

IRELE

Mobile and wireless network

Project report

Authors:

Arico Amaury

Mototani Yuta

Tchakounte Loic

Sipakam Cedric

Professor:

JM. Dricot

TA:

Ladner Navid

Academic year:

2024 - 2025

1	Introduction and concept	1
1.1	Device	1
1.2	Concept	1
2	Threat Model	2
2.1	Data flow diagram	2
2.2	STRIDE Threat Model	3
3	Security Mitigation	5
3.1	Security scheme	5
3.2	Results	6
3.2.1	Morse Code	7
3.2.2	PBKDF2 encryption	7
3.2.3	Encrypted data publishment	8
3.2.4	Decryption at the subscriber	8
3.2.5	Port Knocking	9
3.3	Discussion	9
3.4	Conclusion	10

1.1 Device

1.2 Concept

This is what I wrote in the Threat model section - you can take it as reference to explain the concept more in details with maybe a figure.

In the first chapter, we discussed about the project concept where one ESP32 is used as a data publisher mimicking the data collection of a room we would like the thermal conditions to be controlled via a command center, which is the alias for our second ESP32.

The acquired data are sent from our ESP32 "publisher" to a MQTT server, installed on our machine for the example. The second ESP32 is subscribed to the MQTT server, receiving the published data and in charge of monitoring - controlling the room conditions. Those data are displayed on another MQTT server able to plot data evolution.

2.1 Data flow diagram

In the first chapter, we discussed about the project concept where one ESP32 is used as a data publisher mimicking the data collection of a room we would like the thermal conditions to be controlled via a command center, which is the alias for our second ESP32.

The acquired data are sent from our ESP32 "publisher" to a MQTT server, installed on our machine for the example. The second ESP32 is subscribed to the MQTT server, receiving the published data and in charge of monitoring - controlling the room conditions. Those data are displayed on another MQTT server able to plot data evolution. **Speak about TCP protocol ? - first handshake**

The data flow is shown on the figure 2.1

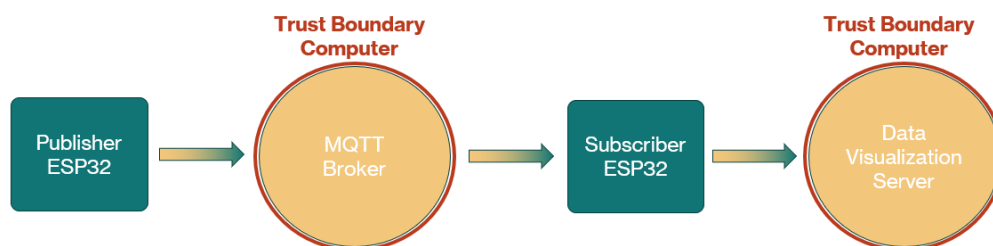


Figure 2.1: Data flow diagram

For completeness, we will consider our data as highly sensitive - with strict security needs - where the room thermal regulation cannot be vulnerable to seizing control attacks or unstablizing temperature attacks for example.

We would therefore need to mitigate all types of attack by integrating a security strategy considering the ressources constraints dependent on the hardware used - in our case, the ESP32 devkit with :

1. 4MB flash memory
2. 520KB RAM
3. Dual-core 32-bit processor

Thus HTTPS / TLS protocol are not adapted for our devices and key cryptography containing complex mathematical operation like RSA cannot be implemented. This suitable security strategy counter acting the whole span of threats while fitting into the hardware and meeting the decyphering process speed will be discussed in the last chapter namely the "Security mitigation".

Before treating about the security strategy, we would need to identify the security threats our model needs to deal with. The STRIDE model will be used in this perspective in the next section.

