MotorIST

Group: T23	Name	IST ID
Student 1:	Miguel dos Santos Raposo	Ist1112167
Student 2:	Vasco Rodrigues das Neves de Magalhães e Silva	Ist199132
Student 3:	Diogo Alexandre Martins Rodrigues	Ist1112192

MEIC-T 2024-2025 – 2nd Period

Final report written for the **Network and Computer Security course**

Master in Computer Science and Engineering

January 2025

Index

1. Introduction

The rapid advancement of automotive technology has enabled the integration of modern management systems in electric vehicles, significantly enhancing user convenience and vehicle functionality. IST has recently introduced a new line of electric cars equipped with cutting-edge features, allowing users to remotely configure and monitor their vehicles through a dedicated application installed on their mobile devices or computers, named MotorIST. These functionalities include remote access to vehicle locks, air conditioning (AC) settings, seat adjustments, and battery level monitoring. Furthermore, the cars support firmware updates from the manufacturer to ensure optimal performance and security over time.

To facilitate seamless communication between users, the manufacturer, and the vehicle, a standardized JSON-based data exchange format is employed. While this innovation offers unparalleled user convenience, it also brings forth critical security and privacy challenges, particularly concerning the sensitive nature of user and vehicle data. In compliance with the General Data Protection Regulation (GDPR), the application needs to ensure the confidentiality, integrity, and authenticity of all communications and documents exchanged in the process.

This report explores the implementation of these security measures to address specific requirements:

- **Confidentiality**: Protecting car configurations from unauthorized access.
- **Integrity**: Ensuring only the rightful owner can modify configurations and that firmware updates originate solely from the manufacturer.
- **Authentication**: Preventing repudiation of firmware updates by the manufacturer.

Additionally, this project focuses on **Security Challenge A**, which addresses the complexities of multi-user scenarios. This includes ensuring data privacy between users, enabling user-specific authorization, and providing accountability for configuration actions. The report details the strategies employed to secure communication and protect user data while maintaining an intuitive and efficient user experience, setting a standard for secure and user-friendly vehicle management systems.

2. Developed Solution

2.1. Secure Documents

The MotorIST aims to ensure the confidentiality and integrity of the configurations sent by the users of each car, preventing any other individual from accessing or altering them. To this end, a library responsible for the protection and verification of these documents, called Secure Docs, was developed. This library is capable of protecting documents through cryptographic techniques, making it computationally impractical for an attacker to reverse them.

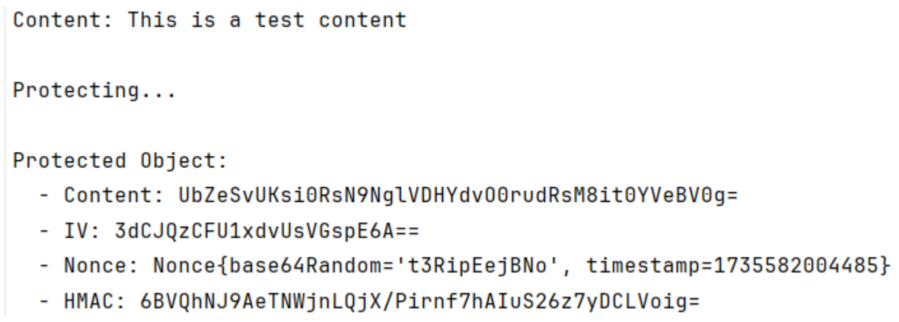
For this purpose, the Java Cryptography Architecture (JCA) was selected as the cryptographic library due to its comprehensive support for a wide range of cryptographic algorithms, proven security framework, and efficient integration capabilities within Java environments.

To ensure document protection, an IV is generated using a Secure Random, and the document is encrypted with AES/CBC/PKCS5Padding using a 256bit secret key. Attending to the need of guaranteeing freshness, a Nonce is optionally created by combining a randomly generated base64 value with a timestamp for its validity period. As the final step, an HMAC is generated over the Nonce, IV, and encrypted content, ensuring their integrity. This process results in the creation of a **ProtectedObject**, which includes the encrypted content, the IV for decryption, the Nonce, and the HMAC.

