Télécom Paris

Projet de filière SR2I208

Communication Bluetooth Sécurisée pour l'Industrie Automobile

Préparé par :

ABI FADEL Najib

FARAH Charbel

KADDOUH Hussein

LTEIF Karen

RABBAT Valentina

YOUNES Hammoud

2023-2024

Sommaire

Résumé Exécutif

Le présent rapport de projet est réalisé dans le cadre du cours de Projet de filière SR2I208 sur la communication TLS over Bluetooth. Ce projet a été mené sous la supervision et avec l'encadrement précieux de plusieurs professeurs que nous tenons à remercier chaleureusement. Nous exprimons notre gratitude à M. Pascal Urien, M. Sébastien Canard, et M. Rida Khatoun pour leur soutien constant tout au long de cette période d'apprentissage et d'accomplissement.

Nous tenons particulièrement à remercier M. Hassane Assaoui pour son accompagnement déterminant dans la réalisation de ce projet. Son expertise et ses conseils ont grandement contribué à la réussite de notre travail.

Ce projet visait à étendre les capacités de communication d'un serveur C et d'un client via TCP/IP en ajoutant une couche TLS (Transport Layer Security), puis en réimplémentant cette configuration pour remplacer le protocole TCP/IP par la technologie Bluetooth. L'objectif final était de créer un client Java intégré à une application Android capable de communiquer de manière sécurisée avec le serveur C via Bluetooth avec TLS.

Ce rapport présente en détail les choix de conception, les défis rencontrés et les solutions apportées à chaque étape du projet. Les résultats obtenus démontrent une compréhension approfondie des technologies impliquées ainsi qu'une application réussie des concepts théoriques dans un contexte pratique.

Nous espérons que ce rapport témoigne de notre engagement, de notre apprentissage et de notre capacité à relever des défis techniques avancés tout en étant reconnaissants envers tous ceux qui ont contribué à rendre cette expérience éducative aussi enrichissante et formatrice.

Introduction

L'industrie automobile connaît une transformation rapide avec l'intégration croissante de capteurs et de technologies de communication avancées au sein des véhicules. Ces progrès permettent la collecte et le partage de données en temps réel, ouvrant la voie à de nouveaux services et à une gestion améliorée de la mobilité urbaine. La communication Bluetooth, traditionnellement utilisée pour des applications de faible puissance et de courte portée, fait désormais son entrée dans ce domaine stratégique.

L'intégration de TLS (Transport Layer Security) sur Bluetooth revêt une importance particulière dans ce contexte. En effet, cette technologie garantit que les données sensibles échangées restent protégées contre les interceptions malveillantes. Cette sécurité renforcée est essentielle pour la confiance des utilisateurs et la fiabilité des systèmes embarqués dans les véhicules connectés et autonomes de demain.

Notre projet se concentre sur le développement d'une application Android sécurisée qui utilise la technologie Bluetooth pour établir une connexion fiable avec un serveur C. Cette application permettra de capturer et d'exporter des messages de sécurité routière, tels que les messages CAM (Cooperative Awareness Message) et DENM (Decentralized Environmental Notification Message), vers le serveur qui seront ensuite mises à disposition pour être exploitées par d'autres utilisateurs de la route, contribuant ainsi à une gestion plus efficace et collaborative de la mobilité urbaine.

Dans cette perspective, nous nous attacherons à résoudre les défis inhérents à l'interopérabilité des technologies envisagées, notamment l'établissement du TLS par-dessus Bluetooth comme protocole de couche 2 et l'interprétation des contenus échangés.

Ce rapport détaille notre approche méthodologique, les technologies utilisées, ainsi que les résultats obtenus, dans le but de démontrer notre engagement envers l'innovation et notre contribution au futur de la mobilité intelligente et connectée.

Aperçu du Projet

Le projet débute par la mise en place d'une communication entre un client C et un serveur via TCP/IP, excluant le protocole HTTP pour se concentrer sur une communication directe entre les deux entités. Cette étape initiale est cruciale pour établir les bases de la communication sécurisée et efficace entre le client et le serveur, en se concentrant sur la gestion des données hors du cadre traditionnel du web.

Ensuite, nous introduisons la sécurité en ajoutant la couche TLS (Transport Layer Security) à cette communication existante. Cette étape garantit que toutes les données échangées entre le client et le serveur sont cryptées et sécurisées, protégeant ainsi la confidentialité et l'intégrité des informations transmises.

Une fois la communication TCP/IP sécurisée établie et validée, notre projet se tourne vers l'adaptation de cette infrastructure pour utiliser la technologie Bluetooth à la place du TCP/IP. Ce changement implique la révision de l'architecture de communication pour s'adapter aux spécificités et aux contraintes de la technologie Bluetooth, tout en maintenant les standards de sécurité et de performance atteints précédemment avec TLS.

Pour compléter cette évolution vers Bluetooth, nous développons un nouveau client en Java, intégré à une application Android. Ce client Java sera conçu pour établir une connexion sécurisée avec le serveur C via Bluetooth avec TLS, en utilisant les fonctionnalités avancées de sécurité et de gestion des données disponibles sur la plateforme Android.

La méthodologie du projet comprendra une phase de conception détaillée, suivie de l'implémentation progressive des composants mentionnés. Chaque étape sera validée par des tests rigoureux pour assurer la sécurité, la fiabilité et la compatibilité des systèmes développés. Des itérations et des ajustements seront effectués en fonction des résultats des tests, garantissant ainsi une solution optimisée et conforme aux exigences spécifiques du domaine de la communication sécurisée dans l'industrie automobile.

Section 1 : Explication des concepts

Transport Layer Security (TLS)

Le protocole TLS (Transport Layer Security) est un protocole cryptographique conçu pour fournir des communications sécurisées sur un réseau informatique. Il est largement utilisé pour sécuriser les connexions Internet, notamment pour le transfert de données sensibles comme les transactions bancaires en ligne, les courriels et les connexions VPN. TLS succède au protocole SSL (Secure Sockets Layer) et apporte des améliorations en matière de sécurité et de performance.

Fonctionnement du Protocole TLS

TLS fonctionne en établissant une session sécurisée entre deux parties, généralement un client et un serveur. La sécurité est assurée par l'utilisation de divers mécanismes cryptographiques, tels que le chiffrement symétrique pour la confidentialité des données, le chiffrement asymétrique pour l'authentification et l'échange de clés, et les fonctions de hachage pour garantir l'intégrité des messages.

Étapes de la Connexion TLS

Le processus de connexion TLS, connu sous le nom de "handshake", comporte plusieurs étapes clés :

1. Client Hello : Le client envoie un message "Client Hello" au serveur, qui comprend des informations telles que la version de TLS, les suites de chiffrement supportées, et des données aléatoires.

2. Server Hello : Le serveur répond avec un message "Server Hello", indiquant la version de TLS et la suite de chiffrement sélectionnées, ainsi que des données aléatoires.

3. Certificat : Le serveur envoie son certificat au client pour permettre l'authentification du serveur. Le certificat contient la clé publique du serveur.

4. Server Key Exchange : Si nécessaire, le serveur envoie un message "Server Key Exchange" contenant des informations supplémentaires pour le chiffrement.

5. Demande de Certificat : Facultatif, le serveur peut demander au client de s'authentifier en fournissant un certificat.

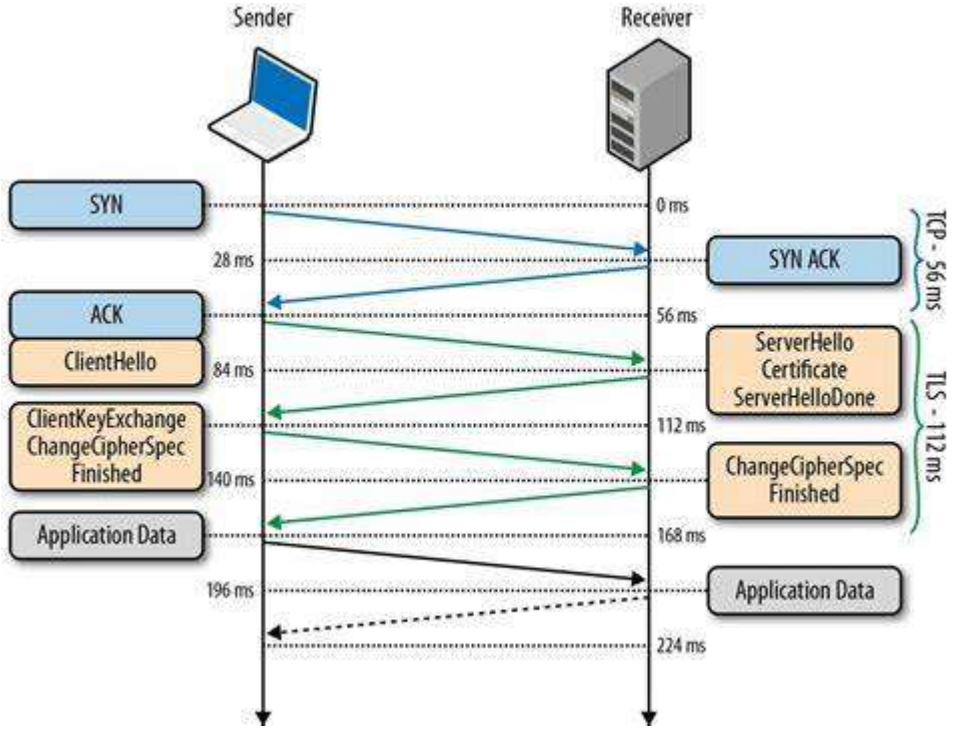
6. Server Hello Done : Le serveur indique qu'il a terminé sa partie du handshake en envoyant un message "Server Hello Done".

7. Client Key Exchange : Le client envoie un message "Client Key Exchange" contenant les informations nécessaires pour que le serveur puisse générer les clés de session.

8. Certificat Verify : Si le client a envoyé un certificat, il envoie un message "Certificat Verify" pour prouver qu'il possède la clé privée correspondant au certificat.

9. Change Cipher Spec : Le client et le serveur envoient chacun un message "Change Cipher Spec" pour indiquer qu'ils vont commencer à utiliser les paramètres de chiffrement négociés.

10. Finished : Les deux parties envoient un message "Finished" pour indiquer que le handshake est terminé. Ce message est chiffré et haché pour garantir l'intégrité du handshake.



https://www.malekal.com/protocole-tls-fonctionnement/

Figure 3 – Etapes de la connexion TLS

Détails Techniques du Handshake TLS

Le handshake TLS établit les paramètres de chiffrement et les clés de session à utiliser pour la communication sécurisée. Les principales phases sont :

- Négociation des paramètres de sécurité : Le client et le serveur négocient les versions de TLS et les suites de chiffrement à utiliser.

- Authentification et échange de clés : Le serveur (et éventuellement le client) s'authentifie en utilisant des certificats, et les clés de session sont échangées en utilisant des algorithmes de chiffrement asymétrique.

- Établissement des clés de session : Les clés symétriques utilisées pour chiffrer les données échangées pendant la session sont générées à partir des informations échangées lors du handshake.

Suites de Chiffrement et Algorithmes Utilisés

TLS utilise une variété de suites de chiffrement, qui sont des combinaisons d'algorithmes pour le chiffrement symétrique, le chiffrement asymétrique, et les fonctions de hachage. Les suites de chiffrement couramment utilisées incluent AES (Advanced Encryption Standard) pour le chiffrement symétrique, RSA (Rivest-Shamir-Adleman) ou Diffie-Hellman pour l'échange de clés, et SHA (Secure Hash Algorithm) pour l'intégrité des messages.

Bluetooth

Introduction au Protocole Bluetooth

Le protocole Bluetooth est une technologie de communication sans fil conçue pour échanger des données sur de courtes distances en utilisant des ondes radio dans la bande de fréquence ISM de 2,4 GHz. Il permet de créer des réseaux personnels sans fil (WPAN) où les dispositifs peuvent se connecter et échanger des informations. Le Bluetooth fonctionne en utilisant un processus appelé "saut de fréquence", où le signal change fréquemment de fréquence pour réduire les interférences et améliorer la sécurité. Un dispositif maître peut se connecter à plusieurs dispositifs esclaves, formant ainsi une piconet, et plusieurs piconets peuvent être interconnectées pour former un scatternet.

Spécificités du Protocole Bluetooth

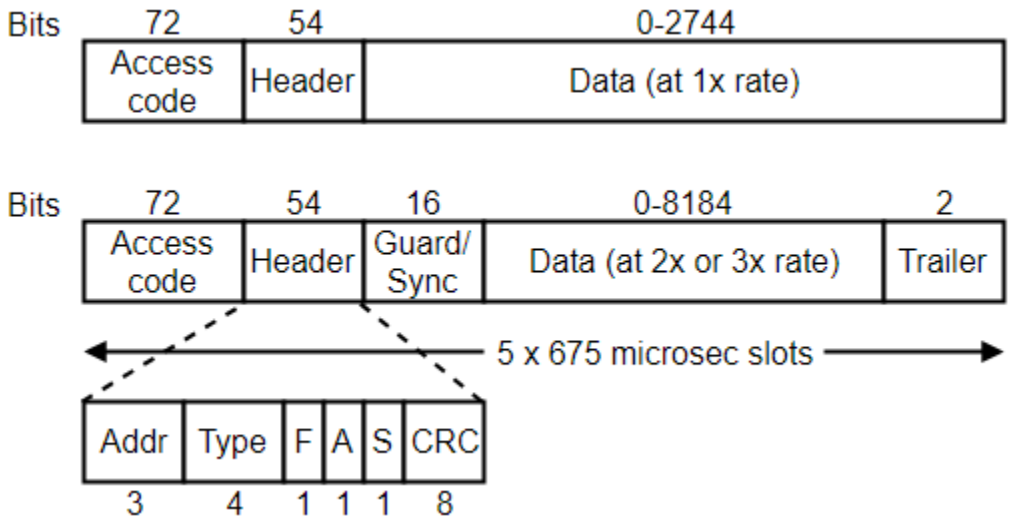
Les spécificités du protocole Bluetooth incluent une portée typique allant jusqu'à 10 mètres, qui peut atteindre jusqu'à 100 mètres avec des dispositifs de classe 1. Les débits de données varient de 1 Mbps pour les versions initiales à 24 Mbps pour Bluetooth 3.0 + HS (High Speed). Le protocole utilise des mécanismes de sécurité tels que le couplage et l'authentification, ainsi que des algorithmes de chiffrement pour protéger les communications. La modulation GFSK (Gaussian Frequency Shift Keying) est utilisée pour transmettre les données de manière fiable.

Applications du Bluetooth

Les applications du Bluetooth sont variées et incluent des périphériques audios sans fil comme les écouteurs et haut-parleurs, des périphériques d'entrée comme les souris et claviers, ainsi que des communications entre smartphones, tablettes et ordinateurs portables. En outre, le Bluetooth est largement utilisé dans la domotique et l'Internet des objets (IoT) pour connecter des capteurs, des lumières intelligentes et divers appareils électroménagers.

Pile de Protocoles et Architecture du Bluetooth

La pile de protocoles et l'architecture Bluetooth sont bien définies. La couche de base, appelée "Baseband", gère la connexion physique et la synchronisation des dispositifs. Le "Link Manager Protocol" (LMP) est responsable de l'établissement et de la gestion des connexions. Le "Logical Link Control and Adaptation Protocol" (L2CAP) fournit des services de multiplexage, de segmentation et de reconstitution des trames de données. Le protocole RFCOMM émule les connexions série RS-232 sur les couches inférieures de Bluetooth, permettant une communication transparente pour les applications nécessitant des connexions série. Enfin, le "Service Discovery Protocol" (SDP) permet aux dispositifs de découvrir les services disponibles sur d'autres dispositifs Bluetooth.

