UNIVERSITÀ DELLA CALABRIA
DIPARTIMENTO DI
INGEGNERIA INFORMATICA,
MODELLISTICA, ELETTRONICA
E SISTEMISTICA
DIMES

Università della Calabria

Telecommunication Engineering: Smart Sensing, Computing and Networking

IoT Security

Project

"IoT Secure System"

Students

Ahamad Mahmod

Mohamad Issam Sayyaf

Academic year 2022 /2023

Table of Contents

Abstract:

This project focuses on the IoT Security Systems. The Security in IoT is vital especially because the number of connected IoT devices expected to reach 10 billion by the end of 2022 [1]. IoT security is important because it is the backbone of the internet. IoT devices are connected to each other, and they are all connected to the internet. This means that if someone hacks into one of these devices, they can hack into all of them. IoT security protects our privacy and data, and it also helps protect our homes.

The main idea of the project builds a secure IoT system that is divided into two parts: the sensing part and actuating part.

In sensing part measure the temperature and humidity, also this system has the capability to sense fire.

In actuating part, the system gives the user ability to control the heating system and contains an Alarm against fire.

The system sends data periodically to the cloud in order to process it and store it in the database. To ensure security in this system, the system used a lot of techniques to archive the security: authentication, authorization, and encryption. These procedures help the system protect the data and maintain its privacy.

1- Introduction:

The rapid proliferation of the Internet of Things (IoT) into diverse application areas such as building and home automation, smart transportation systems, wearable technologies for healthcare, industrial process control, and infrastructure monitoring and control is changing the fundamental way in which the physical world is perceived and managed.

It is estimated that there will be approximately 30 billion IoT devices by 2030. Most of these IoT devices are expected to be of low-cost and wireless communication technology-based, with limited capabilities in terms of computation and storage. As IoT systems are increasingly being entrusted with sensing and managing highly complex ecosystems, questions about the security and reliability of the data being transmitted to and from these IoT devices are rapidly becoming a major concern.

It has been reported in several studies that IoT networks are facing several security challenges [2] including authentication, authorization, information leakage, privacy, verification, tampering, jamming, eavesdropping, etc. IoT provides a network infrastructure with interoperable communication protocols and software tools to enable connectivity to the internet for handheld smart devices (smartphones, personal digital assistants [PDAs], and tablets), smart household apparatus (smart TV, AC, intelligent lighting systems, smart fridge, etc.), automobiles, and sensory acquisition systems [2].

However, the improved connectivity and accessibility of devices present major security concerns for all the parties connected to the network regardless of whether they are humans or machines. The infiltration launched by the Mirai malware on the Domain Name System (DNS) provider Dyn in 2016 through a botnet-based Distributed Denial of Service (DDoS) attack to compromise IoT devices such as printers, IP cameras, residential gateways, and baby monitors represents the fertile ground for cyber threats in the IoT domain [82]. Moreover, the cyber-attack launched at the Ukrainian power grid in 2015 targeting the Supervisory Control and Data Acquisition (SCADA) system caused a blackout for several hours and is a prime example of the gravity of resulting devastation possible through modern day attacks. The main reasons for the security challenges of current information-centric automated systems is their insecure unlimited connectivity with the internet and the non-existent access control mechanisms for providing secure and trustworthy communication. Furthermore, the problem of vulnerabilities in IoT systems arises because of the physical limitations of resource constrained IoT devices (in terms of computing power, on-board storage, and battery life), lack of consensus/standardization in security protocols for IoT, and widespread use of third-party hardware, firmware, and software. These systems are often not sufficiently secure; especially when deployed in environments that cannot be secured/isolated by other means. The resource constraints on typical IoT devices make it impractical to use very complex and time-consuming encryption/decryption algorithms for secure message communication. This makes IoT systems highly susceptible to various types of attacks [2].

For this reason, the project focuses to build a Secure IoT system, the IoT Device sends the data to Cloud "IoT AWS" using communications and security protocols.

2- IoT AWS:

AWS IoT provides cloud services that connect your IoT devices to other devices and AWS cloud services as shown in Fig (1). AWS IoT provides device software that can help you integrate your IoT devices into AWS IoT-based solutions. If your devices can connect to AWS IoT, AWS IoT can connect them to the cloud services that AWS provides [3].

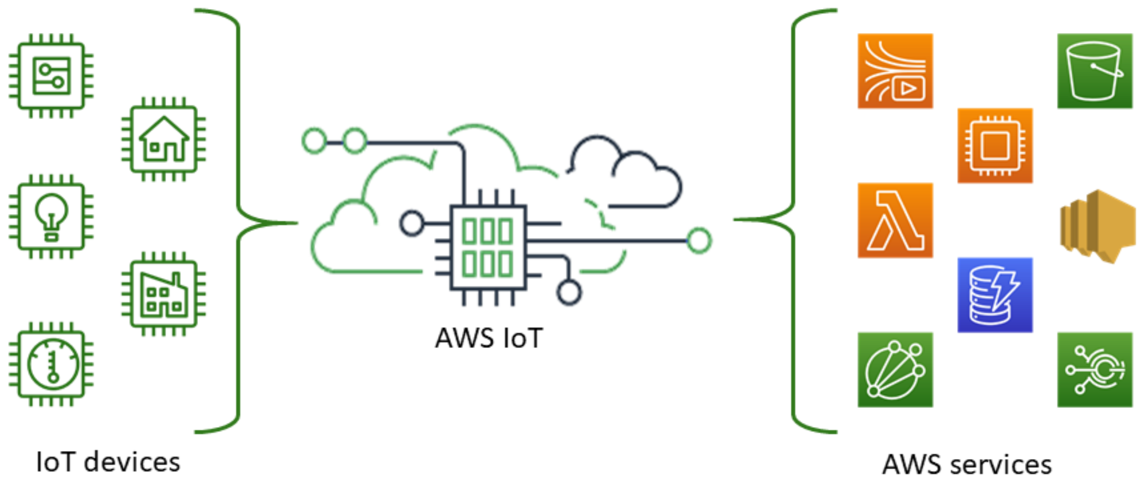


Figure 1: IoT AWS services

AWS IoT lets you select the most appropriate and up-to-date technologies for your solution. To help you manage and support your IoT devices in the field, AWS IoT Core supports these protocols:

- MQTT (Message Queuing and Telemetry Transport)
- MQTT over WSS (Websockets Secure)
- HTTPS (Hypertext Transfer Protocol - Secure)

- LoRaWAN (Long Range Wide Area Network)

The AWS IoT Core message broker supports devices and clients that use MQTT and MQTT over WSS protocols to publish and subscribe to messages. It also supports devices and clients that use the HTTPS protocol to publish messages.

