Crypto-analysis Resistant Digital Key FOB

Jeremy Hein
Adriana Matos
Courtney Eaton

CONCEPT OF OPERATIONS

REVISION – 3
April 30, 2024

CONCEPT OF OPERATIONS
FOR
Crypto-analysis Resistant Digital Key FOB

TEAM <2>

APPROVED BY:

Adriana Matos_____
Project Leader Date

_____
Prof. Kalafatis Date

_____
T/A Date

Change Record

Rev.	Date	Originator	Approvals	Description
-	9/4/2023	Jeremy Hein		Draft Release
1	9/14/2023	Courtney Eaton		Submitted Revision
1a	9/19/2023	Courtney Eaton		Updated Draft for Midterm Report
2	12/1/2023	Courtney Eaton		403 Final Report
3	4/30/2024	Courtney Eaton		Capstone Completed Project Report

Table of Contents

List of Figures

1. Executive Summary

Cybersecurity is becoming a larger issue as nearly all modern, everyday devices contain internet communication capabilities. With increasing dependence on dynamically re-designable software (i.e. software-defined vehicles) and increasingly connected vehicles with autonomous driving capabilities built-in, cybersecurity threats are no longer bound to information but extend out to physical properties and the safety of human beings. Current de facto ways of protecting the SDI (i.e. firmware updates) are Public Key Infrastructure (PKI) for authentication and integrity checks and Key Management System (KMS) for confidentiality. Unfortunately, both of these security services are very expensive to deploy and maintain, as well as being the lowest-hanging fruit/attack surface for malicious hackers. In response to this, Sandia National Laboratories has developed a new way to securely communicate and update firmware within a vehicle by replacing PKI and KMS with dynamic key generation and Zero Trust Architecture. The purpose of our project is to show the capabilities of this technology through the creation of a digital key FOB application that will utilize this new technology in order for the user to securely lock, unlock, and start their car. The applications of this new technology are abundant and it provides the opportunity for a new level of security within the vehicles of the future.

2. Introduction

The purpose of this document is to provide a conceptual framework for building a secure digital key FOB capable of resisting all forms of crypto-analysis attacks. Through a digital key FOB application, this project implements the Zero Trust Application for Vehicle technology (ZAV) which is made to be resistant towards Crypto-analysis attacks against one's vehicle. This project will have the potential to provide a secure communication channel, resistant to cyber attacks (i.e. relay, man-in-the-middle, crypto-analysis attacks, etc.) between the digital key FOB and the vehicle. This ZAV technology can easily be modified to be integrated as a secure way to update firmware in a vehicle Over-the-Air (OTA).

2.1 Background

Many new automobiles now have a keyless entry instead of manually unlocking a car by pressing a button on a car key or turning the key. This keyless entry works by having a proximity key FOB with a short-range radio transmitter/radio frequency identification (RFID) chip and antenna that sends out a distinct rotating coded signal to a receiver unit in the car. When a driver, for example, presses a button on the door handle, the system will transmit a signal to the authorized key FOB to see if it is within range. If the receiver is within range, the key FOB will then respond with its own signal to open the door.[3] This proximity key FOB will only unlock the car if it is close enough on the outside of the car and a button on the door handle is pressed to send out the signal. This method, although relatively secure, is not impervious to relay attacks.

Relay attacks are a form of automotive attack that requires two people and a relay device. One person stands by the automobile with the first relay device and triggers the car to send a signal out to confirm the key FOB is within range. Another person will stand near wherever the key FOB is with the second device. The second person does not need to even enter the building with the key FOB, this can be done by standing outside the building where the key FOB is stored (e.g. a house or place of work). The second device will then transmit the relayed signal to the first device. Once the vehicle receives the signal, it will respond in a normal manner as if the key FOB is near and allow entry into the vehicle. To turn on the automobile, a second relay is needed using essentially the same process. [2]

This relay attack sparked many concerns within the automotive industry, which led to the creation of the Zero Trust Application (ZTA). Sandia National Laboratories patented the Zero Trust Application for Vehicle technology, or ZAV, which is a solution to many car hacking problems. The ZAV technology is a new secure communication that can provide all security attributes namely: confidentiality, integrity, authenticity, nonrepudiation, and availability without the use or reliance on PKI and KMS.

Using a similar algorithm, the digital key FOB will show how the ZAV technology can create a secure connection to communicate between the device application and the vehicle. Once proven to be secure through the digital key FOB, this technology can be adapted to update firmware and protect the vehicle from future attacks.

2.2 Overview

This project will consist of an Android application that will connect to a physical locking mechanism through Bluetooth using a secured communication channel. The Android application will have the capability to securely register the user to a specific car through the use of a unique signature which will be inputted with the user's finger or stylus. Once registered, the user will be able to lock/unlock and turn on/off their car through the application and the microprocessor will use the unique digital behavior of the key FOB (i.e. mobile device, smartphone, tablet) to ensure that the device giving the command to the car is a physically designated key FOB.

The Locking Mechanism will consist of a student-built microprocessor with added CAN bus capabilities to mimic an automobile's internal network system. Using the signal sent from the application and the cryptography software, a secure communication line will be created between the application and the Locking Mechanism. The CAN bus network will ensure that the signal is coming from the owner's device and turn on the necessary Electronic Control Units. Theoretically, if the signal is not coming from the owner's device the locking mechanism will not unlock demonstrating that the message is not replayable, or cryptographically decipherable.

2.3 Referenced Documents and Standards

Referenced Documents:

[1] Choi, Peter. *Zero Trust Application for Vehicle*, Sandia National Laboratories, 2022.

[2] "Keyless Car Theft: What Is a Relay Attack, How Can You Prevent It, and Will Your Car Insurance Cover It?" *Leasing.com*, leasing.com/guides/relay-car-theft-what-is-it-and-how-can-you-avoid-it/.

[3] Shah, Vivek. "Proximity Keys and Other Unlocking Innovations." *CarExpert*, 24 Sept. 2021, www.carexpert.com.au/car-news/proximity-keys-and-other-unlocking-innovations. Accessed 9 Sept. 2023.

Standards:

- IEEE 802.15.1-2002
- IEEE 1363-2000
- IEEE 1363.3-2013
- IEEE 1625-2004
- NIST Cryptographic Standards and Guidelines

3. Operating Concept

3.1 Scope

The scope of this project is to create a proof of concept build that will showcase the capabilities of Zero Trust Authentication in Vehicle safety and security applications through a digital key FOB demonstration. This will be accomplished with an Android application, in-depth backend cryptography for any communication sent through Bluetooth, and a microprocessor to act as the car and interpret the signals. The microprocessor will communicate with a locking and ignition representation to indicate the success of the digital key FOB.

3.2 Operational Description and Constraints

The Crypto-analysis Resistant Digital Key FOB is designed for widespread application across the transportation industry. To use the digital key FOB, the user will open the app and register the device as a digital key FOB with the vehicle ECU. Registration can be achieved by drawing or signing their name on a digital keypad. The device will then pair with the car through a combination of the Diffie-Hellman Key Exchange protocol and the Advanced Encryption Standard (AES) algorithm. Once registration is complete, the user will be able to send lock or unlock and start or turn-off commands through the app to the car, which will be represented by a microcontroller for this project.

The Android app will be developed through Android Studio using Kotlin and Jetpack Compose for the UI and back-end development. The app will use a Bluetooth signal to send data packets to the microcontroller. Once a signal is received and verified to be from a trusted source, the microcontroller will then use the CAN BUS architecture to send commands to a representative car lock and engine ignition devices.

3.3 Subsystem Description

The Crypto-analysis Resistant Digital Key FOB will be made up of an android application, a cryptographic system that handles all communication, and a microprocessor**.** These requirements can be separated into the following subsystems.

Android Application Subsystem:
The Android Application subsystem will integrate with the Communication and Cryptography subsystem to send Bluetooth data packets to the microcontroller. The application will take in the hand signature of the user by using a hand gesture or stylus and provide that data to the Communication subsystem. The app will also take in the user commands and relay them to the Communication and Cryptography subsystem which will facilitate the communication between the mobile device and ECU.

Locking Mechanism Subsystem - Hardware:
This hardware subsystem will include a power source, a Buck-Converter circuit, a microcontroller, LEDs, and a CAN Transceiver. An ESP32 will be used at the necessary microcontroller. The ESP32 provides Bluetooth functionality as well as a CAN Controller for the CAN Bus network needed.

Figure 1: LEDs

The CAN Transceiver will assist the ESP32's CAN Controller in creating the CAN Bus. Two LEDs will be connected to the CAN bus network and function as Electronic Control Units; one green LED to show the car is locked/unlocked and one red LED to show the car's engine is on/off.

Communication and Cryptography Subsystem:
The Communication and Cryptography subsystem will cover the registration and communication processes between the device application and the car. During the registration process, the Z-Hardware Profile (a unique combination of the human signature entered by the user and a pseudo-random number generated by the ZAV mobile application) will be sent to the car with the use of the AES-256 Algorithm where the symmetric key will be created using the Diffie-Hellman key exchange between the car and device. During the communication process, the device uses a non-repeating timestamp and the Z-Hardware Profile to generate the dynamic digital signature using the Secure Hash Algorithm (SHA-256). This will be sent over to the car with the clear text timestamp and command. The car will then make its own dynamic digital signature using clear text timestamp sent by the digital key FOB and its own stored Z-Hardware Profile to verify that the device is the correct device before accepting the command.

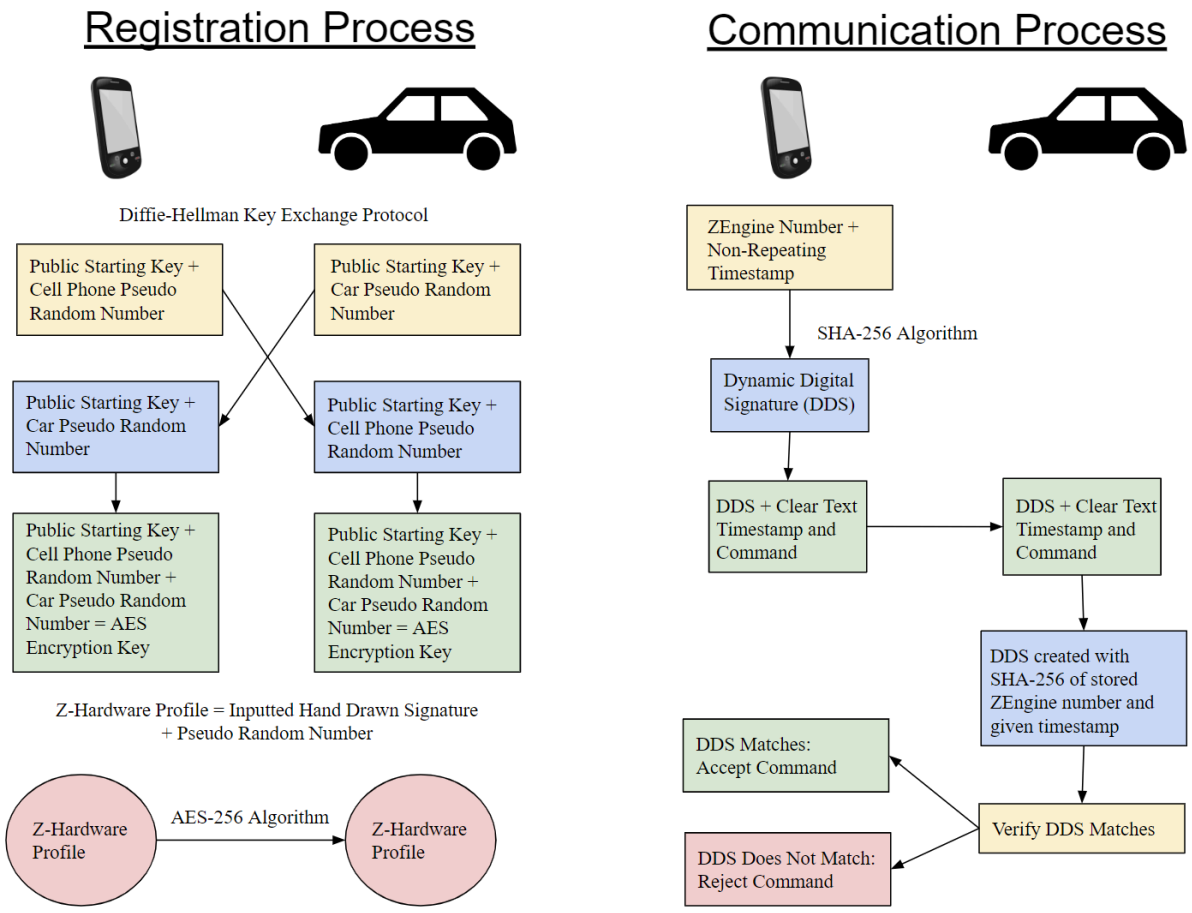


Figure 2: Flow Diagram of Registration and Communication Processes

3.4 Modes of Operations

Registration: The application will enter the registration mode when the user inputs their digital signature into the system, commencing the process for the application and the car microprocessor to exchange the Z-Hardware Profile, and store it for future communication between the digital key FOB and ECU.

Active: The application will enter the active mode automatically after registration while it waits for the user to select either the lock/unlock or start/turn-off commands.

Command: The application will enter the command mode when a command is selected on the application. This will initiate the communication process between the device and the car where the device will create a dynamic digital signature and, along with the unix-time and command, send it to the car where the car will verify the dynamic digital signature before accepting or rejecting the command.

3.5 Users

This digital key solution is intended for use by an everyday user who owns a motorized vehicle and who knows how to use basic digital applications. The Android application will be designed for ease of use and will require no training for operation. The end user will be able to interpret the button commands shown in the application and use them effectively.

3.6 Support

Support will be given to users through a FAQ page in the app showing basic information on app navigation, possible troubleshooting tips, and contact information for technical support services. Since the user will not be able to access the code for the device, explanations for the inner workings of the application will not be provided.

4. Threat Modeling Scenarios & Countermeasures

Below are three possible scenarios for the ZAV technology. The first scenario is the main one that our project will be focusing on, specifically in being able to securely lock, unlock, start, and turn-off a car. Once this project is completed, and it shows the capabilities of the ZAV technology, scenarios two and three highlight other scenarios in which this technology can be applied.

4.1 Anti-Theft Automobiles

Automobile theft is nothing new, but as hackers are able to competently breach older methods of security, new methods are beginning to arise. The ZAV technology provides a solution to this auto theft problem due to its secure cryptography software. Using ZAV and its secure, unique signal, thieves will no longer be able to hack into a car by relaying your keys' (whether they be physical or digital) RFID signal.

4.2 Secure Cloud Protection

Since the Zero Trust Application technology is a secure cryptography software that can generate cryptographic keys on the fly, it is able to store sensitive data onto a cloud-based server and secure that data with the Zero Trust Application without the need for insecure KMS. Having the Zero Trust Application on a personal device will allow only that device to access the sensitive data and no one else. Even if the personal device is hacked, the hacker still will not be able to access the data saved on the cloud if it incorporates what you know into the mix of non-repeating input data for the Communication and Cryptography subsystem to generate the access key.

4.3 Security of Firmware Updates

Most modern automobiles have increased connectivity and active sensing functionality integrated into the vehicle with about 200 million lines of code. Due to the scale and complexity of the coding, frequent software/firmware updates are needed but this leads to hackers being able to infiltrate firmware updates as they are sent over the air to the car. Using the ZAV technology, digital car components can be updated and "will be able to authenticate, encrypt/decrypt, and verify the integrity of 'the' message all in one signal data packet message"[1], thus only allowing firmware updates from the trusted source as well as guaranteed end-to-end encryption/confidentiality.

5. Analysis

5.1 Summary of Proposed Improvements

- Utilization of a digital key FOB allows for the user to always have their key on them preventing misplacing or theft of their car key FOB
- Dynamically changing signals when sending commands prevents relay attacks
- One-time registration creates secure communication between the car and the application
- Application user does not have to manually verify their identity through the use of passwords or biometric information

5.2 Disadvantages and Limitations

- Does not protect against hardware CAN bus injection attacks
- Not built to withstand any infiltration into the software
- Recreating already existing software so will not completely match the technology of Sandia National Laboratories or fully match the capabilities of ZAV

5.3 Alternatives

- PKI and AWS KMS
    - Popular infrastructure that is widely used
    - Less secure with many well-known strategies already in place to hack them
    - Requires the user to remember a password or input their biometrics
- Digital Key FOB provided by Car's Dealership
    - Already connected and compatible with the user's car
    - Can provide more features than just simple locking/unlocking and turning on/off the car
    - Does not provide the same level of security as the Crypto-analysis Resistant Digital Key FOB
- Hardware Key FOB
    - Comes with a key within it to allow for physical locking of the glove box and driver's side door
    - Easy to lose the key FOB or be stolen
    - Is susceptible to car theft through the use of a relay box to capture the signal and transmit it over to the car

5.4 Impact

This project will create a positive impact on the environment as it will allow for people to access and turn on/off their vehicle securely straight from their cell phone. Thus, our project will eliminate the environmental impact of making a physical Key FOB as they will no longer be needed.