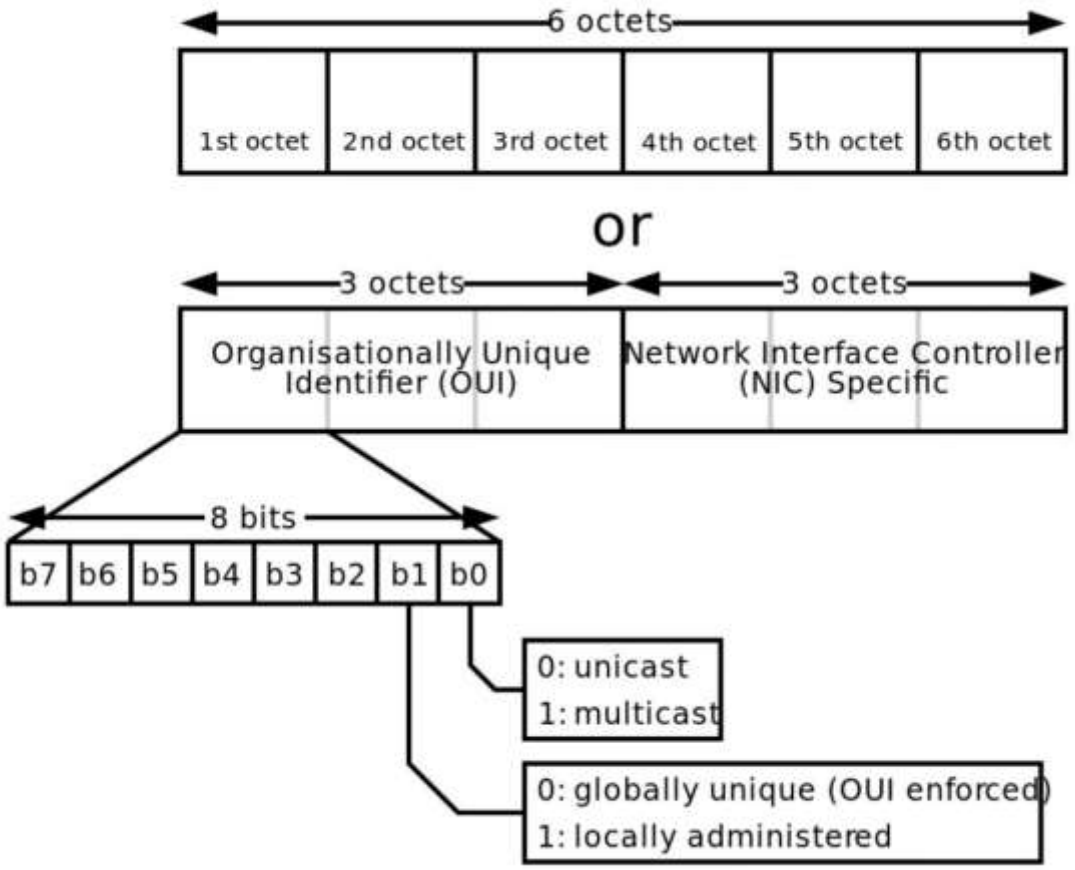


https://www.google.fr/url?sa=i&url=https%3A%2F%2Fnotes.eddyerburgh.me%2Fcomputer-networking%2Fbluetooth&psig=AOvVaw3XdIbZKZe3eA-E56rt5JO6&ust=1719168714672000&source=images&cd=vfe&opi=89978449&ved=0CBIQjhxqFwoTCPCkt9rw74YDFQAAAAAdAAAAABAE

Figure 1 - Format de trame Bluetooth avec débit de données de base

Adressage des Dispositifs Bluetooth

L'adressage des dispositifs Bluetooth se fait à l'aide d'une adresse unique de 48 bits appelée BD_ADDR, similaire à une adresse MAC pour les réseaux Ethernet. Cette adresse est composée de 24 bits pour l'identifiant unique de l'organisation (OUI) et de 24 bits pour l'identifiant unique de l'appareil.



https://www.google.fr/url?sa=i&url=https%3A%2F%2Fsecurity.stackexchange.com%2Fquestions%2F151768%2Fis-it-possible-to-find-the-manufacturer-of-a-ble-device-from-its-mac-address&psig=AOvVaw2gZKGxMpKbsXMhVqISvsSn&ust=1719170835879000&source=images&cd=vfe&opi=89978449&ved=0CBIQjhxqFwoTCPj55fP474YDFQAAAAAdAAAAABAR

Figure 2 - Adressage des appareils Bluetooth

Types de Codes d'Accès

Les types de codes d'accès utilisés incluent le "Channel Access Code" (CAC) pour identifier une piconet spécifique, le "Device Access Code" (DAC) pour l'initialisation de l'invite et les

procédures de correction d'erreurs, et l'"Inquiry Access Code" (IAC) pour identifier les appareils lors des processus d'invite.

Format des Trames Bluetooth

Le format des trames Bluetooth comprend plusieurs composants : un préambule de 4 bits, une synchronisation de 64 bits, un code d'accès de 68 bits, un en-tête de 54 bits qui contient des informations telles que le type de trame et l'adresse de l'appareil, et une charge utile variable pouvant aller jusqu'à 2745 bits.

RFCOMM et son Fonctionnement dans Bluetooth

RFCOMM (Radio Frequency Communication) est un protocole de couche supérieure utilisé pour émuler les ports série RS-232 sur une connexion Bluetooth. Il permet aux applications conçues pour fonctionner avec des ports série traditionnels de fonctionner sans modification majeure sur une liaison Bluetooth. RFCOMM supporte jusqu'à 60 connexions simultanées sur un même lien physique Bluetooth, utilise des techniques de détection et de correction d'erreurs pour assurer une communication fiable, et fournit une interface simplifiée pour les développeurs en cachant la complexité des couches inférieures du Bluetooth.

Intégration de TLS avec Bluetooth

L'intégration de TLS avec Bluetooth fonctionne en encapsulant les données transmises via Bluetooth dans une couche de sécurité fournie par TLS. Comme déjà décrit précédemment, TLS assure la confidentialité et l'intégrité des données échangées grâce à un chiffrement robuste et une authentification mutuelle, tandis que Bluetooth permet une communication sans fil flexible et pratique entre les dispositifs. Cette combinaison est non seulement possible mais également cruciale pour assurer la sécurité des communications sans fil. En intégrant TLS, nous pouvons garantir que même si les données sont transmises sur un canal sans fil potentiellement vulnérable, elles restent protégées et confidentielles. Nous allons expliquer dans les parties suivantes les étapes du déroulement de l'implémentation de cette intégration, en détaillant comment nous avons configuré et testé la communication sécurisée entre le client et le serveur.

Section 2 : Déroulement et méthodologies

Dans cette section, on expliquera le déroulement de notre travail pour établir un client Java communiquant via Bluetooth et implémentant TLS ; TLS nécessite des certificats et des clés privées pour fonctionner. Comme on cherche à mettre en place une authentification à deux facteurs, on va créer à la fois le certificat et les clés pour le serveur et le client en utilisant openssl. Le code utilisé a le format suivant :

```
openssl genpkey -algorithm RSA -out server.key
```

```
openssl req -new -key server.key -out server.csr
```

```
openssl genpkey -algorithm RSA -out server.key
```

On fera cela à la fois pour le serveur et le client, et on utilisera ces éléments pour toutes les applications TLS ci-dessous, que ce soit en TCP ou en Bluetooth.

Communication TCP/IP avec TLS

On a commencé le travail en implémentant un serveur/client TCP/IP avec TLS, étant donné que TCP avec TLS est largement utilisé et bien documenté.

Code en C pour TCP/IP avec TLS

Puisque le serveur final doit être codé en C, nous avons décidé de commencer directement avec ce langage. La bibliothèque standard contient du code permettant d'établir une connexion TCP sous Linux, ainsi que les bibliothèques OpenSSL pour implémenter SSL sur cette connexion. Les certificats et les clés sont directement intégrés dans le code pour être utilisés dans le processus SSL.

Le code se trouve dans le fichier zip joint. Une fois le serveur et le client lancés, la connexion est correctement établie et les messages peuvent être envoyés chiffrés par TLS.

N.B. : Nous compilons notre code avec gcc en utilisant les options -lssl et -lcrypto.

La sortie du serveur sera comme celle montrée dans l'image ci-dessous. On peut effectivement voir que toutes les étapes de la poignée de main sont effectuées avec succès avant que le message chiffré du client soit envoyé, suivi de l'envoi d'un message du serveur au client. Dans la deuxième image, nous pouvons voir que le client reçoit le même message.

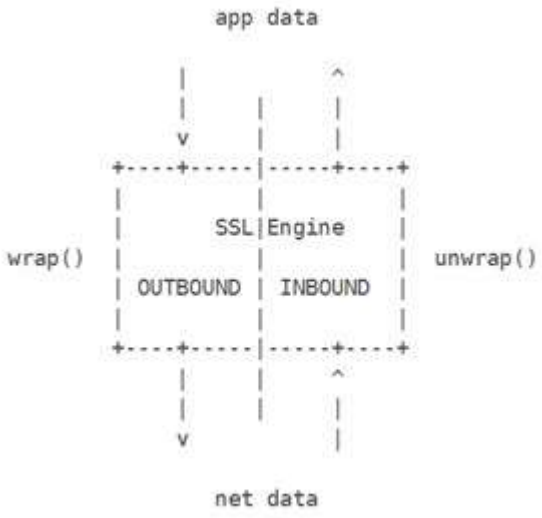
```
netgeek@NetLab64v2:~/Desktop/sr2i208/Latest/servercode$ gcc -o TCP-TLS_server T
P-TLS_server.c -lssl -lcrypto
netgeek@NetLab64v2:~/Desktop/sr2i208/Latest/servercode$ ./TCP-TLS_server
TCP Server Socket Created
Bind Successful to port 5555
Listening...
Connection Established with client at IP 127.0.0.1 and port 49344
SSL_accept:before SSL initialization
SSL_accept:before SSL initialization
SSL_accept:SSLv3/TLS read client hello
SSL_accept:SSLv3/TLS write server hello
SSL_accept:SSLv3/TLS write certificate
SSL_accept:SSLv3/TLS write key exchange
SSL_accept:SSLv3/TLS write certificate request
SSL_accept:SSLv3/TLS write server done
SSL_accept:SSLv3/TLS write server done
SSL_accept:SSLv3/TLS read client certificate
SSL_accept:SSLv3/TLS read client key exchange
SSL_accept:SSLv3/TLS read certificate verify
SSL_accept:SSLv3/TLS read change cipher spec
SSL_accept:SSLv3/TLS read finished
SSL_accept:SSLv3/TLS write change cipher spec
SSL_accept:SSLv3/TLS write finished
Client: HELLO, THIS IS THE CLIENT...
Server: HI, THIS IS THE SERVER, HAVE A NICE DAY!!!
Encrypted message sent to client:
48 49 2C 20 54 48 49 53 20 49 53 20 54 48 45 20
53 45 52 56 45 52 2C 20 48 41 56 45 20 41 20 4E
49 43 45 20 44 41 59 21 21 21
SSL3 alert write:warning:close notify
Connection with client 127.0.0.1 has been closed
```

```
netgeek@NetLab64v2:~/Desktop/sr2i208/Latest/clientcode/C_code$ ./TLS-TCP_client
TCP Client Socket Created
Connected to server at IP 127.0.0.1 and port 5555
Client: HELLO, THIS IS THE CLIENT...
Encrypted message sent to server:
48 45 4C 4C 4F 2C 20 54 48 49 53 20 49 53 20 54
48 45 20 43 4C 49 45 4E 54 2E 2E 2E
Server: HI, THIS IS THE SERVER, HAVE A NICE DAY!!!
Disconnected from the server.
```

Code en Java pour TCP/IP avec TLS

Pour ce qui est de Java, nous utiliserons des bibliothèques standard pour la connexion TCP, et nous utiliserons SSLengine pour chiffrer nos données avant de les envoyer.

SSLengine est mieux expliqué par l'image ci-dessous :



SSLEngine est indépendant de la couche de transport, ce qui nous permet de l'utiliser avec TCP, et plus tard avec Bluetooth, pour envelopper (*wrap()*) et développer (*unwrap()*) les paquets avant de les envoyer et après les avoir reçus.

En se connectant au serveur mentionné précédemment, cela fonctionnera de nouveau.:

```
netgeek@NetLab64v2:~/Desktop/sr2i208/JAVA-TCP$ java ClientTLS.java
About to do handshake...
About to write to the server...
Message sent to the server: Hello I am client!
About to read from the server...
Server response: HI, THIS IS THE SERVER, HAVE A NICE DAY!!!
About to do handshake...
About to close connection with the server...
About to do handshake...
Goodbye!
```

Java requiert l'utilisation de truststores et de keystores pour charger les certificats et clés nécessaires. Ainsi, un truststore et un keystore sont créés en utilisant une combinaison de OpenSSL et Keytool. Voici les étapes à suivre :

Convertir le certificat X.509 et la clé en un fichier PKCS12

```
openssl pkcs12 -export -in server.crt -inkey server.key -out server.p12 -name [some-alias] -CAfile ca.crt -caname root
```

Convertir le fichier PKCS12 en un keystore Java

```
keytool -importkeystore -deststorepass [changeit] -destkeypass [changeit] -destkeystore server.keystore -srckeystore server.p12 -srcstoretype PKCS12 -srcstorepass some-password -alias [some-alias]
```

Le format JKS est utilisé, ce qui fonctionne pour les applications Java normales sur un PC. Plus tard, pour notre client sous Android, le format BKS sera nécessaire.

Communication Bluetooth avec TLS

Nous allons maintenant porter notre travail vers Bluetooth. Nous conserverons une grande partie du code précédent, mais bien sûr, certaines modifications devront être apportées pour le rendre compatible avec Bluetooth.

Code en C pour Bluetooth avec TLS

Pour notre communication via Bluetooth, nous utiliserons une combinaison de l'adresse MAC et du numéro de canal pour établir la connexion entre le serveur et le client.

Nous utiliserons de nouveau la bibliothèque standard, mais nous modifierons certaines fonctions, comme socket, de la manière suivante

```
socket(AF_INET, SOCK_STREAM, 0);
```

To:

```
socket(AF_BLUETOOTH, SOCK_STREAM, BTPROTO_RFCOMM);
```

Ce changement spécifie que nous travaillons sur Bluetooth, plus précisément en RFCOMM. Bien sûr, d'autres modifications doivent être apportées pour s'adapter au nouveau protocole.

Le code SSL devrait rester inchangé car il est indépendant de la couche de transport. Nous définissons le descripteur de fichier de la socket pour qu'il soit SSL.

Code en Java pour Bluetooth avec TLS

Pour ce qui est de Java, puisque nous avons utilisé SSLEngine, un code indépendant de la couche de transport, il suffit de changer la connexion de TCP à Bluetooth RFCOMM. Bien sûr, en réalité, ce n'est pas aussi simple, car TLS sur Bluetooth a très peu de documentation, et les bugs sont difficiles à trouver et à gérer. Mais avec le code entier corrigé, cela fonctionne également, permettant la communication avec le serveur. Vous pouvez voir ci-dessous les paquets de communication TLS Bluetooth, avec une explication détaillée, montrant que cela a fonctionné avec succès.

Comme mentionné précédemment, étant donné que nous travaillons sur Android, nous devons transformer notre truststore et keystore en format BKS. Le fichier jar de Bouncy Castle (lien de téléchargement)est nécessaire pour être utilisé comme fournisseur afin de transformer les stores JKS en BKS .

```
keytool -importkeystore -srckeystore "keystore.jks" -destkeystore "keystore.bks" -srcstoretype JKS -deststoretype BKS -srcstorepass "YOUR_PASSWORD" -deststorepass "YOUR_PASSWORD" -provider "org.bouncycastle.jce.provider.BouncyCastleProvider" -providerPath <Bouncy castle jar file path>
```

Section 3 : Application Android avec Client JAVA

Introduction

Dans le cadre de ce projet, le développement d'un client Java intégré à une application Android représente une composante cruciale. Cette section introduit les principes et méthodes employés pour intégrer efficacement un client Java au sein d'une application Android, permettant une communication sécurisée avec un serveur C via Bluetooth et TLS (Transport Layer Security).

Le client Java pour Android est conçu pour gérer des communications sécurisées par Bluetooth, en utilisant TLS pour chiffrer et sécuriser les données échangées entre les véhicules et les infrastructures. Cela est d'une importance capitale pour garantir la confidentialité et l'intégrité des messages échangés, notamment les messages CAM (Cooperative Awareness Message) et DENM (Decentralized Environmental Notification Message), qui sont essentiels pour la sécurité routière et la gestion de la mobilité urbaine.

Détails de l'Implémentation

Environnement de Développement et Outils

Dans le cadre du développement de notre application Android utilise un client Java pour la communication sécurisée via Bluetooth avec TLS, plusieurs outils et environnements de développement ont été méticuleusement sélectionnés pour optimiser la productivité, la collaboration, et la qualité du code.

1. Environnement de développement intégré (IDE) :

Le développement a été principalement réalisé sous Android Studio, le choix par excellence pour le développement d'applications Android. Cet IDE offre une intégration native avec le SDK Android, des facilités pour gérer les différentes versions d'API Android, et des outils puissants pour le profilage d'applications et le débogage. Android Studio supporte également Kotlin et Java, ce qui est essentiel pour notre projet qui utilise Java pour la mise en œuvre du client.

2. Langages de programmation :

Java a été utilisé pour développer la logique du client, tirant parti de sa robustesse et de son écosystème riche pour gérer les communications réseau sécurisées. Le choix de Java est également justifié par sa portabilité sur différentes plateformes, une caractéristique importante pour les applications devant interagir avec des serveurs et des systèmes embarqués divers.

