

Rapport d'élève Ingénieur Projet de troisième année

Filière : Sécurité et réseaux

SMARTHOUSE

Présenté par : KHEDHAOURIA Eliès & MARCELET Paul

Responsable Isima : Monsieur Alexandre GUITTON

Date de soutenance : 02/07/2024

Campus des Cézeaux . 1 rue de la Chébarde . TSA 60125 . 63178 Aubière CEDEX

Table des matières

1	Introduction	1
2	Contexte du Projet	2
2.1	Analyse du besoin et définition des objectifs	2
2.2	Organisation de la conception à la création	2
3	Conception et Implémentation	5
3.1	Infrastructure et Environnement de Développement	5
3.1.1	Simulation du serveur et architecture réseau	6
3.1.2	Modélisation et Simulation d'une Maison Connectée	10
3.2	Mise en place d'une API Web	10
3.2.1	Architecture logicielle de l'API et choix technologiques	10
3.2.2	Automatisation de l'authentification des maisons	10
3.2.3	Filtrage et récupération des données	10
3.3	Surveillance des données avec une interface graphique	10
3.3.1	Architecture logicielle de l'application SmartHouse Monitoring	10
3.3.2	Intégration et communication avec l'API Web	10
4	Résultats et Discussion	11
4.1	Situation à la fin de l'étude	11
4.2	Analyse des résultats obtenus	11
5	Conclusion	12
5.1	Conclusion du projet	12
5.2	Limites et améliorations possibles	12
A	Annexes	13
A.1	Lexique	13
A.2	Bibliographie	13
A.3	Webographie	13

Table des figures

2.1	Diagramme de Gantt Prévisionnel	4
2.2	Diagramme de Gantt réel	4
3.1	Architecture de l'infrastructure fonctionnelle	5
3.2	Sécurisation de mosquitto	7
3.3	Authentification effectuée: Émission et réception effectuée avec succès	7
3.4	Authentification erronée: Émission et réception non fonctionnel	8

Résumé

Dans le cadre de notre **projet de fin d'études**, nous avons conçu et développé un **système de maison intelligente** capable de transmettre **des données en temps réel de manière sécurisée** vers un serveur distant. L'objectif principal est de mettre en place un **système de monitoring avancé**, permettant à un propriétaire de superviser et d'analyser les données générées par ses équipements connectés.

Pour garantir l'**intégrité et la confidentialité des échanges**, nous avons implémenté une **communication sécurisée basée sur des certificats SSL/TLS** et le protocole **MQTTs**, assurant ainsi une transmission chiffrée et authentifiée entre la maison et le serveur.

Les données collectées par les capteurs sont stockées dans une **base de données à série temporelle** (InfluxDB), spécialement optimisée pour le traitement et l'analyse de données en flux continu.

Une **API centralisée**, développée en **Laravel**, a été mise en place afin de :

- **gérer la création des propriétaires** et l'association sécurisée de leurs équipements.
- **automatiser la génération et la signature des certificats** pour garantir une authentification fiable.
- **offrir une interface d'accès aux données**, permettant aux utilisateurs de récupérer **des données en temps réel ou historiques**, selon différents filtres appliqués à la base de données.

Enfin, une **interface graphique interactive**, développée en **Qt**, permet aux utilisateurs de **visualiser les données en temps réel** sous forme de **graphiques dynamiques**. Cette interface interagit directement avec l'API afin de récupérer et d'afficher **les données filtrées**, qu'elles soient en temps réel ou issues d'une période spécifique dans le passé.

Ce projet intègre **des technologies modernes et des concepts avancés en sécurité, IoT, gestion des bases de données et visualisation de données en temps réel**, assurant ainsi une **infrastructure robuste, fiable et évolutive**.

Abstract

As part of our **final-year engineering project**, we designed and developed a **smart home system** capable of securely transmitting **real-time data** to a remote server. The main objective is to implement an **advanced monitoring system** that allows a homeowner to monitor and analyze data generated by their connected devices.

