Published by :
https://www.ijert.org/
An International Peer-Reviewed Journal

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 10 , October - 2026

Secure IoT Device Management Using Attribute-Based Access Control in AWS Cloud

1st Suthan Renuka
Department of CSE
Sridevi institute of Engineering and Technology
Tumkur, Karnataka,India

2nd **Mahalakshmi** R
Department of CSE
Sridevi institute of Engineering and Technology
Tumkur, Karnataka,India

3rd Namitha G
Department of CSE
Sridevi institute of Engineering and Technology
Tumkur, Karnataka,India

4th Navyashree LV
Department of CSE
Sridevi institute of Engineering and Technology
Tumkur, Karnataka,India

5th Ayesha
Department of CSE
Sridevi institute of Engineering and Technology
Tumkur, Karnataka,India

Abstract - IoT deployments produce a steady stream of sensor data, which makes device identity, protected communication, and user authorization real security concerns. Permissions based on roles alone can be too rigid when access also depends on the operating context. This paper presents a secure IoT management prototype built on AWS IoT Core with application-level Attribute-Based Access Control (ABAC). X.509 credentials authenticate the device, MQTT over TLS protects the communication path, and ABAC checks role, department, location, access level, and device type before any sensor data is released. We also demonstrate a separate emergency rule, triggered by a configurable high-temperature event, and every access outcome is kept in an audit log.

The prototype uses a Python-generated temperature source connected to AWS IoT Core. In our functional trials, permitted users received sensor data, users outside the policy were rejected, and the simulated emergency path was able to authorize an approved healthcare role. The result is a clear split between authenticating the device and authorizing the user, with every decision still traceable.

These results come from functional demonstrations, not from a large statistical benchmark. We exercised five authorization cases, covering both normal and emergency access, and every decision we observed matched the configured policy. The work is therefore a small, reproducible security prototype for controlled IoT data access.

Index Terms - **Internet of Things, AWS IoT Core, Attribute-Based Access Control, ABAC, MQTT, TLS, X.509 Certificate, IoT Security, Cloud Security, Emergency Access Control.**

I. INTRODUCTION

The Internet of Things connects physical devices, sensors, and cloud services so that measurements can be gathered and shared remotely. It is common in healthcare, industrial monitoring, agriculture, smart environments, and home systems. That distributed nature also multiplies the endpoints and communication paths that need protecting, especially when the data being collected is sensitive.

Healthcare deployments are a good example. Devices may report temperature and other patient-related measurements, so only users whose permissions fit the situation should be able to see them. A secure design has to cover two things: the identity of the IoT endpoint and the rights of the person asking for the data.

AWS IoT Core offers managed connectivity for IoT devices and supports MQTT messaging. In this project the device proves its identity with an X.509 certificate and sends sensor data over a TLS-protected MQTT connection.

Authenticating a device, however, says nothing about which person may read its data. That is an authorization problem. RBAC can grant permissions by role, but an IoT request may also depend on department, location, device type, and access level.

ABAC handles this by evaluating several attributes together. Our prototype feeds role, department, location, access level, and device type into the normal access decision.

We also demonstrate emergency handling. The Python sensor can flag a reading as an emergency once a configurable threshold of about 40°C is reached. The subscriber then applies a dedicated emergency authorization rule instead of treating the event as an unrestricted bypass.

The contributions of this project are:

1. AWS IoT Core connectivity for the simulated sensor.
2. X.509-based device identity verification.
3. TLS-protected MQTT communication.
4. Application-level ABAC for user authorization.
5. Multi-attribute, context-sensitive access decisions.
6. Rejection of requests that fail the configured policy.
7. Controlled emergency access for permitted healthcare roles.
8. Local recording of access decisions.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 10 , October - 2026

9. An end-to-end demonstration from sensing through authorization and logging.

II. RELATED WORK

RBAC assigns permissions to predefined roles. It is simple to administer, but a role by itself may not capture the context of an IoT request. A healthcare role, for example, may still need to match a department or hospital location before sensor data is released.