Crypto-analysis Resistant Digital Key FOB

Adriana Matos
Courtney Eaton
Jeremy Hein

FUNCTIONAL SYSTEM REQUIREMENTS

REVISION – 3
April 30, 2024

Functional System Requirements
for
Crypto-analysis Resistant Digital Key FOB

Prepared by:

_____
Project Team Date

Approved by:

_____
Project Leader Date

_____
John Lusher, P.E. Date

_____
T/A Date

Change Record

Rev.	Date	Originator	Approvals	Description
-	9/18/2023	Adriana Matos		Draft Release
1	9/26/2023	Courtney Eaton		Submitted Version
2	12/1/2023	Courtney Eaton		403 Final Report
3	4/30/2024	Courtney Eaton		Capstone Completed Project Report

Table of Contents

List of Tables

List of Figures

1. Introduction

1.1 Purpose and Scope

This specification defines the technical requirements for the development items and support subsystems delivered to the client for the project. Figure 1 shows a representative integration of the project in the proposed CONOPS. The verification requirements for the project are contained in a separate Validation Plan document.

The Crypto-analysis Resistant Digital Key FOB (C-kF) system will provide a secure and efficient solution for managing vehicle access and control. This system includes an Android application, Communication and Cryptography subsystem, and hardware components acting as the vehicle. The Android application will facilitate user and hardware registration by capturing a hand signature through a gesture or stylus input coupled to a pseudo-random number generated (associated with hardware - mobile device) and manage communication with the vehicle's Electronic Control Units (ECUs) to send commands for vehicle unlocking and starting mechanism. The hardware subsystem includes a microprocessor, LEDs, CAN bus network, and Bluetooth connectivity. This will provide seamless hardware-software integration. For initial device/key FOB registration, the Communication and Cryptography subsystem will provide security through the utilization of the Diffie-Hellman Key Exchange and AES-256 algorithm. This ensures the system is impervious to remote crypto-analysis attacks. Once the mobile device and ECUs are “coupled/registered” to each other, the Communication and Cryptography subsystem can ensure the system is resistant to crypto-analysis attacks through the utilization of SHA-256 to verify the device sending the locking/unlocking and starting/stopping engine command. In essence, this system's scope encompasses the transportation industry's need for a secure and user-friendly digital key FOB solution.

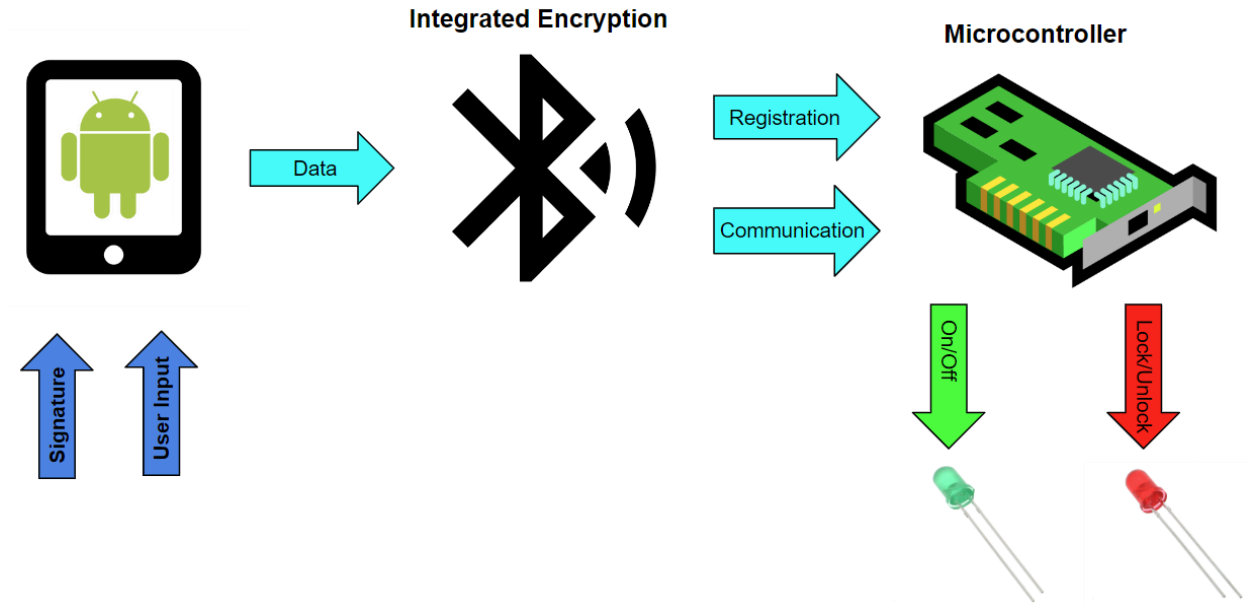


Figure 1. C-kF Conceptual Image

1.2 Responsibility and Change Authority

The team leader, Adriana Matos-Almodovar, is responsible for making sure the project's requirements are met. If any change is made to the project, whether it be the deliverables or final demonstration, it must be approved by the team leader, Adriana Matos-Almodovar, as well as the team's sponsor, Peter Choi.

Subsystem	**Responsible Team Member**
Locking Mechanism-Hardware	Adriana Matos
Android Application	Jeremy Hein
Communication and Cryptography	Courtney Eaton

Table 1. Subsystem Responsibilities

2. Applicable and Reference Documents

2.1 Applicable Documents

The following documents, of the exact issue and revision shown, form a part of this specification to the extent specified herein:

Document Number	**Revision/Release Date**	**Document Title**
IEEE 802.15.1	2002	IEEE Standard for Telecommunications and Information Exchange Between Systems- LAN/MAN - Specific Requirements - Part 15: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Wireless Personal Area Networks (WPANs)
IEEE 1363	2000	IEEE Standard Specifications for Public-Key Cryptography
IEEE 1363.3	2013	IEEE Standard for Identity-Based Cryptographic Techniques using Pairings
IEEE 287.1	2021	IEEE Standard for Precision Coaxial Connectors at RF, Microwave, and Millimeter-Wave Frequencies – Part 1: General Requirements, Definitions, and Detailed Specifications
FIPS PUB 180-4	August 2015	Secure Hash Standard (SHS)
Processing Standard Publication 197	November 26, 2001	Announcing the Advanced Encryption Standard (AES)

Table 2. Applicable Documents

2.2 Reference Documents

The following documents are reference documents utilized in the development of this specification. These documents do not form a part of this specification and are not controlled by their reference herein.

Document Number	Revision/Release Date	Document Title
ISO 11898-1	2015	Road vehicles — Controller area network (CAN) — Part 1: Data link layer and physical signaling

Table 3. Reference Documents

2.3 Order of Precedence

In the event of a conflict between the text of this specification and an applicable document cited herein, the text of this specification takes precedence without any exceptions.

All specifications, standards, exhibits, drawings, or other documents that are invoked as “applicable” in this specification are incorporated as cited. All documents that are referred to within an applicable report are considered to be for guidance and information only, except ICDs that have their relevant documents considered to be incorporated as cited.

3. Requirements

3.1 System Definition

The C-kF will be comprised of three subsystems: the Android Application Subsystem, the Communication and Cryptography Subsystem, and the Locking Mechanism Subsystem. The Android Application subsystem is responsible for creating an Android application that will act as a digital key FOB and is able to communicate with the Locking Mechanism. This will be done by providing an interface for any user to easily be able to send the wanted command to their vehicle. The Communication and Cryptography subsystem is responsible for the Registration and Communication processes which will be implemented on both the Android application and the locking mechanism to securely send information between the mobile device and the vehicle. Finally, the Locking Mechanism subsystem will take the place of the vehicle as a way to demonstrate that the application is successful in sending the commands. It is responsible for mimicking a vehicle’s ECU’s internal communication system to “lock/unlock” or “turn on/off” the vehicle using the digital key FOB Android application. Together these subsystems will work to provide the user with an application to securely access and start their vehicle thus preventing any relay, man-in-the-middle, or other attacks to gain unwanted access to the user’s vehicle.

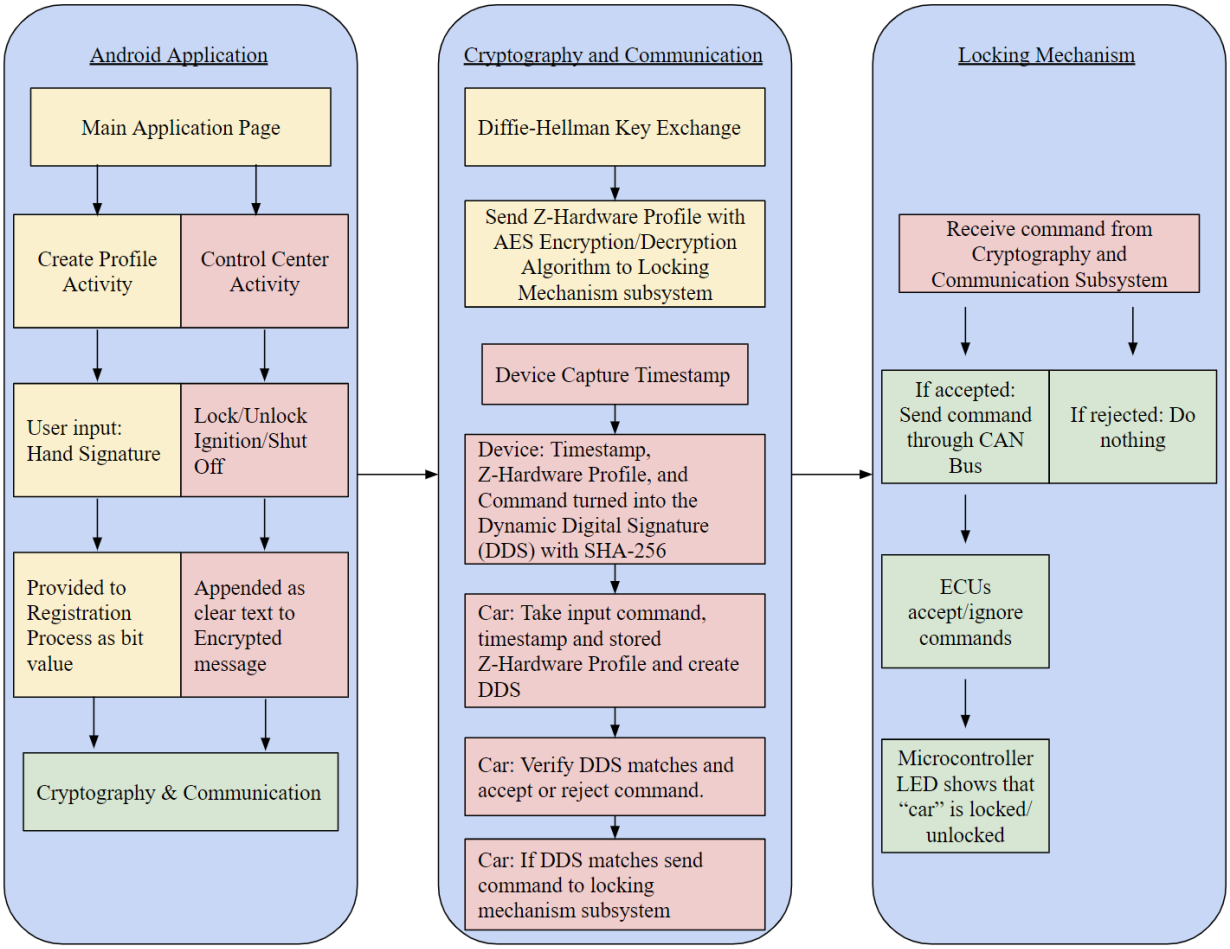


Figure 2. Block Diagram of System

As shown in Figure 2 above, the Android Application subsystem will be connected to the Communication and Cryptography subsystem, and the Communication and Cryptography subsystem will be connected to the Locking Mechanism subsystem.

In order to facilitate the Registration Process (shown by the yellow boxes above), the Android Application subsystem will include a screen that allows the user to enter their vehicle name and a hand signature done with their finger or a stylus. This bitmap image will then be handed over to the Communication and Cryptography subsystem in the form of a hash. This subsystem will convert the hand signature into the Z-Hardware Profile and with the AES-256 encryption/decryption algorithm securely send it to the Locking Mechanism subsystem where it will be stored.

During the secure communication process (shown by the red boxes above), the user will select a command through the application. The device will then capture the unix-time so that the Communication and Cryptography subsystem can use the Z-Hardware profile to modify the captured unix-time and, using SHA-256, turn this combined Z-Hardware profile and unix-time into the Dynamic Digital Signature (DDS). This signature will then be sent over to the vehicle along with the timestamp and command in clear text. The vehicle will then use the shared timestamp and SHA-256 to create its own DDS. The mobile device's DDS will then be verified against the vehicle's DDS. If both DDSs match then the command

will be accepted and sent to the Locking Mechanism subsystem. If they do not match, then the command will simply be ignored.

3.2 Characteristics

3.2.1 Functional / Performance Requirements

3.2.1.1 Android Application Requirements

3.2.1.1.1 Profile Creation/Hand Signature

When creating a vehicle Profile, the user must input a hand signature using either their finger or a stylus. This signature shall be converted to a hash and mixed with a pseudo-random number to be used as a common Z-hardware profile for the mobile device and ECUs.

> *Rationale: This requirement is necessary to provide a non-replicable dynamic value to the Communication and Cryptography subsystem.*

3.2.1.1.2 Multiple Profiles

The user shall be able to create multiple profiles for different vehicles/ECUs and select the profile they want to control from the main page.

> *Rationale: Multiple profiles allow the user to control multiple vehicles with one device which is useful as a large majority of the population uses more than one vehicle.*

3.2.1.1.3 Control Panel

After selecting the desired Vehicle Profile, the user will be able to navigate to a control panel screen and have 2 buttons for controlling the vehicle: Lock/Unlock and Ignition/ Shut-off. The application page will also indicate the pre-existing condition of the vehicle showing if it is On/Off or Locked/Unlocked aiding the user in knowing which command to send.

> *Rationale: The Control Panel is necessary to provide the user with a way of inputting their commands and relaying them to the Communication subsystem.*

3.2.1.1.4 FAQ Page

Each page in the application is included on the navigation bar. The last page in the application leads to the FAQ page that will include basic instructions and guides for operating the software or contacting support.

> *Rationale: Providing the user with basic operating information is an ease-of-use improvement aimed at assisting people with less knowledge of operating digital software and devices and providing users with the ability to troubleshoot the most likely problems they may encounter.*

3.2.1.1.5 Android OS Version

The application will run on Android 10. The app shall have a minimum Android SDK of 29. The compiled SDK will be 31. This Android OS runs on approximately 75% of all Android devices according to Android Studio.

Rationale: An Android application requires an OS to run the software. Android 10 was chosen as it includes modern features and still provides widespread support for older devices.

3.2.1.1.6 Status Messages

Error messages if the registration takes longer than expected will be shown to the user such as “Registration Failure” and “Connection Error”. The inverse of those messages will be displayed for successful operations. Examples are “Registration Successful” and “Connection Successful”.

Rationale: It is necessary to inform the user of the success or failure of their attempted operation for them to have no doubt that the application is performing as intended.

3.2.1.2 Communication and Cryptography Subsystem

3.2.1.2.1 Securely Send/Receive Data

The Android Application and Locking Mechanism should be able to send/receive data securely to avoid man-in-the-middle attacks made to one’s vehicle. To accomplish this, the Diffie-Hellman Key Exchange Protocol shall be used to exchange a one time encryption key between the device and vehicle. The AES-256 Encryption/Decryption Algorithm shall be used to send the Z-Hardware Profile from the device to the vehicle. Finally, SHA-256 shall be used to create the Dynamic Digital Signature for securely sending commands to the vehicle.

Rationale: This requirement is to ensure that the communication between the vehicle and device is secured using the most widely accepted encryption/decryption and hashing algorithms.

3.2.1.3 Locking Mechanism Subsystem

3.2.1.3.1 Accurate Demonstration of Lock/Unlock and Start/Stop

The Locking Mechanism subsystem shall accurately indicate the “vehicle” is locked/unlocked using the LED lights on the PCB. It will also indicate if the “vehicle” is turned on/off through the use of LED lights.

Rationale: This is a basic requirement to show that the demonstration PCB is working properly and the secure communication between the Android Application and the Locking Mechanism is successful.

3.2.1.3.2 Bluetooth Capabilities

The Locking Mechanism subsystem shall have bluetooth capabilities in compliance with the IEEE 802.11ac standards.

Rationale: This is a basic requirement to allow for communication between the Android application and the Locking Mechanism.

3.2.2. Physical Characteristics

3.2.2.1. Structural

The Locking Mechanism will consist of a custom made PCB Microcontroller inside of a housing unit for protection.

Rationale: This is a requirement from the sponsor to mimic a vehicle and show that a digital key FOB can have secure communication with that vehicle.

3.2.3. Electrical Characteristics

3.2.3.1. Inputs

The input into the system shall not cause any damage or malfunction to the main internal communication system (CAN Bus). No man-in-the-middle input attacks should affect the system whatsoever as any data inputs will be confirmed by the Communication and Cryptography subsystem to solely come from a verified device through the Android Application being developed. Along with data input from the application, the mechanism shall also intake power from a wall outlet adapter.

Rationale: Due to the design of the microcontroller being used, the chance of a malfunction due to power failure should be limited.

3.2.3.1.1 Input Voltage Level

The input voltage level for the Locking Mechanism Microcontroller will be about +7V to +12V of DC voltage coming from the power supply adapter. The power shall come from a power supply adapter that is connected to an AC wall socket.

Rationale: An AC wall socket will allow for ease of use of the Locking Mechanism as wall outlets are fairly common. The microcontroller will be able to accept a minimum of 6V and a maximum of 20V using the Jack connector so a input voltage level of 7V to 12V was chosen to provide a buffer in case of power spikes.