To ensure **data integrity and confidentiality**, we implemented a **secure communication protocol based on SSL/TLS certificates** and the **MQTTs protocol**, providing encrypted and authenticated communication between the home and the server.

Sensor data is stored in a **time-series database** (InfluxDB), optimized for real-time data processing and analysis.

A **centralized API**, developed in **Laravel**, has been implemented to:

- **Manage the creation of homeowners** and the secure association of their devices.
- **Automate the generation and signing of certificates** to ensure reliable authentication.
- **Provide a data access interface**, allowing users to retrieve **real-time or historical data** based on various filters applied to the database.

Finally, an **interactive graphical interface**, developed in **Qt**, allows users to **visualize real-time data using dynamic graphs**. This interface directly interacts with the API to retrieve and display **filtered data**, whether in real-time or from a specific historical period.

This project integrates **modern technologies and advanced concepts in security, IoT, database management, and real-time data visualization**, ensuring a **robust, reliable, and scalable infrastructure**.

Chapter 1

Introduction

La **domotique** représente aujourd’hui un enjeu majeur dans le domaine des innovations technologiques. Avec l’essor des **maisons connectées** et des **IoTs**, les utilisateurs peuvent désormais **surveiller** et **contrôler** leur domicile à distance, leur assurant ainsi une amélioration significative en termes de **sécurité**, **d’efficacité énergétique** et de **confort**. Cette évolution s’inscrit dans un contexte plus large dans lequel l’automatisation et la connectivité jouent un rôle crucial dans notre quotidien.

Le **monitoring à distance** des équipements d’une maison constitue un axe fondamental de la domotique moderne. Il permet aux propriétaires d’obtenir une **vue globale de l’état de leur habitation en temps réel**. Cela joue un rôle crucial dans plusieurs domaines:

- il **sécuriser** un domicile, permettant par exemple la détection d’intrusion ainsi que la prévention des cambriolages
- il **optimise énergétiquement** le domicile, par le biais de l’automatisation des objets connectés, en fonction d’horaires programmés, afin d’optimiser la consommation d’énergie.
- et il permet enfin **un confort et un contrôle à distance**, offrant aux propriétaires la possibilité d’activer ou de désactiver certains dispositifs sans être physiquement présent.

Si ces avancées technologiques offrent des opportunités considérables, elles soulèvent néanmoins une problématique critique: la **sécurisation des dispositifs IoTs et du transfert des données**. Aujourd’hui, de nombreux objets connectés sont déployés avec des failles de sécurité importantes souvent négligées par les fabricants et les utilisateurs. Des outils comme **Shodan**, un moteur de recherche spécialisé dans l’identification des appareils connectés exposés sur Internet, mettent en évidence la vulnérabilité de nombreux systèmes IoTs accessibles sans protection adéquate. Cette situation constitue un risque majeur, rendant possible des cyberattaques capables de compromettre l'**intégrité** et la **confidentialité** des données échangées.

Ce rapport décrit une architecture, solution à cette problématique en explorant l’une des applications majeures de la domotique: le **monitoring à distance des capteurs d’une maison connectée**, ainsi que l’**établissement d’une communication sécurisée et authentifiée entre celle-ci et un serveur distant**. L’objectif est de permettre aux utilisateurs de récupérer des **données en temps réel**, issues de capteurs de leur domicile tout en garantissant **une transmission chiffrée** afin de protéger les échanges contre d’éventuelles interceptions malveillantes. C’est à partir de cela que l’on peut définir une problématique à laquelle la solution doit répondre: **Comment développer un système de monitoring en temps réel pour une maison connectée, garantissant la sécurité de la transmission des données tout en renforçant l’authentification des équipements IoTs ?**

Chapter 2

Contexte du Projet

2.1 Analyse du besoin et définition des objectifs

Lorsqu'un résident quitte son domicile, il peut être préoccupé par l'état de sa maison, se demandant si une lumière a bien été éteinte ou si une fenêtre a été correctement fermée. Ces préoccupations sont légitimes, d'autant plus que les statistiques récentes indiquent une augmentation des cambriolages des logements en France. En effet au 30 juin 2024, les forces de sécurité ont enregistré une hausse de 4% des cambriolages de logements sur les douze derniers mois¹. Cette tendance souligne la nécessité de renforcer les dispositifs de sécurité pour protéger les habitations.