Figure 2.2

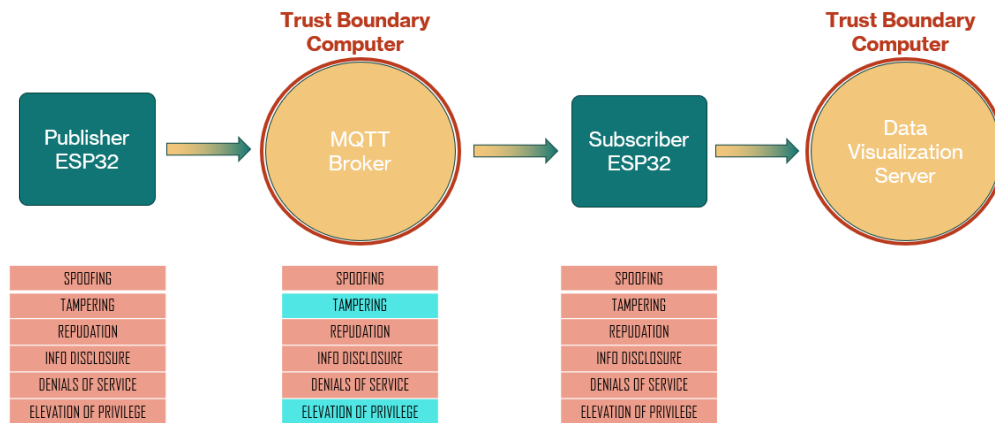


Figure 2.2: STRIDE model verification for project data flow

2.2 STRIDE Threat Model

Take back the STRIDE model used in the security powerpoint and explain each category

1. Spoofing
2. Tampering
3. Repudiation
4. Info Disclosure
5. Denials of service
6. Elevation of privilege

Figure 2.3

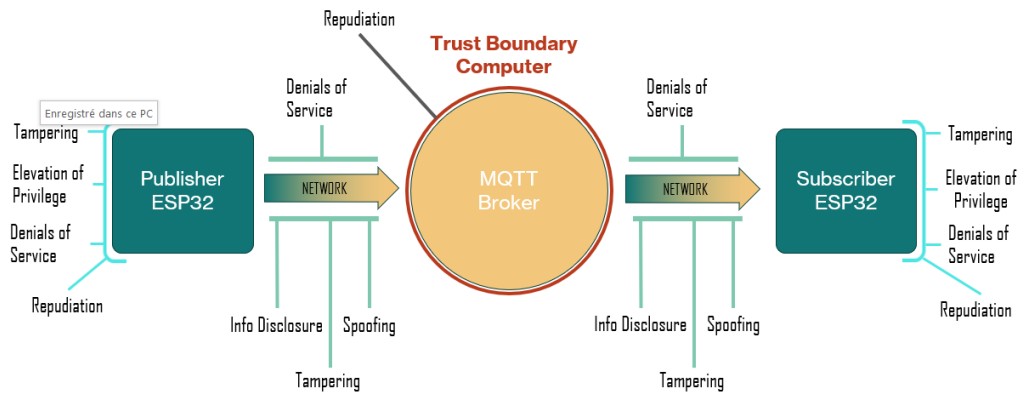


Figure 2.3: STRIDE model and threats localization

Discuss about

1. Hardware being crypted (assumption)
2. Key being encrypted with usage of morse code through PBDFK2 (Additional feature !!) and saved in flash
3. Decryption of the key with morse code before sending payload to get back the plain key
4. Encryption of payload with AES-128 bits encryption (low ressources demand)
5. Data sent to MQTT server (assumed to have a secured hardware and files secured too in the OS)
6. HMAC generation for ensuring authentication (at MQTT server - assumption) at the subscriber
7. Replay detection verification with counter (at MQTT server - assumption) at the subscriber
8. Port knocking implementation to complexify the sniffing (changing port)