3.2.3.1.2 External Commands

The Locking Mechanism shall take in and store the external commands (Z-Hardware Profile) that come from the Android application.

Rationale: This allows for the Z-Hardware profile to be used to verify the identity of the device attempting to give a command to the vehicle.

3.2.3.2. Outputs

3.2.3.2.1 Data Output

The Locking Mechanism's output shall be a visible representation that the system is either locked, unlocked, turned on, or turned off. The locking/unlocking representation will be done by using a red LED: when the LED is on it signifies that the system is "locked" and when the LED is off it signifies that the system is "unlocked". The turned on/off representation will be done by using a green LED: when the system is "on" the LED will turn on and vice versa.

Rationale: This allows for the command passed on from the application to be visibly shown on the Locking Mechanism for demonstration purposes.

3.2.3.3. Connectors

The Locking Mechanism Subsystem shall follow the International Electrotechnical Commission IEC 60130-10:1971 standard for electrical connectors.

Rationale: Conform to the standard for coaxial connectors.

3.2.3.4. Wiring

The Locking Mechanism Subsystem shall reference the ISO 11898-1 standard for CAN bus architecture wiring.

Rationale: Conform to wiring standard.

3.2.4. Environmental Requirements

The C-kF system does not have any environmental requirements.

3.2.5. Failure Propagation

The C-kF system should not allow any errors or communication interruptions of any sort. For project simplicity, a couple of security assumptions are made: 1. hackers do NOT have physical access to the ECUs, and 2. hackers do NOT have control over the mobile device.

3.2.5.1. Wireless Communication Error

Any bluetooth errors will be caught and relayed to the user by the Android Application Subsystem.

Rationale: *This will cover any possible bluetooth error and alert the user of this error.*

3.2.5.2. Cryptography Error

If the registration or communication processes done by the Communication and Cryptography Subsystem take longer than expected, indicating that there was an issue with the processes, then the Android Application will send an error message to the user.

Rationale: This will allow the user of the application to restart the registration or resend the command.

3.2.5.3. Electrical Errors

The Locking Mechanism has a built-in green LED light to show that the system is fully running as it should and that the PCB is connected to power. This LED light turns on for a moment when the PCB is initially connected to power and turns off a moment after. If

this green LED light does not turn on when the system is first connected to power, a problem has occurred in the subsystem and a reset button will be available to reset the power.

Rationale: For demonstration purposes, this will allow the user to reset power and restart the system if any electrical errors occur.

4. Support Requirements

The Android Application will include a FAQ page with basic instructional and operational information allowing users who do not have previous experience with similar apps to navigate and use the interface and controls with ease. There is no warranty for this software as we are not providing a new device but simply installing additional software to an already existing digital infrastructure. If a large company was providing this service a technical support service would be necessary but this project's design scope does not include this solution. When downloading the application from the Google Play Store, the customer will be shown the device requirements for running the application and whether their device meets them.

Appendix A: Acronyms and Abbreviations

Below is a list of common acronyms and abbreviations used in this project.

AES	Advanced Encryption Standard
CAN	Controlled Area Network
C-kF	Crypto-analysis Resistant Digital Key FOB
DDS	Dynamic Digital Signature
ECU	Electronic Control Unit
IEC	International Electrotechnical Commission
ISO	International Organization for Standardization
KMS	Key Management Service
LED	Light-Emitting Diode
OTA	Over-The-Air
PCB	Printed Circuit Board
PKI	Public Key Infrastructure
SDK	Software Development Kit
SHA	Secure Hash Algorithm
V	Volt
ZAV	Zero-Trust Application for Vehicle
ZTA	Zero Trust Application

Appendix B: Definition of Terms

Dynamic Digital Signature - The resulting ciphertext/digest created when the combined Z-Hardware Profile and unix-time are put through the SHA-256 Algorithm.

Z-Hardware Profile - Unique combination of the hand imputed signature (with a finger or stylus) and a pseudo-random number which is used to verify the identity of the device trying to send commands to the vehicle.

Crypto-analysis Resistant Digital Key FOB

Adriana Matos
Courtney Eaton
Jeremy Hein

INTERFACE CONTROL DOCUMENT

REVISION – 3
April 30, 2024

Interface Control Document
FOR
Cypto-analysis Resistant Digital Key FOB

Prepared by:

_____
Project Team Date

Approved by:

Adriana Matos
_____
Project Leader Date

_____
John Lusher II, P.E. Date

_____
T/A Date

Change Record

Rev.	Date	Originator	Approvals	Description
-	9/23/2023	Adriana Matos		Draft Release
1	9/26/2023	Courtney Eaton		Submitted Version
2	12/1/12023	Courtney Eaton		403 Final Report
3	4/30/2024	Courtney Eaton		Capstone Completed Project Report

Table of Contents

List of Figures

1. Overview

This document will provide more detail on the interface between the user, Android Application on the device and Locking Mechanism hardware. It will first describe the physical, thermal, and electrical interfaces of the device and Locking Mechanism. After, the communication interface between the device and Locking Mechanism will be described.

2. References and Definitions

2.1 References

Refer to section 2.1 and 2.2 of the Functional System Requirements Document.

2.2 Definitions

AES	Advanced Encryption Standard
CAN	Controlled Area Network
C-kF	Crypto-analysis Resistant Digital Key FOB
DDS	Dynamic Digital Signature
ECU	Electronic Control Unit
IEC	International Electrotechnical Commission
ISO	International Organization for Standardization
KMS	Key Management Service
LED	Light-Emitting Diode
mm	Millimeter
OTA	Over-The-Air
PCB	Printed Circuit Board
PKI	Public Key Infrastructure
SDK	Software Development Kit
SHA	Secure Hash Algorithm
V	Volt
ZAV	Zero-Trust Application for Vehicle
ZTA	Zero Trust Application

3. Physical Interface

3.1 Weight

The weight of the PCB Locking Mechanism will be about 25 grams in total, not including the housing unit. During the demonstration process a tablet will be used to demonstrate the application's capabilities and the weight of the tablet is determined by the Android device's manufacturer.

3.2 Dimensions

The dimensions of the PCB containing the Locking Mechanism are 68.2 mm x 52.2 mm. The resistors and capacitors within the PCB will mainly be 0402 with 1.0 mm x 0.5 mm dimensions and 0805 with 2.0 mm x 1.2 mm dimensions. The microcontroller's dimensions are 18 mm x 25.5 mm x 3.1 mm

4. Thermal Interface

The microcontroller does not need a heat sink and should be able to dissipate the heat created on its own.

5. Electrical Interface

5.1 Primary Input Power

The Locking Mechanism will be powered by an 120V AC wall outlet. Though the microcontroller does not need to work with 120V AC power, an adapter will be used to step down the convert the 120V AC power to 24V DC power. A buck converter circuit will then step-down the 24V DC input from the barrel jack connector to 3.3V needed by the microcontroller to function.

5.2 Signal Interfaces

The Locking Mechanism will have an internal communication network that allows Electronic Control Units (ECUs) to send internal signals that allow other ECUs to accept or ignore. External signals will be sent over from the user control interface and broadcasted through the microcontroller's internal communication network.

5.3 User Control Interface

The user control interface is the Android device application that acts as the Locking Mechanism's digital key FOB. The user's input signal will be passed through the

Communication and Cryptography subsystem then sent over to the microcontroller to be accepted by the electronic control units within the microcontroller.

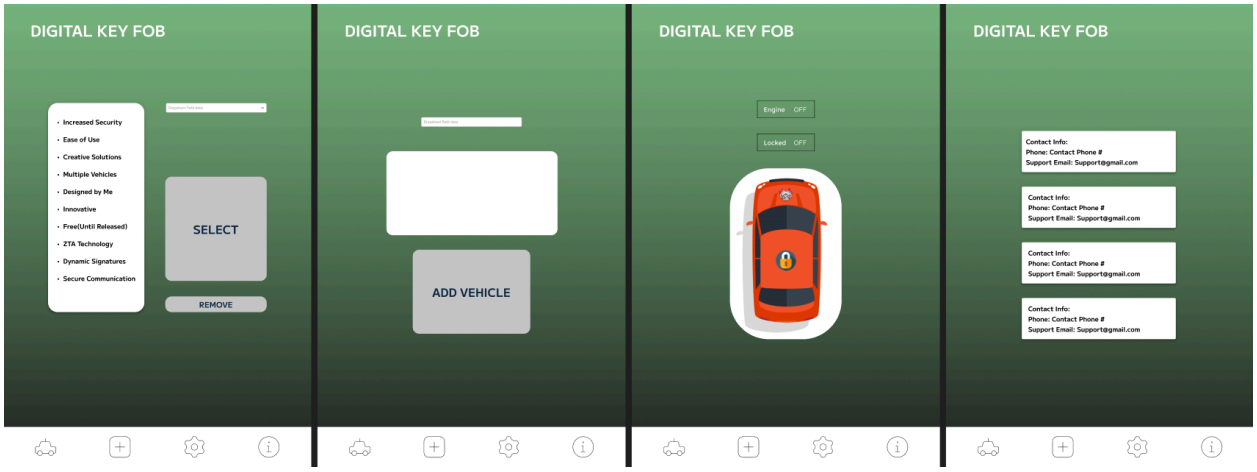


Figure 1: Initial Conceptual UI layout

6. Communications/Device Interface Protocols

6.1 Bluetooth Communication

The microcontroller will have an ESP32 chip assembled with it that has bluetooth capabilities using the IEEE 802.11ac standards. This bluetooth connection will be used so the Android Application can communicate with the Locking Mechanism.

6.2 Diffie-Hellman Key Exchange Protocol

The Diffie-Hellman Key Exchange Protocol will be used during the registration process in order to create and set up the symmetric key to be used by the device and vehicle during the transfer of the Z-Hardware Profile.

6.3 Advanced Encryption Standard (AES), 256 bits

The AES-256 algorithm will be used to securely send the Z-Hardware Profile from the device to the vehicle. It will operate based on the specifications put in place by the National Institute of Standards and Technology (NIST) for this standard.

6.4 Secure Hash Algorithm (SHA), 256 bits

SHA-256 will be used to create the Dynamic Digital Signature from the Z-Hardware Profile and unix-time. It will operate based on the Secure Hash Standard set in place by the NIST.

Crypto-analysis Resistant Digital Key FOB

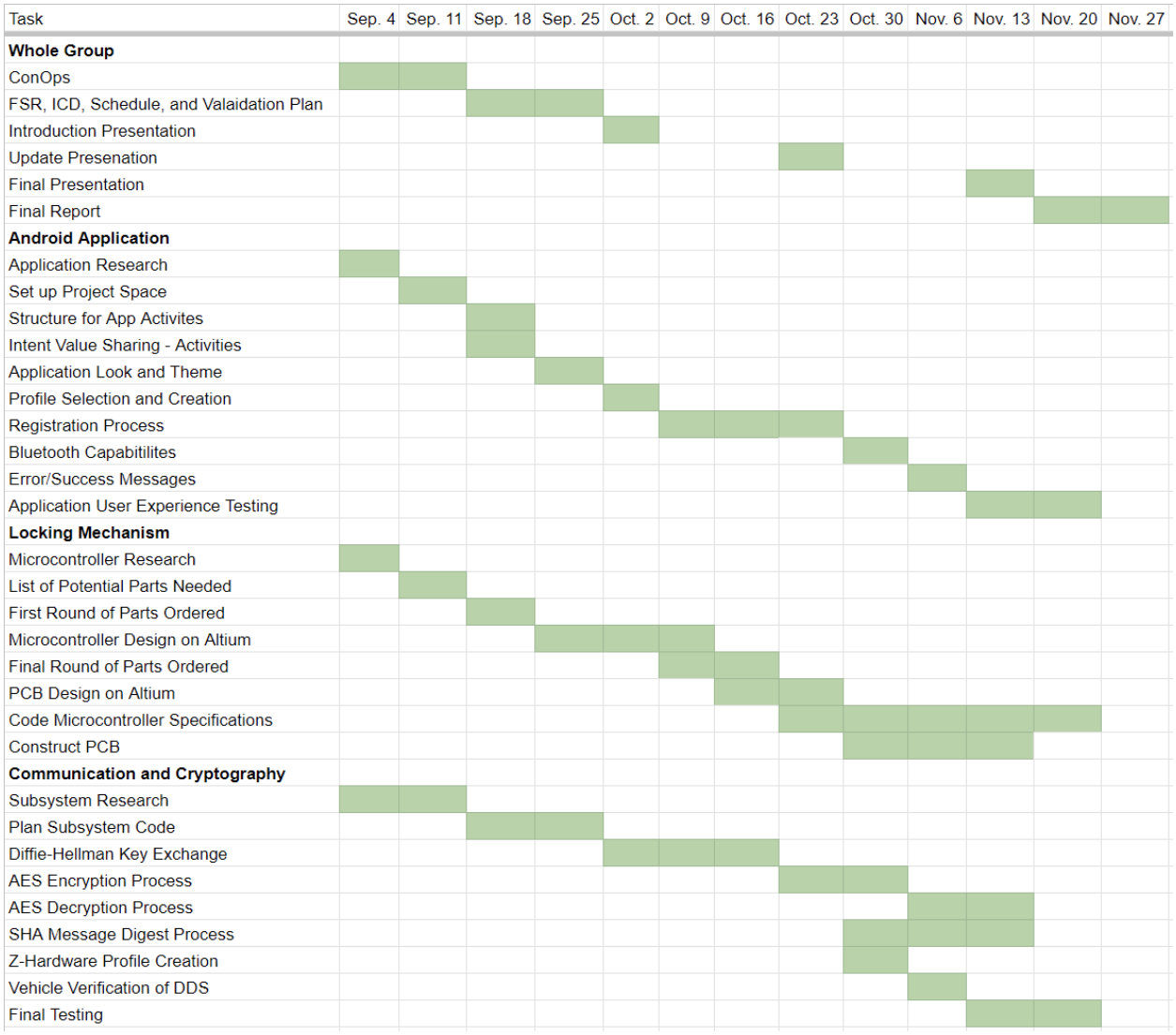
Adriana Matos
Courtney Eaton
Jeremy Hein

SCHEDULE AND VALIDATION PLAN

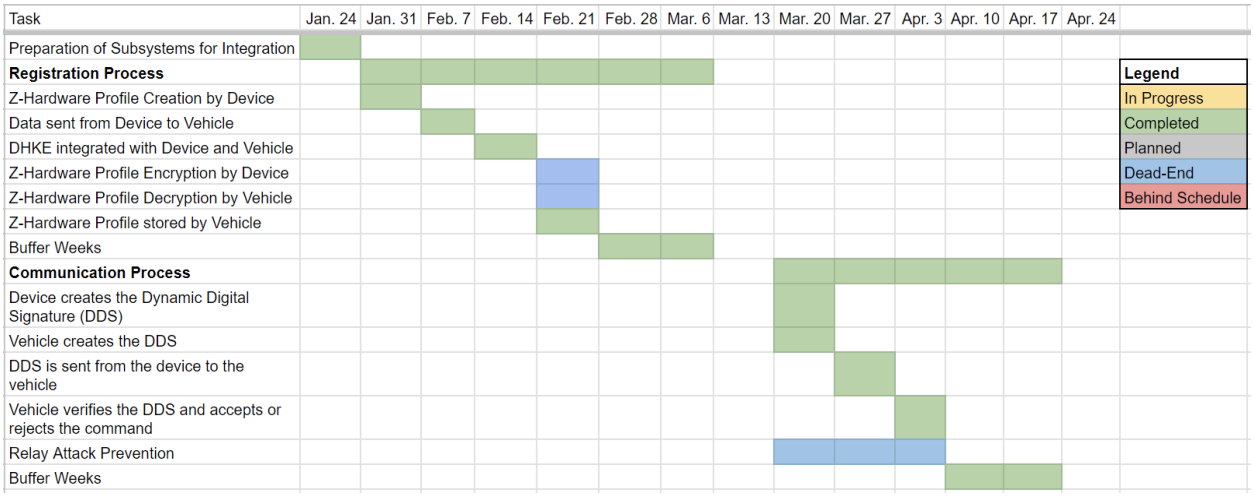
REVISION – 3
April 30, 2024

Schedule

The schedule for the first half of this project (ECEN 403) is shown below. As can be seen, all of the required milestones were met for all three subsystems. There was some slight shuffling in the order of how things were completed as various unplanned problems arose and certain milestones took longer than expected. In the end, however, everything was completed and is in working order.



The schedule for the second half of this project (ECEN 404) is shown below. As can be seen, all of the required milestones were met for the integration of the Registration and Communication processes except the AES-256 Encryption and Decryption algorithm and the Relay Attack Prevention Code which were taken out of the scope of the project. This is explained further in the Communication and Cryptography Subsystem Report.



Validation Plan

The validation plan for the first half of this project (ECEN 403) is shown below. Most of the validation tests were completed with positive results so that the three subsystems are ready to be integrated in ECEN 404.