ABAC offers a broader policy model by evaluating attributes of subjects, devices, resources, actions, or context. NIST SP 800-162 describes authorization as policy evaluation over such attributes. In this prototype, role, department, location, access level, and device type are the main inputs to the decision.

Cloud IoT services already provide device identity and protected communication. AWS IoT Core supports X.509 client certificates and MQTT with TLS, and the TemperatureSensor_01 Thing in our implementation relies on both to connect to the cloud and publish readings.

Existing fine-grained IoT access-control designs can bring in extra policy engines, identity services, or distributed infrastructure. These approaches may support richer deployments, but they also add implementation overhead. This project takes a more compact route: an application-level design that can be demonstrated from device connection all the way to user authorization.

III. PROBLEM STATEMENT

IoT sensors send information to cloud platforms continuously, so the device connection, the network channel, and the user-facing authorization step all need protection. The requirements behind this work are:

1. Unauthorized users should not receive protected sensor information.
2. The IoT endpoint must be authenticated before communication.
3. Sensor messages need protection in transit.
4. Role-only authorization may not be enough for contextual IoT access.
5. Several user and device attributes may need to influence a decision.
6. Emergency conditions may call for a defined exceptional-access path.
7. Security decisions should be kept for later review.

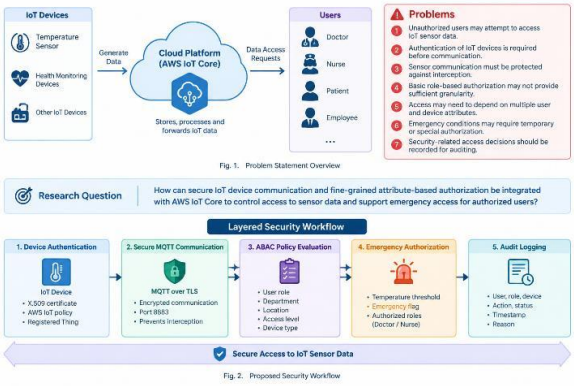
The research question is: How can authenticated IoT communication and multi-attribute user authorization be combined with AWS IoT Core while retaining controlled emergency access and decision traceability?

Our workflow answers this with certificate-based device authentication, TLS-protected MQTT, application-level ABAC, a separate emergency authorization rule, and local audit recording. It is organized as a layered security workflow in which each component has its own responsibility.

IV. PROPOSED SYSTEM

The proposed design uses AWS IoT Core for cloud connectivity and an application-level ABAC layer for user authorization. Its main components are:

1. **IoT Sensor Module**
2. **AWS IoT Core**
3. **X.509 Certificate Authentication**
4. **MQTT over TLS**
5. **ABAC Authorization Module**
6. **Secure Subscriber**
7. **Emergency Access Module**
8. **Audit Logging Module**



The processing path is:

Sensor → AWS IoT Core → MQTT/TLS → Secure Subscriber → ABAC Decision → Data Access → Audit Log

The sensor is a Python simulation. It generates temperature readings and publishes them as JSON to the MQTT topic tied to the registered IoT Thing.

The registered device is:

Thing: TemperatureSensor_01

The associated Thing Type is:

Thing Type: HealthcareSensorType

The Thing is configured with contextual attributes including:

Attribute	Implemented Value
Device Type	TemperatureSensor
Department	Healthcare
Location	Hospital_A
Sensitivity	Normal

These values give the authorization rules their device context.

V. SECURITY ARCHITECTURE

The security architecture keeps device authentication, transport protection, user authorization, emergency handling, and audit recording separate.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 10 , October - 2026

A. Device Authentication

The IoT endpoint presents an X.509 certificate when it connects to AWS IoT Core. The certificate is tied to the registered Thing and its IoT policy, which lets the cloud service authenticate the device.

B. Secure Communication

Sensor traffic travels over MQTT protected by TLS, which secures the path between the simulated device and AWS IoT Core. The secure MQTT port used in the implementation is:

8883

C. Authorization

Once the device connection is established, the application's ABAC component decides whether the requesting user satisfies the sensor-data policy.

D. Emergency Authorization

A reading marked as an emergency triggers a separate authorization check instead of granting access automatically.