To unprotect a document, a **ProtectedObject** is provided, from which the encrypted content and IV are extracted. These are then used to decrypt the document using the same symmetric key. Once decryption is complete, the object is updated to store the decrypted content.

The integrity and validity checks involve verifying both the HMAC and the Nonce. For the HMAC, a new value is generated using the content extracted from the **ProtectedObject** and compared with the HMAC included in the object. For the Nonce, the timestamp is validated to ensure it is within the allowed period, and the base64-encoded random value is checked to confirm it hasn't already been used.

Additionally, the method provides an option to bypass Nonce verification if a Nonce was not used during the protection phase. If both the HMAC and Nonce checks pass, the document is confirmed as authentic and valid.

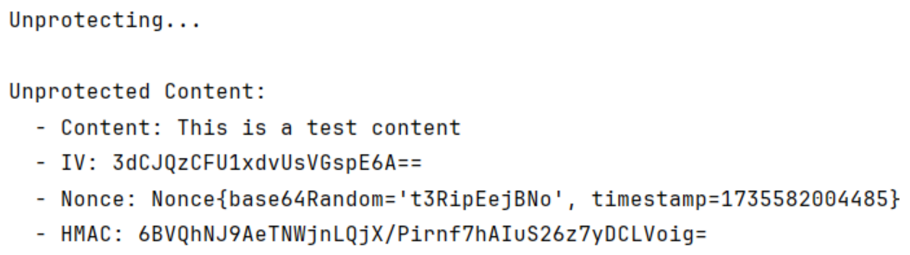


```
Content: This is a test content

Protecting...

Protected Object:
  - Content: UbZeSvUKsi0RsN9NglVDHYdvO0rudRsM8it0YVeBV0g=
  - IV: 3dCJQzCFU1xdvUsVGspE6A==
  - Nonce: Nonce{base64Random='t3RipEejBNo', timestamp=1735582004485}
  - HMAC: 6BVQhNJ9AeTNWjnLQjX/Pirnf7hAIuS26z7yDCLVoig=
```

Figure 1 - Example of protecting a document



```
Unprotecting...

Unprotected Content:
  - Content: This is a test content
  - IV: 3dCJQzCFU1xdvUsVGspE6A==
  - Nonce: Nonce{base64Random='t3RipEejBNo', timestamp=1735582004485}
  - HMAC: 6BVQhNJ9AeTNWjnLQjX/Pirnf7hAIuS26z7yDCLVoig=
```

Figure 2 - Example of unprotecting a document

2.2. System Infrastructure

The architecture of the system has been carefully defined to meet the project's requirements. The following diagram illustrates the key components and their interactions, providing an overview of the system's structure and design.