3. Bibliothèques et frameworks :

- **Bouncy Castle :** Pour la gestion de TLS, la bibliothèque Bouncy Castle a été intégrée. Elle offre des fonctionnalités étendues pour le cryptage et la génération de certificats, essentielles pour notre mise en œuvre de TLS sur Bluetooth.
- **Android Bluetooth API :** Utilisée pour gérer toutes les interactions Bluetooth, permettant de découvrir, de se connecter aux dispositifs Bluetooth, et de communiquer avec eux.

Explication du code

Layout XML (activity_main.xml)

Le fichier activity_main.xml définit l'interface utilisateur pour MainActivity. Il utilise un LinearLayout avec une orientation verticale pour organiser les composants de l'interface utilisateur. Voici un résumé des composants :

- **TextView (@+id/statusBluetooth)** : Affiche le statut du Bluetooth.
- **ImageView (@+id/bluetoothIcon)** : Affiche l'icône Bluetooth indiquant s'il est activé ou désactivé.
- **Boutons** : Quatre boutons pour activer/désactiver le Bluetooth, découvrir des appareils et obtenir les appareils appariés.
- **ListView (@+id/pairedDevices et @+id/deviceListView)** : Liste les appareils appariés et découverts respectivement.
- **TextView (@+id/noDevices)** : Affiche un message lorsqu'aucun appareil n'est trouvé.
- **Boutons pour la connexion TLS** : Deux boutons pour se connecter avec ou sans TLS.
- **LinearLayout (@+id/messageLayout)** : Contient les éléments pour la messagerie après la connexion, incluant un EditText pour taper les messages, un Button pour envoyer les messages, et un autre TextView pour afficher les messages reçus.
- **Button (@+id/disconnectBtn)** : Déconnecte la connexion Bluetooth.

MainActivity.java

Ce fichier contient la logique pour MainActivity, gérant les fonctionnalités Bluetooth et les interactions définies dans le layout XML.

- **Initialisation** : Initialise les composants de l'interface utilisateur, l'adaptateur Bluetooth et les adaptateurs personnalisés pour les listes d'appareils.
- **Permissions et statut Bluetooth** : Vérifie et demande les permissions nécessaires pour les opérations Bluetooth et met à jour le statut Bluetooth en conséquence.
- **Gestion des clics sur les boutons** :
    - **Activer/Désactiver Bluetooth** : Active ou désactive le Bluetooth.
    - **Découvrir les appareils** : Lance la découverte des appareils Bluetooth.
    - **Obtenir les appareils appariés** : Affiche une liste des appareils appariés.

- **Connexion avec/sans TLS** : Établit une connexion Bluetooth en utilisant la méthode choisie (avec ou sans TLS).
- **Envoyer un message** : Envoie un message via la connexion établie.
- **Déconnecter** : Termine la connexion Bluetooth et nettoie les ressources.

BluetoothClientInterface.java

Une interface définissant les méthodes essentielles pour un client Bluetooth qui est implementé par **BluetoothClient.java** et **BluetoothClient_TLS.java**.

- **connect(String macAddress, int channelNumber)** : Se connecte à un appareil.
- **disconnect()** : Se déconnecte de l'appareil.
- **write(byte[] data)** : Envoie des données.
- **read(byte[] buffer)** : Lit des données.

BluetoothClient.java et BluetoothClient_TLS.java

Ces fichiers implémentent la fonctionnalité du client Bluetooth .

- **BluetoothClient** : Établit une connexion Bluetooth standard, gère l'envoi et la réception de données.
- **BluetoothClient_TLS** : Similaire à BluetoothClient, mais ajoute le support pour TLS (Transport Layer Security) pour sécuriser la communication.

BluetoothClientInterface.java

Une interface définissant les méthodes essentielles pour un client Bluetooth :

- **connect(String macAddress, int channelNumber)** : Se connecte à un appareil.
- **disconnect()** : Se déconnecte de l'appareil.
- **write(byte[] data)** : Envoie des données.
- **read(byte[] buffer)** : Lit des données.

CustomAdapter.java

Un adaptateur personnalisé pour remplir le ListView avec les noms des appareils.

TLS.java

Cette classe gère la configuration de TLS pour les connexions sécurisées.

- **Chargement des Keystore et Truststore** : Charge les fichiers de keystore et truststore depuis les ressources brutes.
- **Initialisation des KeyManagerFactory et TrustManagerFactory** : Initialise les gestionnaires de clés et de confiance avec les keystore et truststore chargés.
- **Création de SSLContext** : Crée un contexte SSL avec la version TLS spécifiée.

Layout XML pour CustomAdapter (list_item.xml)

Le fichier list_item.xml définit la mise en page pour chaque élément de la liste affichée par CustomAdapter.

Démonstration

Activation et Désactivation du Bluetooth

Bouton 'Turn On'

Lorsque l'utilisateur appuie sur le bouton **'Turn On'**, l'application vérifie d'abord si les permissions nécessaires au Bluetooth sont accordées (spécifiquement, BLUETOOTH_CONNECT sous Android S et versions ultérieures).
Si les permissions sont accordées, le Bluetooth est activé via la méthode enableBluetooth().
Sinon, l'application demande les permissions nécessaires à l'utilisateur.

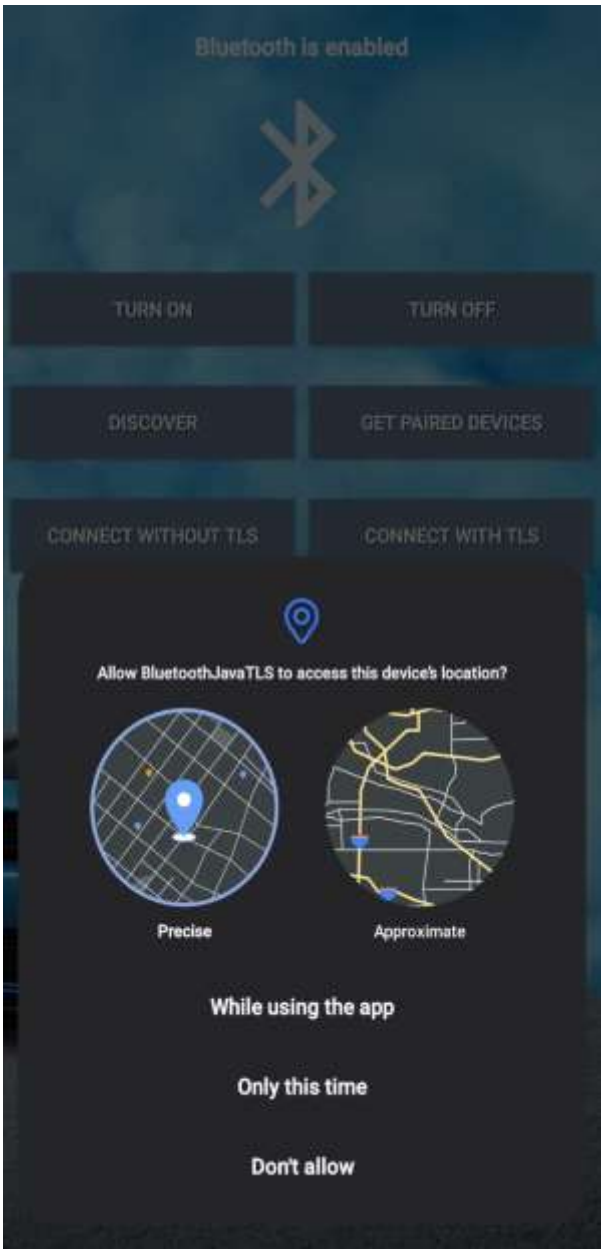
Bouton 'Turn Off' :

En appuyant sur le bouton **'Turn Off'**, l'application effectue une vérification similaire des permissions avant de désactiver le Bluetooth. Si l'utilisateur a déjà accordé les permissions nécessaires, le Bluetooth est désactivé à l'aide de la méthode disableBluetooth(). Si les permissions ne sont pas accordées, l'application initie une demande de permissions auprès de l'utilisateur.

Découverte et Appairage des Dispositifs

Bouton 'Discover' :

Quand l'utilisateur appuie sur le bouton 'Discover', l'application initie le processus de découverte de nouveaux dispositifs Bluetooth. Le code effectue une vérification préliminaire pour s'assurer que le Bluetooth est activé et que les permissions nécessaires sont accordées. Si toutes les conditions sont remplies, la découverte commence, permettant à l'application de détecter et d'afficher uniquement les dispositifs Bluetooth à proximité qui ne sont pas déjà appairés. Ceci aide à éviter la redondance dans la liste affichée et se concentre sur la découverte de nouveaux dispositifs disponibles pour l'appairage. Pendant la découverte, chaque dispositif non appairé détecté est ajouté à la liste des dispositifs découverts et affiché dans l'interface utilisateur, permettant une interaction ultérieure.


Bluetooth is enabled
TURN ON
TURN OFF
DISCOVER
GET PAIRED DEVICES
CONNECT WITHOUT TLS
CONNECT WITH TLS
Allow BluetoothJavaTLS to access this device's location?
Precise
Approximate
While using the app
Only this time
Don't allow


Bluetooth is enabled
TURN ON
TURN OFF
DISCOVER
GET PAIRED DEVICES
CONNECT WITHOUT TLS
CONNECT WITH TLS
DISCONNECT
Discovery started...


Bluetooth is enabled
TURN ON
TURN OFF
DISCOVER
GET PAIRED DEVICES
charbel-VirtualBox (34:CF:F6:86:F6:DB)
CONNECT WITHOUT TLS
CONNECT WITH TLS
DISCONNECT

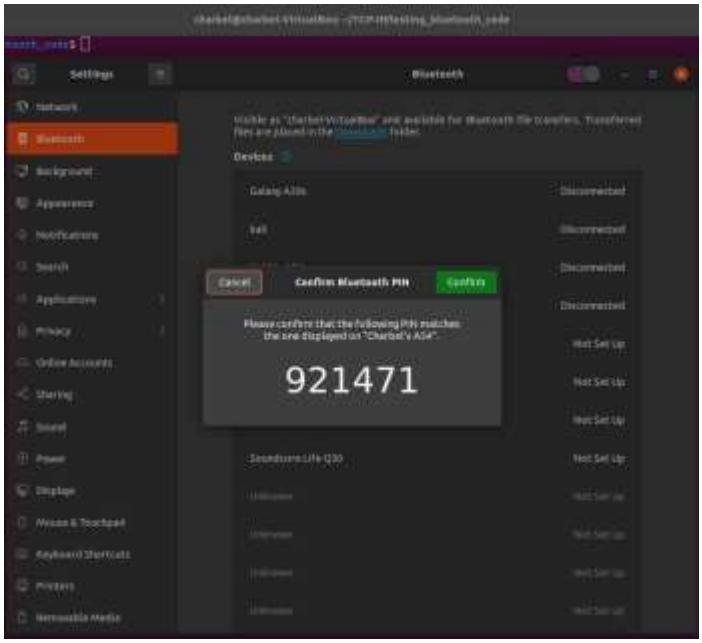
Bouton 'Get Paired Devices' :

En cliquant sur **'Get Paired Devices'**, l'application affiche ceux qui ont déjà été appairés avec le dispositif utilisant l'application. Ce bouton déclenche une fonction qui récupère la liste des dispositifs Bluetooth appairés stockés dans le BluetoothAdapter du dispositif. Ces dispositifs sont ensuite listés dans l'interface utilisateur, permettant à l'utilisateur de voir rapidement quels dispositifs sont déjà connus et potentiellement prêts à être connectés sans nécessiter une nouvelle découverte.

Gestion Avancée des Connexions Bluetooth et Échanges de Messages sans TLS

Sélection et Connexion au Dispositif Bluetooth :

Sélection du dispositif : Après avoir découvert les dispositifs, en sélectionnant un dispositif spécifique dans la liste des appareils découverts, l'application effectuera une action de couplage avec cet appareil sélectionné.

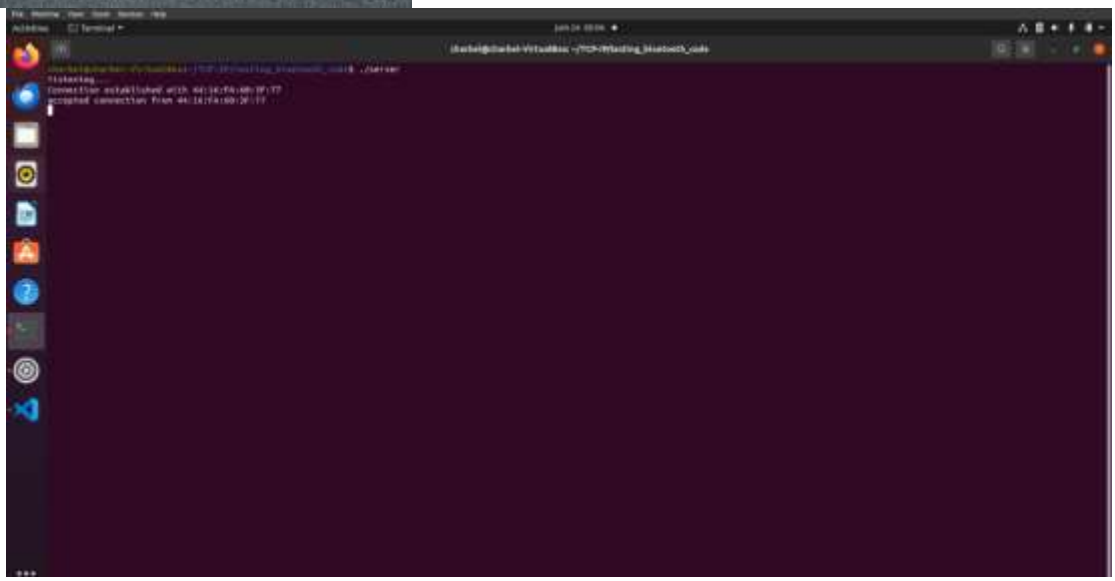


- **Confirmation de la connexion** : Une fois le couplage établie, l'utilisateur retourne à l'application où il peut maintenant initier une connexion sans TLS en appuyant sur le bouton **'Connect Without TLS'**.

Établissement de la Connexion Sans TLS :

En appuyant sur le bouton **'Connect Without TLS'**, l'application tente de se connecter au dispositif sélectionné sans utiliser le protocole TLS.

- **Visuel de confirmation** : Cette capture d'écran du serveur affiche que la connexion est réussie.

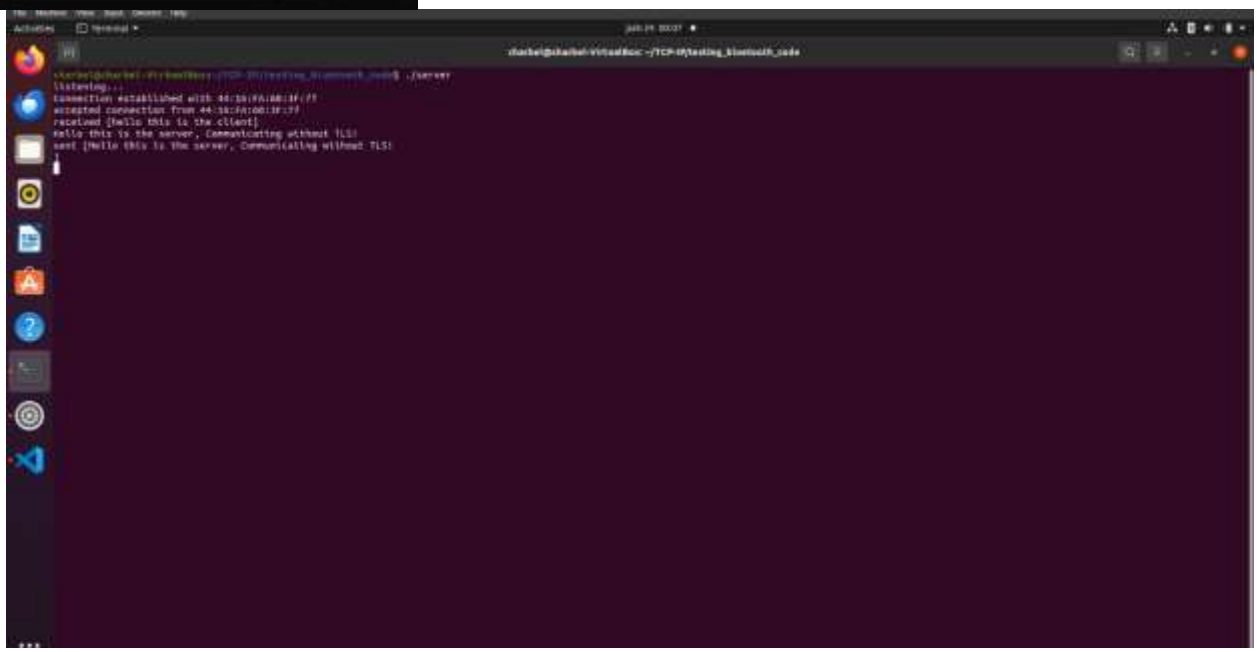


Échange de Messages :

. **Envoi de message depuis l'application**
L'utilisateur saisit un message dans le champ de texte et l'envoie en appuyant sur **'Send'**.