3.1 Security scheme

Figure 3.1

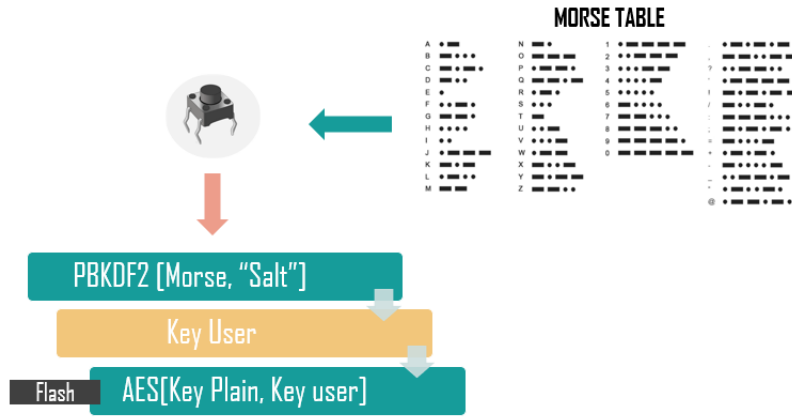


Figure 3.1: Encrypted key generation

Figure 3.2

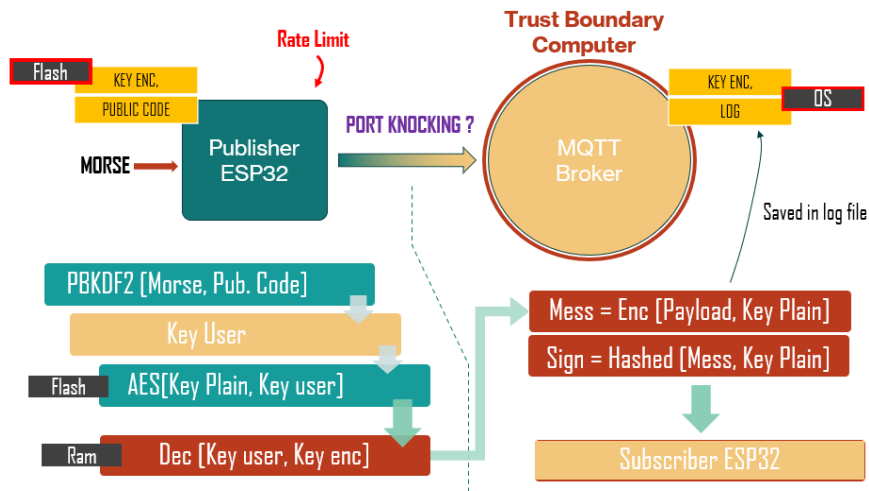


Figure 3.2: Security strategy integration in the project data flow

3.2 Results

In this section, we will review the results obtained during the implementation of the thermal regulation concept - previously discussed - with two ESP32 communicating through a MQTT server in a fully secured environment (authentication, confidentiality and integrity).

We will proceed step by step by confirming the implementation of :

1. The morse code acquisition with debouncing function for security
2. PBKDF2 key generation and decryption of the encrypted key in the flash memory
3. Encryption of the payload with the decrypted key
4. Decryption of the payload at the subscriber

5. Replay detection verification
6. Port knocking verification

3.2.1 Morse Code

The morse code acquisition passes by the implementation of a look-up table translated unique succession of **dots** and **dashes** to an alphabetic letter. These **dots** and **dashes** being generated depending on the duration when the push button has been pressed. Just to ensure the translation code was correctly implemented, the code compares the morse code encoded and the one we should use to decrypt the key stored in the flash. Of course, this verification is not done in practice as it would mean the morse code is stored in the flash memory which could be an additional break in our security strategy - morse code is supposed to be kept secret and shared verbally.

Figure 3.3 shows the results of the last letter encoding (and the the verification of the morse code).

```

Password so far: ACY
Duration: 140
Dot
Duration: 740
Dash
Duration: 138
Dot
Duration: 131
Dot
Translating: .-..
→ Added: L
Password so far: ACYL
✓ PASSWORD ACCEPTED: ACYL

```

Figure 3.3: Morse code implementation Result

3.2.2 PBKDF2 encryption

The following security phase is the decryption of the encrypted key saved in the non-volatile memory through the PBKDF2 encryption and AES decryption. The figure 3.4 shows the following results :

1. Morse code encoded with the push button
2. Derived 16-bytes key generated by the PBKDF2 algorithm (Morse, "Salt", HMAC 256, 1000 iterations)
3. Encrypted 32-bytes key saved in the flash - $\text{Key}_{\text{enc}} = \text{AES}(\text{IV code}, \text{Key}_{\text{plain}})$
4. Decryption of the key - $\text{Key}_{\text{plain}} = \text{Dec}(\text{IV code}, \text{Key}_{\text{enc}})$
5. $\text{Key}_{\text{plain}} = 0x54, 0x68, 0x61, 0x74, 0x73, 0x20, 0x6d, 0x79, 0x20, 0x4b, 0x75, 0x6e, 0x67, 0x20, 0x46, 0x75$ - **Thats my Kung Fu** in 16 ASCII

```

--- PBKDF2 Key Derivation Started (REAL) ---
Password: ACYL
PBKDF2 Key Derivation Complete (Success).
Derived Key: 74257CEC0BFA9DE16A21364B72A29DF1
Print plain key : 5468617473206D79204B756E67204675

```

Figure 3.4: PBKDRF2 encryption verification

3.2.3 Encrypted data publishment

Once the plain key (Key_{plain}) has been generated, the thermal data can be encrypted with AES-128 bits encryption. The figure 3.5 shows the details for the encryption functionality and the results of the encrypted payload.