Parallèlement, la prolifération des dispositifs IoT dans les foyers pose des défis non négligeables en matière de cybersécurité. Bien que ces technologies offrent des avantages indéniables en termes de confort et d'efficacité énergétique, elles peuvent constituer des points d'entrée pour les cybercriminels si elles ne sont pas correctement sécurisées. Il est ainsi essentiel de garantir que seuls les propriétaires autorisés aient accès à ces dispositifs et que les données transmises soient protégées contre toute altération ou interception malveillante.

Fâche à ces constats, notre projet vise à développer une solution permettant la transmission sécurisée issues de capteurs IoTs d'une maison vers un serveur distant. Cette solution devra assurer l'authentification des dispositifs, garantir l'intégrité ainsi que la confidentialité des données, et ainsi permettre aux propriétaires de surveiller à distance l'état de leur domicile en temps réel.

2.2 Organisation de la conception à la création

Dans le cadre du développement de ce projet, nous avons adopté une approche structurée en plusieurs phases allant de la simulation initiale de l'environnement domotique jusqu'à la mise en place d'une infrastructure de communication sécurisée et fiable.

Phase 1: Simulation de l'environnement domotique et émission des données

Avant de mettre en place l'architecture réseau et serveur, nous avons débuté par la simulation logicielle de la maison connectée, en programmant un environnement permettant la génération de données de divers capteurs (température, humidité, lumière...). Cette simulation développée en **Python**, modélise une maison contenant divers équipements IoTs et capteurs, émettant des séries de données à temps réel.

L'objectif principal de cette étape était de tester l'envoi de séries de données, à intervalle régulier, au sein d'un serveur distant (Phase suivante), en utilisant le protocole de communication **MQTT**. À ce

1. source: <https://www.interieur.gouv.fr/actualites/actualites-du-ministere/analyse-conjoncturelle-des-crimes-et-delits> et <https://mobile.interieur.gouv.fr/Interstats/Actualites/Info-Rapide-n-43-La-delinquance-enregistree-par-la-police-et-les>

moment là, la transmission s'effectuait sans authentification ni chiffrement, nous permettant ainsi, de valider l'intégralité du transport, la réception au serveur et d'évaluer aussi les performances du protocole.

Phase 2: Mise en place de l'infrastructure serveur

Une fois la simulation fonctionnelle, nous avons déployé une **infrastructure serveur** sous une machine virtuelle **Ubuntu**, utilisant l'hyperviseur **Virtualbox** avec un accès par pont en configuration réseau. Ce serveur assure le rôle de récepteur des données envoyées par la maison connectée.

Afin de permettre la réception ainsi que le stockage des données, nous avons mis en place plusieurs composants essentiels:

- **Mosquitto**: Un broker **MQTT** permettant la gestion des messages entre les maisons connectées et le serveur aux divers topics.
- **InfluxDB**: Une **base de données à séries temporelles**, choisie pour sa capacité à stocker et traiter efficacement des flux de données en temps réel.
- **Telegraf**: Un agent de collecte des données utilisé pour formaliser et structurer les données reçues depuis le **Broker** avant leur insertion dans la base de données.

À la fin de cette étape, après configuration des composants, l'infrastructure était fonctionnelle, mais vulnérable : les données envoyées par les capteurs n'étaient pas protégées et n'importe quel utilisateur pouvait intercepter ou publier des messages MQTT sur le serveur, compromettant ainsi l'intégrité du système.

Phase 3: Sécurisation des échanges et authentifications des Maisons

Afin de garantir l'**authenticité des émetteurs** et de protéger les données échangées, nous avons implanté une **authentification basée sur des certificats SSL/TLS** pour le protocole MQTT. Cette sécurisation repose sur l'**utilisation de certificats clients** générés par une autorité de certification interne au serveur, exigeant une **authentification mutuelle** entre la maison et le serveur pour toute communication MQTT ainsi que le **chiffrement des échanges** grâce à TLS qui empêche toute interception des données transmises.