E. Audit Logging

Both accepted and rejected authorization outcomes are written to the local access log.

VII. ATTRIBUTE-BASED ACCESS CONTROL

ABAC is the main mechanism for controlling human access to the sensor data. The implementation combines several attributes rather than relying on the user's role alone. The decision attributes are:

Attribute	Description
User Role	Doctor, Nurse, Patient, Employee, etc
Department	Healthcare, Finance, etc
Location	Hospital_A, Hospital_B, etc
Access Level	Standard, High, Emergency
Device Type	TemperatureSensor

A simplified form of the normal rule is:

Access = role condition AND department condition AND location condition AND valid access level

For a normal request, the application first verifies department and location, then checks whether the role is permitted for the target IoT device.

The normal healthcare roles permitted by the implementation are:

- Doctor
- Nurse
- Admin

Patient users and users from unrelated departments fail the normal policy.

Illustrative authorization cases

For TemperatureSensor_01:

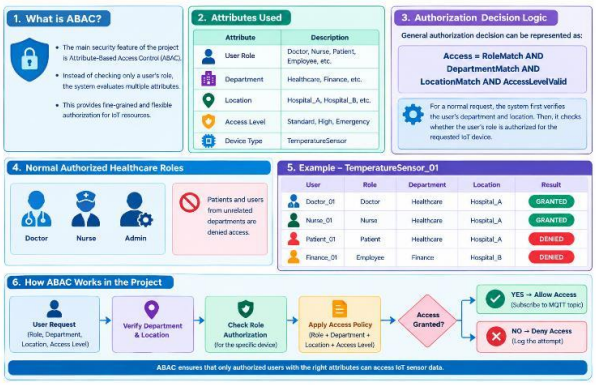
Doctor_01
- Role = Doctor
- Department = Healthcare
- Location = Hospital_A
- Result = GRANTED

Nurse_01
- Role = Nurse
- Department = Healthcare
- Location = Hospital_A
- Result = GRANTED

Patient_01
- Role = Patient
- Department = Healthcare
- Location = Hospital_A
- Result = DENIED

Finance_01
- Role = Employee
- Department = Finance
- Location = Hospital_B
- Result = DENIED

These examples show that access depends on the configured combination of attributes, not simply on whether the user exists.



Emergency access is implemented as a separate policy path. A configurable threshold of approximately 40°C is used only to simulate an emergency event in the prototype. An example event looks like this:

Temperature = 44.33°C
Emergency = True

When the threshold is met, the sensor payload carries an emergency indicator and the subscriber starts the emergency authorization check.

The emergency rule permits selected healthcare roles, including the following, to receive emergency access:

- Doctor
- Nurse

For the demonstration, the system creates an emergency user context:

User: Emergency_Doctor_01
Role: Doctor
Department: Emergency
Location: Hospital_B
Access Level: Emergency

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 10 , October - 2026

The resulting decision is:

ACCESS = GRANTED

The emergency outcome is also written to the audit log.

The threshold and emergency flag are demonstration controls, not medically validated detection criteria.

IX. IMPLEMENTATION

The prototype is implemented in Python with AWS IoT Core.

A. AWS IoT Core

AWS IoT Core serves as the managed cloud endpoint. The registered device is:

TemperatureSensor_01

The device is associated with HealthcareSensorType.

B. X.509 Certificate

An X.509 certificate is generated and associated with the IoT Thing. During connection establishment it is used together with the device private key to set up an authenticated connection.

C. AWS IoT Policy

The device uses the AWS IoT policy named TemperatureSensorPolicy.

TemperatureSensorPolicy

The policy defines the permissions attached to the device connection.

D. MQTT Topic

Sensor data is published to:

temperature/TemperatureSensor_01

A normal payload has the following structure:

```
{
    "device_id": "TemperatureSensor_01",
    "device_type": "TemperatureSensor",
    "department": "Healthcare",
    "location": "Hospital_A",
    "temperature": 29.07,
    "unit": "Celsius"
}
```

When an emergency is simulated, the payload adds an emergency field:

```
{
    "device_id": "TemperatureSensor_01",
    "device_type": "TemperatureSensor",
    "department": "Healthcare",
    "location": "Hospital_A",
    "temperature": 44.33,
    "unit": "Celsius",
    "emergency": true
}
```

E. Secure Subscriber

The subscriber may connect and subscribe only after the application's authorization check permits the request. A permitted doctor can subscribe to the temperature topic and receive sensor messages, while a patient who fails the ABAC policy is blocked before the protected subscription is granted.