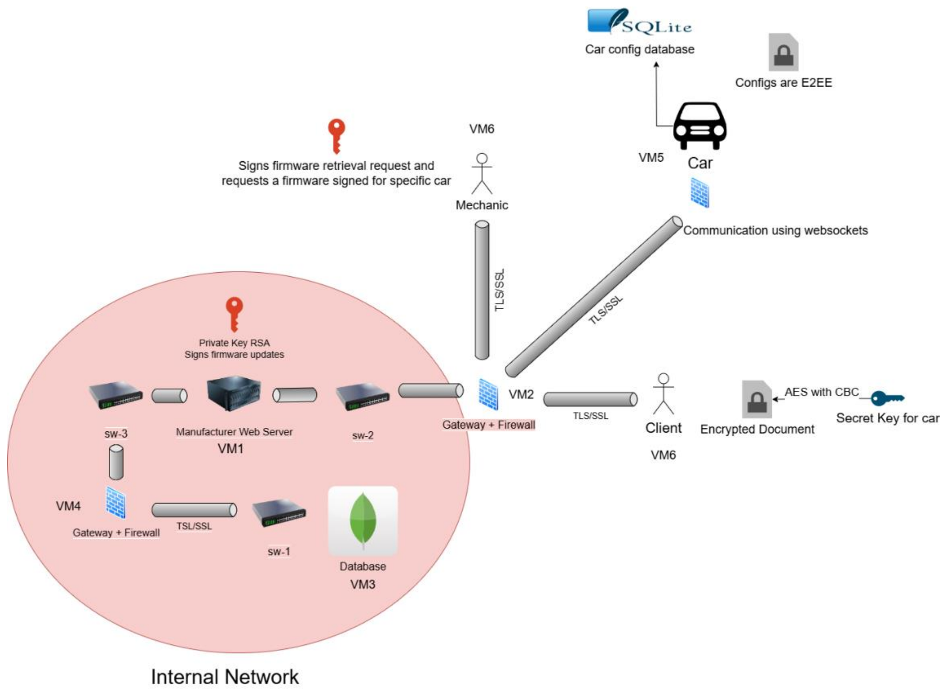


Figure 3 - System Architecture

The system consists of an internal network hosting the car manufacturer's web server and a MongoDB database for storing user configurations, firmware updates, and registered application users. Additionally, it includes a gateway connecting to the external network and another gateway dedicated to managing and restricting traffic directed to the database.

Outside this network are the cars and users, whether they are customers or mechanics, who will interact with the web server to enable communication between them.

To ensure secure communication between these machines, TLS has been implemented. The server, the car, and the MongoDB database each hold certificates signed by a Certificate Authority (CA), stored in a keystore known as a Truststore, which enables seamless authentication during the process. Communication between regular users and the server employs standard TLS, while communication between the web server and the database, as well as between the web server and the car, utilizes mTLS with mutual certificate authentication. The project employs a flat trust model, relying on a single CA to sign all certificates used in the application, with its public key universally recognized by all entities involved.

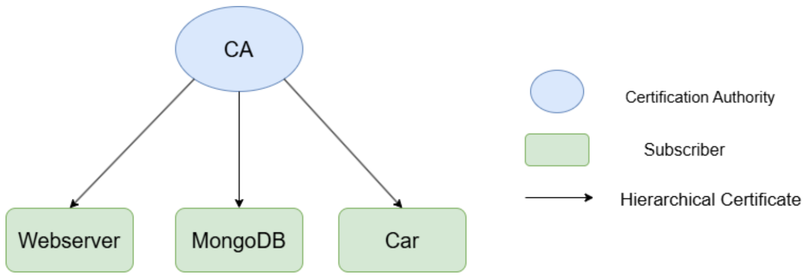


Figure 4 - Flat trust model

Maintaining secure communication between the car and the server is not sufficient to achieve the desired interaction between them. Additionally, the goal is to avoid having a car behave like a server—listening on a port and waiting for requests. To address this, communication is implemented using WebSockets. The car establishes a direct connection with the server, and whenever data needs to be sent, the server relays a message to the car, through the websocket tunnel, in a format it can interpret and process. The server then waits for the car's response. It's important to note the car implements a firewall that blocks all incoming new connections and only the car itself can try and open new connections.

To safeguard the machines within the internal network, a firewall was deployed on the gateway that interfaces with external systems, effectively controlling the traffic allowed into the network. Its primary function is to route all HTTPS traffic to the web server while blocking any other traffic. Since the web server handles all HTTPS requests, it resides within the network's DMZ. To further enhance security, a second gateway was introduced to filter traffic from the server to the database, protecting the database from potential threats. This setup ensures that, even if the DMZ is compromised, the database remains shielded by an additional layer of protection. The second gateway strictly allows traffic destined for the database and originating from the server, preventing any other machine from accessing the database.

This implementation could be significantly improved by introducing an additional machine to serve as the application's frontend, which would be situated within the network's DMZ. The backend, along with the database, would be placed behind the second gateway and would handle all of the application's core logic—such as sending messages to the car and managing interactions with the MongoDB database. This improvement would strengthen the overall infrastructure, as even if the DMZ were compromised, an attacker would be limited in their ability to exploit the application's functionality. They would only have control over the frontend, which contains no critical logic and merely acts as an interface to communicate with the backend, relaying user requests. This separation ensures that the backend and database remain secure, maintaining the integrity of the system.

This infrastructure was deployed using six virtual machines, as depicted in Figure 3. Each machine within the internal network was assigned a unique IP address.