Test Name	Success Criteria	Person Responsible	Status	Date Tested
App Functionality	Application will start on device running a compatible OS	Jeremy	Completed	10/25/2023
App Navigation	Users will be able to freely navigate app without getting stuck on any page with no errors	Jeremy	Completed	10/20/2023
Intent Value Sharing	Activities will share necesary information when certain fields are filled or buttons pressed	Jeremy	Completed	10/21/2023
Application Design	Application will be viewable as a product with modern designs and layouts	Jeremy	Completed	11/20/2023
Hand Signature	During registration process, application will take in a user gesture and convert it to an image	Jeremy	Completed	10/22/2023
Registation Procedure	User will create a Car Profile and the app will store the profile parameters	Jeremy	Completed	10/22/2023
Communication Procedure	Application will relay user commands to the Communication Subsystem	Jeremy	Completed	11/11/2023
FAQ Page	User will be able to enter FAQ Page from starting screen	Jeremy	Completed	10/20/2023
Bluetooth Capability	Application will be able to connect to Bluetooth devices	Jeremy	Completed	11/11/2023
Red/Green LED Test	Will turn on when a specific voltage is applied and will turn off when opposite is applied	Adriana	Completed	11/22/2023
ESP 32 Chip/CAN Transceiver	CAN Bus Network sends and recieves information	Adriana	In-Progress	
Power	Microcontroller is able to turn on/off with no problems or excessive heat dissipation	Adriana	Completed	11/14/2023
Overall Microcontroller	Whole system works together and no interruppted current paths	Adriana	Completed	11/14/2023
Bluetooth	Microcontroller is able to accept data over Bluetooth interface	Adriana	Completed	11/22/2023
	Hand signature image turns into a unique bit number	Jeremy/Courtney	Completed	11/6/2023
Z-Hardware Profile	Signature bit number is combined with a pseudo-random number	Courtney	Completed	11/6/2023
	Device and vehicle are able to create their own pseudo-random number combined with the public starting key	Courtney	Completed	10/24/2023
Diffie-Hellman Key Exhcnage Protocol	Device and vehicle create AES symmetric key from the other's sent information	Courtney	Completed	10/24/2023
AES Encyption Code	Z-Hardware Profile is Encrypted	Courtney	Completed	11/27/2023
AES Decrpyiton Code	Z-Hardware Profile is Decrypted	Courtney	Completed	11/25/2023
	Z-Hardware Profile is created into the Dynamic Digital Signature for the device	Courtney	Completed	11/7/2023
SHA Algorithm Code	Z-Hardware Profile is created into the Dynamic Digital Signature for the vehicle	Courtney	Completed	11/25/2023
Verification Code	Vehicle is accurately able to verify if Dynamic Digital Siganture created by the device matches that created by the car	Courtney	Completed	11/14/2023

The updated validation plan for this project (created during ECEN 404) is shown below. All of the validation requirements were tested and met showing that the overall Crypto-analysis Resistant Digital Key FOB is in working order as described in the CONOPS, FSR, and ICD.

Test Name	Requirement	Success Criteria	Methodology	Responsible Party	Status
App Functionality	FSR 3.2.1.1.5	Application will start on device running a compatible OS	Test that app is functional on all Android OS above Android 10 and on different device manufacturers.	Jeremy	Completed
App Navigation	ICD 5.3	Users will be able to freely navigate app without getting stuck on any page with no errors	Have multiple Users test application and attempt to use every function within the application to ebsure no crashes or application breaking errors.	Jeremy	Completed
Intent Value Sharing	FSR 3.2.1.1.2	Activities will share necesary information when certain fields are filled or buttons pressed	Application will save certain feilds of information when traveling between activities.	Jeremy	Completed
Application Design	ICD 5.3	Application will be viewable as a product with modern designs and layouts	The application is reviewed by user testing to ensure it is visually acceptable.	Jeremy	Completed
Hand Signature	FSR 3.2.1.1.1	During registration process, application will take in a user gesture and convert it to an image	Run registration process thuroughly and ensure each instance creates a unuiqe hash from the signature pad.	Jeremy	Completed
Registation Procedure	FSR 3.2.1.1.2	User will create a Car Profile and the app will store the profile parameters	View application database to ensure profiles are being stored correctly and each paramter is being captured as desired.	Jeremy	Completed
Communication Procedure	FSR 3.2.1.1.3	Application will relay user commands to the Communication Subsystem	Test to ensure that application is integrated with the communciation subsystem.	Jeremy	Completed
FAQ Page	FSR 3.2.1.1.4	User will be able to enter FAQ Page from starting screen	The application is reviewed by user testing to ensure this page is accessable.	Jeremy	Completed
Bluetooth Capability	ICD 6.1	Application will be able to connect to Bluetooth devices	Test to ensure application connects and can transfer data successfully without connection interuptions. Testing is limited since virtual devices can't utilize bluetooth	Jeremy	Completed
Red/Green LED Test	FSR 3.2.1.3.1	Will turn on when a specific voltage is applied and will turn off when opposite is applied	Visually confirm if LED turns on/off when signal is sent from Serial Bluetooth Terminal Application on Android device	Adriana	Completed
Power	ICD 5.1	Microcontroller is able to turn on/off with no problems or excessive heat dissipation	Connect MCU to power and leave on to test excess heat dissipation after about 5 minutes. Make sure computer recognizes MCU when connected through UART.	Adriana	Completed
Overall Microcontroller	FSR 3.2.1.3	Whole system works together and no interruppted current paths	Test copper paths on PCB, while connected to power, with a multimeter and verify proper voltage	Adriana	Completed
Bluetooth	ICD 6.1	Microcontroller is able to accept data over Bluetooth interface	Verify on Serial Monitor that data is accepted	Adriana	Completed

Test Name	Requirement	Success Criteria	Methodology	Responsible Party	Status
Hash Map Creation	FSR 3.2.1.1.1	Hand signature image turns into a unique bit number	Visually confirm Hand Signature is stored as a hash map number.	Jeremy/Courtney	Completed
Z-Hardware Profile Creation	FSR 3.2.1.2.1	Signature bit number is combined with a pseudo-random number	Run Z-Hardware creation code and ensure that inputted Z-Hardware profile is accuratly combined with the Pseudo-Random number.	Courtney	Completed
DHKE Protocol - Part 1, Pre-Bluetooth	ICD 6.2	Device and vehicle are able to create their own pseudo-random number combined with the public starting key	Run DHKE Pt. 1 code and use output terminal to confirm proper operation.	Courtney	Completed
DHKE Protocol - Part 2, Post-Bluetooth	ICD 6.2	Device and vehicle create AES symmetric key from the other's sent information	Run DHKE code for Java and C++ (manually moving the needed values) and confirm that final encryption key is the same for both.	Courtney	Completed
AES Encyption Code	ICD 6.3	Z-Hardware Profile is encrypted with AES in Java based on the specifications by NIST	Enter plain text and ensure that program is able to turn it into a cipher text and back into the same plain text.	Courtney	Completed
AES Decrpyiton Code	ICD 6.3	Z-Hardware Profile is decrypted with AES in C++ based on the specifications by NIST	Enter plain text and ensure that program is able to turn it into a cipher text and back into the same plain text.	Courtney	Completed
SHA DDS Creation - Device	ICD 6.4	Z-Hardware Profile is created into the DDS for the device based on NIST requirements	A plain text entered into the code creates a hash text that matches the one created for the vehicle.	Courtney	Completed
SHA DDS Creation - Vehicle	ICD 6.4	Z-Hardware Profile is created into the DDS for the vehicle based on NIST requriements	A plain text entered into the code creates a hash text that matches the one created for the device.	Courtney	Completed
Verification Code	FSR 3.2.1.2.1	Vehicle is accurately able to verify if Dynamic Digital Siganture created by the device matches that created by the car	The code terminal is used to ensure that the code accuratly says if the hardcoded Z-Hardware profile (used for testing) and the entered Z-Hardware profile match or not.	Courtney	Completed

Legend
Android Application Subsystem
Locking Mechanism Subsystem
Communication and Cryptography Subsystem
Integration

Z-Hardware Profile Creation - Device	FSR 3.2.1.2.1	Z-Hardware Profile is created on the device and stored.	Visually confirm Z-Hardware profile is stored in the database.	Whole Group	Completed	
DHKE Protocol	ICD 6.2	Symmetric Encryption Key is created on the device and vehicle hardware and match eachother.	Run the DHKE and confirm both the device and vehicle have the same encryption key.	Whole Group	Completed	
Z-Hardware Profile Encryption	ICD 6.3	Z-Hardware Profile is encrypted with AES on the device and sent to the vehicle.	Confirm the plain text Z-Hardware profile is able to be turned into a cypher text and recived by the vehicle.	Whole Group	Postponed	
Z-Hardware Profile Decryption	ICD 6.3	Z-Hardware Profile is received by the vehicle, decrypted, and stored.	Visually confirm receved cypher text is able to be decrypted into the original Z-Hardware profile and stored in the vehicle hardware.	Whole Group	Postponed	
User Interface - Communication Process	ICD 6.4	Command entered by the user is taken in by the device and Dynamic Digital Signature is created.	Click a command within the app and visually confirm the DDS is created from the Z-Hardware Profile and command by looking at recieved string on vehicle terminal.	Whole Group	Completed	
DDS Sent via Bluetooth	ICD 6.1	Device's DDS and the entered command are sent via bluetooth to the vehicle.	Visually confirm DDS is accuratly sent to the vehicle	Whole Group	Completed	
DDS Creation - Vehicle	ICD 6.4	Vehicle is able to create a DDS from given command.	Visually confirm DDS is made from stored Z-Hardware profile and sent command.	Whole Group	Completed	
DDS Verification	FSR 3.2.1.2.1	Vehicle is able to verifiy given DDS with it's own and the command is accepted or rejected.	Run verification code and ensure that command is sent when Z-Hardware profiles should match and is not sent when they do not.	Whole Group	Completed	
Relay Attack Prevention		Use device and vehicle built in time to confirm the identity of the device.	Run verification code and ensure that if the command is only taken in if it comes within the correct time frame.	Whole Group	Postponed	

Crypto-analysis Resistant Digital Key FOB

Jeremy Hein

ANDROID APPLICATION SUBSYSTEM REPORT

REVISION – 2
April 29, 202

SUBSYSTEM REPORT
FOR
Crypto-analysis Resistant Digital Key FOB

TEAM <2>

APPROVED BY:

_____
Project Leader Date

_____
Prof. Kalafatis Date

_____
T/A Date

Change Record

Rev.	Date	Originator	Approvals	Description
1	12/3/2023	Jeremy Hein		Submitted Revision
2	4/29/2024	Jeremy Hein		Submitted Revision

Table of Contents

List of Figures

List of Tables

Definitions

Jetpack Compose: A modern toolkit for building native Android UI
XML: A markup language for Android Studio UI development

Android Application Subsystem Report

1. Subsystem Introduction

The Android application subsystem provides an interface where the user can create car profiles to send commands to their vehicles. The application sends these commands using Bluetooth data packets. The details surrounding the methods used for accomplishing these features are in the following sections.

2. Subsystem Details

2.1 Room Database

The Android Application Subsystem utilizes Room Database to create a streamlined developer-friendly interface for SQLite. Room provides tools such as data access objects (DAO) and Compile-time verification. DAO helps ensure a structured approach to database interactions and queries. Compile-time verification alleviates runtime errors and provides seamless integration with Android architecture which in turn supports a cleaner app structure.

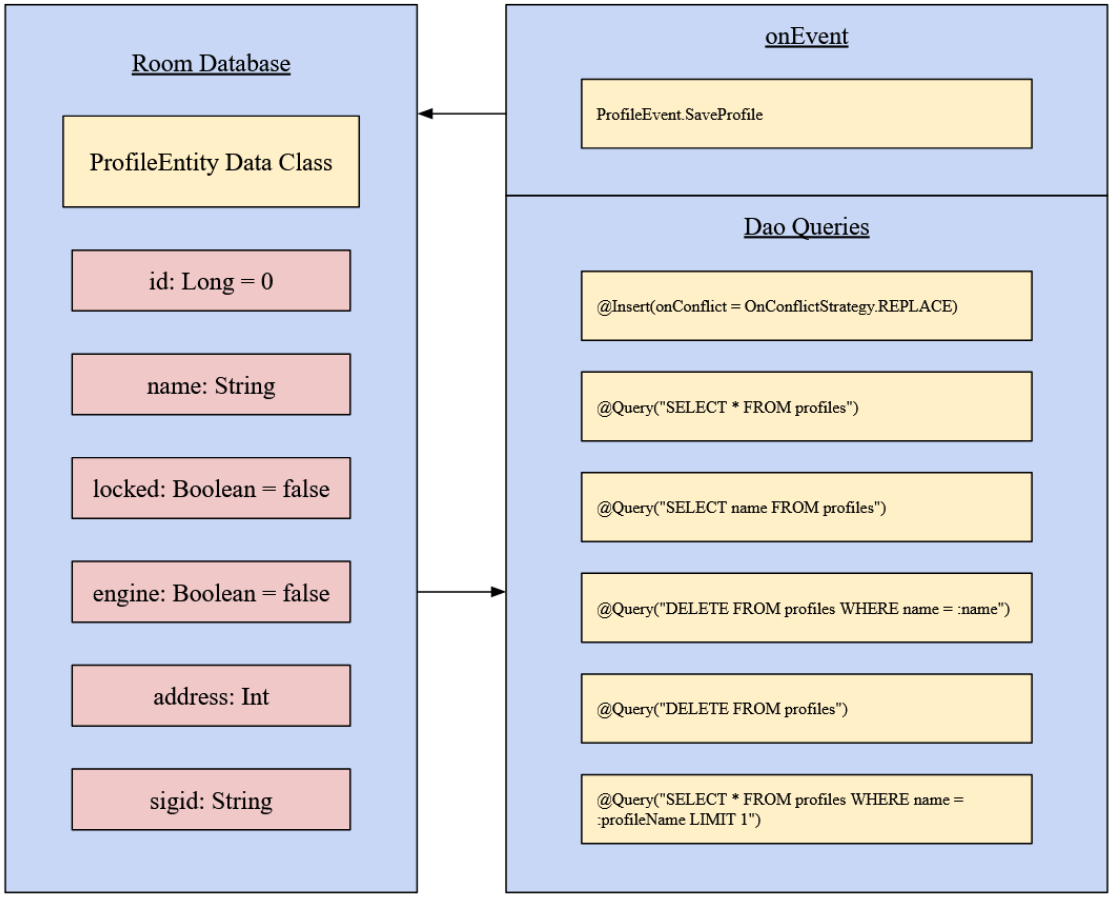


Figure 1: Flow Diagram of Database

2.1.1 Profile Event

See Figure 1:

- Extracts profile details from the current activity state and calls the SaveProfile function. The SaveProfile Function then calls the Insert DAO query to save the profile to the database

2.1.2 DAO Database Queries

See Figure 1:

- Insert or replace a profile in the database
- Query all profiles and observe changes as a Flow
- Query all profile names and observe changes as a Flow
- Delete a profile by its specific name
- Delete all profiles from the database
- Query a profile by its name and return a single result as a ProfileEntity

2.1.3 ProfileEnitity Class

See Figure 1:

- id: Autogenerated consecutive number
- name: Profile Name
- locked: Locked State
- engine: Engine State
- address: Vehicle Bluetooth MAC address
- sigid: Signature Bitmap hash

2.2 Preference Manager

The Preferences Manager streamlines user profile selection in the application. This manager handles the storage and retrieval of user preferences for display in the UI. This allows the activities in the application to seamlessly update in response to user choices. When the user selects a profile from the dropdown menu in the start activity, the selected profile is then loaded from the database into the Preferences Manager.

2.3 Bluetooth Connection Handler

A Bluetooth Controller manages the Bluetooth connections within the application. This Controller establishes, maintains, and handles the communication between devices through Bluetooth by utilizing task functions such as device discovery, pairing, and data transfer.

2.4 UI Layout

The UI is composed of four distinct pages. Users can move between pages using the navigation bar. Users will encounter a start screen upon opening the application. A one-time prompt asking for Bluetooth permissions is then displayed. If the user does not allow

Bluetooth permissions the app will not function until the user reopens the app and allows permissions. The user then progresses to a profile creation page, engages with a functional control screen, and accesses information in the FAQ section.

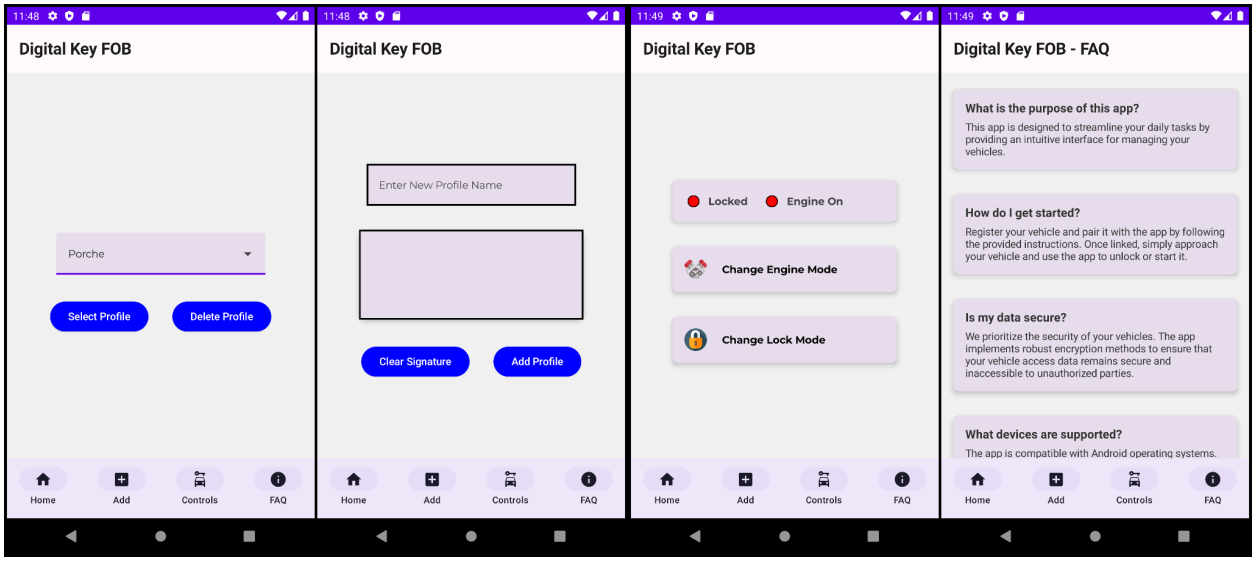


Figure 2: Current UI Iteration

3. Subsystem Validation

3.1 App Functionality

App functionality validation is centered on ensuring the smooth execution of the application on any compatible operating system. This involves confirming that the app runs seamlessly, meeting the basic compatibility requirements of the targeted Android OS. The required minimum OS for this application is Android 10. The following table shows the list of devices the app has been virtually or physically tested on.