AWS IoT Core for LoRaWAN helps you connect and manage wireless LoRaWAN (low-power long-range Wide Area Network) devices. AWS IoT Core for LoRaWAN replaces the need for you to develop and operate a LoRaWAN Network Server (LNS).

In general, the Internet of Things (IoT) consists of the key components shown in this Fig (2).

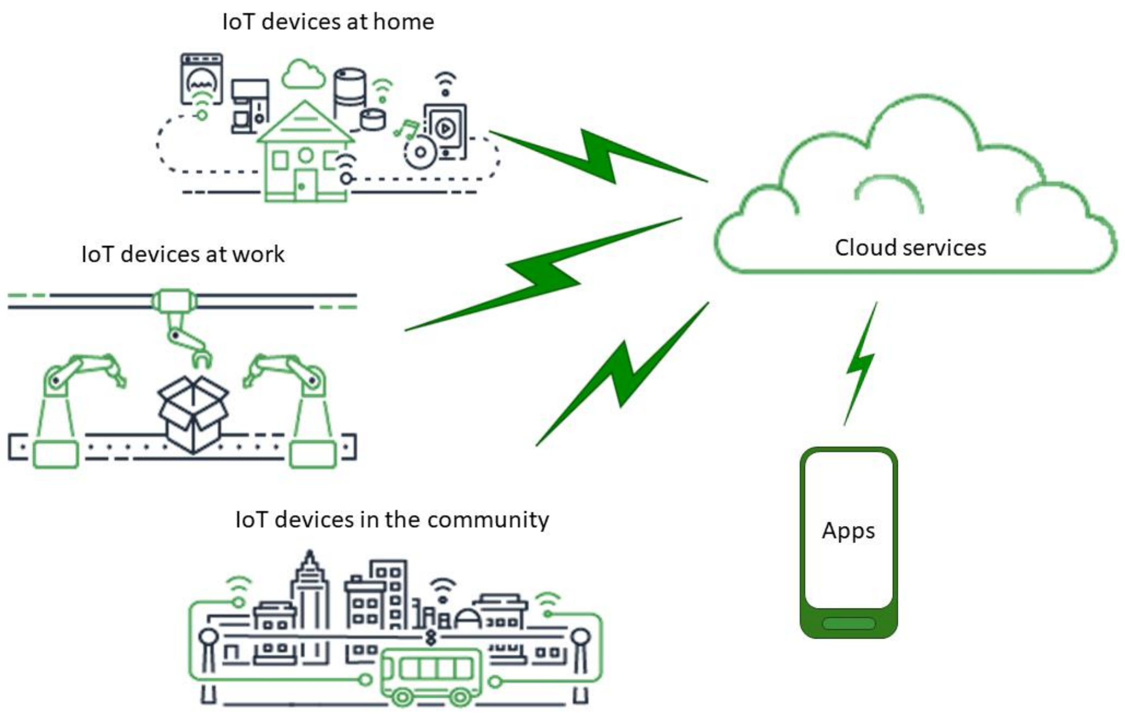


Figure 2: The main component of IoT

Cloud services:

Apps give end users access to IoT devices, and the features provided by the cloud services to which those devices are connected.

Cloud services are distributed, large-scale data storage and processing services that are connected to the internet. Examples include:

IoT connection and management services AWS IoT is an example of an IoT connection and management service.

Compute services, such as Amazon Elastic Compute Cloud and AWS Lambda Database services, such as Amazon DynamoDB.

3- Communication and Secure Protocol:

In this section will discuss the MQTT protocol as a communication protocol, and TLS protocol as Secure protocol.

3-1- MQTT Protocol:

MQTT is a publish/subscribe protocol that is lightweight and requires a minimal footprint and bandwidth to connect an IoT device shown in Fig (3). Unlike HTTP's request/response paradigm, MQTT is event-driven and enables messages to be pushed to clients. This type of architecture decouples the clients from each other to enable a highly scalable solution without dependencies between data producers and data consumers [4].

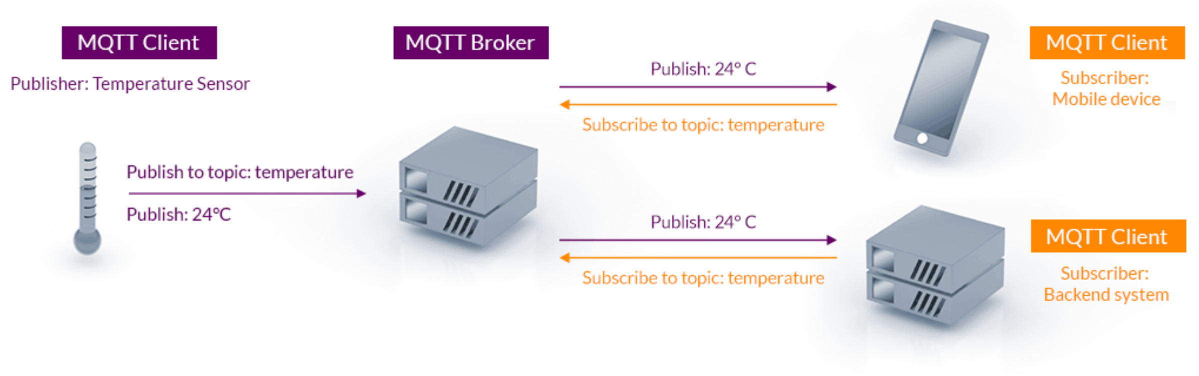

MQTT Client
Publisher: Temperature Sensor
Publish to topic: temperature
Publish: 24°C
MQTT Broker
Publish: 24° C
Subscribe to topic: temperature
Publish: 24° C
Subscribe to topic: temperature
MQTT Client
Subscriber: Mobile device
MQTT Client
Subscriber: Backend system

Figure 3: MQTT Publish / Subscribe Architecture

The key benefits of MQTT are:

1. Lightweight and efficient to minimize resources required for the client and network bandwidth.
2. Enables bidirectional communication between devices and servers. Also, enabling broadcasting messages to groups of things.
3. Scales to millions of things.