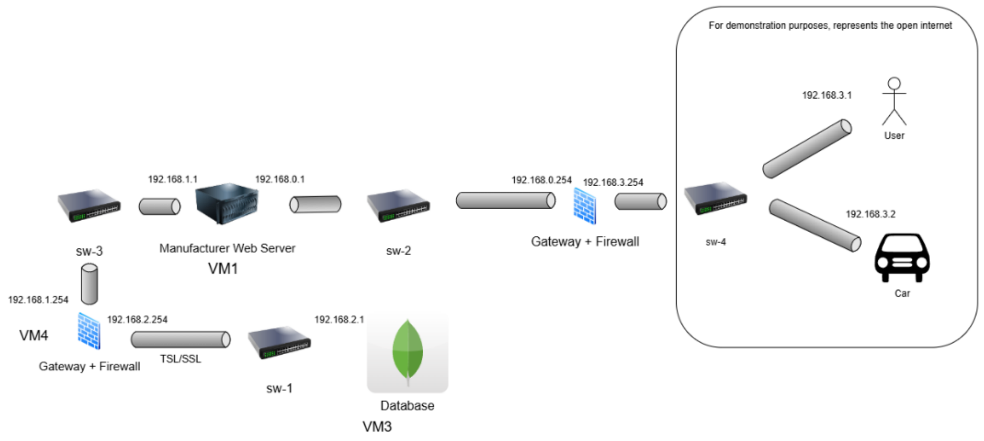


Figure 5 - IP assigning for the internal network

As highlighted earlier, our application leverages two distinct types of databases:

- Manufacturer – The application utilizes a non-relational MongoDB database to store all users, configurations, and firmware updates. To uphold data security and protect sensitive information, all configurations are securely stored in an encrypted format, where only the car and user know the secret key.
- Car – A relational SQLite database is used to store the car's general information and details of the users paired with it. To maintain user privacy and security, each paired user's current configuration is securely stored in an encrypted format using the user's unique secret key. This ensures that configurations are isolated and cannot be accessed or viewed by other users, preserving data confidentiality within the shared system.

All user configurations are encrypted at rest in the databases.

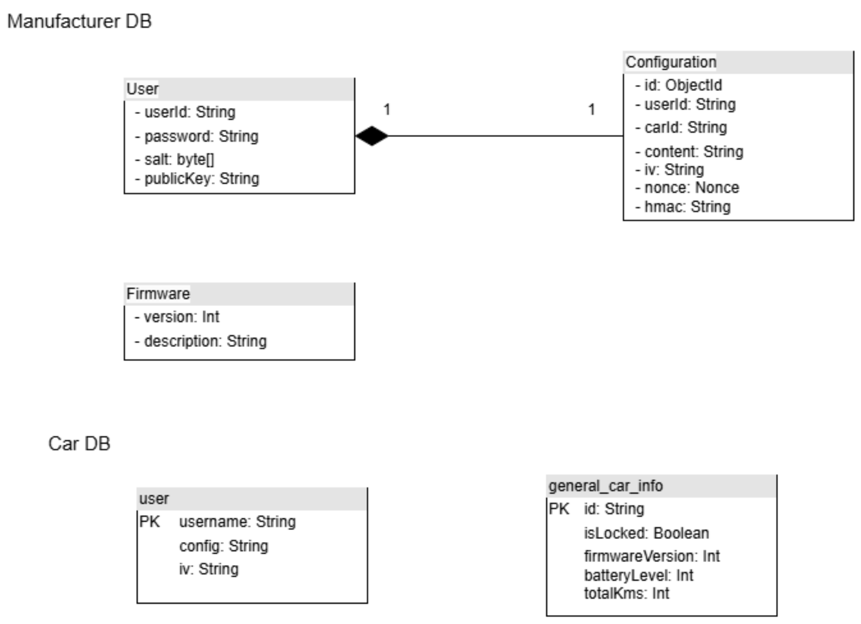


Figure 6 - Databases representation

2.3. Application Execution Flow

MotorIST users can pair with their car to establish a symmetric key, enabling secure access to view and update their personal configurations. Users can also access general car information at any time, which is shared among all paired users, while ensuring that individual configurations remain private and accessible only to their respective owners. Furthermore, users have the option to unpair from the car at any point, which permanently removes their configuration from the system, ensuring their data is no longer associated with the vehicle.