Ce mécanisme permet ainsi de **garantir l'identité des dispositifs connectés** et d'empêcher ainsi toute interception des données par un acteur non autorisé.

Phase 4 : Développement d'une API centralisant l'accès aux données

Afin de faciliter l'accès aux données, d'éviter une exposition directe du broker MQTT et aussi de permettre par la suite la création d'interfaces fonctionnant sur divers plateformes, nous avons développé une **API Rest sous Laravel**, jouant deux rôles importants:

- **Gestion de propriétaires et authentification**: l'API permet la **création de propriétaires** assurant une authentification unique et sécurisée. Un nouvel utilisateur peut en effet s'enregistrer en tant que propriétaire, l'API lui générera ainsi un **token unique**, qui servira **d'identifiant primaire** pour toutes les interactions futures entre les propriétés de l'utilisateur et le système. Elle générera de plus, un **certificat client signé** par l'autorité de certification accompagné d'une clé privée, permettant ainsi une authentification sécurisée lors des échanges avec le broker MQTT.
- **API de relais et récupération des données**: l'API agit également en tant que **relais sécurisé** entre les propriétaires et les données stockées au sein du serveur. L'API est capable d'extraire les séries temporelles stockées au sein de **InfluxDB**, en les filtrant en fonction des critères demandés pour chaque utilisateurs (temps réel, historique...). Ainsi, les utilisateurs peuvent accéder unique-

ment aux données qui leur sont destinées, garantissant l'intégrité et la confidentialité des échanges.

Phase 5: Développement d'une interface graphique permettant un affichage concret des données

Afin de permettre une visualisation claire des données issues des capteurs, nous avons eu l'idée de développer une interface graphique utilisant **Qt**. Cette interface permet aux propriétaires d'interagir avec l'**API** et d'accéder aux informations de leur maison de manière ergonomique.

Le but de l'interface QT est de récupérer les données via l'API en appliquant divers critères de filtrage, et placer les résultats affichés sous forme de graphique dynamique, afin de faciliter l'analyse ainsi que la supervision de la maison connectée.

Le choix d'utiliser Qt comme technologie, est liée à plusieurs raisons:

- **Demande élevée en entreprise:** Qt est largement utilisé dans l'industrie pour le développement d'interfaces utilisateur performantes et multiplateformes.
- **Complémentarité avec les connaissances acquises en C++:** jusqu'à présent, le programme de formation avait principalement abordé le C++ de manière théorique. Ce projet était donc une opportunité idéale pour appliquer concrètement ses connaissances en développant une interface interactive et fonctionnelle.



Figure 2.1 – Diagramme de Gantt Prévisionnel



Figure 2.2 – Diagramme de Gantt réel

Chapter 3

Conception et Implémentation

3.1 Infrastructure et Environnement de Développement

Cette section détaille l'ensemble de l'infrastructure mise en place, depuis la simulation logicielle d'une maison connectée jusqu'à l'infrastructure serveur dédiée. Cette dernière assure la **réception**, le **traitement** ainsi que le **stockage** sécurisé des données issues des différents capteurs IoT simulés. Nous présenterons les diverses technologies utilisées au sein de l'infrastructure, ainsi que la façon dont elles ont été intégrées afin de garantir une cohérence globale avec les objectifs initiales du projet.

Ce projet a été réalisé en **local**, sous la forme d'une **simulation globale**. Le serveur utilisé repose sur une machine virtuelle exécutant **Ubuntu Server**, configurée localement via **un accès réseau en mode pont**. Cette configuration permet au serveur d'obtenir une adresse IP dédiée sur le même réseau local que la machine hôte, facilitant ainsi une connectivité réseau directe et simplifiée avec la maison connectée simulée. Dans ce contexte, notre simulation considère que le serveur et l'émetteur sont présents au sein du même réseau local, en l'absence de solution cloud externe.