4. Specifies Quality of Service (QoS) levels to support message reliability.
5. Supports persistent sessions between device and server that reduces reconnection time required over unreliable networks.
6. Messages can be encrypted with TLS and support client authentication protocols.

At the core of MQTT is the **MQTT broker** and the **MQTT clients**. The broker is responsible for dispatching messages between the sender and the rightful receivers. An MQTT client publishes a message to a broker and other clients can subscribe to the broker to receive messages. Each MQTT message includes a topic. A client publishes a message to a specific topic and MQTT clients subscribe to the topics they want to receive. The MQTT broker uses the topics and the subscriber list to dispatch messages to appropriate clients.

An MQTT broker is able to buffer messages that can't be dispatched to MQTT clients that are not connected. This becomes very useful for situations where network connections are unreliable. To support reliable message delivery, the protocol supports 3 different types of service messages: 0 - at most once, 1 - at least once, and 2 - exactly once.

3-2- SSL/TLS Protocol:

SSL (Secure Socket Layer) and TLS (Transport Layer Security) are popular cryptographic protocols that are used to imbue web communications with integrity, security, and resilience against unauthorized tampering. PKI uses the TLS protocol to establish secure connections between clients and servers over the internet, ensuring that the information relayed is encrypted and unable to be read by an external third party [5].

In theory, web connections are completely possible without TLS to secure them. However, without a security protocol in place, the communication would be rendered completely open to external access. If a browser connected to the website of an online store, and a user had to enter their credentials to log in, those credentials could easily be lifted by an observing party.

TLS, at its core, serves to provide end-to-end encryption for all data transmitted from one point to another, and uses cryptography to ensure that only the two transacting bodies are capable of reading this information. Every service in the world now mandates that connections are secure by TLS – leading browsers do not allow users to access websites without a valid TLS connection.

TLS has the following benefits:

- The contents of the connection remain encrypted, private, and fully secure – and cannot be easily deciphered by malicious actors.
- The connection is only made if it is reliable – this reliability check is a part of TLS communications, and is enforced by the exchange of a Message Authentication Code.
- The use of PKI and TLS certificates ensures that the identities of both communicating parties are verified.

3-2-1- The TLS Handshake

When two systems that leverage TLS attempt to connect, each system will make an effort to verify that the other supports TLS. This process is called the TLS handshake, and it is here that both parties decide upon the TLS version, encryption algorithm, cipher suite etc. that will be used in the procedure. Once a TLS handshake has been successfully executed, both systems start exchanging data on a secure line.

Note: A working knowledge of PKI and its constituents, such as keys, may prove to be useful prior to understanding TLS. The encryption and decryption are carried out with the help of cryptographic devices called keys. In public key cryptography, Public keys are used to encrypt information, while secret Private keys can be used to decrypt that information. Since two different keys are involved, this technique is called ‘asymmetric cryptography’, as opposed to ‘symmetric cryptography’, where a single key can perform both encryption and decryption. More on this below.

Every TLS handshake follows the same basic steps as shown in Fig (4). For the sake of simplicity, let’s assume a browser (a client) is attempting to connect to a server, which hosts a website:

- The client requests the server to open a secure line, and the server responds by displaying a list of TLS versions and cipher suites it is compatible with. Once they agree upon common ones to use in the transaction, they begin the handshake.
- The server sends a copy of its public key, attached to its digital certificate, to the client. The client checks the certificate to verify that the server is legitimate, and if it is, proceeds with the transaction.
- The client uses the server’s public key and its private key to encrypt a ‘session key’ which is a key that will be used by both parties to encrypt and decrypt information in this particular session. The session key becomes invalid as soon as the connection is terminated.

- Both parties test the connection by sending each other encrypted messages. If the other can decrypt them using the session key, the connection has been successfully secured.

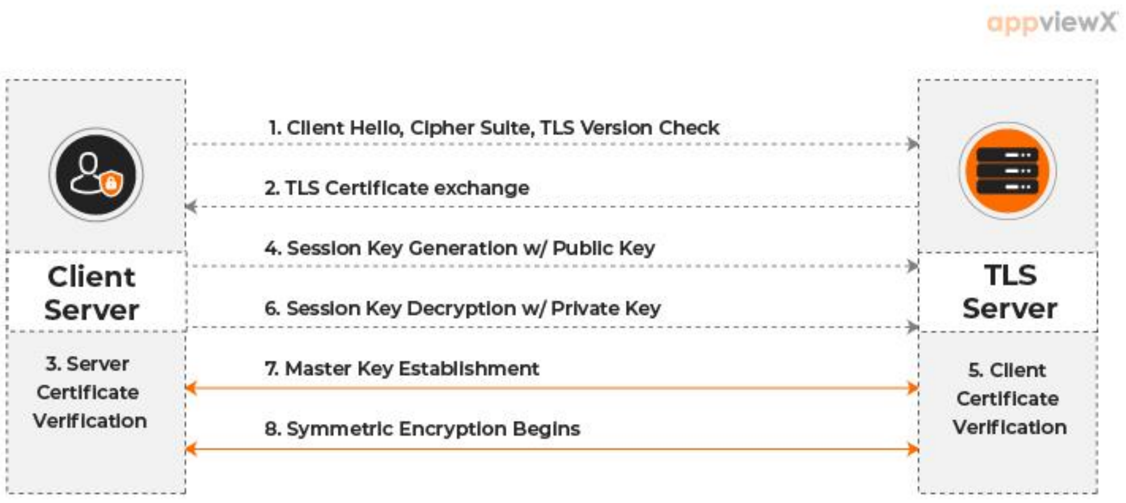

appviewX
Client Server
3. Server Certificate Verification
1. Client Hello, Cipher Suite, TLS Version Check
2. TLS Certificate exchange
4. Session Key Generation w/ Public Key
6. Session Key Decryption w/ Private Key
7. Master Key Establishment
8. Symmetric Encryption Begins
TLS Server
5. Client Certificate Verification

Figure 4: TLS handshake steps

3-2-2- Asymmetric vs. Symmetric Encryption

The above example was an instance where both asymmetric and symmetric encryption was used to secure a connection.