1. Thermal conditions gathered in a java script object notation (readable text to send structured data)
2. IV and plain key shown for transparency in **Arduino String**
3. Hmac and cypher payload generation with AES-128 bits encryption.
4. Publishment of the data on the MQTT server after client connexion on MQTT server on dedicated port.

```

***** PUBLISMENT OF ENCRYPTED DATA TO MQTT SERVER*****
Sending json doc{"temp":22.79999924,"hum":50,"heat":22.4411087,"dew":11.85450268,"comfort":100}
Check AES_IV key : 4NZ8IXQ0aKz9HmFMIOeoBg==
Check AES key : VGhhdHMgbXkgS3VuZyBGdQ==
HMAC => OG4l14VdqCc0ZKpwfE3H7NDByHTGygIfdFDSRF8u7dc=
Encrypted Data => 4NZ8IXQ0aKz9HmFMIOeoBgEAAAC1i3xyD4qPgAwv0c/00RnuRvYQ7Jz1INyr/Z7aDZH3evdjYjkt6Gx

```

Figure 3.5: Encrypted data publishment on MQTT server

3.2.4 Decryption at the subscriber

The second ESP32 subscribed to the same MQTT server - listening to the same port as the publisher - receives the cypher message containing HMAC signature and encrypted payload. From the inverse encryption process, the cypher is decrypted and the result details are shown in the figure 3.6 and 3.7.

1. Encrypted message and HMAC signature sent to the subscriber
2. Payload decrypted with the plain key (decrypted at the subscriber with the same secret morse code)
3. Display of the counter countering replay attacks.

```

Message arrived on topic: sensor/DHT11/all
Message: 4NZ8IXQ0aKz9HmFMIOeoBgEAAAC1i3xyD4qPgAwv0c/00RnuRvYQ7Jz1INyr/Z7aDZH3evdjYjkt6G
Message arrived on topic: sensor/DHT11/all_Hmac
Message: OG4l14VdqCc0ZKpwfE3H7NDByHTGygIfdFDSRF8u7dc=

```

Figure 3.6: Subscription to MQTT server topics

```
Decrypted json doc:
{"temp":22.79999924,"hum":50,"heat":22.4411087,"dew":11.85450268,"comfort":100}
Decrypted values:
Temp: 22.80
Hum: 50.00
Heat: 22.44
Dew: 11.85
Comfort: 100
Counter: 1
```

Figure 3.7: Decryption verification

3.2.5 Port Knocking

To complete the results section, the terminal shows the results got during data sending experiment from the subscriber side.

One can observe that the port is switching once the packet has been received - **Switching Suscribed port : 1900**. This port switch simulates a synchronised port switching between the publisher - MQTT server and the subscriber.

Figure 3.8

```
Decrypted json doc:
{"temp":22.79999924,"hum":50,"heat":22.4411087,"dew":11.85450268,"comfort":100}
Decrypted values:
Temp: 22.80
Hum: 50.00
Heat: 22.44
Dew: 11.85
Comfort: 100
Counter: 1
Switching Suscribed port : 1900
Connecting to MQTT...Connected!
Temp:22.50 Hum:52.00 Index:22.16 Dew:12.17 Comfort_OK
Temp:22.60 Hum:50.00 Index:22.22 Dew:11.67 Comfort_OK
***** PUBLISMENT OF PLAIN DATA TO MQTT SERVER*****
Sending data to Port 1950 for data Visualisation
```

Figure 3.8: Port switching for sniffing security improvement

3.3 Discussion

Discuss about :

1. the assumption the MQTT server and the ESP32 flashes were secured - hardware level - trust zone area !
2. That the port knocking can work only if the server switches from one listening port to another listening port in synchronization with the ESP32
3. Explain that attacks like low-level instructions counting can be used to understand the encryption we used (last year course with Tobias)?

4. Explain the maybe AES 128-bits encryption is a bit light for sensitive data ? I do not know but maybe be good to check

3.4 Conclusion