Nous verrons que le serveur joue un rôle central au sein de cette infrastructure. En effet, il héberge l'ensemble des services essentiels au fonctionnement de la solution réceptrice: le **broker MQTT**, l'**API REST** développée sous Laravel, ainsi que l'ensemble des bases de données qu'elles soient à **séries temporelles ou relationnelles**.

Concernant l'infrastructure émettrice, afin de simuler les équipements IOTs de la **maison connectée**, nous avons utilisé le **language Python** du fait de la disponibilité étendue des bibliothèques réseau faciles à implémenter, facilitant ainsi la mise en œuvre rapide et fiable de la simulation.

Dans les sous-sections suivantes, nous détaillerons la mise en place précise de chacun des composants techniques essentiels à la réalisation de nos objectifs.

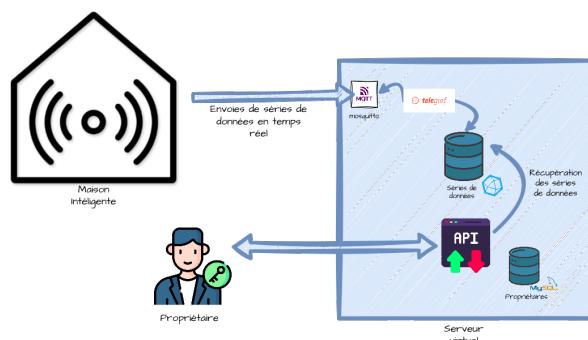


Figure 3.1 – Architecture de l'infrastructure fonctionnelle

Le schéma ci-dessus décrit l'infrastructure globale mis en place au sein de la section.

3.1.1 Simulation du serveur et architecture réseau

Déploiement d'un Broker MQTT sécurisé

Afin de garantir une transmission en temps réel des données IoTs entre la maison connectée et le serveur, nous avons trouver plus judicieux de choisir le protocole **MQTT**.

MQTT est un protocole de messagerie léger basé sur le modèle **publish/subscribe**, initialement conçu pour des environnements où la bande passante est limitée et la latence est faible. MQTT est, dans notre contexte, parfaitement adapté. En effet, le protocole a initialement été conçu afin d'être adapté aux applications IoTs, grâce à sa faible consommation de ressources, sa simplicité d'implémentation et son support pour des communications asynchrones et pouvant être sécurisées par le biais de la technologie **SSL/TLS**.

Il est important de savoir que MQTT repose sur trois éléments fondamentaux:

- **le Broker (serveur de message):** Element central de l'architecture MQTT agissant comme un relais entre les clients acheminant les messages publiés et les abonnés appropriés. Il en existe un certain nombre, et dans le cadre de notre projet, nous avons choisi d'utiliser **Mosquitto**.
- **les Topics (Sujets de message):** Il s'agit d'une chaîne hiérarchique permettant d'identifier une catégorie de messages, leur format est composé de "/" permettant de séparer les sous-topics. Dans le cadre de notre projet nous avons justement exploité ce système de chaînes hiérarchiques afin d'organiser d'une manière précise, l'ensemble des types d'**iots** situés dans les différentes salles des divers **maisons** des différents **propriétaires**.
- **les clients (publishers et subscribers):** MQTT fait la distinction entre deux types de clients:
 - **les publishers (éditeurs)**, ceux qui envoient des messages à un topic spécifique, sans se soucier du destinataire, il s'agit dans le cadre de notre projet, de la **maison connectée** qui envoie des séries de données aux différents topics correspondants.
 - **les subscribers (abonnés)** ceux qui écoutent au sein d'un topic et reçoivent les messages spécifiques correspondants, dans notre cas, il s'agit des **propriétaires** des maisons.