Asymmetric encryption was used to create the session key. But from that point onwards, the session key was used to bilaterally encrypt and decrypt the entire information flow between both parties. Here's why:

Asymmetric encryption is mathematically resource-intensive. The encryption-decryption operations involving two keys from both parties takes a heavy toll on the processing unit powering this process. If a system were configured to handle an entire connection this way – by using a public key to encrypt and a private key to decrypt it – it would probably give out within the first few minutes. It's much more secure than its symmetric counterpart, but cannot be facilitated in an efficient manner.

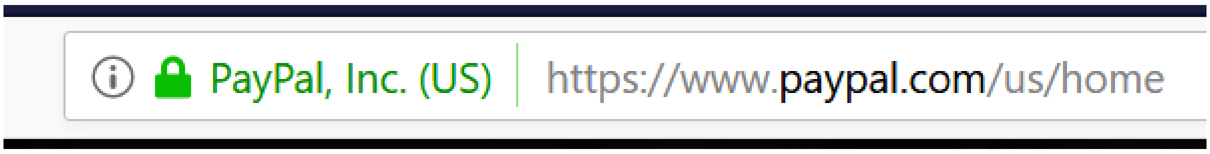
However, symmetric encryption, which makes use of a single, shared key to encrypt and decrypt, is not that resource-intensive. This is precisely why asymmetric encryption is used to establish a secure link between two parties, and used to generate the session key which, in theory, only those two parties can possibly know about. Now, symmetric encryption can be used to secure the connection, given the additional layer of security added to it by the first step.

3-2-3- SSL/TLS certificates

Digital certificates were mentioned earlier. In general, digital certificates are digital documents that are 'signed' by trusted authorities, and act as documents of ownership of a public key. By extension, they serve to validate the legitimacy of a server or a client. However, there are several types of digital certificates available. The term 'x.509 certificates' is used to differentiate SSL/TLS certificates with other kinds of digital certificates (code-signing certificates, for instance).

As the previous section made clear, digital certificates are important pieces in the public key cryptography domain. They're attached to public keys, and are proof that the holder of the public key is actually the legitimate owner. This is because digital certificates are signed, sold, and issued by bodies called 'Certificate Authorities' (or CAs, for short), which are trusted bodies responsible for verifying the authenticity of anyone who requests a certificate. Since major operating systems and browsers have 'trust stores' consisting of these CAs built into them, browsers will automatically trust certificates issued by major CAs.

Certificates are key to making websites easily recognizable to users as a trusted, secure page. Webpages with valid SSL/TLS certificates installed on them will have 'https' preceding the name of the website in the search bar, given that the certificate has been installed correctly. In some browsers, the use of a valid Extended Validation certificates (a type of TLS certificate), will cause the HTTPS padlock to turn green, providing visitors with additional assurance that they are on a legitimate website, like so:



Note: The certificates issued by CAs which are used to secure web connections that use TLS are called TLS certificates – this term is interchangeably used with 'SSL certificates' due to the term being coined back when SSL was in mainstream use.

4- IoT Device:

An embedded device system generally runs as a single application. However, these devices can connect through the internet connection, and able communicate through other network devices as shown in Fig (5).

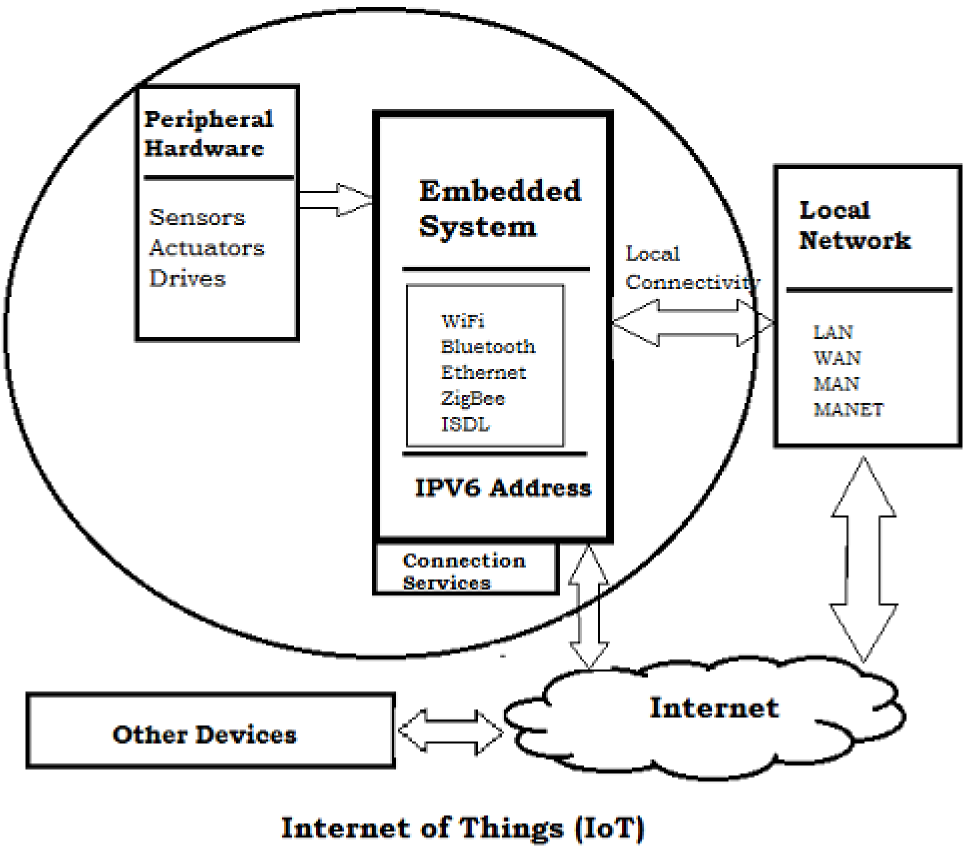


Figure 5: IoT Embedded system device

The embedded system can be of type microcontroller or type microprocessor. Both of these types contain an integrated circuit (IC).

The essential component of the embedded system is a RISC family microcontroller like Motorola 68HC11, PIC 16F84, Atmel 8051 and many more. The most important factor that differentiates these microcontrollers with the microprocessor like 8085 is their internal read and writable memory. The essential embedded device components and system architecture are specified below as shown in Fig (6).