F. Audit Log

The prototype keeps authorization records in the local access.log file. A normal entry contains fields such as:

User=Doctor_01
Role=Doctor
Device=TemperatureSensor_01
Action=READ
Status=GRANTED
Reason=All ABAC conditions satisfied

An emergency record contains fields such as:

User=Emergency_Doctor_01
Role=Doctor
Device=TemperatureSensor_01
Action=EMERGENCY_READ
Status=GRANTED
Reason=Emergency access granted to authorized healthcare role

X. EXPERIMENTAL RESULTS AND DISCUSSION

We evaluated the integrated demonstration through device connection, secure message publication, normal authorization, emergency authorization, rejection of an unauthorized user, and audit recording. The tests were functional; they were not meant as a large-scale latency or throughput benchmark.

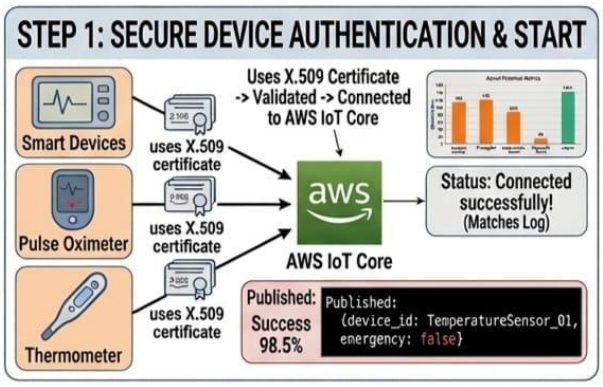


Fig. 1. Device authentication and secure publication sequence.

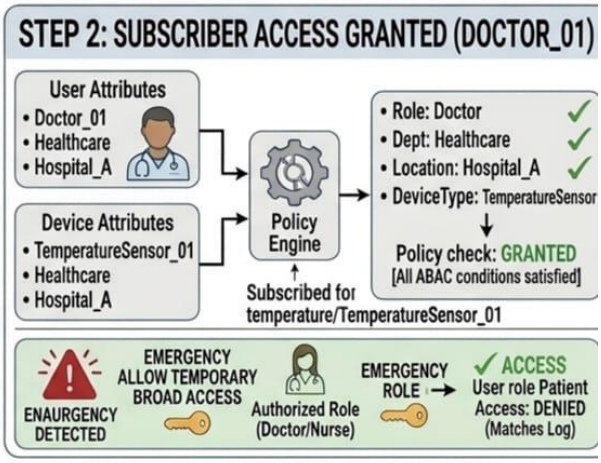


Fig. 2. ABAC evaluation leading to an authorized subscriber.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 10 , October - 2026

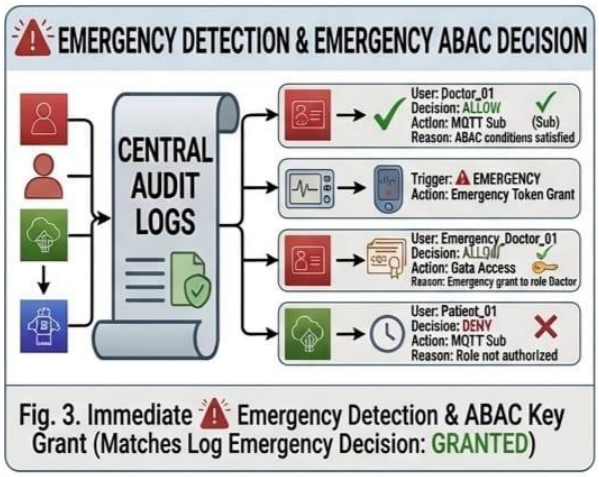


Fig. 3. Simulated emergency event followed by emergency ABAC evaluation.