- **Réception du message sur le serveur** : Cette capture d'écran montre que le message envoyé depuis l'application a bien été reçu par le serveur.

- **Envoi de réponse depuis le serveur** : Cette capture d'écran du serveur montre que le processus d'envoi d'un message de réponse vers l'application est en cours.



- **Réception de la réponse dans l'application** :

Cette image montre le message reçu du serveur.

Déconnexion et Messages de Fin de Session

Bouton 'Disconnect' :

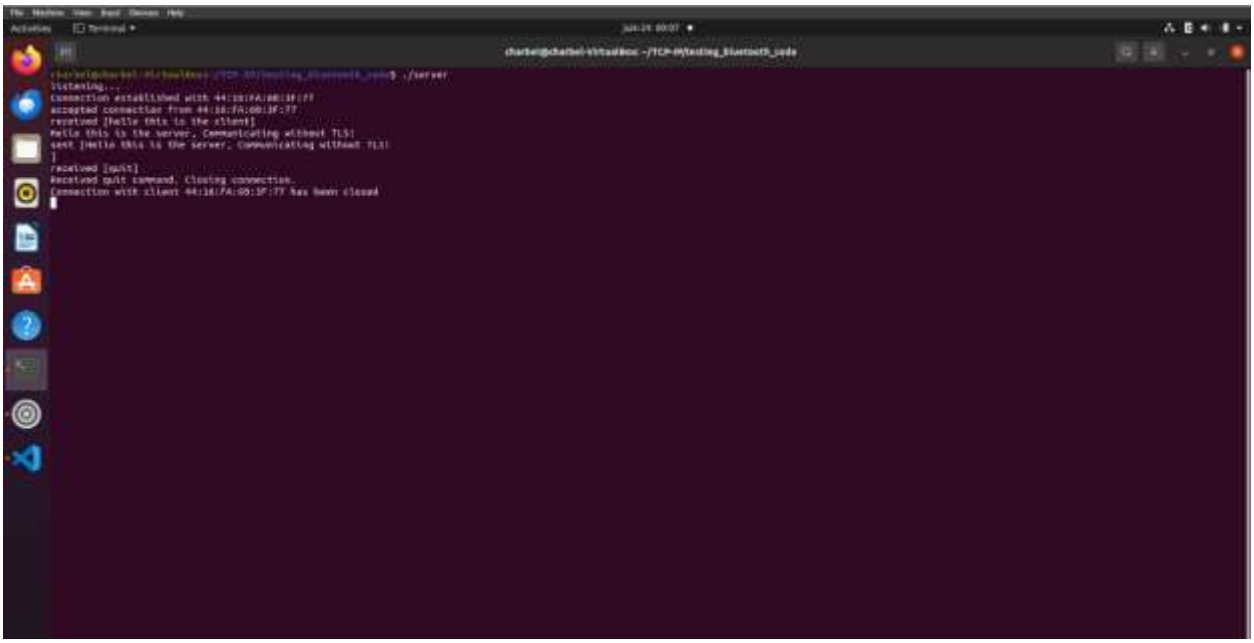
•Lorsque l'utilisateur appuie sur le bouton 'Disconnect', l'application envoie un message de type "quit" au serveur pour signaler la fin de la connexion. Ce message est programmé pour initier la procédure de déconnexion dans le code de l'application.

Échange de Messages lors de la Déconnexion :

- **Envoi du message de fin de session par l'application :**

Quand le bouton 'Disconnect' est pressé, l'application envoie un message explicite, typiquement "Client: Client Ended Connection", qui est transmis au serveur. Ce message indique que la connexion va être terminée du côté du client.

•**Réception et réponse du serveur :** Le serveur reçoit le message de fin de session du client et répond avec un message de confirmation de la fermeture de la connexion, par exemple "Server: Server Ended Connection". Ce message de réponse est ensuite reçu et affiché par l'application sur l'interface utilisateur du client.

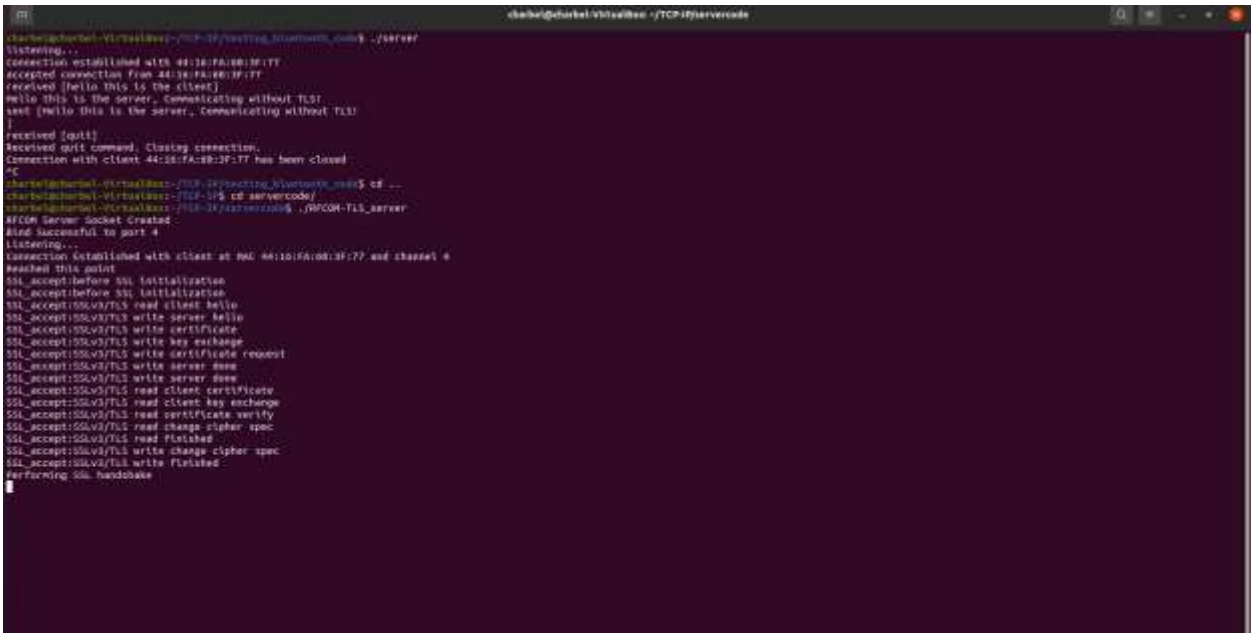

./server
listening...
connection established with 44:1B:FA:0B:3F:77
accepted connection from 44:1B:FA:0B:3F:77
received [hello this is the client]
Hello this is the server. Commpicating without TLS!
sent [Hello this is the server. Communicating without TLS!
]
received [quit]
Received quit command. Closing connection.
[connection with client 44:1B:FA:0B:3F:77 has been closed

Gestion Avancée des Connexions Bluetooth et Échanges de Messages avec TLS

Établissement de la Connexion Avec TLS :

- En appuyant sur le bouton **'Connect With TLS'**, l'application tente de se connecter au dispositif sélectionné en utilisant le protocole TLS pour une communication sécurisée.

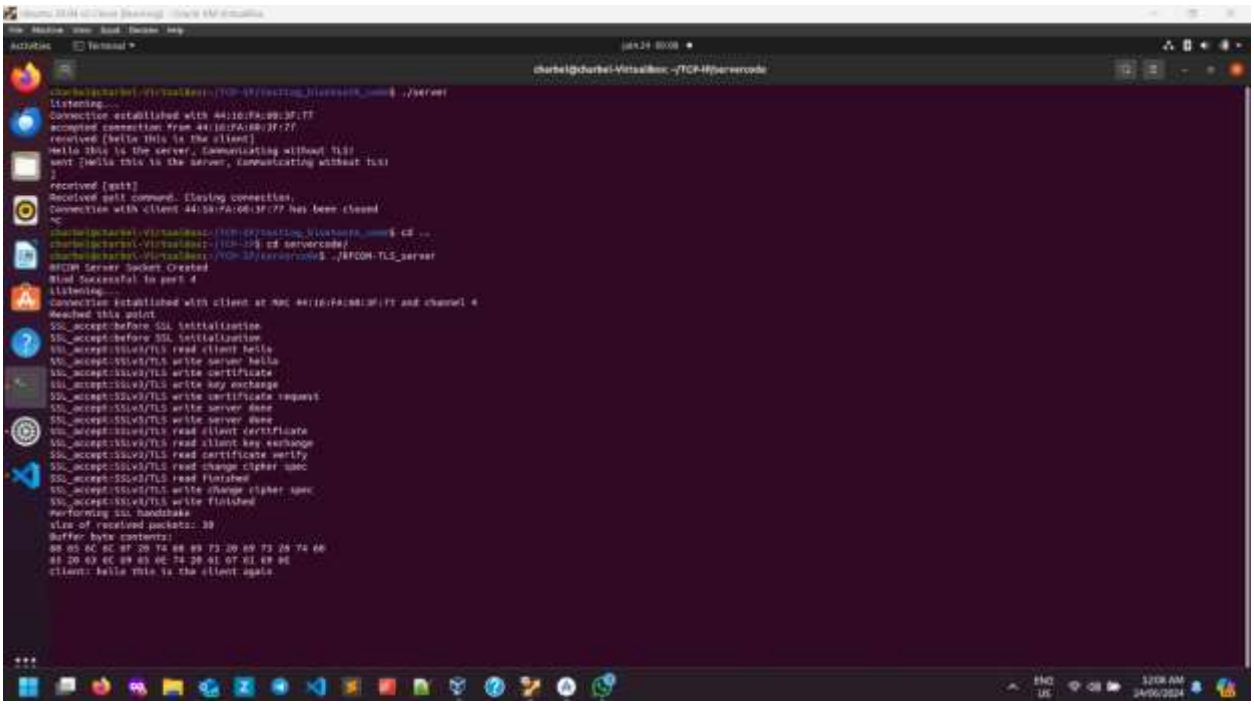
•**Visuel de confirmation** : Cette image du serveur indique que la connexion sécurisée a réussi.


./server
Listening...
Connection established with 44:16:FA:80:3F:77
accepted connection from 44:16:FA:80:3F:77
received [hello this is the client]
Hello this is the server, Communicating without TLS!
sent [Hello this is the server, Communicating without TLS!
]
received [quit]
Received quit command. Closing connection.
Connection with client 44:16:FA:80:3F:77 has been closed
^C
cd ..
cd servercode/
./RFCOM-TLS_server
RFCOM Server Socket Created
Bind Successful to port 4
Listening...
Connection Established with client at MAC 44:16:FA:80:3F:77 and channel 4
Reached this point
SSL_accept:before SSL initialization
SSL_accept:before SSL initialization
SSL_accept:SSLv3/TLS read client hello
SSL_accept:SSLv3/TLS write server hello
SSL_accept:SSLv3/TLS write certificate
SSL_accept:SSLv3/TLS write key exchange
SSL_accept:SSLv3/TLS write certificate request
SSL_accept:SSLv3/TLS write server done
SSL_accept:SSLv3/TLS write server done
SSL_accept:SSLv3/TLS read client certificate
SSL_accept:SSLv3/TLS read client key exchange
SSL_accept:SSLv3/TLS read certificate verify
SSL_accept:SSLv3/TLS read change cipher spec
SSL_accept:SSLv3/TLS read finished
SSL_accept:SSLv3/TLS write change cipher spec
SSL_accept:SSLv3/TLS write finished
Performing SSL handshake

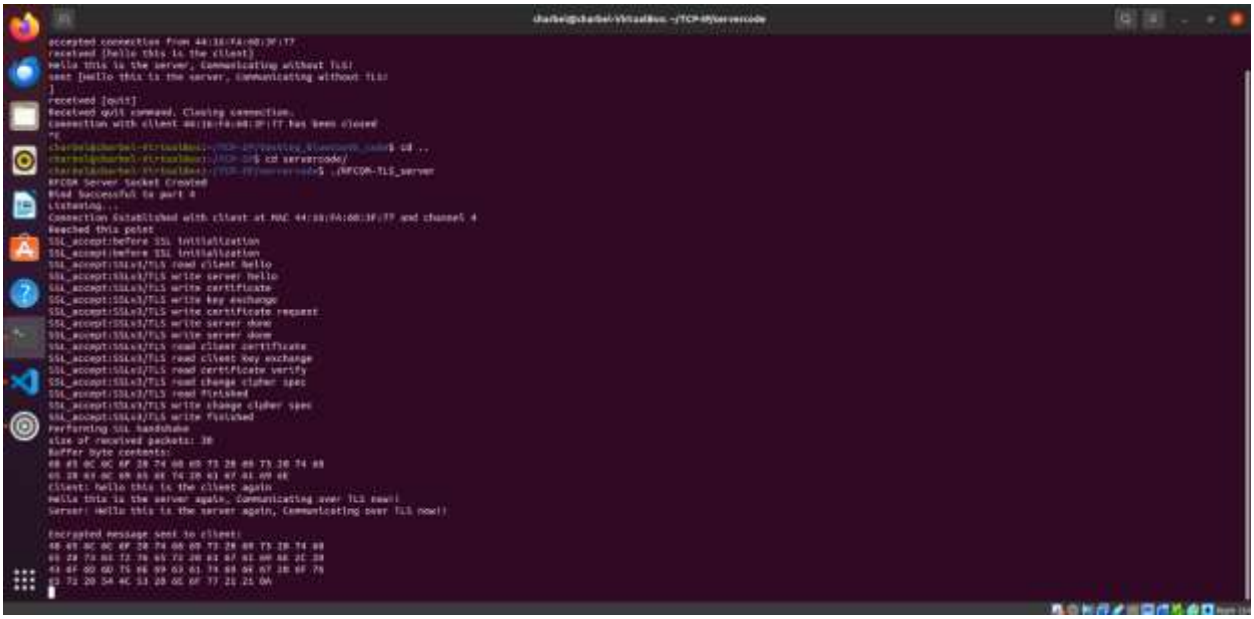
Échange de Messages Sécurisés :

- **Envoi de message sécurisé depuis l'application** : L'utilisateur saisit un message dans le champ de texte et l'envoie de manière sécurisée en appuyant sur 'Send'.

- **Réception du message sécurisé sur le serveur** : Cette image du serveur montre que le message envoyé depuis l'application avec TLS est bien arrivé au serveur.



- **Envoi de réponse sécurisée depuis le serveur** : Cette capture d'écran du serveur montre l'envoi d'une réponse sécurisée vers l'application.



- **Réception de la réponse sécurisée dans l'application** : Enfin, cette image de l'application montre le message sécurisé reçu du serveur.

Bouton 'Disconnect' avec TLS :

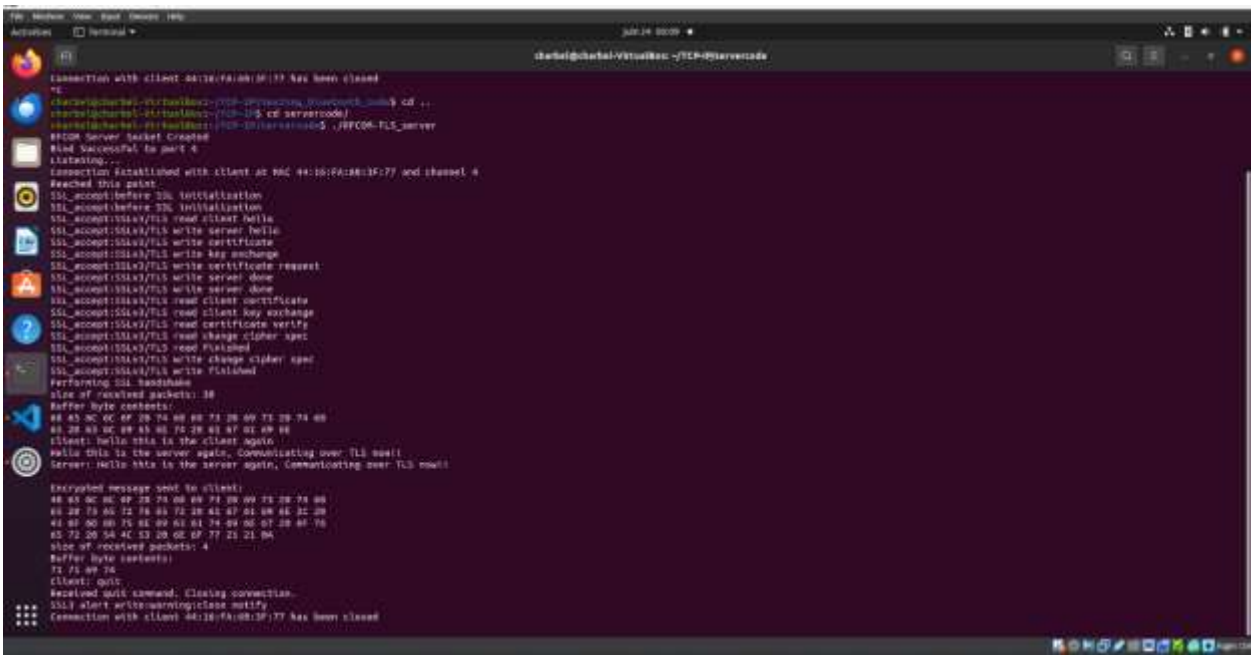
- Lorsque l'utilisateur appuie sur le bouton 'Disconnect', l'application envoie un message de type "quit" au serveur pour signaler la fin de la connexion sécurisée.

