name (full name) []:Gujarat
Locality Name (eg. city) []:Rajkot
Organization Name (eg. company) []:RKU
Organizational Unit Name (eg. section) []:School of Engineering
Common Name (eg. fully qualified host name) []:www.rku.ac.in
Email Address []:dhaval.nimavat@gmail.com

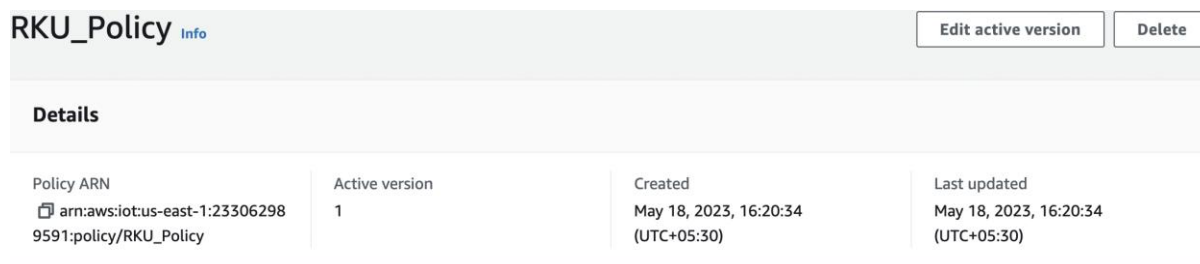
Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
dhavalnimavat@Dhavalns-MacBook-Air ~ %
```

Fig 5. Generating digital certificate using the previous step's private key

Used the OpenSSL – LibreSSL 2.8.3 command to sign the x509 certificate. The next example creates a custom digital certificate named RKU_CSR.csr using the previous step's private key (dmn_private.key) and the signing request.

Step 4: Declare custom MQTT Policy

The policy of AWS IoT Core is based on JSON format. Each policy contains various standards which are responsible for establishing connections, retaining messages, publishing messages, receiving messages and also subscribing to topics[13]. Figure 6 describes the custom policy on AWS as “RKU_Policy”.



The screenshot shows the AWS IoT Core console interface for a policy named 'RKU_Policy'. At the top right, there are two buttons: 'Edit active version' and 'Delete'. Below the header, there is a 'Details' section with a table of key-value pairs.


Policy ARN	Active version	Created	Last updated
 arn:aws:iot:us-east-1:23306298:9591:policy/RKU_Policy	1	May 18, 2023, 16:20:34 (UTC+05:30)	May 18, 2023, 16:20:34 (UTC+05:30)

Fig 6. Policy for publisher/subscriber

iot:Connect - Ability to connect to the AWS IoT Core broker. Every time a **iot:Connect** request is issued to the broker, the **iot:Connect** authorization is verified. Two clients with the same client ID cannot be connected to the message broker at the same time. The broker shuts the current connection once the second client connects. Use the **iot:Connect** permission to restrict connections to just authorized clients with a certain client ID.

iot:GetRetainedMessage - Permission to retrieve the contents of a single retained message. Messages that were published with the **RETAIN** flag set and saved by AWS IoT Core are referred to as retained messages.

iot:ListRetainedMessages - Represents the ability to access summary information about the account's retained messages but not the messages' contents.

iot:Publish - Denotes the ability to publish MQTT topic. Each **PUBLISH** request is issued to the broker, and this permission is verified. This may be used to enable clients to publish to certain topic patterns.

iot:Receive - The authorization to receive messages from AWS IoT Core. Every time a message is provided to a client, the **iot:Receive** permission is verified. It can use this permission to revoke rights to clients who are currently subscribing to a subject since it is checked on every delivery.

iot:Subscribe - Enables to subscribe to the topic filter represented by this object. This permission is verified every time a **SUBSCRIBE** request is issued to and for the broker. Allow customers to subscribe to subjects that fit specified topics such as **design**[14].

Step 5: Allow specific or all publishers/subscriber request

Once the policy has been established, it is important to use "Action" to provide access to all or a subset of clients. For instance, we only grant iot:Connect to all connected devices as described in figure 7.

```
Active version: 1 Builder JSON  
  
{  
  "Version": "2023-05-19",  
  "Statement": [  
    {  
      "Effect": "Allow",  
      "Action": "iot:Connect",  
      "Resource": "*"   
    }  
  ]  
}
```

Fig 7. Allowing pub/sub for iot:Connect only

Step 6: Distribute generated certificate and private key among the authorized publisher/subscriber.