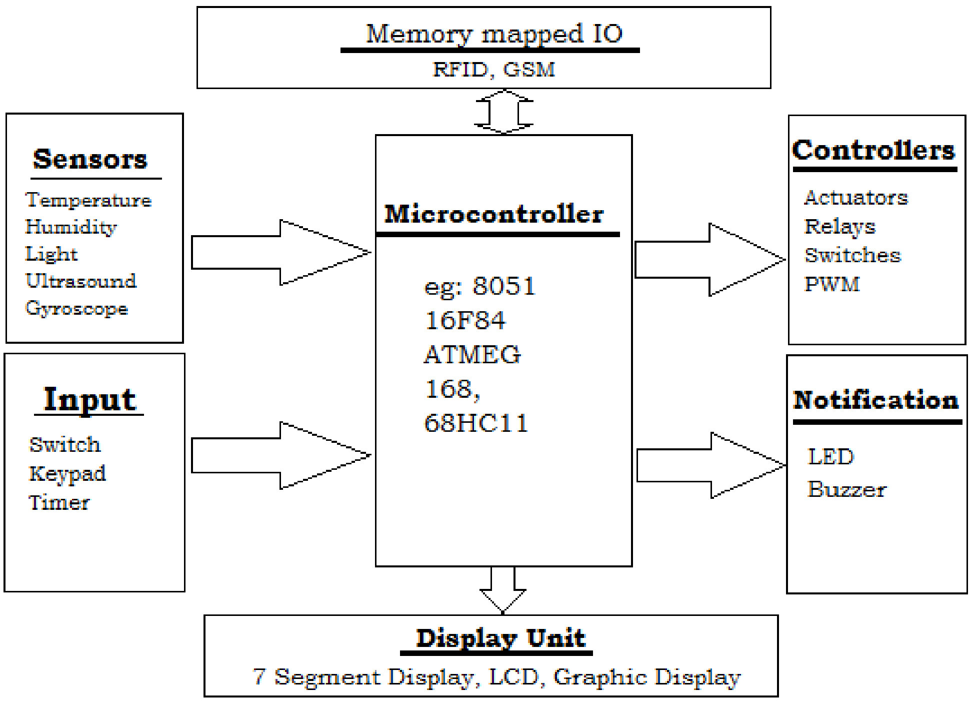


Figure 6: Embedded device components and system architecture

5- The project:

5-1- Main Idea of the project:

The main idea of the project as shown Fig (7) measures the temperature and humidity, in addition to monitoring the fire alarm.

The project works to send the data to the cloud periodically to register this data in the database and process the data if the fire alarm is activated the cloud will send a notification through SMS/email to the user.

The project gives the capability to the user for monitor the system and control of the heating it remotely using node-red.

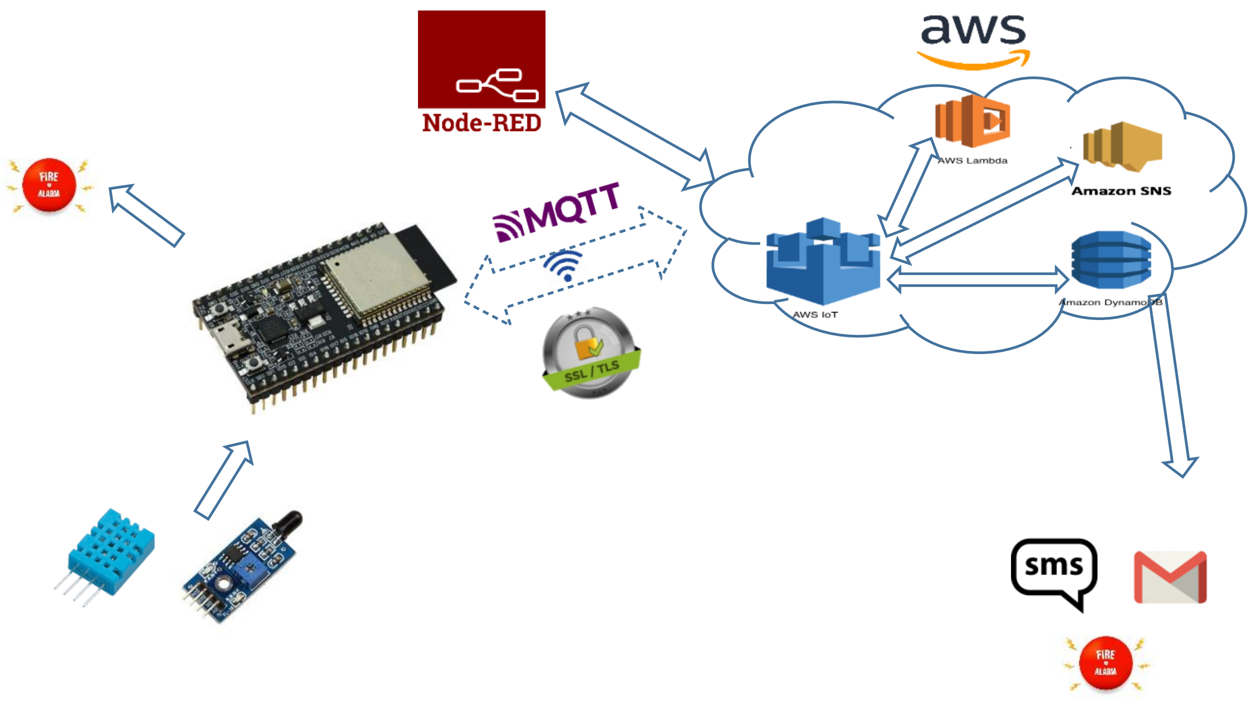
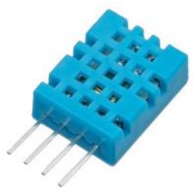


Figure 7:The main idea of the project

5-2- Components:

The main components that are used in the project shown in table 1.

Table 1: Main components of the project

Name	Description	Figure
ESP32	ESP32 is a series of low-cost, low-power system on a chip microcontroller with integrated Wi-Fi and dual-mode Bluetooth. The ESP32 series of microcontrollers are based on the ARM Cortex-M0 and M4 microprocessor cores. They are designed for use in small, low-power devices, and are particularly well-suited for use in Internet of Things (IoT) applications. Some features of the ESP32 include: • Support for 802.11b/g/n Wi-Fi • Support for Bluetooth 4.2 and Bluetooth 5 • A wide range of peripherals, including ADC, PWM, I2C, I2S, UART, and more • On-chip security features, including secure boot and flash encryption	
DHT11	The DHT11 sensor measures both temperature and humidity and can communicate the data to a microcontroller using a single wire digital interface. The DHT11 has a relatively low accuracy compared to other temperature and humidity sensors, with a temperature accuracy of +/- 2 degrees Celsius and a humidity accuracy of +/- 5% relative humidity. Despite	

	this, it is still a popular choice for many projects due to its low cost and ease of use. To use the DHT11 sensor with a microcontroller, you will need to connect it to one of the microcontroller's digital input/output (I/O) pins, and use a library to communicate with the sensor and read the temperature and humidity data.	
Flam Sensor	A flame sensor is a device that is used to detect the presence of a flame or fire. Infrared flame detectors: These sensors use an infrared (IR) sensor to detect the presence of a flame by measuring the IR emissions of a fire.	