Échange de Messages lors de la Déconnexion avec TLS :

- **Envoi du message de fin de session par l'application :** Quand le bouton 'Disconnect' est pressé, l'application envoie un message explicite, "Client: Client Ended Connection", qui

est transmis au serveur. Ce message indique que la connexion va être terminée du côté du client.

- **Réception et réponse du serveur :** Le serveur reçoit le message de fin de session du client et répond avec un message de confirmation de la fermeture de la connexion, "Server: Server Ended Connection". Ce message de réponse est ensuite reçu et affiché par l'application sur l'interface utilisateur du client.


Connection with client 44:10:FA:08:3F:77 has been closed
^C
cd ..
cd servercode/
$ ./RFC08-TLS_server
RFC08 Server Socket Created
Bind Successful to port 4
Listening...
Connection Established with client at MAC 44:10:FA:08:3F:77 and channel 4
Reached this point
SSL_accept:before SSL initialization
SSL_accept:before SSL initialization
SSL_accept:SSLv3/TLS read client hello
SSL_accept:SSLv3/TLS write server hello
SSL_accept:SSLv3/TLS write certificate
SSL_accept:SSLv3/TLS write key exchange
SSL_accept:SSLv3/TLS write certificate request
SSL_accept:SSLv3/TLS write server done
SSL_accept:SSLv3/TLS write server done
SSL_accept:SSLv3/TLS read client certificate
SSL_accept:SSLv3/TLS read client key exchange
SSL_accept:SSLv3/TLS read certificate verify
SSL_accept:SSLv3/TLS read change cipher spec
SSL_accept:SSLv3/TLS read finished
SSL_accept:SSLv3/TLS write change cipher spec
SSL_accept:SSLv3/TLS write finished
Performing SSL handshake
size of received packets: 38
Buffer byte contents:
client: hello this is the client again ::
Hello this is the server again, Communicating over TLS now!!
Server: Hello this is the server again, Communicating over TLS now!!
Encrypted message sent to client:
size of received packets: 4
Buffer byte contents:
71 75 69 74
Client: quit
Received quit command. Closing connection.
SSL3 alert write:warning:close notify
Connection with client 44:10:FA:08:3F:77 has been closed

Section 4 : Analyse de trafic et interprétation

Objectif :

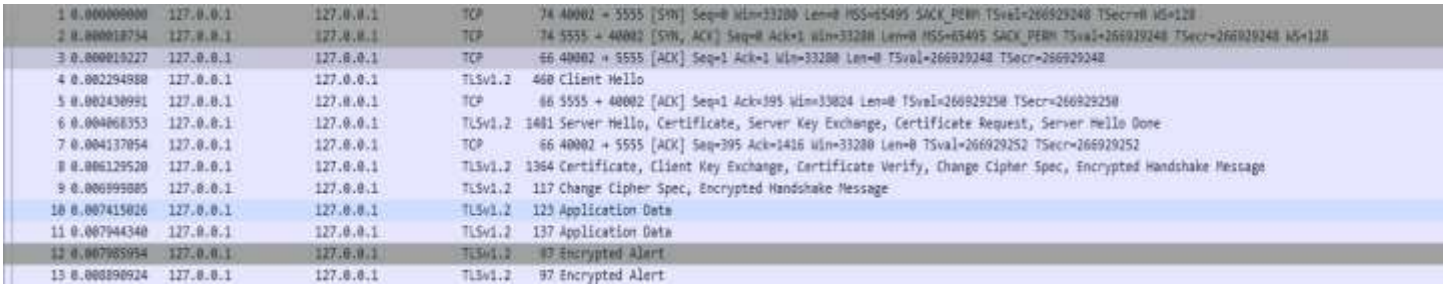
L'objectif de cette section est de fournir une analyse détaillée du trafic capturé afin de comprendre le fonctionnement du protocole TLS (Transport Layer Security) sur TCP/IP ainsi que sur Bluetooth.
En examinant les paquets échangés entre le client et le serveur, nous pourrons évaluer l'efficacité de TLS pour sécuriser les communications dans ces deux contextes distincts. Cette analyse nous permettra également d'identifier les points forts et les faiblesses de notre implémentation de TLS.

Pourquoi analyser le trafic ?

L'analyse de trafic est une étape cruciale pour assurer la sécurité des communications réseau. En étudiant les paquets échangés, nous pouvons vérifier que les mécanismes de chiffrement et d'authentification sont correctement appliqués et qu'il n'y a pas de vulnérabilités exploitables. L'analyse de trafic aide également à comprendre le comportement des protocoles en situation réelle, à diagnostiquer les problèmes de performance, et à s'assurer que les communications sont conformes aux spécifications de sécurité établies.

Pour mener cette analyse, nous avons principalement utilisé Wireshark, un outil d'analyse de trafic réseau reconnu pour sa puissance et convivialité. Wireshark permet de capturer et d'inspecter les paquets réseau à un niveau granulaire, offrant une interface graphique intuitive pour explorer les différentes couches de protocoles.

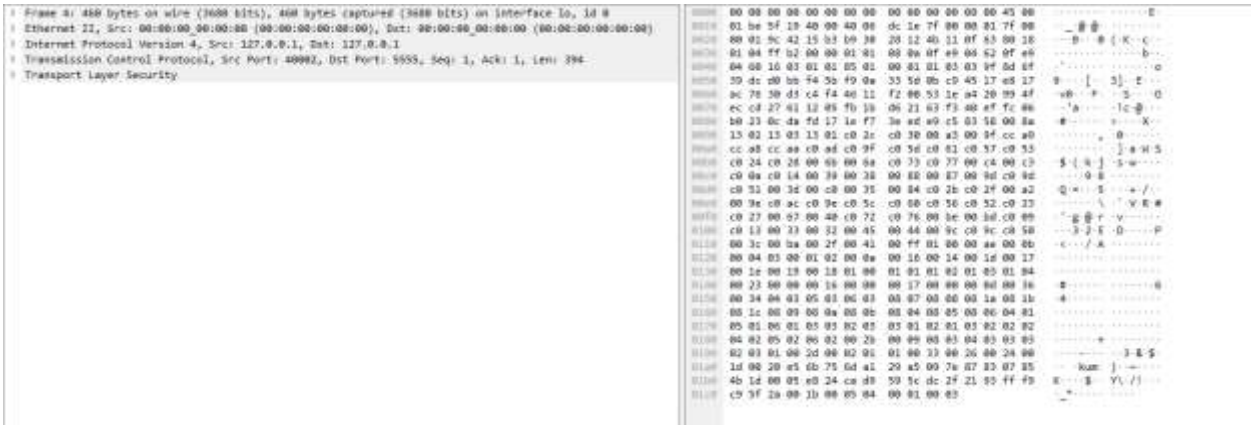
Analyse de Trafic avec TLS en TCP/IP



Dans cette capture d'écran prise de WireShark, on peut voir les types de paquets échangés entre le serveur et le client après l'établissement de la session.
Pour redonner une idée brève sur chaque paquet :
-Client Hello : Initie la communication, propose les paramètres de sécurité.
-Server Hello : Accepte les paramètres, envoie des informations nécessaires.
-Certificate : Authentifie le serveur.
-Server Key Exchange : Envoie des informations pour la génération des clés.
-Client Key Exchange : Transmet la clé de pré-maître secrète.
-Change Cipher Spec : Indique le début de l'utilisation des algorithmes de chiffrement négociés.
-Finished : Confirme la fin du handshake.

Nous allons maintenant examiner en détail chaque type de paquet capturé dans Wireshark pour illustrer le processus de handshake TLS et la transmission des données chiffrées.

Client Hello :



Passons à une petite analyse détaillée des bits :

****TLS Record Layer****

- `16`: Content Type (Handshake)
- `03 01`: Version (TLS 1.0
- `01 85`: Length (389 bytes)

****Handshake Protocol: Client Hello****

- `01`: Handshake Type (Client Hello)
- `00 01 81`: Length (385 bytes)
- `03 03`: Version (TLS 1.2)
- `9f 8d 6f 39 ...`: Random (32 bytes)
- `20`: Session ID Length (32)

****Cipher suites****

- `00 8a`: Cipher Suites Length (138 bytes)
- `13 02`: TLS_AES_256_GCM_SHA384
- `13 03`: TLS_CHACHA20_POLY1305_SHA256
- ...

- `00`: No compression

****Extensions****

- `00 2b 00 09 08 03 04 03 03 03 02 03 01`: Supported Versions
- `00 2d 00 02 01 01`: Psk Key Exchange Modes
- `00 33 00 26 00 24 00 1d 00 20 e5 6b 75 6d a1 29 a5 09 7e 87 83 07 85 4b 1d 00 05 e8 24 ca d9 59 5c dc 2f 21 93 ff f9 c9 5f 2a`: Key Share

Ce qui est important à noter est le suivant :

Le message `Client Hello` initie la communication sécurisée en proposant des paramètres de sécurité au serveur. Ce message inclut :
1. **Content Type** : 16 (Handshake)
2. **Version** : 03 01 (TLS 1.0)
3. **Length** : 01 85 (389 bytes)
4. **Handshake Type** : 01 (Client Hello)
5. **Client Version** : 03 03 (TLS 1.2)
6. **Random** : Une valeur aléatoire de 32 bytes, utilisée pour la génération des clés de session.
7. **Session ID Length** : 32 bytes
8. **Cipher Suites Length** : 138 bytes
9. **Cipher Suites** : Liste des suites de chiffrement proposées.
10. **Compression Methods** : No compression (00)
11. **Extensions** : Inclut des extensions comme le SNI (Server Name Indication).

Ce paquet est crucial pour négocier les paramètres de sécurité qui seront utilisés tout au long de la session TLS.

Server Hello :
Les bits obtenus sont nombreux dans ce cas, alors nous allons nous limiter à analyser les bits essentiels :

-02: Handshake Type (Server Hello)
-00 00 59: Length (89 bytes)
-03 03: Version (TLS 1.2)
-0f 96 f8 e4 b6 3d 37 71 9d 7d 6e a8 3d 3b ff ff af db 74 6d 79 da 3a 05 0d 97 0a 1c a4 cc 20 a4 : Random (32 bytes)
-20: Session ID Length (32 bytes)
-b6 a1 a4 7b 6f ba 74 bd b1 8f 9a 47 08 72 ba 17 74 2e f8 94 ec 69 df a2 90 c4 c2 8b 98 35 bd 2f: Session ID(32 bytes)
-c0 30: Cipher Suite (TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384)
-00: Compression Method (null)
-00 11: Extensions Length (17 bytes)
-ff 01 00 01 00: Extension (Renegotiation Info)
-00 0b 00 04 03 00 01 02: Extension (EC Point Formats)
-00 17 00 00: Extension (Extended master key length 0)

Interprétation

Le message Server Hello répond au Client Hello en confirmant les paramètres de sécurité pour la session TLS. Voici une explication détaillée des éléments :

-Content Type : 16 (Handshake)
-Version : 03 03 (TLS 1.2)
-Length : 00 5d (93 bytes)
-Handshake Type : 02 (Server Hello)
-Version : 03 03 (TLS 1.2)
-Random : Une valeur aléatoire de 32 bytes, utilisée pour la génération des clés de session.

-Session ID Length : 32 bytes
-Cipher Suite : c0 30 (TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384)
-Compression Method : null (00)
-Extensions Length : 00 11 (17 bytes)

-Extensions :

-ff 01 00 01 00 : Renegotiation Info

-00 0b 00 04 03 00 01 02 : EC Point Formats
-00 17 00 00 : Session Ticket

Ce paquet est crucial pour confirmer les paramètres de sécurité proposés par le client et ajouter les informations nécessaires pour sécuriser la session TLS.

Certificate, Client Key Exchange, Certificate Verify, Change Cipher Spec, Encrypted Handshake Message

TLS Record Layer
16: Content Type (Handshake)
03 03: Version (TLS 1.2)
03 a3: Length (931 bytes)

TLS Record Layer: Certificate
0b: Handshake Type (Certificate)
00 03 9f: Length (927 bytes)

Certificat

Le certificat contient plusieurs informations encodées en ASN.1 DER, dont :

30 82 03 95 30 82 02 7d 02 14 22 09 a1 b4 73 ab b4 3b b7 af fa ae 0a 86 91 e1 8b f6 aa c3 30 0d 06 09 2a 86 48 86 f7 0d 01 01 0b 05 00 30 81 86 31 0b 30 09 06 03 55 04 06 13 02 41 55 31 0f 30 0d 06 03 55 04 08 0c 06 73 74 61 74 65 42 31 0e 30 0c 06 03 55 04 07 0c 05 63 69 74 79 42 31 12 30 10 06 03 55 04 0a 0c 09 63 6f 6d 70 61 6e 79 42 5d 31 11 30 0f 06 03 55 04 0b 0c 08 73 65 63 74 69 6f 6e 42 31 0f 30 0d 06 03 55 04 03 0c 06 64 6f 6d 61 69 6e 31 1e 30 1c 06 09 2a 86 48 86 f7 0d 01 09 01 16 0f 65 6d 61 69 6c 40 65 6d 61 69 6c 2e 63 6f 6d 30 1e 17 0d 32 34 30 35 32 38 30 39 32 32 33 33 5a 17 0d 32 35 30 35 32 38 30 39 32 32 33 33 5a 30 81 86 31 0b 30 09 06 03 55 04 06 13 02 41 55 31 0f 30 0d 06 03 55 04 08 0c 06 73 74 61 74 65 42 31 0e 30 0c 06 03 55 04 07 0c 05 63 69 74 79 42 31 12 30 10 06 03 55 04 0a 0c 09 63 6f 6d 70 61 6e 79 42 5d 31 11 30 0f 06 03 55 04 0b 0c 08 73 65 63 74 69 6f 6e 42 31 0f 30 0d 06 03 55 04 03 0c 06 64 6f 6d 61 69 6e 31 1e 30 1c 06 09 2a 86 48 86 f7 0d 01 09 01 16 0f 65 6d 61 69 6c 40 65 6d 61 69 6c 2e 63 6f 6d 30 82 01 22 30 0d 06 09 2a 86 48 86 f7 0d 01 01 01 05 00 03 82 01 0f 00 30 82 01 0a 02 82 01 01 00 d0 f4 c7 cf b9 71 f7 c5 c5 ce 3f c3 b6 f9 6e 92 28 29 57 81 9f b5 38 7e d6 1a c6 c7 e5 ba 61 3b d7 4a ec f9 4a 35 c6 83 30 15 39 81 ec 37 a2 52 7f a6 f8 63 57 52 b5 b1 5c 49 4b 66 4b 33 ef f1 56 e1 ee 73 1b f4 c8 ab b5 30 54 9e bd 67 cf ef 8e 9f 64 7e 00 c3 15 d5 c0 df f6 ed b8 aa 79 f2 16 26 d6 4f 2b 37 70 01 ae ee a8 83 86 b1 78 d4 41 74 a2 bf f1 97 bf 63 9a ff f6 51 ed 86 7b 6c 5c f3 3d 89 39 9c f6 1f 85 0a 65 4d 96 c1 25 82 54 d8 61 3c 5a b9 d8 6f 4e 48 fa 45

0d 15 c1 39 57 5e a3 4f 39 49 a1 96 27 56 fb 91 a0 82 94 eb 3f e1 a4 1b d4 5f ce 19 46 82 38 56 cd bf cc 57 c9 3b 7e f0 dc 87 be 74 49 8c 70 6e d0 9c c8 2a 13 4e 07 9f c8 fd 10 ab 21 a1 ab e2 78 2d 1c 61 4a 61 92 ec ed 41 21 4c f5 19 c2 9d 2e cf 49 b5 14 ab 8a 93 88 6b b4 31 0c 5a 53 d8 e9 34 a8 eb 02 03 01 00 01 30 0d 06 09 2a 86 48 86 f7 0d 01 01 0b 05 00 03 82 01 01 00 74 42 c0 89 5e f7 b3 d6 60 b7 77 08 da c9 6c 99 34 55 06 c9 24 7e 23 37 b1 d0 6b c3 61 36 66 1e 2e c6 f8 4b 23 10 d0 d5 0d 5c c7 e9 09 3a 37 89 9d e1 4b d0 ac 50 7e e4 67 58 33 d2 c1 42 a1 de 35 f6 b3 c4 ff f8 09 d9 0b 48 d7 53 dd 45 35 6e e9 b2 17 2f 2d 9a 18 48 46 48 c9 27 bd 14 d3 f0 97 94 68 31 8a 2b 06 88 be cc 92 f6 f5 ab 3e d7 05 41 55 4e ec 72 f9 1d e8 cb 7f 7f e4 25 3d 0e 8b 29 ec 68 ae 34 ff 8a 90 22 42 ca d0 ef af 72 9e 9a f7 e0 d7 eb 16 93 32 11 59 7a eb a9 44 a7 7e 77 30 be 38 6f 49 09 42 dc aa 02 ca cf 6c c2 90 87 f8 d3 0f 71 69 f7 3a 70 98 2c 73 a6 54 a7 58 e6 14 15 81 29 ee 11 ab b1 4f 1a 7f 4d 6d b4 4b e7 54 9a ad 71 d4 23 a9 b6 5e 9d a1 72 fe 2b 0b 92 d6 85 ad e9 a2 76 7b 1c 12 b7 d8 15 49 fc f9 6b 0d d2 16 04 a8 49 b5 f1 72 7f e6 a4 05 43

Le certificat inclut plusieurs champs tels que Issuer, Subject, Validity, Subject Public Key Info et d'autres champs spécifiques au format de certificat X.509.