Downloading certificate and private key files to install on publisher subscriber devices so that they connect to AWS as described in figure 8.

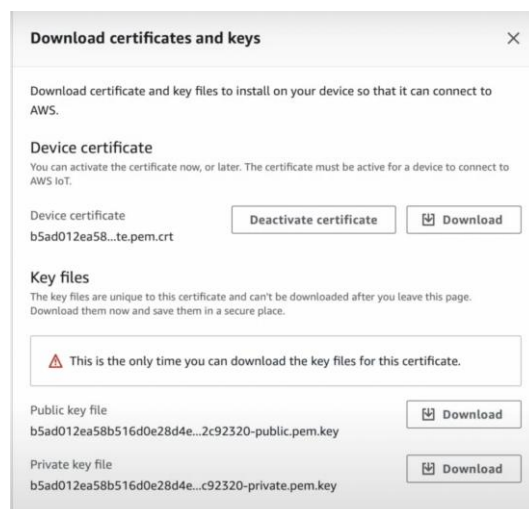


Fig 8. Digital x509Certificate and Private Key

Step 7: Configure MQTT broker services with the given certificate and key using node-red.

Within an AWS IoT context, the Node-RED platform has been used in a specific case to create and put to use digital signature and policy techniques. Node-RED is a visual programming tool that enables users to build and deploy applications by connecting several nodes, which represent various services or functionalities. In an AWS IoT context[15], Node-RED can possibly be used to coordinate

the necessary setups for digital signatures and regulations[10], [16] Which is shown in figures 9 and 10 respectively.

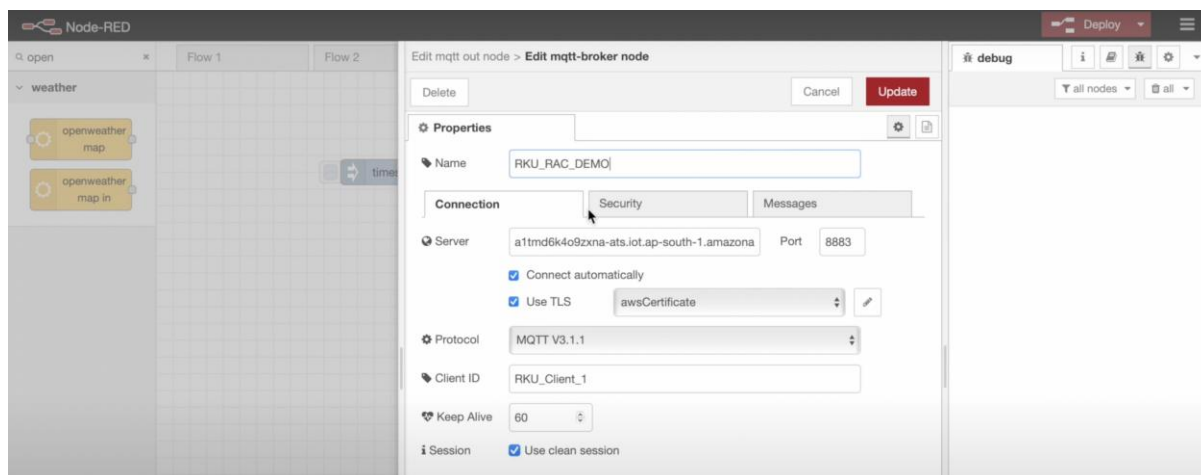


Fig 9. Node-Red, AWS MQTT Broker configuration

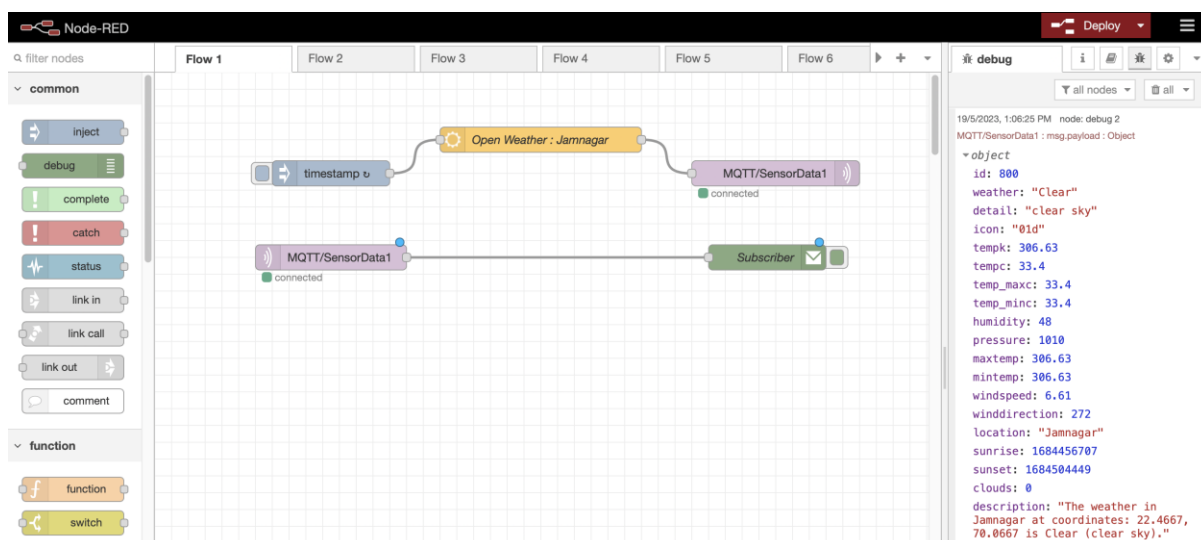


Fig 10. Flow on Node-Red

Case Study of Open Weather API as publisher and the local client running on <https://127.0.0.1:1880> as a subscriber using customised AWS IoT Core Concepts, IoT Topic “MQTT/SernsorData1”, custom private key “dmn_private.key”, generated digital certificate “RKU_CSR.csr” and policy “RKU_Policy”[12].

4. Comparative Analysis

In the given research work we analysed and compared various cloud security like AWS(Amazon Web Service), Microsoft Azure and GCP(Google Cloud Platform) as shown in Table 1.