5-3- Workflow:

The state machine that describe the work of esp32 is shown in Fig (8)

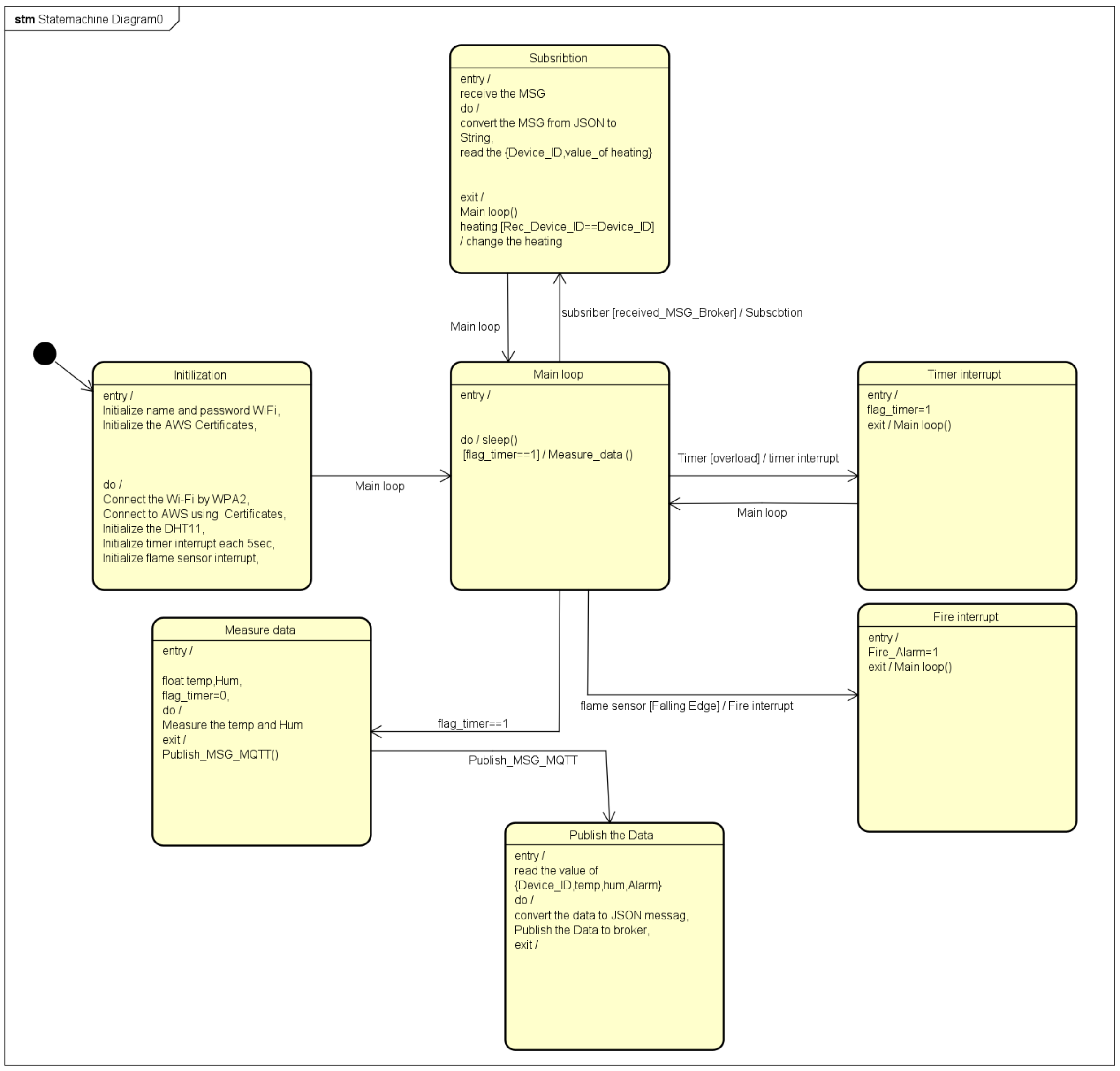

stm Statemachine Diagram0
Subsribtion
entry / receive the MSG
do / convert the MSG from JSON to String, read the {Device_ID,value_of heating}
exit / Main loop() heating [Rec_Device_ID==Device_ID] / change the heating
Main loop
subsriber [received_MSG_Broker] / Subscbtion
Initilization
entry / Initialize name and password WiFi, Initialize the AWS Certificates,
do / Connect the Wi-Fi by WPA2, Connect to AWS using Certificates, Initialize the DHT11, Initialize timer interrupt each 5sec, Initialize flame sensor interrupt,
Main loop
Main loop
entry /
do / sleep() [flag_timer==1] / Measure_data ()
Timer [overload] / timer interrupt
Main loop
Timer interrupt
entry / flag_timer=1
exit / Main loop()
Fire interrupt
entry / Fire_Alarm=1
exit / Main loop()
flame sensor [Falling Edge] / Fire interrupt
Measure data
entry / float temp,Hum, flag_timer=0,
do / Measure the temp and Hum
exit / Publish_MSG_MQTT()
flag_timer==1
Publish_MSG_MQTT
Publish the Data
entry / read the value of {Device_ID,temp,hum,Alarm}
do / convert the data to JSON messag, Publish the Data to broker,
exit /

Figure 8: State machine of the project

5-4- Security Aspects:

First level of security: The esp32 connect with Wi-Fi using WPA2 protocol, So the connection between node and Access point is encrypted and secure.

Second level of security: The connection between the node and AWS server is secure by using SSL/TLS with MQTT for this reason the node uses 3 certificate to make authentication and encryption.

Third level of security: The policies of the certificate are restricted in the Amazon So these certificates do not have the right to access to different services.