Physical Device Tested	Android OS
Samsung Galaxy Tab A8	Android 13
Virtual Device Tested	**Android OS**
Google Pixel 2	Android 10-11
Google Pixel C Tablet	Android 12-14

Table 1: Tested Devices

3.2 App Navigation

User experience is significantly influenced by app navigation. This validation assesses the clarity, intuitiveness, and logical flow of the app's navigation system to enhance user-friendliness.

The original design featured a variety of buttons that led to the respective pages. This design was hampered by the clumsiness of Android XML view layouts which prevented a

more uniform navigation scheme. To improve user experience and fulfill the validation requirements, the original design was replaced with a Jetpack Compose UI which allows for a more straightforward application. This new design features a simple four-button navigation bar for centralized app movement and adds a more modern feel.

3.3 Value Sharing

The Preference Manager stores the currently selected profile for use across different app activities. This value sharing allows the user to move between application pages while retaining the selected profile or vehicle state.

3.4 Application Design

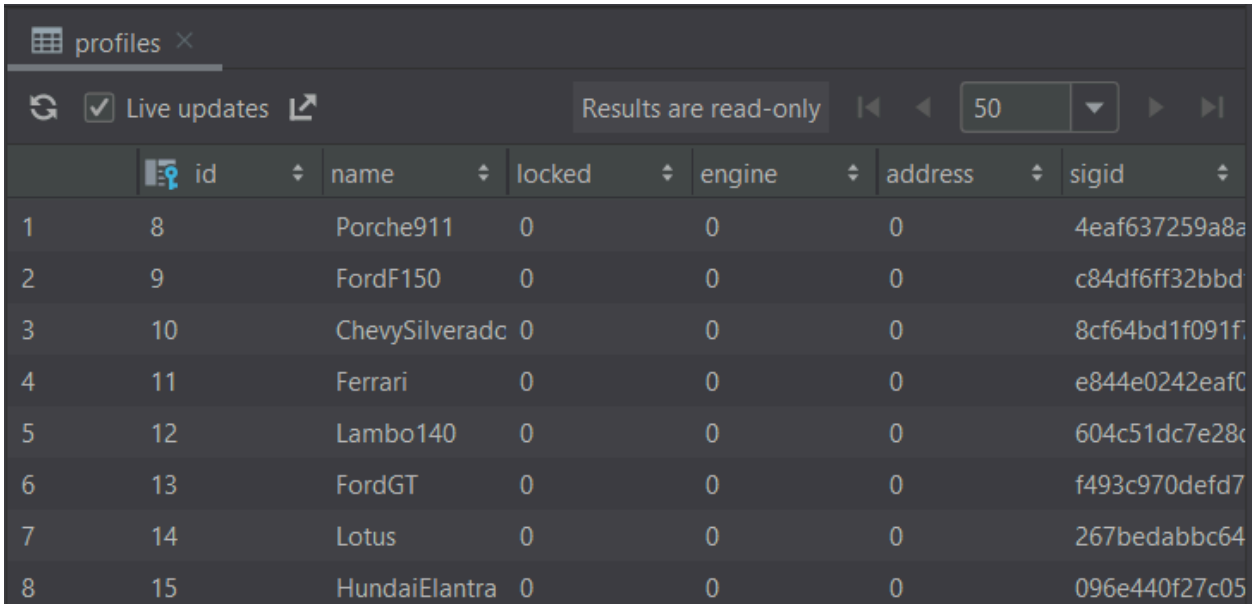
Beyond aesthetics, the application design validation confirms that the visual elements not only look appealing but also maintain a cohesive and recognizable identity. This app was intentionally designed to allow users of all ages to quickly recognize each specific use of the UI layout. Error and Success messages were also designed to alert the user of potential errors or steps they need to take to make use of all available functionality.

Message Cause	Output Text to User
User Attempts to send command with no device connected	No Device Connected
User Attempts to send command with device connected	Message Sent
User Deletes Profile	Profile Deleted: (Selected Profile)
User Attempts to Delete Profile with no Profile Selected	No profile selected for deletion
User Creates Profile	Profile created
User Attempts to Create profile with no signature	Please Sign Signature pad
User Attempts to use a duplicate name or does not enter anything	Please enter a unique and non-duplicate profile name

Table 2: Application User Alerts

3.5 Registration Procedure

The registration procedure for each car profile consists of the user entering a unique username and signing a signature pad. These two events create an onboarding process that is cohesive and user-friendly. Once the user selects “Add Profile”, the application converts the signature data to a secure hash and associates it with the unique profile name. The user can create unlimited profiles within this application.



profiles

Live updates | Results are read-only | 50

	id	name	locked	engine	address	sigid
1	8	Porche911	0	0	0	4eaf637259a8a
2	9	FordF150	0	0	0	c84df6ff32bbd
3	10	ChevySilveradc	0	0	0	8cf64bd1f091f
4	11	Ferrari	0	0	0	e844e0242eaf0
5	12	Lambo140	0	0	0	604c51dc7e28c
6	13	FordGT	0	0	0	f493c970defd7
7	14	Lotus	0	0	0	267bedabbc64
8	15	HundaiElantra	0	0	0	096e440f27c05

Figure 3: Room Database

3.6 Communication Procedure

The communication procedure is centered on the app's conversion of the signature bitmap to a hash and its transfer to the Communication and Cryptography subsystem. This unique hash, tied to each profile, is stored securely in the Room Database.

3.7 FAQ Page

The FAQ page provides clear, comprehensive, and accurate answers to common user queries, contributing to a better understanding of the application's features and functionalities. This page includes a placeholder for company support and contact information that could be populated if the application was used commercially.

Question	**Answer**
What is the purpose of this app?	This app is designed to streamline your daily tasks by providing an intuitive interface for managing your vehicles.
How do I get started?	Register your vehicle and pair it with the app by following the provided instructions. Once linked, simply approach your vehicle and use the app to unlock or start it.
How does the signature pad work?	The app utilizes a unique signature pad feature to generate a random number, which is then used to ensure the uniqueness of the vehicle ID.
Is my data secure?	We prioritize the security of your vehicles. The app implements robust encryption methods to ensure that your vehicle access data remains secure and inaccessible to

	unauthorized parties.
What devices are supported?	The app is compatible with Android operating systems. It's optimized for the latest smartphone models to ensure a reliable and consistent experience.
How can I contact support?	For any assistance, questions, or concerns, you can contact our support team at digitalkeyfob@gmail.com. Our dedicated support staff is ready to assist with any issues you might encounter while using the app.

Figure 4: FAQ Page

3.8 Bluetooth Capability

The Bluetooth capability validation ensures that the application can successfully establish a connection with the user's vehicle. For testing purposes, a Raspberry Pi simulated the vehicle. The image below displays the output of a Python Bluetooth script that opens a socket connection to listen for incoming pairing requests. The script output demonstrates the application's ability to connect to the Raspberry PI, receive and send confirmation data, and close the connection. This validation affirms the functionality of Bluetooth communication within the application.

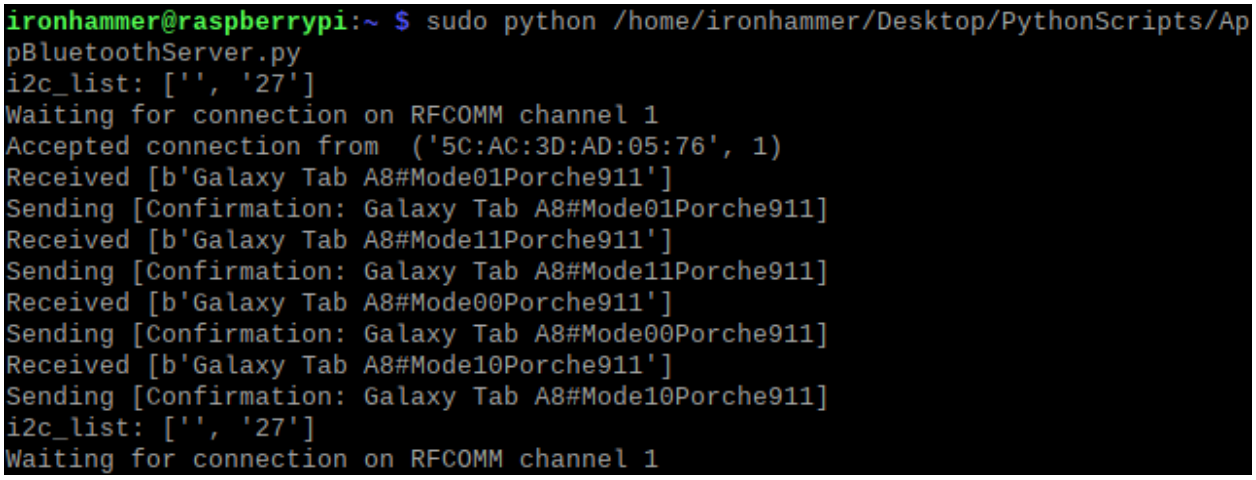


Figure 5: Raspberry Pi Console Output

4. Subsystem Conclusion

The Android Application subsystem functions as expected, successfully meeting the validation points set for the full project. In the second semester, application development focused on expanding and streamlining the Bluetooth functionality to align with the security requirements integral to the overall project. UI updates were also important as certain functions were fine-tuned for a smoother user experience after user testing.

Crypto-analysis Resistant Digital Key FOB

Adriana Matos

LOCKING MECHANISM SUBSYSTEM REPORT

REVISION – 1
December 3, 2023

SUBSYSTEM REPORT
FOR
Crypto-analysis Resistant Digital Key FOB

TEAM <2>

APPROVED BY:

Adriana Matos_____
Project Leader Date

_____
Prof. Kalafatis Date

_____
T/A Date

Change Record

Rev.	Date	Originator	Approvals	Description
1	12/3/2023	Adriana Matos		Submitted Revision
2	04/29/2024	Adriana Matos		Final Revision

Table of Contents

List of Figures

1. Subsystem Introduction

The Locking Mechanism Subsystem is the hardware component of the project that is constructed to mimic an automobile. It will use a microcontroller, a CAN transmitter, and two LED lights to show that the “car” can successfully receive encrypted signals from a digital key fob to lock/unlock the car as well as turn on/off the engine.

2. Subsystem Details

A block diagram of the subsystem is shown below in **Figure 1**.

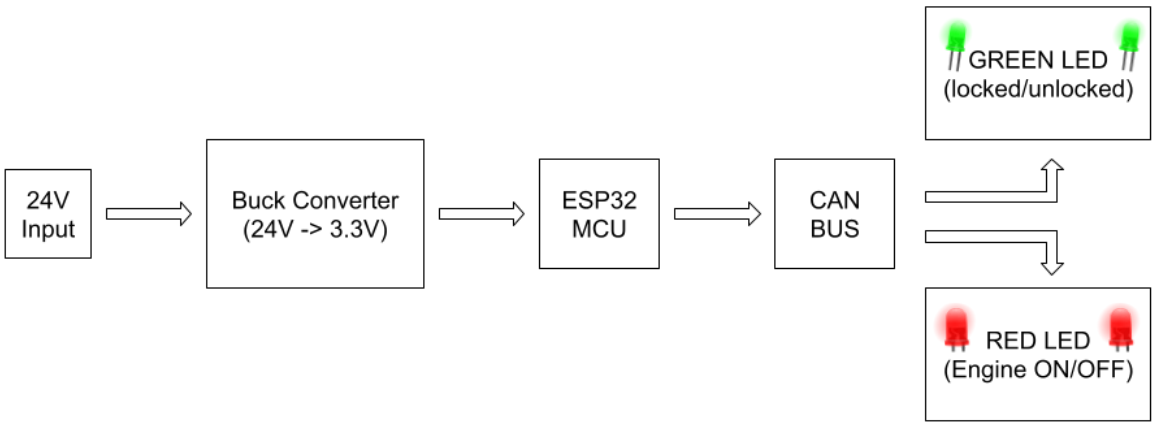


Figure 1: Functional Block Diagram of Locking Mechanism Subsystem

The primary challenge of the Locking Mechanism Subsystem was figuring out how to efficiently design a Printed Circuit Board (PCB) that has all the functionalities needed without any prior knowledge of Altium PCB Design. The PCB needed to contain a microcontroller with Bluetooth and CAN bus functionality and LEDs to demonstrate the mechanism accepted the signal. An ESP32-WROOM microcontroller was chosen since it has the necessary Bluetooth functionality as well as a CAN Controller to operate the CAN bus. An additional CAN Transceiver was necessary to properly use the CAN Controller within the ESP32. A barrel jack connector was chosen to provide the input power for the mechanism, but since the barrel jack connector outputs 24V at 1A, a buck converter was used to step down the voltage from 24V to 3.3V. The outputted 3.3V is what is then used to power the ESP32 as well as the CAN Transceiver. After following the typical application circuits provided within the datasheets of each component, the final schematic design, as shown in **Figure 2**, was created. After the final schematic was done, the PCB layout was created which can be seen in **Figure 3.**

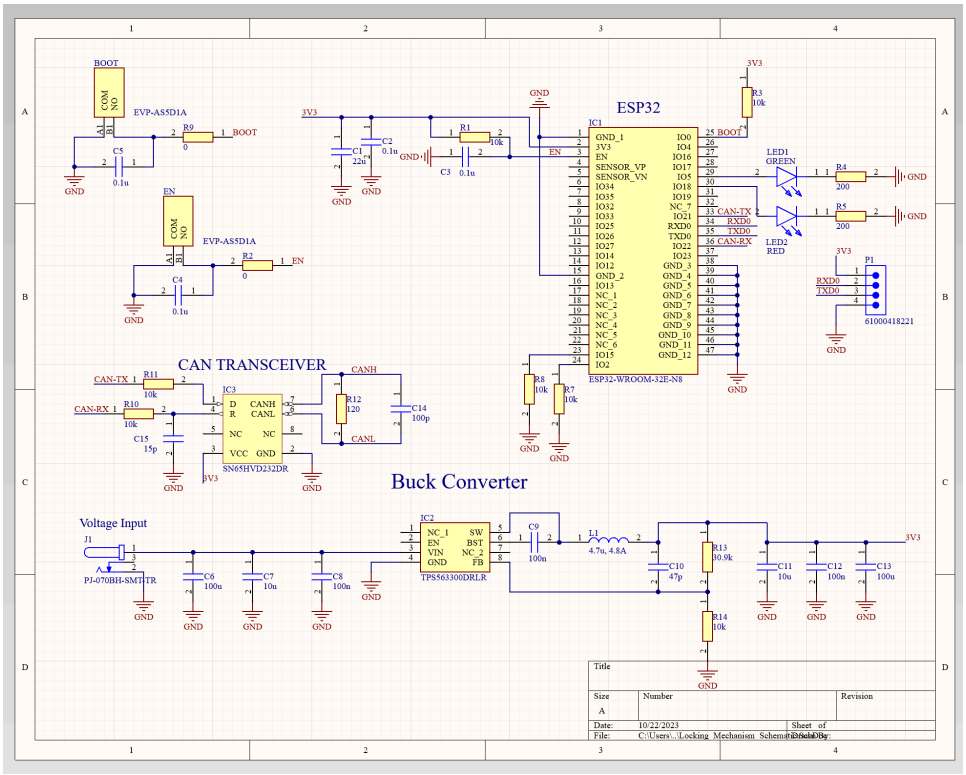


Figure 2: Final Locking Mechanism Schematic Design

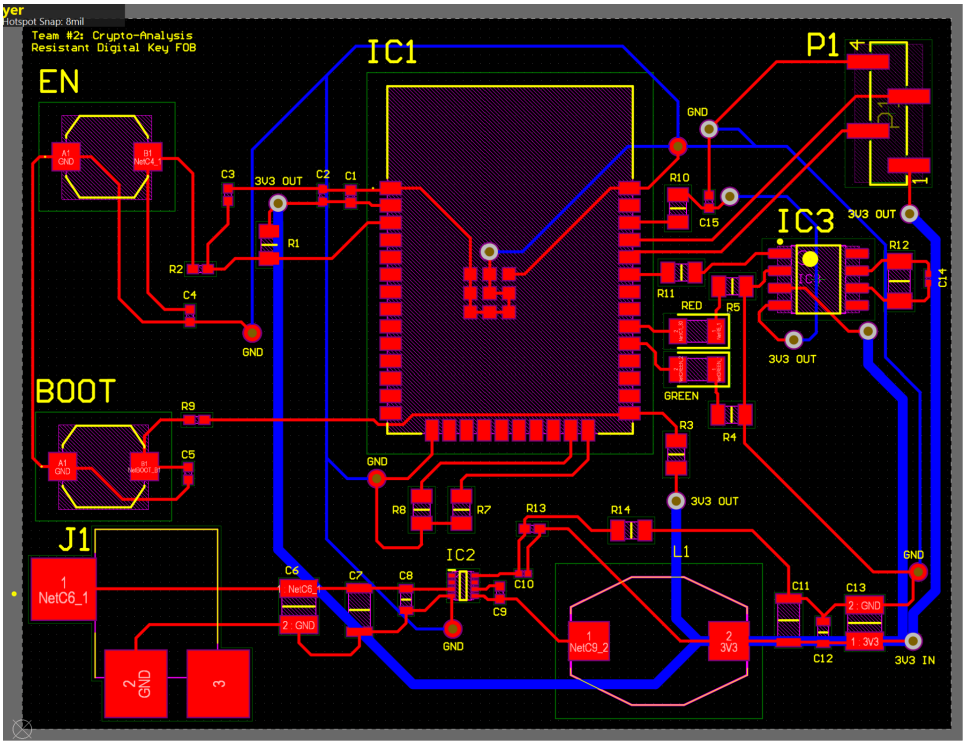


Figure 3: Final Locking Mechanism PCB Design

Another challenge encountered when creating the PCB was underestimating how small some components were and soldering them on properly without causing any shorts or other errors. With soldering practice and a bit of help, the PCB was done and ready for validation testing, as shown in **Figure 4**.