TLS Handshake Protocol: Client Key Exchange

10: Handshake Type (Client Key Exchange)

00 00 21: Length (33 bytes)

Les données de Client Key Exchange incluent les informations nécessaires pour que le serveur puisse générer la clé de session symétrique à partir de la clé publique du client.

TLS Handshake Protocol: Certificate Verify

0f: Handshake Type (Certificate Verify)

00 01 04: Length (260 bytes)

Certificate Verify est une étape où le client prouve qu'il possède la clé privée correspondant au certificat envoyé précédemment.

Change Cipher Spec

14: Content Type (Change Cipher Spec)

03 03: Version (TLS 1.2)

00 01: Length (1 byte)

01: Indique que le Change Cipher Spec est en cours

Encrypted Handshake Message

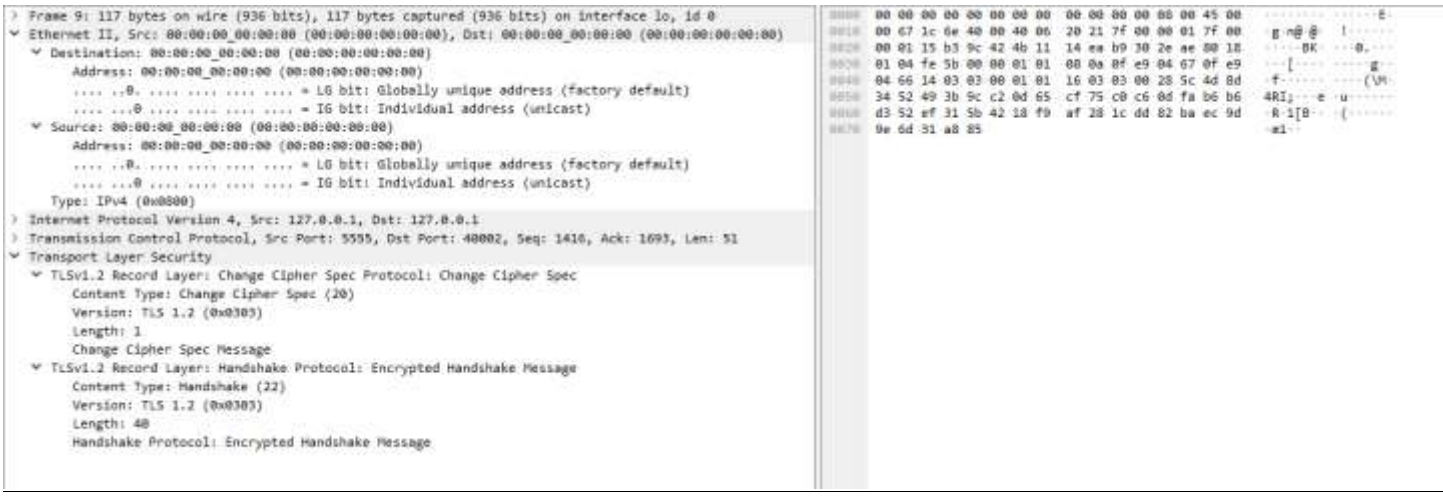
16: Content Type (Handshake)

03 03: Version (TLS 1.2)

00 28: Length (40 bytes)

L'Encrypted Handshake Message est le dernier message du handshake, chiffré pour garantir la confidentialité et l'intégrité de la communication.

Change Cipher Spec, Encrypted Handshake Message



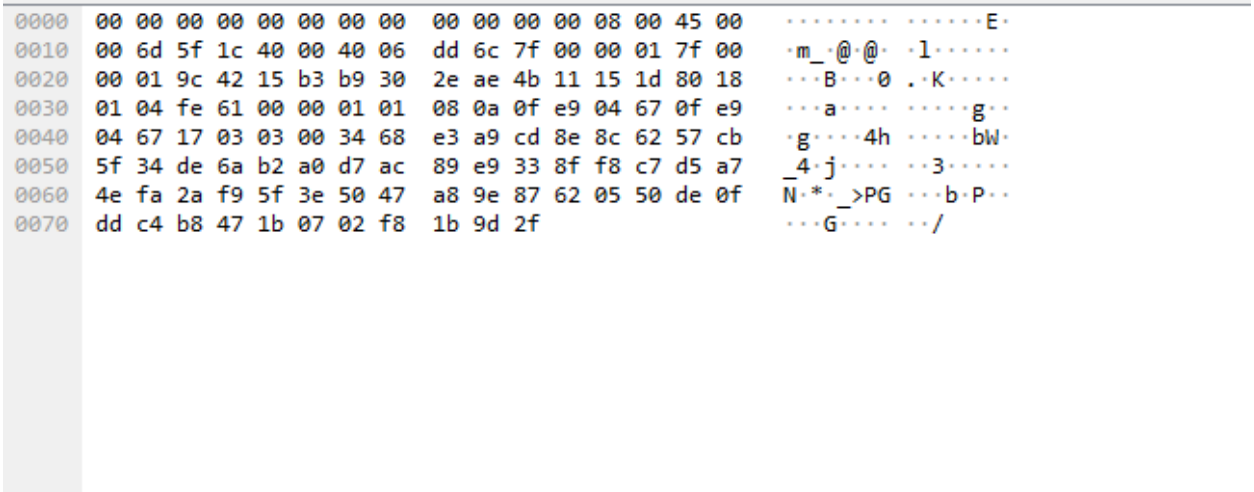
TLS record layer :
-14: Content Type (Change Cipher Spec)
-03 03: Version (TLS 1.2)
-00 01: Length (1 byte)
-01: Indique que le Change Cipher Spec est en cours

Encrypted Handshake Message
-16: Content Type (Handshake)
-03 03: Version (TLS 1.2)
-00 28: Length (40 bytes)
-Données du Handshake Message suivent ici (chiffrées)

-Le message Change Cipher Spec est simple et indique que le client passera aux nouveaux paramètres de chiffrement. C'est une étape critique car elle marque la transition vers une communication sécurisée.

-L'Encrypted Handshake Message contient des informations de fin de handshake qui sont chiffrées, telles que le Finished message qui est une preuve que le handshake s'est déroulé correctement en utilisant les clés établies.

Application Data :



-17: Content Type (Application Data)
-03 03: Version (TLS 1.2)
-00 34: Length (52 bytes)
-Encrypted Application Data

Les octets restants représentent les données d'application cryptées. Étant donné que les données sont cryptées, elles ne sont pas lisibles par un humain. Cependant, la présence de ces données indique que l'application utilise activement la connexion sécurisée établie pour transmettre sa charge utile.

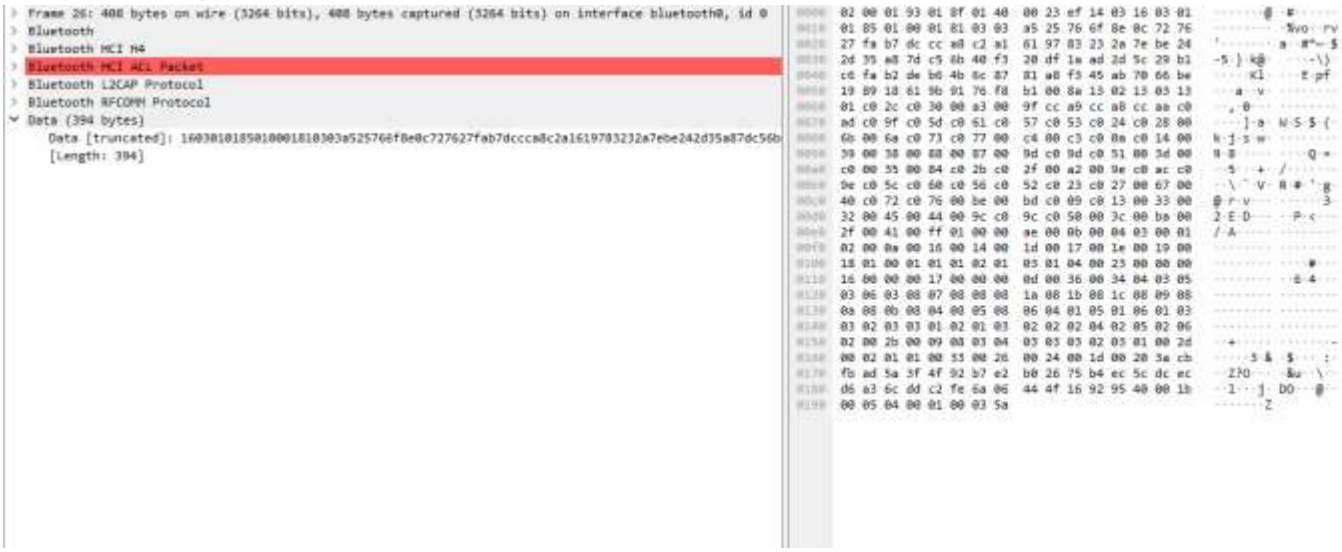
Pareil pour le dernier paquet qui est aussi un paquet « Application data »

Analyse de Trafic avec Bluetooth et TLS

	1	0.000000	controller	host	HCI_EVT	37 Rcvd LE Meta (LE Extended Advertising Report)
	2	0.240781	controller	host	HCI_EVT	258 Rcvd Extended Inquiry Result
*	3	1.551426	localhost ()	remote ()	L2CAP	17 Sent Connection Request (RFCOMM, SCID: 0x0040)
	4	1.600593	controller	host	HCI_EVT	8 Rcvd Number of Completed Packets
	5	1.603853	remote ()	localhost ()	L2CAP	21 Rcvd Connection Response - Success (SCID: 0x0040, DCID: 0x0040)
	6	1.603857	remote ()	localhost ()	L2CAP	32 Rcvd Configure Request (DCID: 0x0040)
	7	1.603914	localhost ()	remote ()	L2CAP	32 Sent Configure Request (DCID: 0x0040)
	8	1.603982	localhost ()	remote ()	L2CAP	23 Sent Configure Response - Success (SCID: 0x0040)
	9	1.655860	remote ()	localhost ()	L2CAP	23 Rcvd Configure Response - Success (SCID: 0x0040)
	10	1.655935	localhost ()	remote ()	RFCOMM	13 Sent SABM Channel=0
	11	1.677161	controller	host	HCI_EVT	8 Rcvd Number of Completed Packets
	12	1.706850	controller	host	HCI_EVT	8 Rcvd Number of Completed Packets
	13	1.708673	controller	host	HCI_EVT	8 Rcvd Number of Completed Packets
	14	1.739823	remote ()	localhost ()	RFCOMM	13 Rcvd UA Channel=0
	15	1.739882	localhost ()	remote ()	RFCOMM	23 Sent UIH Channel=0 -> 4 MPX_CTRL DLC Parameter Negotiation (PN)
	16	1.791804	remote ()	localhost ()	RFCOMM	23 Rcvd UIH Channel=0 -> 4 MPX_CTRL DLC Parameter Negotiation (PN)
	17	1.791866	localhost ()	remote ()	RFCOMM	13 Sent SABM Channel=4 (Unknown)
	18	1.811891	controller	host	HCI_EVT	8 Rcvd Number of Completed Packets
	19	1.836536	controller	host	HCI_EVT	8 Rcvd Number of Completed Packets
	20	1.836722	remote ()	localhost ()	RFCOMM	13 Rcvd UA Channel=4
	21	1.836744	localhost ()	remote ()	RFCOMM	17 Sent UIH Channel=0 -> 4 MPX_CTRL Modem Status Command (MSC)
	22	1.870111	remote ()	localhost ()	RFCOMM	17 Rcvd UIH Channel=0 -> 4 MPX_CTRL Modem Status Command (MSC)
	23	1.870180	localhost ()	remote ()	RFCOMM	17 Sent UIH Channel=0 -> 4 MPX_CTRL Modem Status Command (MSC)
	24	1.891813	remote ()	localhost ()	RFCOMM	17 Rcvd UIH Channel=0 -> 4 MPX_CTRL Modem Status Command (MSC)
	25	1.891876	localhost ()	remote ()	RFCOMM	14 Sent UIH Channel=4 UID
	26	1.891940	localhost ()	remote ()	RFCOMM	488 Sent UIH Channel=4
	27	1.905941	controller	host	HCI_EVT	8 Rcvd Number of Completed Packets
	28	1.906728	controller	host	HCI_EVT	8 Rcvd Number of Completed Packets
	29	1.907894	controller	host	HCI_EVT	8 Rcvd Number of Completed Packets
	30	1.939790	remote ()	localhost ()	RFCOMM	14 Rcvd UIH Channel=4 UID
	31	1.960893	controller	host	HCI_EVT	8 Rcvd Number of Completed Packets
	32	1.961111	remote ()	localhost ()	HCI_ACL	349 Rcvd [Reassembled in #35]
	33	1.961112	remote ()	localhost ()	HCI_ACL	349 Rcvd [Continuation to #32] [Reassembled in #35]
	34	1.970085	remote ()	localhost ()	HCI_ACL	338 Rcvd [Continuation to #32] [Reassembled in #35]
	35	1.991295	remote ()	localhost ()	RFCOMM	9 Rcvd UIH Channel=4
	36	2.022338	remote ()	localhost ()	HCI_ACL	349 Rcvd [Reassembled in #37]
	37	2.022342	remote ()	localhost ()	RFCOMM	29 Rcvd UIH Channel=4
	38	2.023820	localhost ()	remote ()	HCI_ACL	1026 Sent [Reassembled in #39]
	39	2.024089	localhost ()	remote ()	RFCOMM	9 Sent UIH Channel=4
	40	2.024171	localhost ()	remote ()	RFCOMM	296 Sent UIH Channel=4
	41	2.057957	controller	host	HCI_EVT	8 Rcvd Number of Completed Packets
	42	2.116050	controller	host	HCI_EVT	8 Rcvd Number of Completed Packets
	43	2.117866	controller	host	HCI_EVT	8 Rcvd Number of Completed Packets
	44	2.151847	remote ()	localhost ()	HCI_ACL	349 Rcvd [Reassembled in #47]
	45	2.151851	remote ()	localhost ()	HCI_ACL	349 Rcvd [Continuation to #44] [Reassembled in #47]
	46	2.151851	remote ()	localhost ()	HCI_ACL	338 Rcvd [Continuation to #44] [Reassembled in #47]
	47	2.177489	remote ()	localhost ()	RFCOMM	9 Rcvd UIH Channel=4
	48	2.200621	remote ()	localhost ()	RFCOMM	152 Rcvd UIH Channel=4
	49	2.204533	localhost ()	remote ()	RFCOMM	70 Sent UIH Channel=4
	50	2.264778	remote ()	localhost ()	RFCOMM	84 Rcvd UIH Channel=4
	51	2.266178	localhost ()	remote ()	RFCOMM	44 Sent UIH Channel=4
	52	2.266639	localhost ()	remote ()	RFCOMM	13 Sent DISC Channel=4
	53	2.274954	controller	host	HCI_EVT	8 Rcvd Number of Completed Packets
	54	2.281877	remote ()	localhost ()	RFCOMM	44 Rcvd UIH Channel=4
	55	2.281880	remote ()	localhost ()	RFCOMM	13 Rcvd DISC Channel=4
	56	2.281937	localhost ()	remote ()	RFCOMM	13 Sent UA Channel=4
	57	2.306546	controller	host	HCI_EVT	8 Rcvd Number of Completed Packets
	58	2.325024	controller	host	HCI_EVT	8 Rcvd Number of Completed Packets
	59	2.325206	remote ()	localhost ()	RFCOMM	13 Rcvd UA Channel=4
	60	2.325207	remote ()	localhost ()	RFCOMM	13 Rcvd DM Channel=4
	61	2.325224	localhost ()	remote ()	RFCOMM	13 Sent DM Channel=4
	62	2.325874	controller	host	HCI_EVT	8 Rcvd Number of Completed Packets
	63	2.401192	controller	host	HCI_EVT	8 Rcvd Number of Completed Packets
	64	2.501299	controller	host	HCI_EVT	60 Rcvd LE Meta (LE Extended Advertising Report)

Dans cette capture d'ecran prise de wireshark, on peut voir les types de paquets échangés entre le serveur et le client après l'etablissement de la session.