Tel qu'expliqué, nous avons choisis de déployer **Mosquitto** au sein du serveur. Malgré le fait qu'il existe d'autres solutions telles que **HiveMQT**, **EMQX** ou **RabbitMQ MQTT**, pouvant faire office de **broker**, nous avons choisi **Mosquitto**, du fait qu'il est conçu pour être **ultra-léger**, consommer **très peu de mémoire et de CPU même sous forte charge**, et convient aussi bien **aux petits réseaux IoT qu'aux grandes infrastructures**.

De plus, **Mosquitto** permet une installation assez rapide de son service, et la configuration s'effectue via un seul fichier : **mosquitto.conf**. Enfin, il supporte des fonctionnalités de sécurité avancées que nous avons mis en place au sein de notre serveur, telles que le support **TLS/SSL** afin de chiffrer les communications.

Pour sécuriser les connexions et assurer l'authenticité des communications, nous avons implémenté MQTT sécurisé (MQTTs) basé sur SSL/TLS. Nous avons utilisé l'outil **OpenSSL** pour créer une autorité de certification interne **CA** ainsi que pour **générer et signer automatiquement** des certificats clients **client.crt** et leurs clés privées associées **client.key**. Chaque maison simulée dispose donc de trois certificats essentiels pour établir une connexion sécurisée.

Voici ainsi l'architecture de sécurisation de **Mosquitto**:

- la création d'une Autorité de Certification (CA) dédiée pour la **génération et la signature des certificats**.

- l'utilisation de certificats clients signés (certificat client et clé privée) pour chaque maison voulant envoyer des données au sein d'un serveur.
- l'utilisation du protocole MQTTS, permettant une **authentification mutuelle** et un chiffrement systématique des communications, empêchant ainsi toute interception ou injection malveillante de données.

Pour faciliter la compréhension de la sécurisation de Mosquitto, et l'intégration du protocole MQTTS au sein de l'architecture, voici un schéma explicatif:

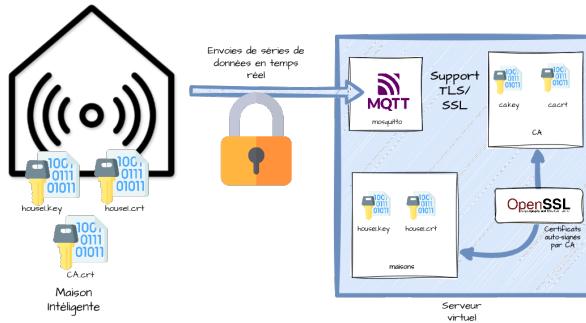


Figure 3.2 – Sécurisation de mosquitto

Afin de vérifier le fonctionnement des certificats ainsi que l'envoi de données au sein du broker mosquitto du serveur virtuel, nous avons à ce moment là testé avec les commandes **mosquitto_pub** côté client et les logs temps réel côté serveur¹.

Ainsi, après génération et auto-signature des certificats, les figures ci-dessous montrent ce qu'il se passe lorsque l'authentification est valide c'est à dire lorsque le **client.key** et le **client.crt** correspondent bien, et lorsque ces derniers sont invalides:

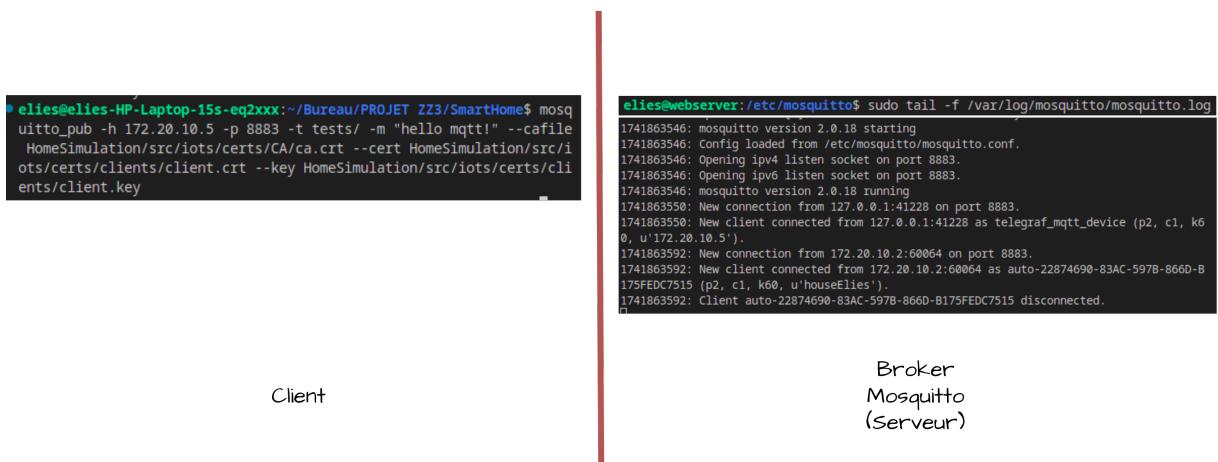


Figure 3.3 – Authentification effectuée: Émission et réception effectuée avec succès

Dans la figure ci-dessus, nous pouvons nous apercevoir (par la capture d'écran du résultat du log instantané du broker) que la connexion de l'émetteur, s'est correctement déroulée, et que ce client a été correctement authentifiée par le nom d'utilisateur **houseElies**.

1. Au sein de la configuration de mosquitto, nous avons défini un chemin de log accessible en temps réel via la commande linux `tail -f /var/log/mosquitto/mosquitto.log`

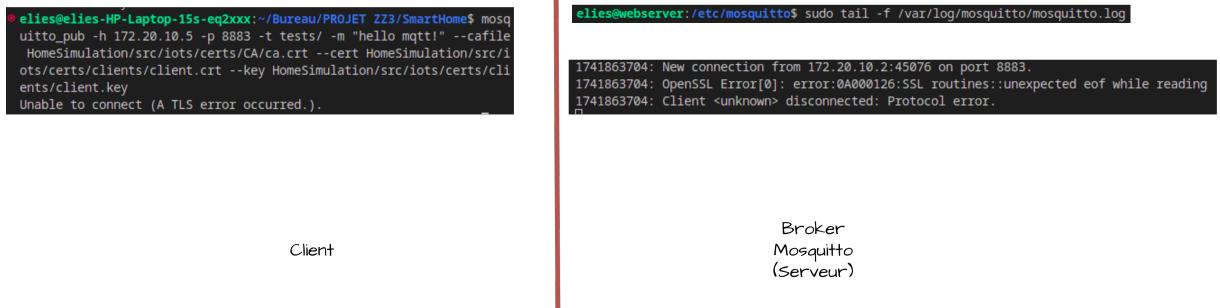


Figure 3.4 – Authentification erronée: Émission et réception non fonctionnel

À la différence de la figure précédente, afin de prouver qu'il est nécessaire de disposer des certificats clients valides, nous avons intentionnellement modifié la clé générée de manière à ce qu'elle soit erronée. Nous pouvons ainsi constater qu'il y a une erreur lors de la connexion de l'emetteur et que ce dernier n'est pas parvenu à publier, ni à être identifié par le broker.

Intégration d'une base de données à séries temporelles

L'émission sécurisée des données depuis une adresse IP distante vers le serveur étant désormais établie, l'étape suivante consiste à assurer le stockage efficace et structuré de ces informations. L'objectif de ce projet, étant non seulement de permettre l'émission de grandes séries données en temps réel, mais aussi de pouvoir interroger et analyser des données collectées à une date ou une période spécifique. C'est pourquoi, nous avons jugé indispensable d'intégrer une base de données adaptées à des contraintes temporelles.

Afin de répondre à ce problème de contrainte temporelle, nous avons utilisé une **base de donnée à séries temporelles (TSDB)**. Une base de donnée à séries temporelles est un type de base de données optimisée pour stocker et interroger des données indexées par le temps. A la différence des bases de données classiques tels que **MySQL** ou **PostgreSQL**, stockant les données sous forme de relation statiques, une **TSDB** permet de gérer des séries de données continus, optimisant les requêtes temporelles et stockant de manière efficace les données horodatées.