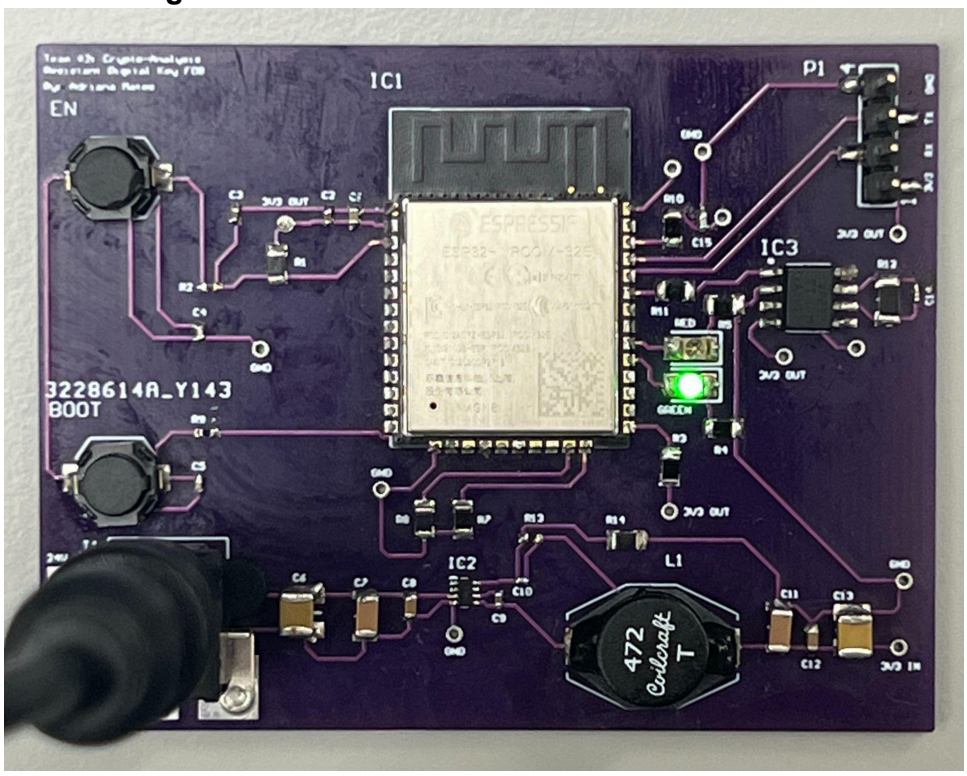


Figure 4: Completed PCB

Once subsystem integration started, ESP-IDF and Visual Studio Code were used to program the PCB. Referencing the Espressif website for ESP-IDF, the Bluetooth Acceptor example code was used as a starting point for the overall code. Since integration of the Communication and Cryptography subsystem was necessary, an external library, namely CryptoPP, needed to be integrated into the “esp-idf” folder on the computer for the library to be compiled at the same time the project compiles. This led to the creation of a “CMakeLists.txt” file for the library and a custom Partition Table to be made to ensure proper compilation of the project as well as carving out the necessary space for the application of the code. The use of Non-Volatile Storage in the ESP32 was used to store the various Z-hardware profiles sent over by the application during the Registration process. The ESP32 then read the data stored when needed for the Communication process and appended this data and some data included in the received signal to a string to be put through the SHA256 code. If the received hash from the application and the created hash from the ESP32 match then the command sent is accepted and processed. If the hashes do not match the ESP32 is programmed to ignore the command sent from the application.

3. Subsystem Validation

The subsystem was meant to be tested once the PCB was fully assembled. The PCB underwent three different validation tests: ensure the complete system works when

connected to power, ensure the microcontroller is able to connect to an Android device through Bluetooth and ensure the red and green LEDs are functioning properly when a specific signal is sent from the Android device. The first test, ensuring the system works when connected to power, was tested once the PCB was fully constructed and before the microcontroller was programmed with the Bluetooth and LED functionalities. As shown in **Figure 4**, when connected to power a green LED is lit to show that the system is working properly and the microcontroller is being supplied with the necessary 3.3V.
Once the system proved that it was working when connected to power, the microcontroller was programmed to properly function with Bluetooth and turn the LEDs on/off. The next two tests were tested together when an Android device was connected through Bluetooth and using a Bluetooth terminal application on the device, signals were sent to turn the LEDs on and off. **Figures 5 & 6** show the LED outputs when a ‘1’ signal is sent to turn on the green LED and a ‘2’ signal is sent to turn on the red LED. **Figures 5 & 6** also show the terminal screen showing the returned message when the signal was received.

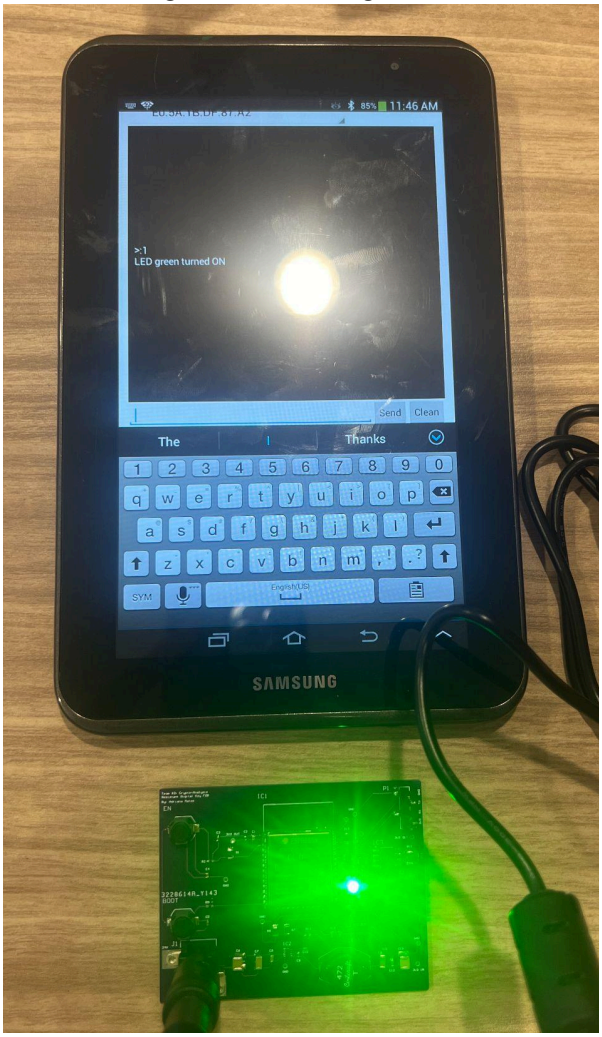


Figure 5: Green LED Output

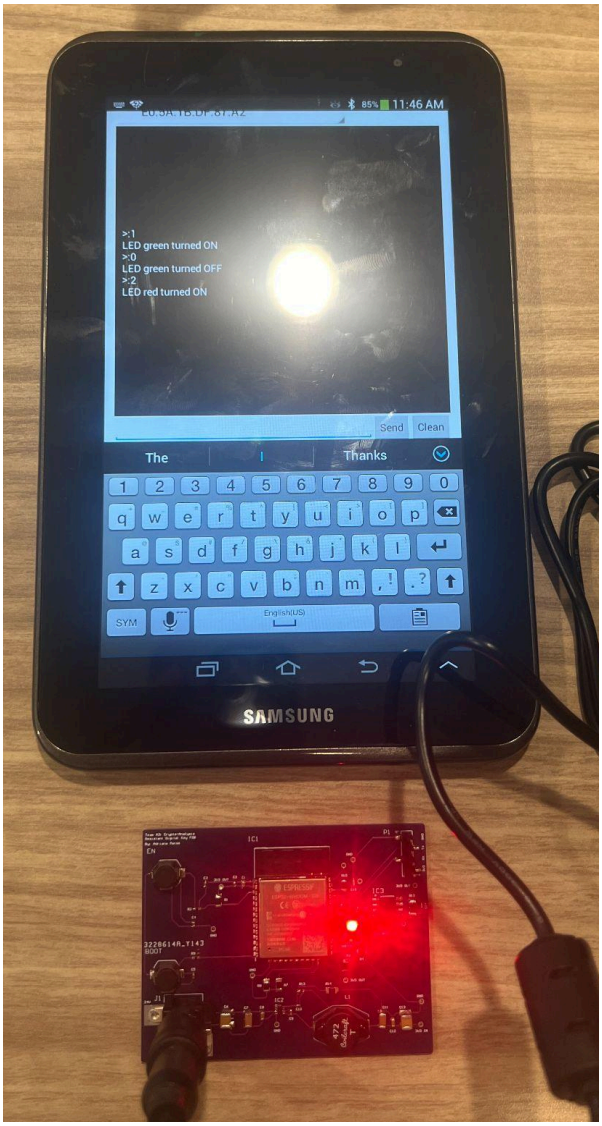


Figure 6: Red LED Output

Ensuring Bluetooth connection between the PCB and application was validated by monitoring the output of the terminal connected to the PCB. The Bluetooth Acceptor example code included error checking that would output on the terminal when a connection was made, when a connection was interrupted and disconnected the PCB and application, or when a connection could not be made. To validate if the cryptography code and CryptoPP library integration were successful, print statements were added to the code to print every step onto the monitor. This ensured that the code was working as expected, but also proved helpful when decoding was necessary if something did not work properly. Validation was also done by physically viewing the function of the LEDs when a command was sent through. Since the LED function was tested with a Bluetooth terminal application downloaded from the Google Play store, if they did not turn on when a command was sent then there was a problem with the code programmed on the ESP32 and not with the soldering of the PCB. Various profiles were created to test the Non-Volatile Storage of the ESP32, as well as testing to see if the ESP32 could properly read/write to the Non-Volatile Storage.

4. Subsystem Conclusion

The subsystem proved to work as expected. The only test left is to ensure the CAN bus functionality of the ESP32 can properly work with the CAN Transceiver and the LEDs to mimic the internal communication network of an automobile. For demonstration purposes, a simple Bluetooth program was created to show that the Bluetooth was able to function properly, but as the subsystems begin to be interfaced, a new code will be written to work with the Application Subsystem. The encryption/decryption code from the Communication and Cryptography Subsystem will also be added to provide secure communication between the device and the “vehicle”. When interfaced with the other subsystems, it will be able to communicate with the Digital Key FOB Application through a Bluetooth connection and receive encrypted signals to turn the LEDs on and off as the signals are sent from the application.

After creating the Bluetooth connection code using ESP-IDF and Visual Studio Code and integrating it with the other two subsystems, the system proved to work as expected. Unfortunately, due to time constraints, the implementation of the Relay Attack prevention code was not possible. Since the ESP32 does not hold its own time, it needed to be coded to connect to a server to obtain the Unix time needed for the Relay Attack prevention code. This meant that along with Bluetooth code, the ESP32 also needed Wi-fi connection code; this proved to be difficult to implement due to the already implemented Bluetooth code as well as compilation issues. With the upcoming final demo and the Relay Attack prevention code not being a part of the starting project, it was decided that it would not be implemented and the remaining time would be used for project testing and validation. After the integration was finalized, the subsystem proved to work well with the entire project with minimal to no errors if any.

Crypto-analysis Resistant Digital Key FOB

Courtney Eaton

COMMUNICATION AND CRYPTOGRAPHY SUBSYSTEM REPORT

REVISION – 2
April 30, 2024

SUBSYSTEM REPORT
FOR
Crypto-analysis Resistant Digital Key FOB

TEAM <2>

APPROVED BY:

_____
Project Leader Date

_____
Prof. Kalafatis Date

_____
T/A Date

Change Record

Rev.	Date	Originator	Approvals	Description
1	12/3/2023	Courtney Eaton		Submitted Revision
2	4/30/2024	Courtney Eaton		Capstone Completed Project Report

Table of Contents

List of Figures

1. Subsystem Summary

The Communication and Cryptography subsystem is responsible for the cryptography algorithms and protocols that make up the Registration and Communication Processes of the application, as shown in figure 1 below. For the Registration process, which registers a user's vehicle with the application, the Communication and Cryptography subsystem is responsible for creating the Z-Hardware Profile and coding the necessary encryption and decryption with the use of AES-256 to send the Z-Hardware Profile from the device to the vehicle. For the Communication process, when the user enters a command (lock/unlock or start/stop engine), the Communication and Cryptography subsystem is responsible for the creation of the Dynamic Digital Signature (DDS, a unique combination of unix-time and the Z-Hardware profile) with the use of SHA-256 and data from the Android Application. It is also responsible for verifying the vehicle and device DDS matches. All of these components have been coded, debugged, and validated to be in working order.

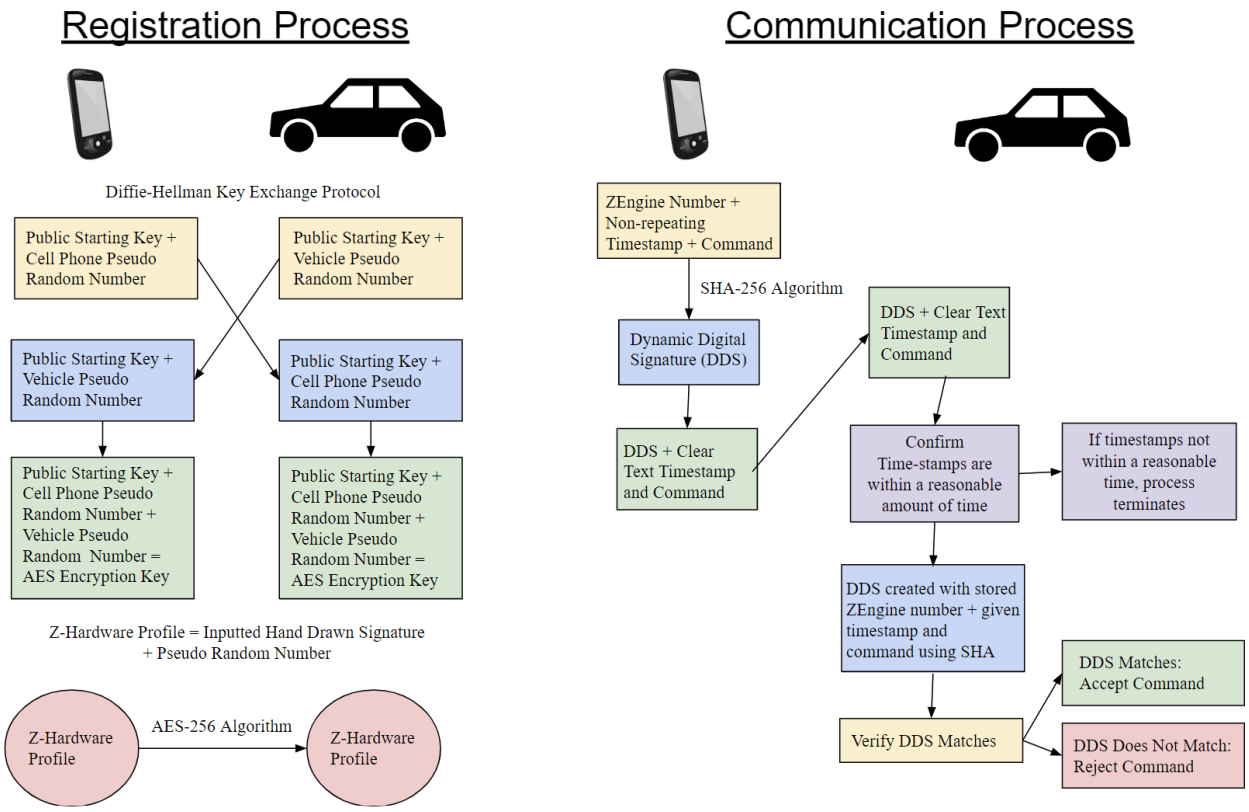


Figure 1: Subsystem Block Diagram

2. Diffie-Hellman Key Exchange (DH)

2.1 Background

The Diffie-Hellman Key Exchange is a protocol developed to effectively set up a shared key between two devices without anyone who overhears the exchange being able to obtain the key. The process is shown in figure 2 using colors to represent the numbers in the algorithm. Essentially, DH is done by first the two devices agreeing on two unique, predetermined numbers which make up the public key (common paint). This first number is a large prime number (p) and the second is one of its primitive primes (known as the generator, g). Both devices then create their own secret pseudo-random numbers, “a” for the first device (Alice) and “b” for the second (Bob). Next, the public key is combined through the use of the power and modulus functions with “a” and “b” to create “A” (light orange) and “B” (light blue), respectively. The two devices then share their “A” and “B” values with each other. Finally, using the power and modulus functions and their original “a” and “b” secret numbers, the same secret key is determined for both devices. For our project, DH is used to set up the symmetric key for the AES-256 algorithm. Device 1 (A) was the Android application and its code was done in Kotlin while device 2 (B) was the vehicle and its code was done in C++.