4.1 Authentication and Authorization

For Authentication and authorization, AWS offers IAM (Identity & Access Management) as per Figure 11, whereas other cloud services like Microsoft Azure and Google Cloud Platform (GCP) supports only Sign-On support which can be easily attacked by DDoS (Distributed Denial of Services). AWS IAM ensures the identity and access management of each publisher or subscriber using fine-grained permission by identifying them by the workforce and workload. IAM also defined custom organizational or institutional policies for what resources can be accessible by authorized publishers or subscribers. AWS also has plenty of distributed data centres around the globe, ensuring data redundancy, availability and planning capacity. Likewise, Microsoft Azure and Google Cloud Platform (GCP) also have plenty of data centres offering data storage [17].

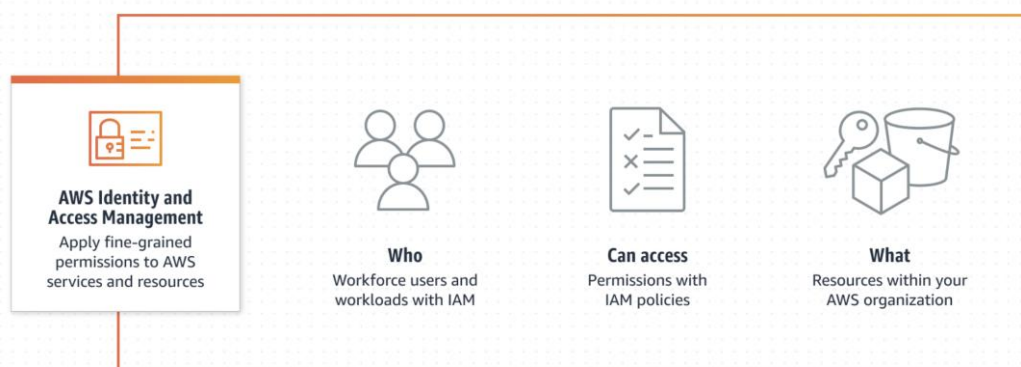


Fig 11. AWS IAM (Identity and Access Management)

4.2 Protection

AWS Shield is a service for threat protection. Apart from Microsoft Azure and GCP which offers protection from DDoS attack, The AWS shield provides DDoS detection and mitigation benefits for all publishers and subscribers connected via MQTT Broker, based on middleware architecture. AWS Shield Services also enable the entire middleware architecture to get protected from various network and transport layer attacks. Using AWS Shield following attacks can be detected and prevented [18].