To pair a user with a car, the user must be physically present during the process. When the pairing option is selected on the car's console, the car generates a unique code that the user must enter into their device, ensuring they have physical access to the vehicle. Once the code is entered, it is sent to the server, which has already received the car's code. The server compares the hashes of both codes to verify if they match. If the validation is successful, the server sends a confirmation message to the car and stores the default configuration associated with the user in its database.

Upon receiving the success message, the car displays the generated symmetric key, which the user must input into their application. This establishes a shared secret key between the car and the user, enabling secure encryption and decryption of the configurations. The car maintains a unique secret key for each paired user, ensuring complete isolation and security of individual configurations, preventing unauthorized access between users. This process ensures both security and personalized access management.

A potential improvement to this process would be the use of a QR Code displayed on the car's screen instead of a plain text code. This adjustment would streamline the user experience, making the pairing process more intuitive, efficient, and user-friendly.

Once the secret key is established, the user can perform more advanced operations. For instance, to update a configuration, the user encrypts the configuration document and transmits it to the server. The server, in turn, forwards the encrypted object, referred to as the ProtectedObject, to the car. The car decrypts the object and validates its integrity, ensuring the configuration remains unaltered during transmission.

If the verification is successful, the car updates its local database with the new configuration and sends a success message back to the server. The server then stores the ProtectedObject received from the user in its own database and confirms the update by responding to the client with a success message.