Une des plus connues est **InfluxDB**. Développée par **InfluxData**, cette base de données répond parfaitement au problème du fait qu'elle permet l'enregistrement de données évoluant dans le temps et facilite leur analyse. A la différence de ses concurrents, elle permet une écriture des données rapide, ce qui est essentiels pour le temps réel. De plus, elle compresse les données de manière efficace, permettant ainsi l'optimisation de l'espace de stockage. Enfin, un des avantages clés pour notre projet, est le fait que cette base de données est capable de traiter des millions de points de données par seconde.

Elle dispose de plus d'une **API Web** permettant ainsi un accès distant des données, nous verrons par la suite que cette **API** nous sera essentiels pour accéder aux données transférées.

Cependant, la subtilité, est que **InfluxDB** ne supporte pas nativement le protocole **MQTT**, il est donc nécessaire d'utiliser un composant intermédiaire afin d'assurer la transmission des messages depuis le protocole MQTT vers la base de données.

Pour cela, nous avons intégré **Telegraf**, un agent de collecte de métrique conçu afin d'intéragir avec différents systèmes de monitoring et bases de données.

Nous avons ainsi configuré **Telegraf** pour écouter les messages publiés sur **Mosquitto** et les reformater avant de les insérer directement au sein de la base de données.

Telegraf propose deux formats principaux pour insérer des données au sein d'**InfluxDB**:

- **Line Protocol:** Ce format propre à **InfluxDB** optimise le stockage ainsi que la vitesse d'écriture, il s'agit d'une syntaxe représentée sous la forme **measurement,tag=value timestamp**. Bien que très performant, ce format peut être complexe à manipuler et requiert une structure rigide des données envoyées.
- **JSON:** Ce format est plus lisible et universellement reconnu, facilitant le traitement et la manipulation des données. Il permet une structuration claire des informations envoyées, avec des clés explicites pour chaque valeur de mesure, ce qui facilite le débogage et l'analyse des données stockées.

Ainsi, pour des raisons de clarté et de lisibilité, nous avons opté pour le format **JSON**, cela nous facilitera ainsi l'écriture des algorithmes dédiés à l'envoi de données maison.

Afin d'effectuer cela, nous avons configuré le fichier **telegraf.conf** auquel nous avons utilisé le plugin **mqtt_consumer**.

Ce plugin va permettre à **Telegraf** de s'abonner à tout les **topics MQTT** auxquels par la suite nous aurons besoin de récupérer les messages publiés au format **JSON**, afin d'**orchestrer automatiquement** la structuration des données au sein des tables **InfluxDB**, et de faire correspondre **les colonnes aux valeurs JSON** issues des clés correspondantes. Nous verrons plus en détail à la **sous-section suivante** comment nous avons choisi de formaliser les données et ainsi structurer la base de données.

Le schéma **ci-dessous** décrit le but de **Telegraf au sein de notre architecture** par un exemple simple de données à envoyer.

Formalisation des données entre Mosquitto et InfluxDB

3.1.2 Modélisation et Simulation d'une Maison Connectée

Conception de l'architecture logicielle de la simulation

Implémentation du protocole MQTTs

Structuration et formalisation des données échangées

3.2 Mise en place d'une API Web

3.2.1 Architecture logicielle de l'API et choix technologiques

3.2.2 Automatisation de l'authentification des maisons

Mise en place d'une base de données MySQL

Signature automatique des certificats

3.2.3 Filtrage et récupération des données

Communication avec InfluxDB API

3.3 Surveillance des données avec une interface graphique

3.3.1 Architecture logicielle de l'application SmartHouse Monitoring

3.3.2 Intégration et communication avec l'API Web

Authentification des maisons

Affichage des données récupérées

Chapter 4

Résultats et Discussion

4.1 Situation à la fin de l'étude

4.2 Analyse des résultats obtenus

Chapter 5

Conclusion

5.1 Conclusion du projet

5.2 Limites et améliorations possibles

Appendix A

Annexes

A.1 Lexique

A.2 Bibliographie

A.3 Webographie

Bibliography