4.2.1 Network Volume Attack

Within layer 3, IoT architecture also contains the Infrastructure layer attack vectors. Due to this attack vector, the entire capacity of middleware architecture denies the services of authorized publishers and subscribers [18], [19].

4.2.2 Network Protocol Attack

Within layer 4, A network protocol attack is a SYN (Synchronization Packets) flood, which harms the connections between pub/sub-like client servers, it also affects load balances and firewalls. This attack can be also volumetric, as a larger SYN flood may harm the entire IoT middleware architecture [20].

4.2.3 Application Layer Attack

Within layer 7, this type of attack attempts to deny services of authorized pub/sub by flooding architecture with a bunch of queries which are suitable as per the target environment. It also knows as Web Request Flood[17], [21].




Security Service	AWS (Amazon Web Service)	Microsoft AZURE	GCP (Google Cloud Program)
Physical Security	 Plenty of data centres ensure redundancy, availability and capacity planning	 A wide range of regions across the globe ensure redundancy and compliance	 Google Cloud GCP Data Centre ensures single failure circumvention and data residency
Authentication and Authorization	IAM (Identity and Access Management)	Single Sing-In Support	OAuth 2.0
Protection	Shield	DDoS	Cloud Armor
Data Encryption	KMS (Key Manager Service)	SSE (Storage Service Encryption)	Standard Encryption

Table 1. Security Comparison for AWS vs Azure vs GCP

4.3 Data Encryption

Amazon Web Services provides a wide range of data encryption to ensure the security and privacy of IoT Architecture. Following key benefits of AWS Encryption standard over remaining Cloud Providers.

4.3.1 S3-Encryption

Encryption at rest is supported by Amazon S3 (Simple Storage Service). Server-side encryption (SSE) can be used using customer-supplied keys (SSE-C), AWS Key Management Service (KMS), or managed keys from Amazon S3 (SSE-S3). Data is encrypted by SSE before being stored, and it is then decrypted when retrieving the data.

4.3.2 AWS Key Management Service

Pub/Sub can generate and maintain encryption keys using the managed service provided by AWS KMS. For a number of AWS services, including Amazon S3, Amazon EBS (Elastic Block Store),

Amazon RDS (Relational Database Service), and others, they manage keys using AWS KMS. They can safely create, rotate, and manage the encryption key lifecycle using AWS KMS.

4.3.3 EBS Encryption

Options for data-at-rest encryption are available with Amazon EBS volumes. It has the option of specifying a customer master key (CMK) from AWS KMS for encryption or using the standard encryption (Amazon EBS managed keys). Data will be kept secured throughout its entire lifespan thanks to encrypted EBS volumes. KMS remains the default key for EBS encryption which required KMS actions such as kms:CreateGrant (for giving permission), kms:Decrypt (for decryption of message) and kms:reEncrypt (to re-encrypt the message)[22].

4.3.4 AWS Certificate Manager

ACM enables to provision, management, and deploy SSL/TLS certificates for use with AWS services. It simplifies the process of obtaining and renewing SSL certificates, ensuring secure communication between IoT middleware architecture[23].

4.3.5 AWS CloudHSM

AWS CloudHSM (Hardware Security Module) is a dedicated hardware appliance that provides secure key storage and cryptographic operations. It allows to generation and uses encryption keys within a secure hardware environment, providing an extra layer of protection.

4.3.6 AWS Database Encryption

AWS offers encryption options for various database services like Amazon RDS, Amazon Aurora, Amazon DynamoDB, and Amazon Redshift. It encrypts data at rest using AWS KMS and encrypts data in transit using SSL/TLS.

4.3.7 AWS Transit Gateway Network Manager

AWS Transit Gateway Network Manager allows encrypting the network traffic between on-premises networks and AWS using IPsec VPN connections or AWS Direct Connect.

4.3.8 AWS CloudTrail

AWS CloudTrail provides detailed logging and auditing of API calls made within its own AWS account. It captures data such as the identity of the caller, timestamp, and details of the API request. It can enable encryption for CloudTrail logs and store them in an S3 bucket using SSE-S3 or SSE-KMS.