Analyse du Paquet RFCOMM Contenant TLS : Client Hello



Le paquet commence par 16 03 01, ce qui est typique pour les paquets TLS. Voici une décomposition des premiers octets :

-16 : Content Type (Handshake)
-03 01 : Version (TLS 1.0)
-01 85 : Length (389 octets)

Dans ce cas, c'est un *Client Hello*

Le reste du paquet contient les données spécifiques au handshake TLS. Continuons à interpréter ces données :

-Content Type (Handshake) : 16
-Version (TLS 1.0) : 03 01
-Length : 01 85 (389 octets)
-Handshake Type (Client Hello) : 01
-Length : 00 01 81 (385 octets)
-Version : 03 03 (TLS 1.2)
-Random : a5 25 76 6f 8e 0c 72 76 27 fa b7 dc cc a8 c2 a1 61 97 83 23 2a 7e be 24 2d 35 a8 7d c5 6b 40 f3
-`20`: Session ID length (32 bytes)
-`df 1a ad 2d 5c 29 b1 c6 fa b2 de b6 4b 6c 87 81 a8 f3 45 ab 70 66 be 19 89 18 61 9b 91 76 f8 b1`: Session ID
- `00 8a`: Cipher suites length (138 bytes)
-Cipher Suites : 13 02 13 03 13 01 c0 2c c0 30 00 a3 00 9f cc a9 cc a8 cc aa c0 ad c0 9f c0 5d c0 61 c0 57 c0 53 c0 24 c0 28 00 6b 00 6a c0 73 c0 77 00 c4 00 c3 c0 0a c0 14 00 39 00 38 00 88 00 87 00 9d c0 9d c0 51 00 3d 00 c0 00 35 00 84 c0 2b c0 2f 00 a2 00 9e c0 ac c0 9e c0 5c c0 60 c0 56 c0 52 c0 23 c0 27 00 67 00 40 c0 72 c0 76 00 be 00 bd c0 09 c0 13 00 33 00 32 00 45 00 44 00 9c c0 9c c0 50 00 3c 00 ba 00 2f 00 41 00 ff
-Compression Methods Length : 01

-Compression Methods : 00
-Extensions Length : 00 ae
-Extensions : Diverses extensions comme 00 0b 00 04 03 00 01 02, etc

Analyse du Paquet HCI_ACL Contenant TLS : Server Hello

```
0000  02 00 21 58 01 fd 03 40  00 21 ef f0 07 16 03 03   ··!X···@ ·!·····
0010  00 41 02 00 00 3d 03 03  2f 9f 1e 92 fc 41 b4 69   ·A···=·· /····A·i
0020  a0 cf 5c 24 a7 bd dd 6e  fe a3 fa 65 78 82 8f 7a   ··\$···n ···ex··z
0030  5d 44 7c 0b e9 93 2b 3f  00 c0 30 00 00 15 ff 01   ]D|···+? ··0·····
0040  00 01 00 00 0b 00 04 03  00 01 02 00 23 00 00 00   ········ ····#···
0050  17 00 00 16 03 03 03 a1  0b 00 03 9d 00 03 9a 00   ········ ········
0060  03 97 30 82 03 93 30 82  02 7b 02 14 09 b9 55 5d   ··0···0· ·{····U]
0070  9e e5 f0 d1 03 4c 09 84  51 6e 36 10 fa 53 70 cf   ·····L·· Qn6··Sp·
0080  30 0d 06 09 2a 86 48 86  f7 0d 01 01 0b 05 00 30   0···*·H· ·······0
0090  81 85 31 0b 30 09 06 03  55 04 06 13 02 41 55 31   ··1·0··· U····AU1
00a0  0f 30 0d 06 03 55 04 08  0c 06 73 74 61 74 65 41   ·0···U·· ··stateA
00b0  31 0e 30 0c 06 03 55 04  07 0c 05 63 69 74 79 41   1·0···U· ···cityA
00c0  31 11 30 0f 06 03 55 04  0a 0c 08 63 6f 6d 70 61   1·0···U· ···compa
00d0  6e 79 41 31 11 30 0f 06  03 55 04 0b 0c 08 73 65   nyA1·0·· ·U····se
00e0  63 74 69 6f 6e 41 31 0f  30 0d 06 03 55 04 03 0c   ctionA1· 0···U···
00f0  06 64 6f 6d 61 69 6e 31  1e 30 1c 06 09 2a 86 48   ·domain1 ·0···*·H
0100  86 f7 0d 01 09 01 16 0f  65 6d 61 69 6c 40 65 6d   ········ email@em
0110  61 69 6c 2e 63 6f 6d 30  1e 17 0d 32 34 30 35 32   ail.com0 ···24052
0120  38 30 39 32 32 30 33 5a  17 0d 32 35 30 35 32 38   8092203Z ··250528
0130  30 39 32 32 30 33 5a 30  81 85 31 0b 30 09 06 03   092203Z0 ··1·0···
0140  55 04 06 13 02 41 55 31  0f 30 0d 06 03 55 04 08   U····AU1 ·0···U··
0150  0c 06 73 74 61 74 65 41  31 0e 30 0c 06            ··stateA 1·0··
```

Le paquet commence par 16 03 03, ce qui est typique pour les paquets TLS. Voici une décomposition des premiers octets :

-16 : Content Type (Handshake)
-03 03 : Version (TLS 1.2)
-00 41 : Length (65 octets)

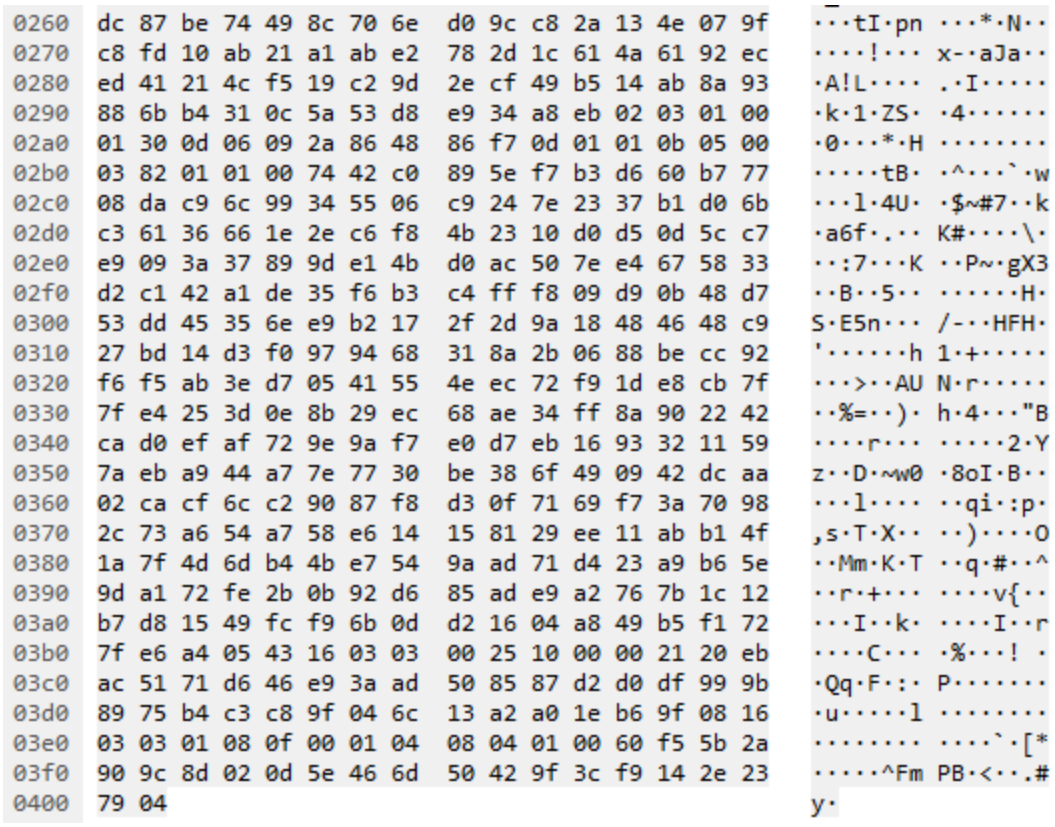
Dans ce cas, c'est un *Server Hello*

-16 : Content Type (Handshake)
-03 03 : Version (TLS 1.2)
-00 41 : Length (65 octets)
-02 : Handshake Type (Server Hello)
-00 00 3d : Length (61 octets)
-03 03 : Version (TLS 1.2)
-2f 9f 1e 92 fc 41 b4 69 a0 cf 5c 24 a7 bd dd 6e fe a3 fa 65 78 82 8f 7a 5d 44 7c 0b e9 93 2b 3f : Random (32 octets)
-c0 30 : Cipher Suite (TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384)

-00 : Compression Method (null)
-00 15 : Extensions Length (21 octets)
-ff 01 00 01 00 : Extension (Renegotiation Info)
-00 0b 00 04 03 00 01 02 : Extension (EC Point Formats)
-00 23 00 00 : Extension (Session Ticket)
-00 17 00 00 : Extension (Extended Master Secret)

Analyse du paquet 38 HCI-ACL contenant TLS :

```
0000  02 00 01 fd 03 fd 03 40  00 23 ef f0 07 16 03 03   ······@ ·#·····
0010  03 a3 0b 00 03 9f 00 03  9c 00 03 99 30 82 03 95   ········ ····0···
0020  30 82 02 7d 02 14 22 09  a1 b4 73 ab b4 3b b7 af   0··}··"· ··s··;··
0030  fa ae 0a 86 91 e1 8b f6  aa c3 30 0d 06 09 2a 86   ········ ··0···*·
0040  48 86 f7 0d 01 01 0b 05  00 30 81 86 31 0b 30 09   H····... ·0··1·0·
0050  06 03 55 04 06 13 02 41  55 31 0f 30 0d 06 03 55   ··U····A U1·0···U
0060  04 08 0c 06 73 74 61 74  65 42 31 0e 30 0c 06 03   ····stat eB1·0···
0070  55 04 07 0c 05 63 69 74  79 42 31 12 30 10 06 03   U····cit yB1·0···
0080  55 04 0a 0c 09 63 6f 6d  70 61 6e 79 42 5d 31 11   U····com panyB]1·
0090  30 0f 06 03 55 04 0b 0c  08 73 65 63 74 69 6f 6e   0···U··· ·section
00a0  42 31 0f 30 0d 06 03 55  04 03 0c 06 64 6f 6d 61   B1·0···U ····doma
00b0  69 6e 31 1e 30 1c 06 09  2a 86 48 86 f7 0d 01 09   in1·0··· *·H·····
00c0  01 16 0f 65 6d 61 69 6c  40 65 6d 61 69 6c 2e 63   ···email @email.c
00d0  6f 6d 30 1e 17 0d 32 34  30 35 32 38 30 39 32 32   om0···24 05280922
00e0  33 33 5a 17 0d 32 35 30  35 32 38 30 39 32 32 33   33Z··250 52809223
00f0  33 5a 30 81 86 31 0b 30  09 06 03 55 04 06 13 02   3Z0··1·0 ···U····
0100  41 55 31 0f 30 0d 06 03  55 04 08 0c 06 73 74 61   AU1·0··· U····sta
0110  74 65 42 31 0e 30 0c 06  03 55 04 07 0c 05 63 69   teB1·0·· ·U····ci
0120  74 79 42 31 12 30 10 06  03 55 04 0a 0c 09 63 6f   tyB1·0·· ·U····co
0130  6d 70 61 6e 79 42 5d 31  11 30 0f 06 03 55 04 0b   mpanyB]1 ·0···U··
0140  0c 08 73 65 63 74 69 6f  6e 42 31 0f 30 0d 06 03   ··sectio nB1·0···
0150  55 04 03 0c 06 64 6f 6d  61 69 6e 31 1e 30 1c 06   U····dom ain1·0··
0160  09 2a 86 48 86 f7 0d 01  09 01 16 0f 65 6d 61 69   ·*·H···· ····emai
0170  6c 40 65 6d 61 69 6c 2e  63 6f 6d 30 82 01 22 30   l@email. com0··"0
0180  0d 06 09 2a 86 48 86 f7  0d 01 01 01 05 00 03 82   ···*·H·· ········
0190  01 0f 00 30 82 01 0a 02  82 01 01 00 d0 f4 c7 cf   ···0···· ········
01a0  b9 71 f7 c5 c5 ce 3f c3  b6 f9 6e 92 28 29 57 81   ·q····?· ··n·()W·
01b0  9f b5 38 7e d6 1a c6 c7  e5 ba 61 3b d7 4a ec f9   ··8~···· ··a;·J··
01c0  4a 35 c6 83 30 15 39 81  ec 37 a2 52 7f a6 f8 63   J5··0·9· ·7·R···c
01d0  57 52 b5 b1 5c 49 4b 66  4b 33 ef f1 56 e1 ee 73   WR··\IKf K3··V··s
01e0  1b f4 c8 ab b5 30 54 9e  bd 67 cf ef 8e 9f 64 7e   ·····0T· ·g····d~
01f0  00 c3 15 d5 c0 df f6 ed  b8 aa 79 f2 16 26 d6 4f   ········ ··y··&·O
0200  2b 37 70 01 ae ee a8 83  86 b1 78 d4 41 74 a2 bf   +7p····· ··x·At··
0210  f1 97 bf 63 9a ff f6 51  ed 86 7b 6c 5c f3 3d 89   ···c···Q ··{l\·=·
0220  39 9c f6 1f 85 0a 65 4d  96 c1 25 82 54 d8 61 3c   9·····eM ··%·T·a<
0230  5a b9 d8 6f 4e 48 fa 45  0d 15 c1 39 57 5e a3 4f   Z··oNH·E ···9W^·O
0240  39 49 a1 96 27 56 fb 91  a0 82 94 eb 3f e1 a4 1b   9I··'V·· ····?···
0250  d4 5f ce 19 46 82 38 56  cd bf cc 57 c9 3b 7e f0   ·_··F·8V ···W·;~·
```



```
0260  dc 87 be 74 49 8c 70 6e  d0 9c c8 2a 13 4e 07 9f   ···tI·pn ···*·N··
0270  c8 fd 10 ab 21 a1 ab e2  78 2d 1c 61 4a 61 92 ec   ····!··· x-·aJa··
0280  ed 41 21 4c f5 19 c2 9d  2e cf 49 b5 14 ab 8a 93   ·A!L···· ..I·····
0290  88 6b b4 31 0c 5a 53 d8  e9 34 a8 eb 02 03 01 00   ·k·1·ZS· ·4·····
02a0  01 30 0d 06 09 2a 86 48  86 f7 0d 01 01 0b 05 00   ·0···*·H ········
02b0  03 82 01 01 00 74 42 c0  89 5e f7 b3 d6 60 b7 77   ·····tB· ·^···`·w
02c0  08 da c9 6c 99 34 55 06  c9 24 7e 23 37 b1 d0 6b   ···l·4U· ·$~#7··k
02d0  c3 61 36 66 1e 2e c6 f8  4b 23 10 d0 d5 0d 5c c7   ·a6f·.·· K#····\·
02e0  e9 09 3a 37 89 9d e1 4b  d0 ac 50 7e e4 67 58 33   ··:7···K ··P~·gX3
02f0  d2 c1 42 a1 de 35 f6 b3  c4 ff f8 09 d9 0b 48 d7   ··B··5·· ······H·
0300  53 dd 45 35 6e e9 b2 17  2f 2d 9a 18 48 46 48 c9   S·E5n··· /-··HFH·
0310  27 bd 14 d3 f0 97 94 68  31 8a 2b 06 88 be cc 92   '·····h 1·+·····
0320  f6 f5 ab 3e d7 05 41 55  4e ec 72 f9 1d e8 cb 7f   ···>··AU N·r·····
0330  7f e4 25 3d 0e 8b 29 ec  68 ae 34 ff 8a 90 22 42   ··%=··)· h·4···"B
0340  ca d0 ef af 72 9e 9a f7  e0 d7 eb 16 93 32 11 59   ····r··· ·····2·Y
0350  7a eb a9 44 a7 7e 77 30  be 38 6f 49 09 42 dc aa   z··D·~w0 ·8oI·B··
0360  02 ca cf 6c c2 90 87 f8  d3 0f 71 69 f7 3a 70 98   ···l···· ··qi·:p·
0370  2c 73 a6 54 a7 58 e6 14  15 81 29 ee 11 ab b1 4f   ,s·T·X·· ··)····O
0380  1a 7f 4d 6d b4 4b e7 54  9a ad 71 d4 23 a9 b6 5e   ··Mm·K·T ··q·#··^
0390  9d a1 72 fe 2b 0b 92 d6  85 ad e9 a2 76 7b 1c 12   ··r·+··· ····v{··
03a0  b7 d8 15 49 fc f9 6b 0d  d2 16 04 a8 49 b5 f1 72   ···I··k· ····I··r
03b0  7f e6 a4 05 43 16 03 03  00 25 10 00 00 21 20 eb   ····C··· ·%···! ·
03c0  ac 51 71 d6 46 e9 3a ad  50 85 87 d2 d0 df 99 9b   ·Qq·F·:· P·······
03d0  89 75 b4 c3 c8 9f 04 6c  13 a2 a0 1e b6 9f 08 16   ·u·····l ········
03e0  03 03 01 08 0f 00 01 04  08 04 01 00 60 f5 5b 2a   ········ ····`·[*
03f0  90 9c 8d 02 0d 5e 46 6d  50 42 9f 3c f9 14 2e 23   ·····^Fm PB·<···#
0400  79 04                                             y·
```

Ce paquet contient une partie importante d'une transmission TLS, indiquée par la présence des en-têtes et des structures courantes du protocole TLS.