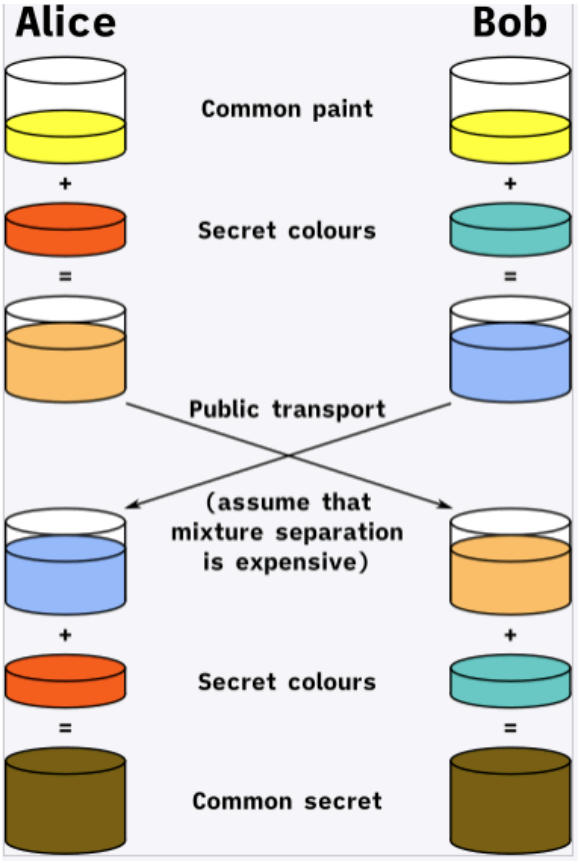


Figure 2: Diffie-Hellman Key Exchange Concept [1]

2.2 Validation Results

The results of the DH validation tests can be seen below. It is important to note that these tests are done with much smaller numbers than what will eventually be needed. Although many people were consulted and many hours were spent trying to find a way in which would allow for larger numbers, it was unable to be achieved. This problem is due to the fact that the largest number C++ can work with is a long long int data type which is only 64-bits but it is wanted for the final secret key to be 124-bits.

Regardless, DH is completed and proof-of-concept can be seen below in figures 3 and 4 where there are two different instances of DH being executed. Both instances show the outputs from the device in the dark gray boxes with a red outline where “P” and “G” are the public key numbers, “A val” is the pseudo-random number “a”, and “Vehicle DHKE Portion” is the final sent number “A”. The code in the light gray box with the blue outline is from the vehicle where “B value to Device” is the final sent number “B”. As can be seen both tests verify that the device and vehicle are able to obtain the same secret key (sk) value.

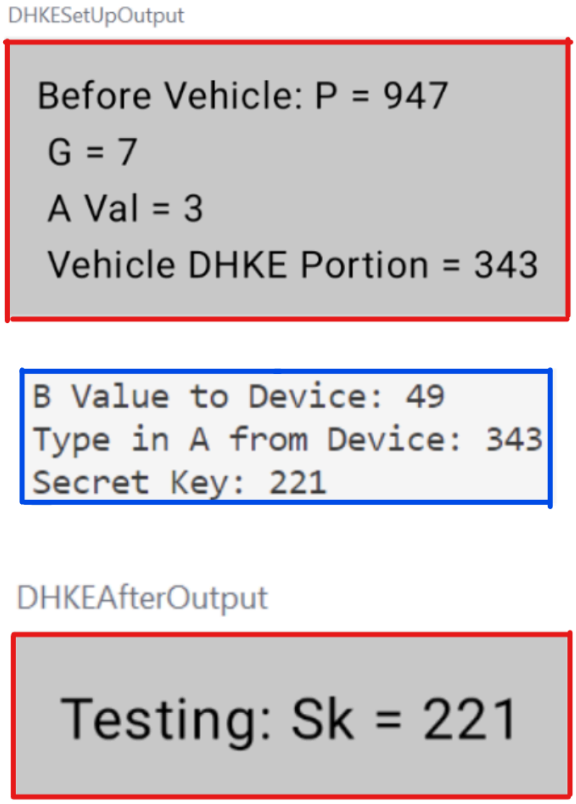


Figure 3: DH Test #1

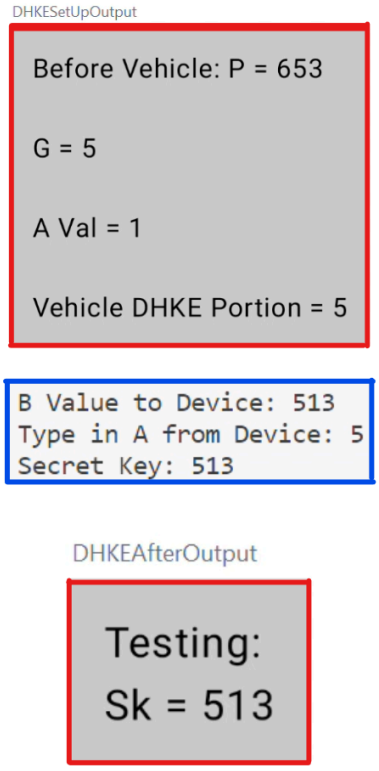


Figure 4: DH Test #2

2.3 Learnings from Integration

The issues regarding expanding DH to larger numbers was something that was never able to be securely remedied. Although functions could be created to do the required math through strings in C++, these functions would decrease the overall security of the protocol as any error in the code, no matter how small, would greatly hinder the security of the entire protocol. Since there was no cryptography export working on the project to ensure that there were no breaches of security from these created functions, it was too large of a risk to incorporate. Instead, as the one of the main security advantages of DH comes from the variation in the secret keys that are generated, it was decided to decrease the values of P and G to 13 and 7, respectively so that the amount of possible secret keys could be increased from 36 (where it was in December 2023) to 256 possible keys.

During the research done to investigate the best way to increase the security of DH, it was found that this protocol is not the most efficient or secure method to set up the encryption key for the AES-256 algorithm. Instead, Rivest-Shamir-Adleman (RSA) would be a more effective algorithm to use for the encryption key. RSA is an asymmetric encryption/decryption algorithm which uses public and private keys to encrypt and decrypt information. RSA works by using mathematics so that the receiver of the information (in this case, the vehicle) would have its own previously set up private and public keys. The public key would then be used by the sender of the information (in this case, the device) to encrypt the data and send it to the receiver (vehicle) where it would then be decrypted using the private key. Incorporating this algorithm into the Registration Process as a whole, the setting up of the Z-Hardware profile would be no different, however when it comes time to send the profile to the vehicle, AES-256 would encrypt the Z-Hardware profile while RSA would be used to encrypt the AES-256 encryption key. The encrypted AES-256 key and Z-Hardware profile would then be sent over to the vehicle where RSA would be used to decrypt the AES-256 encryption key. Once this key is encrypted, it would then be used for AES-256 to decrypt the Z-Hardware profile before it is stored by the vehicle. Unfortunately, due to time constraints for this project, RSA was not implemented. As the DH key exchange is still a secure method (even if not the most secure) to set up the encryption key for AES-256 it was used for the final product.

It was also found during the integration stage of the project that quantum computers have the capability to crack DH. Fortunately, quantum computers are only being utilized currently in academia and they are not powerful enough to be able to crack DH, just yet. However, NIST has recently released new algorithms to help combat this issue. With these algorithms being so new and not fully explored by the cryptographic community into their security and reliability as well as quantum computing not being a large threat right now, it was decided to continue with DH for this project. In the future, however, as quantum computing grows, these new algorithms would need to be utilized.

3. Advanced Encryption Standard, 256 bits (AES-256)

3.1 Background

The Advanced Encryption Standard is a Federal Information Processing Standards approved encryption and decryption algorithm chosen by the National Institute of Standards and Technology (NIST) to replace the Data Encryption Standard (DES). This algorithm is referred to as either the AES algorithm (which can also include the key length as in the case of AES-256) or the Rijndael algorithm after its creators. Even though the algorithm standard came out in 2001, it is still considered to be the most secure algorithm today. It has been approved by the National Security Agency (NSA) for use in the US government and has become an industry standard for encrypting information. The 256 bit key length is considered to be the most secure key length making the algorithm virtually uncrackable [2]. For the purpose of this project, AES-256 is used to encrypt the Z-Hardware profile before it is sent over to the vehicle via bluetooth then decrypt it once it gets there. This allows for safe transfer of the Z-Hardware profile so it can be stored by both the device and vehicle and used to confirm the commands are coming from the right device without needing to transfer it ever again.

3.2 Validation Results

Below shows the results from the validation tests for the AES-256 algorithm. Figures 5 and 6 show the algorithm output for the device done in Kotlin. For the final project only the encryption will be needed for the device but for the purposes of the final demonstration both the encryption and decryption were implemented to show that the code did correctly encrypt the inputted data. There are two tests shown below with two different inputs that are both able to produce the correct output.

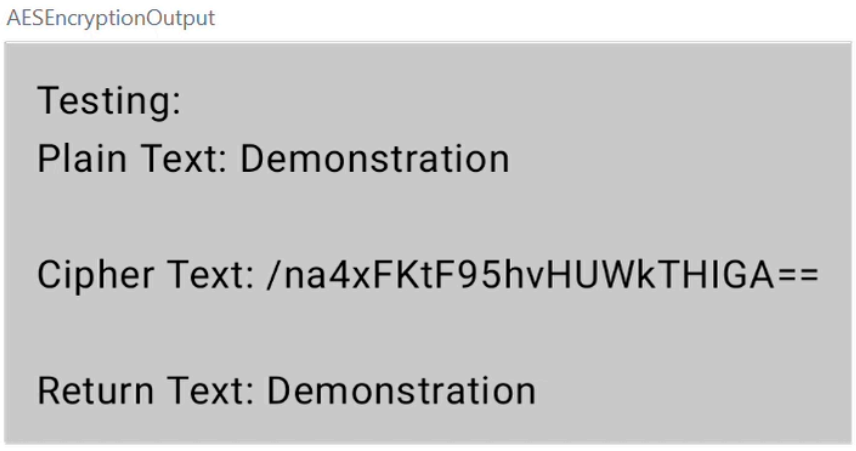


Figure 5: AES-256 Device Test #1

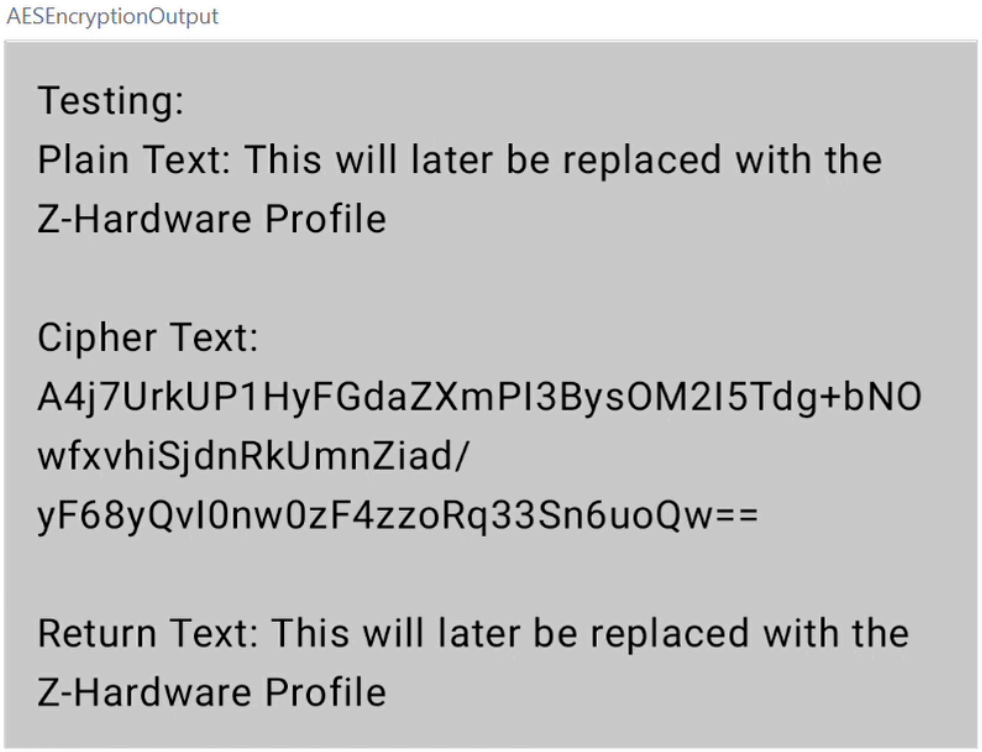


Figure 6: AES-256 Device Test #2

Figures 7 and 8 show the algorithm output for the vehicle done in C++. For the final project, only the decryption will be needed for the vehicle but the encryption was implemented as well to show that it is able to accurately decrypt the encrypted data.

```
AES-256
Starting Phrase text: Demonstration
Key: 4E8608EB89ED7E792D813E25BBE26F07
IV: EC8E0B42F2D40141B11FE5A6D4562D89
Cipher Text: B0A229FC0A96069559760D8326A4CB20
Plain Text: Demonstration
```

Figure 7: AES-256 Vehicle Test #1

```
AES-256
Starting Phrase text: This will be replaced with the ZHardware Profile
Key: F2FE838C49E93FC6639FCD00E9BC8AA9
IV: BA4A1F07F56172C78AB2D2C856EF24C2
Cipher Text: 9E0C123CECCB5CEE2BAA46BA0DAD10ABC4D1D2ED3C8D8082725DAADE466
6B89416C4559707C174D2A3F88C429C8A6BC6E254C0B4D2940BD31B8CF61449FC9BC3
Plain Text: This will be replaced with the ZHardware Profile
```

Figure 8: AES-256 Vehicle Test #2

3.3 Learnings from Integration

During the integration of the C++ and Kotlin versions of the AES-256 algorithms with each other and the other subsystems, an issue arose in being able to get the C++ and Kotlin versions of the algorithm to work together. The final design required for the Z-Hardware profile to be encrypted by the device which operates in Kotlin and then be decrypted by the vehicle which operates in C++. Thus, two versions of the AES-256 algorithm were coded during the subsystem creation (ECEN 403). However, when the two versions were integrated it was discovered that the two libraries used (one in C++ and one in Kotlin) take in the needed parameters (encryption key, initialization vector, and text to be encrypted) for the AES-256 algorithm differently, thus making each of them have two different cipher texts and incompatible with each other.

Many different solutions were tested to fix these issues including adjusting the padding, encoding, and variable types of the input parameters. However, when all of these solutions were unsuccessful, the sponsor of the project was reached out to for assistance. A meeting was held with the main software coding lead at Sandia National Laboratories and some tips were given to possibly fix this issue, but even Sandia did not have a solution for this issue. Because of this, the inclusion of AES-256 into the final product became out of the capabilities of the project team and was taken out. The final registration process was adjusted to start with the creation of the Z-Hardware profile followed by the Diffie-Hellman Key Exchange (as it was already integrated into the final product) then the Z-Hardware profile is sent unencrypted via bluetooth from the device to the vehicle where it is then stored.

4. Secure Hash Algorithm, 256 bits (SHA-256)

4.1 Background

The Secure Hash Algorithm (256 bits) is a hashing algorithm with a final digest/output size of 256 bits. It was created by the NIST and NSA to replace SHA 1 which was becoming susceptible to brute force attacks and SHA-256 is one of the most widely used hashing algorithms [3]. Hashing is when a plain text is scrambled to the point that it is impossible to tell what the original text was. This scrambled text is called the cipher text or message digest. This hashing process is irreversible however if the input is not changed the output will remain the same regardless of how many times the algorithm is run. For our project SHA-256 is going to be used to validate that the command is coming from a previously registered device. To complete this, the Z-Hardware profile, timestamp, and command will be run through the SHA-256 to create the Dynamic Digital Signature (DDS). This signature will be created separately on the device and vehicle and the device's DDS will be sent to the vehicle via bluetooth to compare it with the one the vehicle made. If both DDSs match then the command will be accepted. If they do not match then the command must have come from an unauthorized source and will be ignored.

4.2 Validation Results

Below shows validation test results for the SHA-256. Figures 9 and 10 show two tests for the device code (done in Kotlin) with both tests being able to produce a digest.