Fourth level of security: The monitor data is done by Node-red, also the access to node-red is secured by username and password.

6-Results:

The system was successfully able to measure temperature, humidity and detect fire as intended in the sensing part. The hardware of the system is shown in Fig (9).

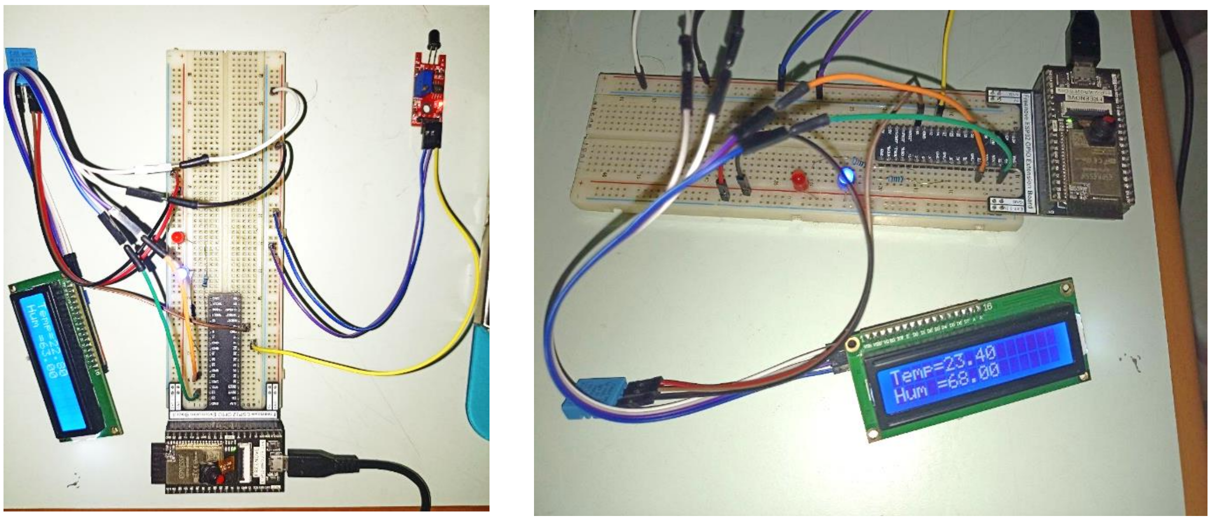


Figure 9: Hardware of the system

In Fig 10, you can observe the security aspect of the node-red. The node-red prompts the user to enter their username and password to grant them access to the dashboard. The figure also displays the data of the system on the dashboard.

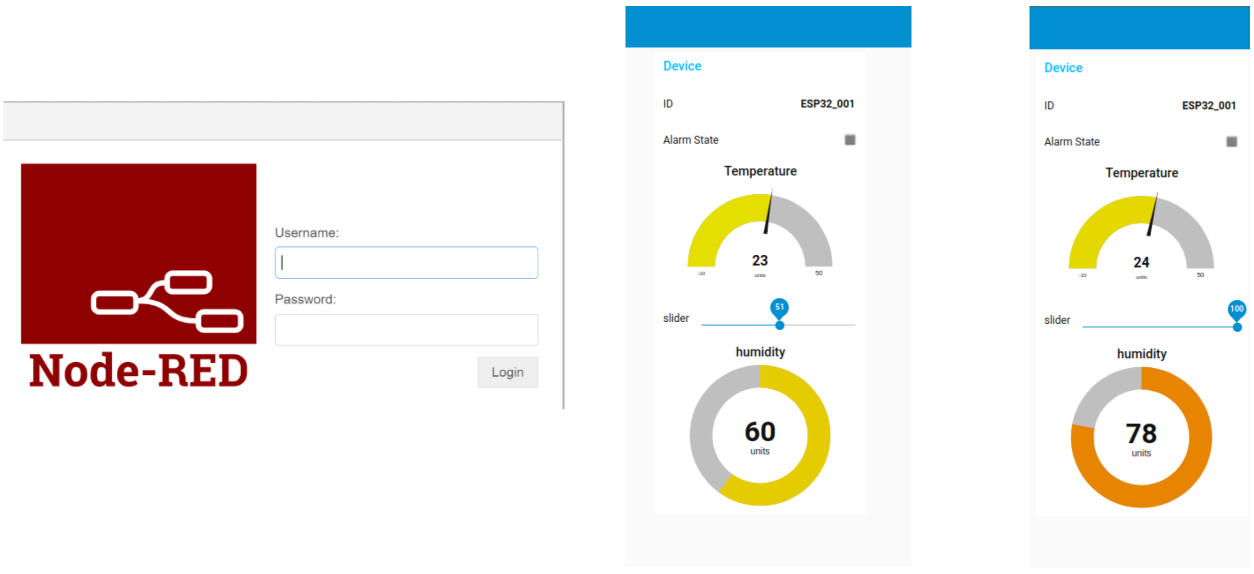
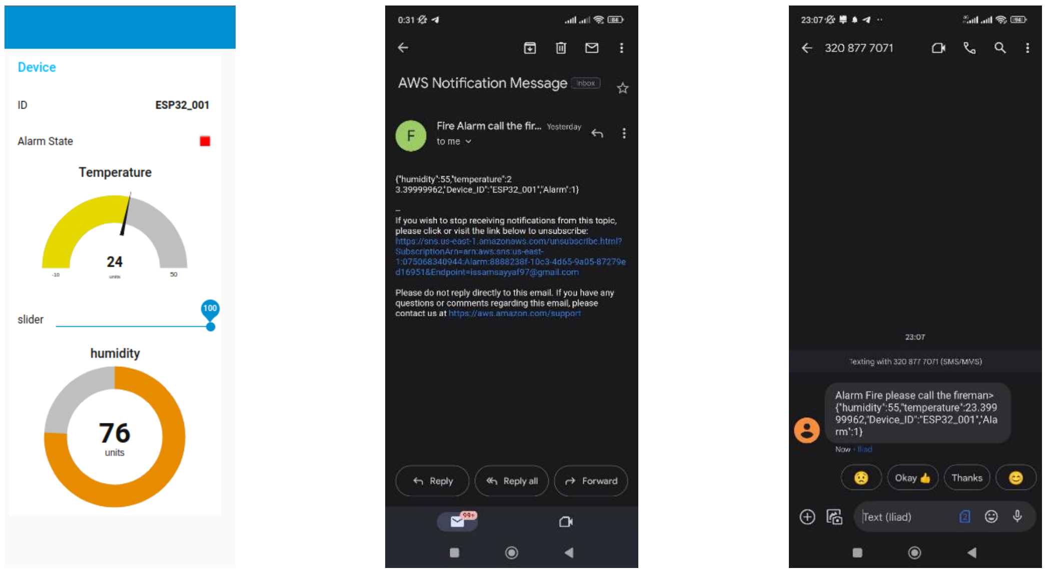


Figure 10: Node-red Dashboard

When the fire sensor detects a fire, it will trigger an alarm, causing the system to activate the alarm LED on the circuit and display an alarm on the dashboard, as well as sending SMS and email notifications to alert of the fire, as depicted in Fig (11).