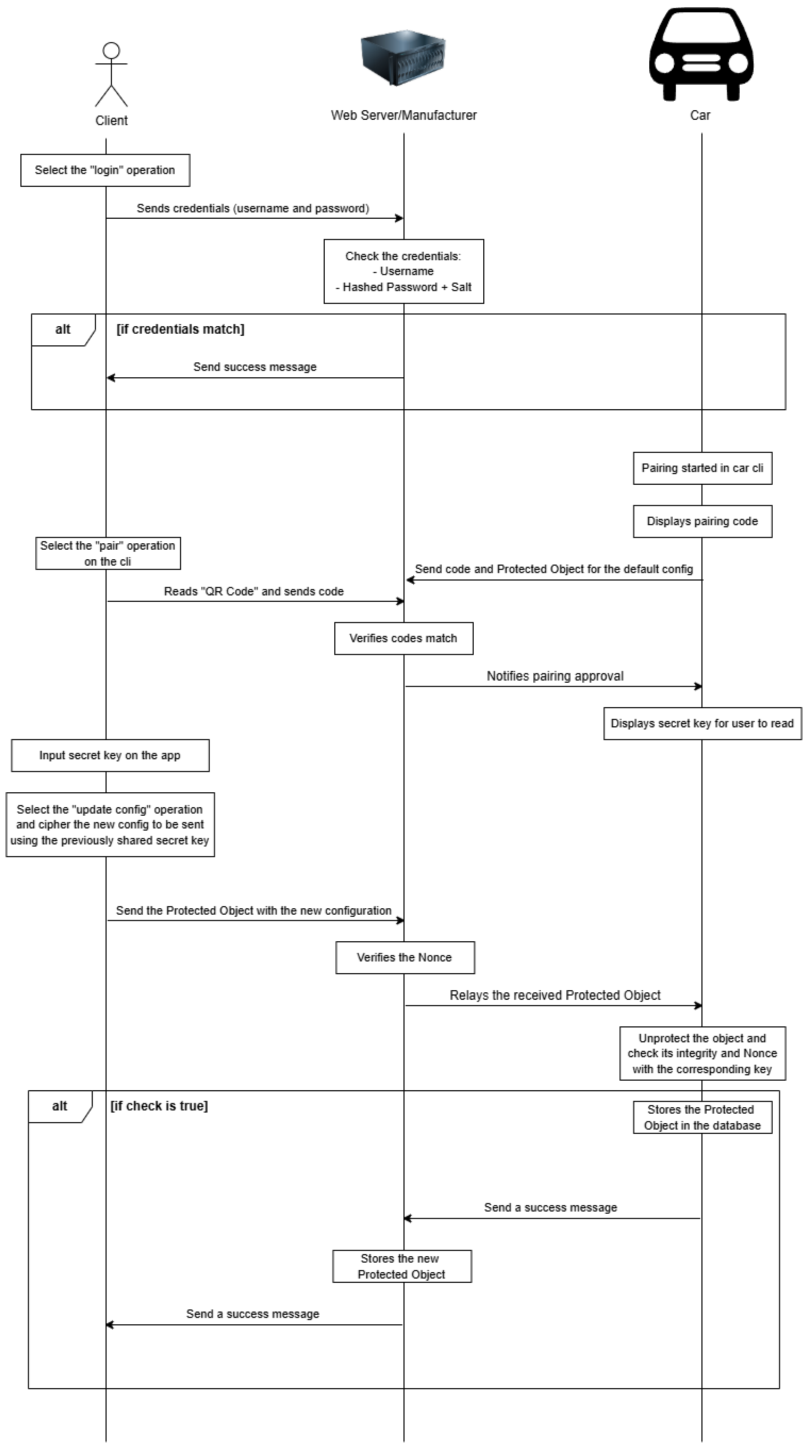

Client
Web Server/Manufacturer
Car
Select the "login" operation
Sends credentials (username and password)
Check the credentials:
- Username
- Hashed Password + Salt
alt
[if credentials match]
Send success message
Pairing started in car cli
Displays pairing code
Select the "pair" operation on the cli
Send code and Protected Object for the default config
Reads "QR Code" and sends code
Verifies codes match
Notifies pairing approval
Displays secret key for user to read
Input secret key on the app
Select the "update config" operation and cipher the new config to be sent using the previously shared secret key
Send the Protected Object with the new configuration
Verifies the Nonce
Relays the received Protected Object
Unprotect the object and check its integrity and Nonce with the corresponding key
alt
[if check is true]
Stores the Protected Object in the database
Send a success message
Stores the new Protected Object
Send a success message

Figure 7 - Pairing and update config flow

To unpair, the process follows a similar approach, however the encrypted content contains a confirmation message explicitly requested from the user. This step ensures that only the user's configurations are deleted, preventing the removal of configurations belonging to others. Instead of updating configurations in the databases, this process securely deletes them, completing the unpairing operation.

To retrieve a configuration, the process is streamlined by interacting solely with the server, as it maintains the current configurations for each application user, acting as a cache. This eliminates the need to communicate directly with the car. The server returns the stored ProtectedObject, which the user decrypts using their secret key, ensuring the data's integrity before proceeding.

To access the car's general information, the execution flow must involve direct communication with the car. This is because the data is dynamically updated based on the car's usage, and the server does not continuously refresh or store these values. When requested, the server forwards a query to the car, which responds with the general information in an encrypted format. This ensures that only the intended user, equipped with the corresponding secret key, can decrypt and interpret the data. Once the user receives the encrypted response, they decrypt it, verify its integrity, and access the updated information securely.

Registered mechanics are recognized by the server, with their public keys securely stored in the database. Within the application, their primary role is to download and install firmware updates on the cars. To initiate the update process, the mechanic must sign the car's unique identifier and submit it to the server. The server then verifies the signature to ensure that the request is coming from an authorized, registered mechanic. Once the mechanic's identity is confirmed, the server returns the manufacturer-signed firmware, which is packaged in a JSON file for the mechanic to install on the car. This process ensures security, authenticity and non-repudiation of firmware updates.

Upon receiving the firmware, the mechanic installs it on the car by transferring the JSON file via USB. On the car's console, the mechanic selects the option to install the firmware and specifies the file location. The car then verifies the manufacturer's signature, checks the version of the firmware and updates its database with the new general information to reflect the latest version. The update is applied only if the signature is valid and the new version surpasses the current one, ensuring that only authorized and necessary upgrades are implemented, effectively protecting the car's system from unauthorized firmware updates.