AWS offers a wide range of encryption options to help you protect your data both at rest and in transit, allowing you to meet your specific security requirements. It's important to consult the AWS

documentation and security best practices for detailed guidance on implementing encryption for your specific use cases.

5. Conclusion

Offering security with cloud storage continues to be difficult for all of the current cloud providers. As part of the research work, we analysed the security features of many cloud platforms, including AWS (Amazon Web platforms), Microsoft Azure, and GCP (Google Cloud Platform). Using the proposed algorithm, We also implemented the middleware security aspect using OpenSSLLibreSSL 2.8.3 on the AWS platform by offering AWS IoT Core concepts and AWS MQTT broker by applying X.509 digital certification by defining our own authentication and authorization by customizing Amazon EBS encryption standard and IAM (Identity and Access Management), own 4096-bit private key defined as Amazon KMS (Key Management Service) and AWS IoT core devices policy to the existing IoT environment. Enhanced middleware architecture also ensures not only DDoS attacks prevention but also protects against threats such as network volume attacks, network protocol attacks and application layer attacks.

6. Reference

- [1] G. Nimavat Dhaval M. and Raiyani Ashwin, "A Study on MQTT Protocol Architecture and Security Aspects Within IoT Paradigm," in *Intelligent Computing and Networking*, V. B. and K. A. Balas Valentina Emilia and Semwal, Ed., Singapore: Springer Nature Singapore, 2023, pp. 61–72.
- [2] N. Dhaval and R. Ashwin, *Study on Security Issues and Threats for MQTT with IoT Paradigm*. [Online]. Available: www.rku.ac.in
- [3] J. Kotak, A. Shah, A. Shah, and P. Rajdev, "A comparative analysis on security of MQTT brokers," in *2nd Smart Cities Symposium (SCS 2019)*, 2019, pp. 1–5. doi: 10.1049/cp.2019.0180.
- [4] J. Kotak, A. Shah, A. Shah, and P. Rajdev, "A comparative analysis on security of MQTT brokers," in *2nd Smart Cities Symposium (SCS 2019)*, 2019, pp. 1–5. doi: 10.1049/cp.2019.0180.
- [5] F. Chen, Y. Huo, J. Zhu, and D. Fan, "A Review on the Study on MQTT Security Challenge," in *2020 IEEE International Conference on Smart Cloud (SmartCloud)*, 2020, pp. 128–133. doi: 10.1109/SmartCloud49737.2020.00032.
- [6] C.-S. Park and H.-M. Nam, "Security Architecture and Protocols for Secure MQTT-SN," *IEEE Access*, vol. 8, pp. 226422–226436, 2020, doi: 10.1109/ACCESS.2020.3045441.
- [7] L. Staglianò, E. Longo, and A. E. C. Redondi, "D-MQTT: design and implementation of a pub/sub broker for distributed environments," in *2021 IEEE International Conference on*

- Omni-Layer Intelligent Systems (COINS)*, 2021, pp. 1–6. doi: 10.1109/COINS51742.2021.9524110.
- [8] K. Terada, S. Ohno, H. Mukai, K. Ishibashi, and T. Yokotani, “Proposal of MQTT distributed broker control mechanism,” in *2020 International Conference on Information Networking (ICOIN)*, 2020, pp. 402–404. doi: 10.1109/ICOIN48656.2020.9016508.
- [9] D. Selimović, A. Salkanović, and M. Tomić, “Application of MQTT Based Message Brokers for IoT Devices Within Smart City Solutions,” in *2022 45th Jubilee International Convention on Information, Communication and Electronic Technology (MIPRO)*, 2022, pp. 428–433. doi: 10.23919/MIPRO55190.2022.9803388.
- [10] H. Yujia, H. Yongfeng, and C. Fu, “Research on Node Authentication of MQTT Protocol,” in *2020 IEEE 11th International Conference on Software Engineering and Service Science (ICSESS)*, 2020, pp. 405–410. doi: 10.1109/ICSESS49938.2020.9237678.
- [11] F. A. Shodiq, R. R. Pahlevi, and P. Sukarno, “Secure MQTT Authentication and Message Exchange Methods for IoT Constrained Device,” in *2021 International Conference on Intelligent Cybernetics Technology & Applications (ICICyTA)*, 2021, pp. 70–74. doi: 10.1109/ICICyTA53712.2021.9689126.