En-tête TLS Record Layer

Les octets suivants indiquent qu'il s'agit d'un enregistrement TLS:
16: Content Type (Handshake)
03 03: Version (TLS 1.2)
03 a3: Length (931 octets)

Handshake TLS :

0b 00 03 9f 00 03 9c 00 03 99 30 82 03 95
0b : Handshake Type *(Certificate)*
00 03 9f : Length (927 octets)
00 03 9c : Certificate Length (924 octets)

Contenu du Certificat :

30 82 03 95 30 82 02 7d 02 14 22 09 a1 b4 73 ab b4 3b b7 af fa ae 0a 86 91 e1 8b f6 aa c3 30 0d 06 09 2a 86 48 86 f7 0d 01 01 0b 05 00 30 81 86 31 0b 30 09 06 03 55 04 06 13 02 41 55 31 0f 30 0d 06 03 55 04 08 0c 06 73 74 61 74 65 42 31 0e 30 0c 06 03 55 04 07 0c 05 63 69 74 79 42 31 12 30 10 06 03 55 04 0a 0c 09 63 6f 6d 70 61 6e 79 42 5d 31 11 30 0f 06 03 55 04 0b 0c 08 73 65 63 74 69 6f 6e 42 31 0f 30 0d 06 03 55 04 03 0c 06 64 6f 6d 61 69 6e 31 1e 30 1c 06 09 2a 86 48 86 f7 0d 01 09 01 16 0f 65 6d 61 69 6c 40 65 6d 61 69 6c 2e 63 6f 6d 30 1e 17 0d 32 34 30 35 32 38 30 39 32 32 33 33 5a 17 0d

32 35 30 35 32 38 30 39 32 32 33 33 5a 30 81 86 31 0b 30 09 06 03 55 04 06 13 02 41 55 31 0f 30 0d 06 03 55 04 08 0c 06 73 74 61 74 65 42 31 0e 30 0c 06 03 55 04 07 0c 05 63 69 74 79 42 31 12 30 10 06 03 55 04 0a 0c 09 63 6f 6d 70 61 6e 79 42 5d 31 11 30 0f 06 03 55 04 0b 0c 08 73 65 63 74 69 6f 6e 42 31 0f 30 0d 06 03 55 04 03 0c 06 64 6f 6d 61 69 6e 31 1e 30 1c 06 09 2a 86 48 86 f7 0d 01 09 01 16 0f 65 6d 61 69 6c 40 65 6d 61 69 6c 2e 63 6f 6d 30 82 01 22 30 0d 06 09 2a 86 48 86 f7 0d 01 01 01 05 00 03 82 01 0f 00 30 82 01 0a 02 82 01 01 00 d0 f4 c7 cf b9 71 f7 c5 c5 ce 3f c3 b6 f9 6e 92 28 29 57 81 9f b5 38 7e d6 1a c6 c7 e5 ba 61 3b d7 4a ec f9 4a 35 c6 83 30 15 39 81 ec 37 a2 52 7f a6 f8 63 57 52 b5 b1 5c 49 4b 66 4b 33 ef f1 56 e1 ee 73 1b f4 c8 ab b5 30 54 9e bd 67 cf ef 8e 9f 64 7e 00 c3 15 d5 c0 df f6 ed b8 aa 79 f2 16 26 d6 4f 2b 37 70 01 ae ee a8 83 86 b1 78 d4 41 74 a2 bf f1 97 bf 63 9a ff f6 51 ed 86 7b 6c 5c f3 3d 89 39 9c f6 1f 85 0a 65 4d 96 c1 25 82 54 d8 61 3c 5a b9 d8 6f 4e 48 fa 45 0d 15 c1 39 57 5e a3 4f 39 49 a1 96 27 56 fb 91 a0 82 94 eb 3f e1 a4 1b d4 5f ce 19 46 82 38 56 cd bf cc 57 c9 3b 7e f0 dc 87 be 74 49 8c 70 6e d0 9c c8 2a 13 4e 07 9f c8 fd 10 ab 21 a1 ab e2 78 2d 1c 61 4a 61 92 ec ed 41 21 4c f5 19 c2 9d 2e cf 49 b5 14 ab 8a 93 88 6b b4 31 0c 5a 53 d8 e9 34 a8 eb 02 03 01 00 01 30 0d 06 09 2a 86 48 86 f7 0d 01 01 0b 05 00 03 82 01 01 00 74 42 c0 89 5e f7 b3 d6 60 b7 77 08 da c9 6c 99 34 55 06 c9 24 7e 23 37 b1 d0 6b c3 61 36 66 1e 2e c6 f8 4b 23 10 d0 d5 0d 5c c7 e9 09 3a 37 89 9d e1 4b d0 ac 50 7e e4 67 58 33 d2 c1 42 a1 de 35 f6 b3 c4 ff f8 09 d9 0b 48 d7 53 dd 45 35 6e e9 b2 17 2f 2d 9a 18 48 46 48 c9 27 bd 14 d3 f0 97 94 68 31 8a 2b 06 88 be cc 92 f6 f5 ab 3e d7 05 41 55 4e ec 72 f9 1d e8 cb 7f 7f e4 25 3d 0e 8b 29 ec 68 ae 34 ff 8a 90 22 42 ca d0 ef af 72 9e 9a f7 e0 d7 eb 16 93 32 11 59 7a eb a9 44 a7 7e 77 30 be 38 6f 49 09 42 dc aa 02 ca cf 6c c2 90 87 f8 d3 0f 71 69 f7 3a 70 98 2c 73 a6 54 a7 58 e6 14 15 81 29 ee 11 ab b1 4f 1a 7f 4d 6d b4 4b e7 54 9a ad 71 d4 23 a9 b6 5e 9d a1 72 fe 2b 0b 92 d6 85 ad e9 a2 76 7b 1c 12 b7 d8 15 49 fc f9 6b 0d d2 16 04 a8 49 b5 f1 72 7f e6 a4 05 43

Handshake Protocol: Client Key Exchange:

Pubkey length: 32 bytes

Pubkey: eb ac 51 71 d6 46 e9 3a ad 50 85 87 d2 d0 df 99 9b 89 75 b4 c3 c8 9f 04 6c 13 a2 a0 1e b6 9f 08

Paquet 48 RFCOMM :

```
0000  02 00 21 93 00 8f 00 40  00 21 ef 14 01 97 a9 7c   ··!····@ ·!·····|
0010  39 1c b2 f0 c0 b7 ba 0d  5f 35 6d 4d 49 f6 ad b6   9·······  _5mMI···
0020  91 34 25 6b 4d 48 2b 70  c0 1d 4b b9 b3 b1 e5 11   ·4%kMH+p  ··K·····
0030  ae ec ac 5d 9b 49 98 01  ff 0d f2 ce cc 1d 86 5c   ···]·I··  ·······\
0040  79 55 c9 3b c1 cd 55 d4  15 5f b6 13 96 2f cd 7e   yU·;··U·  ·_···/·~
0050  c1 77 7c 5c d4 20 85 4b  01 c9 fc 3e 4b 5b 2e f1   ·w|\· ·K  ···>K[.·
0060  49 37 d1 93 14 03 03 00  01 01 16 03 03 00 28 cf   I7····· ·  ·····(·
0070  88 89 c2 04 94 01 41 96  1c 7b ce 8d 7f 6d 4f 45   ·····A·  ·{···mOE
0080  49 17 c3 b4 52 b9 9c e2  62 a4 65 bf ba 3b 2c 31   I···R···  b·e··;,1
0090  0e b6 39 b5 7c 65 c4 80                           ··9·|e··
```

En-tête TLS Record Layer

14 03 03 00 01

-14 : Content Type *(Change Cipher Spec)*
-03 03 : Version (TLS 1.2)
-00 01 : Longueur (1 octet)

Contenu du Change Cipher Spec
01

En-tête TLS Record Layer

16 03 03 00 28
-16 : Content Type (Handshake)
-03 03 : Version (TLS 1.2)
-00 28 : Longueur (40 octets)

Contenu du Handshake (*Finished*)

cf 88 89 c2 04 94 01 41 96 1c 7b ce 8d 7f 6d 4f
45 49 17 c3 b4 52 b9 9c e2 62 a4 65 bf ba 3b 2c
31 0e b6 39 b5 7c 65 c4 80

Ce paquet fait partie du handshake TLS et contient deux messages importants :

-Change Cipher Spec : Le client indique qu'il va commencer à utiliser les clés nouvellement négociées pour chiffrer les messages suivants.
-Finished : Le client envoie un hash de tous les messages échangés durant le handshake afin de vérifier l'intégrité et la réussite du processus de handshake.

Section 5 : Défis et Solutions

Les défis rencontrés lors de l'utilisation de SSLEngine pour établir le handshake TLS et du développement de l'application sur Android Studio ont été substantiels, mais grâce à une approche méthodique et à l'utilisation de ressources appropriées, nous avons pu surmonter ces obstacles. Néanmoins, ces expériences ont enrichi notre compréhension des technologies de sécurité et de développement mobile, et nous ont permis de créer une application robuste et sécurisée répondant aux exigences de notre projet.

Utilisation de SSLEngine pour Établir le Handshake TLS

L'un des principaux défis rencontrés lors du développement de notre projet a été l'utilisation de SSLEngine pour établir le handshake TLS entre le client et le serveur. SSLEngine est une API complexe fournie par Java pour les communications sécurisées via TLS/SSL, et sa maîtrise nécessite une compréhension approfondie des concepts de cryptographie et de protocoles de sécurité.

Défis Rencontrés

- Complexité de l'API : SSLEngine est conçu pour une flexibilité maximale, mais cela se traduit par une complexité accrue. La gestion des états et des transitions entre ces états au cours du handshake TLS nécessite une gestion rigoureuse des buffers de lecture et d'écriture.

- Debugging et Diagnostic : Les erreurs et échecs de handshake ne fournissent pas toujours des messages d'erreur explicites. Déboguer ces problèmes a nécessité une analyse minutieuse des logs et une compréhension approfondie des échanges de messages TLS.

- Compatibilité : Assurer la compatibilité entre le client écrit en Java utilisant SSLEngine et le serveur écrit en C a posé des défis, notamment en ce qui concerne la gestion des certificats et la négociation des suites cryptographiques.

Solutions Apportées

- Documentation et Ressources : L'équipe a consacré du temps à l'étude approfondie de la documentation officielle de SSLEngine, ainsi qu'à des ressources complémentaires telles que des tutoriels et des forums spécialisés.

- Logs Détaillés : Pour faciliter le diagnostic, nous avons intégré des mécanismes de logging détaillés pour suivre chaque étape du handshake TLS et identifier précisément où et pourquoi les échecs se produisaient.

- Tests Incrémentaux : Nous avons adopté une approche de développement incrémentale, testant chaque petite partie de l'implémentation TLS avant de les intégrer ensemble. Cela nous a permis d'isoler et de résoudre les problèmes plus efficacement.

Développement de l'Application sur Android Studio

Le développement de l'application Android utilisant Android Studio pour communiquer avec le serveur C via Bluetooth avec TLS a également présenté des défis spécifiques.

Défis Rencontrés

- Intégration Bluetooth et TLS : Combiner les communications Bluetooth avec les protocoles TLS a été particulièrement difficile. Chaque technologie a ses propres exigences et complexités, et les faire fonctionner ensemble de manière transparente a nécessité des ajustements minutieux.

- Performance et Efficacité : Optimiser l'application pour qu'elle fonctionne de manière fluide tout en gérant les communications sécurisées et sans fil a nécessité une attention particulière à la gestion des ressources et à l'optimisation du code.

Solutions Apportées

- Utilisation des Bibliothèques Appropriées : Nous avons utilisé des bibliothèques spécifiques, telles que Bouncy Castle pour TLS sur Android et des bibliothèques Bluetooth robustes, pour simplifier l'implémentation et garantir la sécurité et la fiabilité.

- Tests sur Dispositifs Réels : Pour surmonter les limitations des émulateurs Android, nous avons effectué des tests exhaustifs sur des dispositifs physiques pour s'assurer que la communication Bluetooth avec TLS fonctionnait correctement dans des conditions réelles.

- Optimisation du Code : Nous avons optimisé le code pour minimiser l'impact sur les performances de l'application.

Conclusion

Notre projet a démontré la faisabilité et les avantages de l'intégration de TLS sur Bluetooth pour les communications sécurisées dans le domaine automobile. En développant une application Android capable de se connecter de manière fiable et sécurisée à un serveur C via Bluetooth, nous avons posé les bases d'une infrastructure de communication robuste et sécurisée pour les véhicules connectés. Cette infrastructure est essentielle pour la transmission de messages de sécurité routière, tels que les CAM (Cooperative Awareness Message) et DENM (Decentralized Environmental Notification Message), garantissant ainsi la confidentialité et l'intégrité des données échangées et démontrant les vastes potentielles utilisations de cette preuve de concept (PoC) dans l'industrie automobil.

En conclusion, notre projet ouvre la voie à de nombreuses applications innovantes dans l'industrie automobile, où la sécurité des communications joue un rôle crucial. En adoptant TLS sur Bluetooth, les véhicules connectés peuvent désormais échanger des informations critiques en toute sécurité, renforçant la confiance des utilisateurs et soutenant le développement de nouvelles technologies de mobilité urbaine. Les travaux futurs pourraient inclure l'extension de cette technologie à d'autres types de communications sans fil et l'intégration avec des systèmes de gestion de flotte pour une optimisation globale des transports.

Références

Github Link to The Full Project:

https://github.com/CharbelFarah057/Bluetooth_TLS.git

Other references:

1. Bluetooth SIG, Inc. "Bluetooth Technology Website." Bluetooth.com. https://www.bluetooth.com/.

2. Dierks, Tim, and Eric Rescorla. "The Transport Layer Security (TLS) Protocol Version 1.2." IETF Datatracker. Internet Engineering Task Force (IETF), August 2008. https://datatracker.ietf.org/doc/html/rfc5246.

3. Rescorla, Eric. "The Transport Layer Security (TLS) Protocol Version 1.3." IETF Datatracker. Internet Engineering Task Force (IETF), August 2018. https://datatracker.ietf.org/doc/html/rfc8446.

4. "SSL." OpenSSL. OpenSSL Project. https://www.openssl.org/docs/man1.1.1/man3/SSL.html.

5. "ENGINE." OpenSSL. OpenSSL Project. https://www.openssl.org/docs/man1.1.1/man3/ENGINE.html.

6. wolfSSL. "Documentation." wolfSSL. https://www.wolfssl.com/documentation/.

7. Huang, Albert. "Introduction to Bluetooth Programming with BlueZ." CSAIL, MIT. https://people.csail.mit.edu/albert/bluez-intro/index.html.

8. Cloudflare. “What Is TLS? Transport Layer Security Explained.” Cloudflare. https://www.cloudflare.com/fr-fr/learning/ssl/transport-layer-security-tls/.

9. SSL.com. “What Is SSL/TLS? An In-Depth Guide.” SSL.com. https://www.ssl.com/fr/article/quel-est-ssl-tls-an-in-depth-guide/.

10. Intel. “How Does Bluetooth Work?” Intel. https://www.intel.fr/content/www/fr/fr/products/docs/wireless/how-does-bluetooth-work.html.