AWS Notification Message Inbox

Fire Alarm call the fir... Yesterday
to me

{"humidity":55,"temperature":23.39999962,"Device_ID":"ESP32_001","Alarm":1}

--
If you wish to stop receiving notifications from this topic, please click or visit the link below to unsubscribe:
https://sns.us-east-1.amazonaws.com/unsubscribe.html?SubscriptionArn=arn:aws:sns:us-east-1:075068340944:Alarm:8888238f-10c3-4d65-9a05-87279ed16951&Endpoint=issamsayyaf97@gmail.com

Please do not reply directly to this email. If you have any questions or comments regarding this email, please contact us at https://aws.amazon.com/support

320 877 7071

23:07

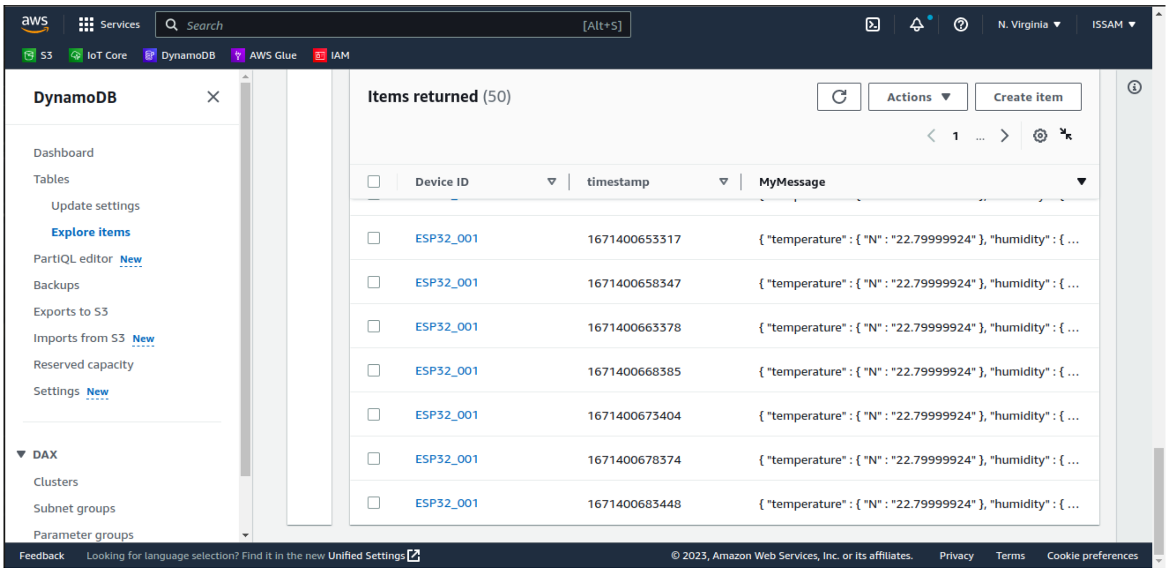
Texting with 320 877 7071 (SMS/MMS)

Alarm Fire please call the fireman> {"humidity":55,"temperature":23.39999962,"Device_ID":"ESP32_001","Alarm":1}

Now · Iliad

Figure 11: Alarm response

In this project, the sensing data and all information related to the system is stored in a database hosted on Amazon Web Services (AWS). As shown in Figure 12, this allows for easy access to the data, as well as providing a secure and reliable platform for storing the data long-term. The use of a cloud-based database also enables real-time monitoring and analysis of the system data, providing valuable insights into the performance and behavior of the system.



Items returned (50)

Device ID	timestamp	MyMessage
ESP32_001	1671400653317	{ "temperature" : { "N" : "22.79999924" }, "humidity" : { ...
ESP32_001	1671400658347	{ "temperature" : { "N" : "22.79999924" }, "humidity" : { ...
ESP32_001	1671400663378	{ "temperature" : { "N" : "22.79999924" }, "humidity" : { ...
ESP32_001	1671400668385	{ "temperature" : { "N" : "22.79999924" }, "humidity" : { ...
ESP32_001	1671400673404	{ "temperature" : { "N" : "22.79999924" }, "humidity" : { ...
ESP32_001	1671400678374	{ "temperature" : { "N" : "22.79999924" }, "humidity" : { ...
ESP32_001	1671400683448	{ "temperature" : { "N" : "22.79999924" }, "humidity" : { ...

Figure 12: AWS database.

Conclusion:

In conclusion, the project aimed to develop a secure IoT system for collecting and storing data from IoT devices. The system was able to measure temperature, humidity and detect fire, and provided a user interface for controlling the heating system and activating the fire alarm. The system used various security measures, such as authentication, authorization, and encryption, to ensure the privacy and security of the data being transmitted to the cloud. The data was stored in a database hosted on Amazon Web Services. The sending data to AWS is done by using secure MQTT protocol with SSL/TLS encryption.

The results of this project demonstrate the feasibility of creating a secure IoT system, highlighting the importance of implementing robust security measures in IoT devices. This project serves as a model for future IoT security solutions and provides valuable insights into the challenges and opportunities in this field. The project demonstrates the potential for further research in the area of IoT security, to continue improving the security and reliability of IoT systems.

References:

[1]. https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/

[2]. Jurcut, Anca & Ranaweera, Pasika & Xu, Lina. (2019). Introduction to IoT Security. 10.1002/9781119471509.w5GRef260.

[3]. https://aws.amazon.com/iot/

[4]. https://mqtt.org/

[5]. https://www.appviewx.com/education-center/what-is-tls-ssl-protocol/

[6]. https://nodered.org/

[7]. https://www.arduino.cc/

[8]. https://www.espressif.com/en