- [12] T. K. Boppana and P. Bagade, “Security risks in MQTT-based Industrial IoT Applications,” in *2022 IEEE International Conference on Omni-layer Intelligent Systems (COINS)*, 2022, pp. 1–5. doi: 10.1109/COINS54846.2022.9854993.
- [13] F. B. Setiawan and Magfirawaty, “Securing Data Communication Through MQTT Protocol with AES-256 Encryption Algorithm CBC Mode on ESP32-Based Smart Homes,” in *2021 International Conference on Computer System, Information Technology, and Electrical Engineering (COSITE)*, 2021, pp. 166–170. doi: 10.1109/COSITE52651.2021.9649577.
- [14] E. Longo, A. E. C. Redondi, M. Cesana, and P. Manzoni, “BORDER: A Benchmarking Framework for Distributed MQTT Brokers,” *IEEE Internet Things J*, vol. 9, no. 18, pp. 17728–17740, 2022, doi: 10.1109/JIOT.2022.3155872.
- [15] R. Johari, S. Bansal, and K. Gupta, “Routing in IoT using MQTT Protocol,” in *2020 12th International Conference on Computational Intelligence and Communication Networks (CICN)*, 2020, pp. 1–5. doi: 10.1109/CICN49253.2020.9242600.
- [16] H. Kim *et al.*, “Petification: Node-RED Based Pet Care IoT Solution Using MQTT Broker,” in *2022 13th International Conference on Information and Communication Technology Convergence (ICTC)*, 2022, pp. 25–29. doi: 10.1109/ICTC55196.2022.9952423.
- [17] A. S. Suwardi Ansyah *et al.*, “MQTT Broker Performance Comparison between AWS, Microsoft Azure and Google Cloud Platform,” in *2023 International Conference on Recent Trends in Electronics and Communication (ICRTEC)*, 2023, pp. 1–6. doi: 10.1109/ICRTEC56977.2023.10111870.
- [18] A. B. M. Sultan, S. Mehmood, and H. Zahid, “Man in the Middle Attack Detection for MQTT based IoT devices using different Machine Learning Algorithms,” in *2022 2nd International*

- Conference on Artificial Intelligence (ICAI)*, 2022, pp. 118–121. doi: 10.1109/ICAI55435.2022.9773590.
- [19] S. Andy, B. Rahardjo, and B. Hanindhito, “Attack scenarios and security analysis of MQTT communication protocol in IoT system,” in *2017 4th International Conference on Electrical Engineering, Computer Science and Informatics (EECSI)*, 2017, pp. 1–6. doi: 10.1109/EECSI.2017.8239179.
- [20] L. Hao, X. Yu, T. Zhang, and H. Schulzrinne, “Distributed MQTT Brokers at Network Edges: A Study on Message Dissemination,” in *2022 IEEE International Conferences on Internet of Things (iThings) and IEEE Green Computing & Communications (GreenCom) and IEEE Cyber, Physical & Social Computing (CPSCom) and IEEE Smart Data (SmartData) and IEEE Congress on Cybermatics (Cybermatics)*, 2022, pp. 17–24. doi: 10.1109/iThings-GreenCom-CPSCom-SmartData-Cybermatics55523.2022.00044.
- [21] G. Potrino, F. de Rango, and A. F. Santamaria, “Modeling and evaluation of a new IoT security system for mitigating DoS attacks to the MQTT broker,” in *2019 IEEE Wireless Communications and Networking Conference (WCNC)*, 2019, pp. 1–6. doi: 10.1109/WCNC.2019.8885553.
- [22] R. Kawaguchi and M. Bandai, “A Distributed MQTT Broker System for Location-based IoT Applications,” in *2019 IEEE International Conference on Consumer Electronics (ICCE)*, 2019, pp. 1–4. doi: 10.1109/ICCE.2019.8662069.
- [23] G. Potrino, F. de Rango, and A. F. Santamaria, “Modeling and evaluation of a new IoT security system for mitigating DoS attacks to the MQTT broker,” in *2019 IEEE Wireless Communications and Networking Conference (WCNC)*, 2019, pp. 1–6. doi: 10.1109/WCNC.2019.8885553.