To ensure secure user authentication, a straightforward login mechanism has been implemented. Authentication occurs exclusively with the server, as users, by holding a secret key tied to their car, inherently secure access to their configurations, ensuring they are only readable by the user. Currently, credentials are passed and verified with each request to the server's endpoints. However, a potential future improvement would be to implement session management using cookies, allowing for seamless user experience by maintaining authenticated sessions and eliminating the need for repeated credential verification throughout the user's interaction with the system.

2.4. Security Challenge

A key security challenge addressed in this project was enabling multiple users to share the same car while ensuring strict privacy and isolation of their individual configurations. Each user is assigned a unique secret key during the pairing process, which safeguards their data and prevents unauthorized decryption or access to configurations belonging to others.

To support shared access, the car also maintains general information, such as battery level and total kilometers driven, which is relevant to all users. This data is stored in plain text within the car's database, being encrypted with the requesting user's key upon request, keeping the information private between the car and the users. This ensures universal accessibility without compromising the privacy of user-specific settings.

Furthermore, an append-only log file was introduced to record every action performed by users on the car. This immutable audit trail enhances accountability, providing a clear and tamper-proof history of who performed what actions and when.

2.5 Threat Model

In the context of automotive systems, a comprehensive threat model is critical for assessing vulnerabilities and potential attack vectors that could compromise the security and privacy of users and vehicle systems. The primary threat actors range from external attackers to insiders with legitimate access to the system, each possessing unique motives and methods for exploiting weaknesses. These threats can exploit various attack vectors such as network-based vulnerabilities, unauthorized access, data tampering, and physical compromises.

External attackers, including hackers and network-based adversaries, may target unsecured communications to intercept or manipulate data flowing between the vehicle and user devices. Malicious users, such as those sharing vehicles, could attempt to exploit weak authentication mechanisms to access sensitive configurations. Insider threats, such as mechanics or employees, present additional risks by abusing authorized access to alter critical vehicle functions or install malicious software. These actors may also leverage insider knowledge to bypass safeguards or exploit system weaknesses unnoticed. Attackers may exploit common vulnerabilities like weak passwords or unsecured firmware updates, allowing unauthorized parties to compromise the system.

By identifying these potential threats and attack vectors, the threat model enables targeted strategies for risk mitigation, helping secure vehicle systems against a wide array of malicious activities.

Table 1 - Threat model

Threat	Impact	Mitigation
Unencrypted communication	Data theft	Enforcing TLS for all communications with certificates signed by a trusted CA.
Weak user credentials	Unauthorized access	Require strong passwords and implement rate-limiting for failed login attempts.
Replay attacks	Unauthorized actions	Include timestamps and nonces in all critical requests.
Firmware spoofing	Car compromise	Verify firmware authenticity using digital signatures from the manufacturer.
Data tampering	Misconfiguration or misuse	Use HMAC for integrity checks and log all changes in an immutable audit trail.
Insider abuse	Privacy breach	Log and monitor all actions performed by mechanics and employees.
Database compromise	Data exposure or tampering	Encrypt sensitive data at rest and implement access controls for the database.
Unauthorized configuration access	Privacy violation	Use unique symmetric keys for each user and isolate user configurations.

3. Conclusion

This project highlights the critical importance of secure communication and robust data protection in modern automotive systems, particularly in the context of electric vehicles equipped with advanced management functionalities. By addressing the unique challenges posed by sensitive user and vehicle data, the work emphasizes the necessity of balancing security requirements with usability in innovative transportation solutions.

Key achievements include the implementation of confidentiality measures to protect user data, the establishment of strict authorization protocols for multi-user interactions, and the creation of an auditable system that ensures accountability for configuration actions. Furthermore, the project successfully met all specified requirements: **SR1** (Confidentiality), **SR2** (Integrity for user configurations), **SR3** (Integrity for firmware updates), **SR4** (Authentication), as well as the multi-user scenario challenges **SRA1** (Data privacy), **SRA2** (Authorization), and **SRA3** (Authenticity). These measures collectively contribute to a secure and efficient framework for vehicle management through the MotorIST application.

Throughout the project, challenges related to managing multi-user scenarios and ensuring GDPR compliance provided valuable insights into the design of secure systems. The experience underscored the importance of cryptographic safeguards and clear communication protocols in achieving both security and functionality.