Experimental Summary

We executed five authorization scenarios. Doctor_01, Nurse_01, and Emergency_Doctor_01 were granted access, while Patient_01 and Finance_01 were rejected under the configured rules. All five observed decisions matched the expected policy outcomes, a 100% decision-match rate for these test cases. This is not a statistical accuracy measure.

The complete demonstration followed this path:

IoT Sensor → X.509 Authentication → AWS IoT Core → MQTT/TLS → ABAC Evaluation → Normal or Emergency Access → Audit Logging

The local audit file records the user, role, device, action, status, and reason, which gives a basic trace of both successful and rejected requests.

Overall, the functional evaluation confirms that the components work together in the demonstration environment: the device authenticates, messages travel over the protected MQTT channel, ABAC controls subscriber access, the emergency rule applies when required, and the decision is logged.

XI. LIMITATIONS AND FUTURE WORK

The present version relies on a Python sensor simulation, application-level ABAC, a limited attribute set, a configurable simulated emergency trigger, and local audit storage. Future work can connect physical IoT hardware, introduce richer and centrally managed policies, add attributes such as time or device health, replace the simulated emergency condition with trusted sensing, move audit records to centralized cloud storage, evaluate larger device and user populations, and explore anomaly detection.

XII. CONCLUSION

This project demonstrates a practical IoT security workflow in which AWS IoT Core, X.509 device identity, MQTT over TLS, and application-level ABAC work together to control access to sensor data. The authorization policy uses role, department, location, access level, and device type. Normal users are granted or rejected according to these conditions, while the emergency path applies a separate rule for approved healthcare roles. Functional testing covered five scenarios, and the decisions matched expectations in every case. Local audit records make the important access events traceable. Within the scope of the prototype, the work shows that authenticated device communication and contextual user authorization can be combined without adding an external policy platform.

XII. COMPARATIVE RESULTS

The graphs summarize the executed authorization cases and the difference in decision attributes between role-only access control and the implemented ABAC approach. The comparison is capability-oriented, not a measured performance benchmark.

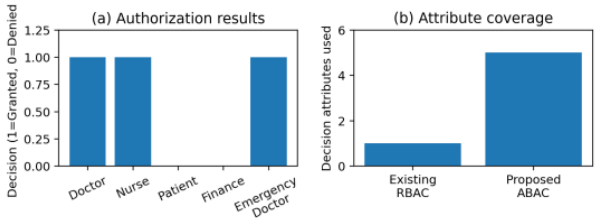


Fig. 4. Comparative results: executed authorization outcomes and attribute coverage.

XIV. REFERENCES

[1] C. Hu, D. Ferraiolo, R. Kuhn, et al., Guide to Attribute Based Access Control (ABAC) Definition and Considerations, NIST SP 800-162, 2019.

[2] Amazon Web Services, "AWS IoT Core Developer Guide," including MQTT/TLS device communication and authentication guidance, AWS Documentation, 2026.

[3] Amazon Web Services, "Authentication in AWS IoT Core," AWS Documentation, 2026.

[4] Amazon Web Services, "X.509 Client Certificates in AWS IoT Core," AWS Documentation, 2026.

[5] Amazon Web Services, "Device Communication Protocols in AWS IoT Core," AWS Documentation, 2026.

[6] OASIS, *MQTT Version 3.1.1*, OASIS Standard, Oct. 2014.

[7] E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.3," RFC 8446, Internet Engineering Task Force, Aug. 2018.

[8] M. Rose, "The Simple Network Management Protocol (SNMP)," Internet Engineering Task Force, RFC 1157, May 1990.

[9] National Institute of Standards and Technology, *Security and Privacy Controls for Information Systems and Organizations*, NIST Special Publication 800-53, Rev. 5, 2020.

[10] National Institute of Standards and Technology, *IoT Device Cybersecurity Capability Core Baseline*, NISTIR 8259A, 2020.

[11] Amazon Web Services, "AWS IoT Security Best Practices," AWS Documentation, 2026.

[12] Amazon Web Services, "Connect Devices to AWS IoT Core," AWS Documentation, 2026.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)