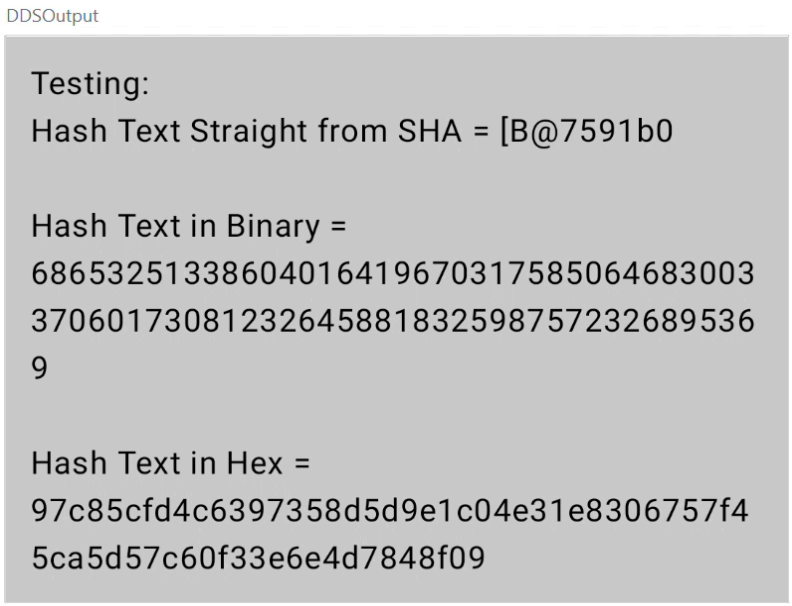


Figure 9: SHA-256 Device Test #1

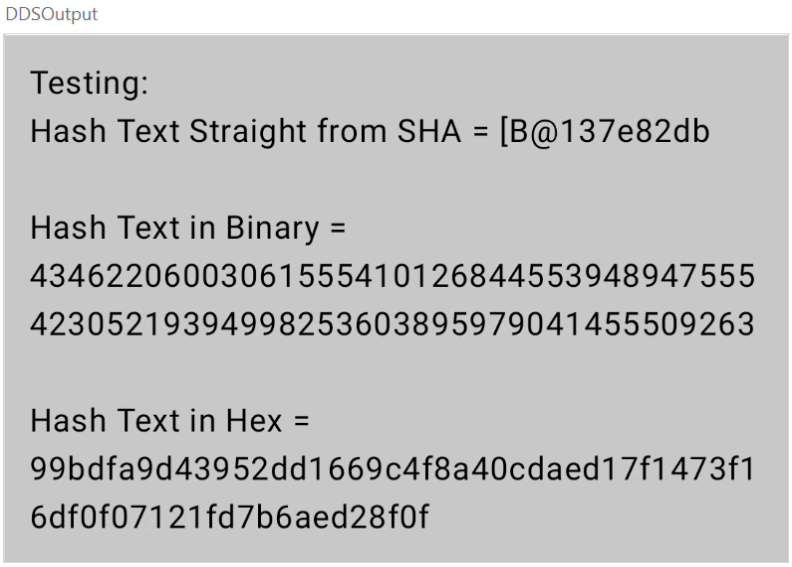


Figure 10: SHA-256 Device Test #2

Figures 11 and 12 show the two test results for the vehicle SHA-256 done in C++. Both of these tests were also able to produce an output.

Figure 11: SHA-256 Vehicle Test #1

```
SHA-256
Starting Text: This will be replaced with the Dynamic Digital Signature
and Command
Final Dynamic Digital Signature Text: B1B97B9024D4E000556BD4FD94D38D93FC
2DF3C2FC2096E5FAD80918332F74D3
```

Figure 12: SHA-256 Vehicle Test #2

5. Miscellaneous

5.1 Z-Hardware Profile Creation

The device (using Kotlin) creates the Z-Hardware Profile through taking the bitmap of the hand drawn signature from the Android Application and combining it with a pseudo-random number to create the final Z-Hardware profile. Once created, this profile will be securely stored by the device as well as being sent to and stored by the vehicle. Figures 13 and 14 below show the validation test results for the creation process.

ZHardwareOutput

```
Testing:
Bit Map in BigInteger:
968977336104434177578798380765397883776568712953149781601174439937990222530119

Pseudo-Random Number:
105779215109418528651849066348596109187094951298770465983529887802125742132271

Z-Hardware Profile:
289034540135660684804924957913469974955608553305823377116258683352922215464844
```

Figure 13: Z-Hardware Creation Test #1

ZHardwareOutput

```
Testing:
Bit Map in BigInteger:
791391444463615533723635075341647087555115571128583459233990180554022798345887

Pseudo-Random Number:
9331059757197326687187532707976316868886591197791512396664266893573741699865 2

Z-Hardware Profile:
437555682560873677646671099054560529362559440137614312790738430475284043072 39
```

Figure 14: Z-Hardware Creation Test #2

5.2 Dynamic Digital Signature Verification

The vehicle (using C++) verifies that the Dynamic Digital Signatures (DDS) match by taking in the data from the device and comparing it to the vehicle. The data from the device will consist of the first 64 hex digits (equating the 256-bit output of SHA-256) being the DDS and the rest of the data will contain the timestamp and command. Thus, this verification code will extract the device's DDS from the data before comparing it. For demonstration purposes, the code outputs a short message stating whether or not they match but when integrated it will pass along the command if it does match and do nothing if it does not. Figures 15 and 16 below show the verification test results with the programmed vehicle DDS for demonstration purposes being "C0mmun1c8t10nAndCr7ptographySubsystemTeam2CapstoneProjectECEN403".

```
Enter DDS and Command from Vehicle: C0mmun1c8t10nAndCr7ptographySubsystemTeam2CapstoneProjectECEN403ExtraSuffToBeCutOff
DDSs match: Send Command to Vehicle.
```

Figure 15: Dynamic Digital Signature Verification Test #1

```
Enter DDS and Command from Vehicle: ThisWillNotMatch
DDSs do not match: No further action.
```

Figure 16: Dynamic Digital Signature Verification Test #2

5.3 Relay Attack Prevention Code

Current key FOBs send a signal to the car continuously letting the car know that it is nearby and when the Key FOB is close enough and the lock or unlock button is pressed on the car

door the car opens. Recently, thieves have used a loophole in this process and will place a relay device close to a key FOB capturing the signal being sent to the car telling it to unlock and then they will play this signal back when near the car causing it to unlock. With the original scope of the project, the Crypto-analysis Resistant Digital key FOB is also susceptible to this relay attack. Roughly three weeks before the final project (ECEN 404) demonstration it was proposed by the sponsor to include code in which would prevent these relay attacks. This code would work by using the unix-time captured by the device for the DDS and compare it to the unix-time captured by the vehicle. If the unix-time of the device (captured when the DDS is sent) and the unix-time of the vehicle (captured when the DDS is received) is within two (2) seconds then the rest of the Communication process would continue with the DDS verification. If the unix-times are not within two seconds, then the Communication process would discontinue and no command to the vehicle would be accepted. Thus, if a signal to the vehicle was captured during the Communication process and relayed later, then this relay attack prevention code would prevent access to the vehicle as the unix-times would be more than two seconds apart.

The ESP32 chip that is used in the microcontroller hardware which acts as the vehicle does not keep its own time. Thus, in order for the vehicle hardware to be able to capture its own unix-time the ESP32 chip would have to connect via wifi to a server controlled by the NIST which keeps track of the unix-time. It would then take the unix-time from this server and use that to compare with the unix-time sent by the device. However, when it was attempted to connect the ESP32 chip to wifi, the code clashed with the code that was already implemented for the bluetooth connection. With this issue arising so close to the final project (ECEN 404) demonstration, it was decided that there was not enough time for the relay attack prevention code to be implemented into the project.

6. Conclusion

Overall, the Communication and Cryptography subsystem was able to operate as required. All of the tests done on the various algorithms and protocols resulted in correct outputs. During the integration phase of our project, all of the algorithms and protocols were connected to the Android Application and Locking Mechanism subsystems so that they can work with each other and the other subsystems to send data between the device and hardware. Overall, the Communication and Cryptography subsystem is able to fully support all of the backend processes and algorithms to allow for secure communication between the Android Application and Locking Mechanism.

7. References

1. Cobb, Michael. “What Is the RSA Algorithm? Definition from Searchsecurity.” *Security*, TechTarget, 4 Nov. 2021, www.techtarget.com/searchsecurity/definition/RSA.
2. “Diffie–Hellman Key Exchange.” *Wikipedia*, Wikimedia Foundation, en.wikipedia.org/wiki/Diffie%E2%80%93Hellman_key_exchange.
3. “Everything You Need to Know about AES-256 Encryption.” *Kiteworks*, www.kiteworks.com/risk-compliance-glossary/aes-256-encryption/.
4. Jena, Baivab Kumar. “What Is SHA-256 Algorithm: How It Works and Applications: Simplilearn.” *Simplilearn.Com*, Simplilearn, 29 Aug. 2023, www.simplilearn.com/tutorials/cyber-security-tutorial/sha-256-algorithm.

Crypto-analysis Resistant Digital Key FOB

Courtney Eaton
Adriana Matos
Jeremy Hein

FULL SYSTEM REPORT

REVISION – 1
April 30, 2024

FULL SYSTEM REPORT
FOR
Crypto-analysis Resistant Digital Key FOB

TEAM <2>

APPROVED BY:

_____
Adriana Matos Date

_____
Prof. Kalafatis Date

Zhewen Hu
_____
T/A Date

Change Record

Rev.	Date	Originator	Approvals	Description
1	4/30/2024	Courtney Eaton		Capstone Completed Project Report

Table of Contents

List of Figures

1. Overview

Sandia National Laboratories sponsored this project to showcase their new ZTa encryption scheme. The selected mode for the proof of concept build was a Digital Key FOB utilizing a Bluetooth connection for relaying the encrypted messages. The project was separated into three different subsystems: Application development, Cryptography, and PCB/vehicle design. These three subsystems each play a role in demonstrating the ZTa technology. The application acts as the Digital Key FOB responsible for generating the Z-hardware profile and initiating communication with the PCB. The PCB design focuses on creating a device that can act as a vehicle when receiving signals from the Digital Key FOB application. This culminates in the cryptography subsystem that acts as an intermediary between the app and the PCB by writing cryptography code that allows both the app and PCB to communicate securely.

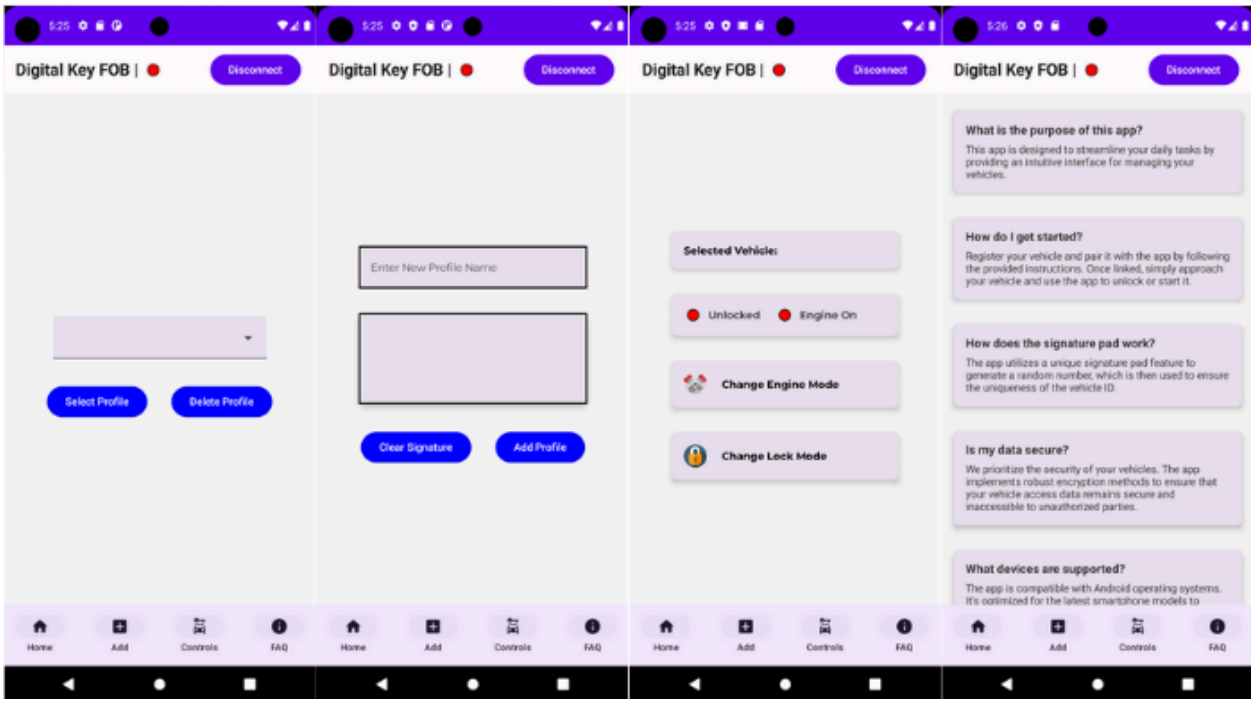


Figure 1: UI for application design

The application was designed using Android Studio as it is more compatible with Bluetooth applications and generally has better development support. The application also utilizes Jetpack Compose for its UI design. The PCB was designed with Altium software and contains an ESP32 chip for handling Bluetooth communication and holding enough storage for the necessary cryptography libraries. The Cryptography subsystem focused on creating sections of code that could be easily integrated into both ends of the communication to create the secure communication mentioned above.

2. Execution Plan

The original design focused on the Registration and Communication processes which utilize the Diffie-Hellman Key exchange, the AES cryptography library, and SHA hashing. However, problems with AES communication between the different coding languages resulted in a few changes to the original design.

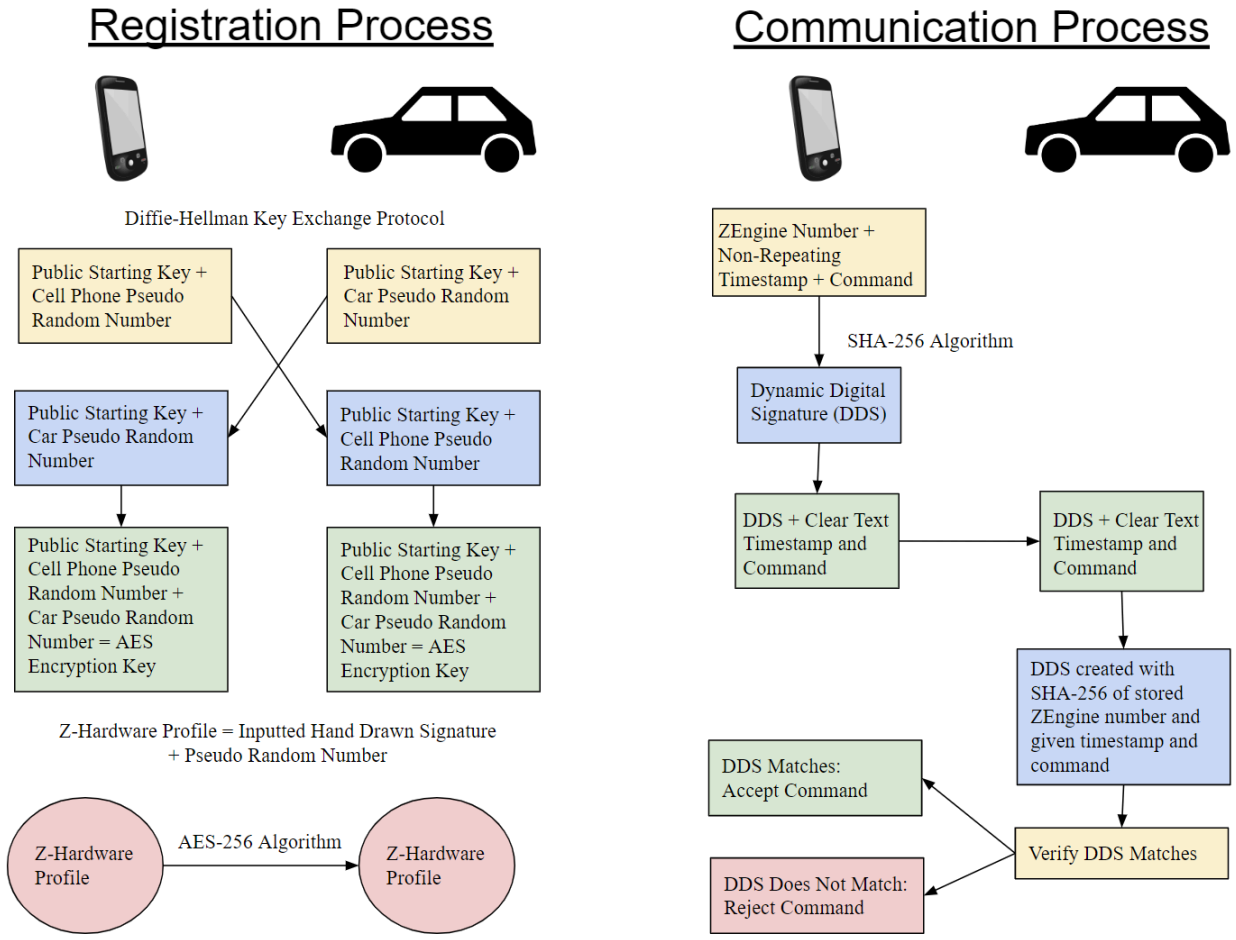


Figure 2: Original system design

AES was replaced with a plaintext exchange of the Z-Hardware profile key and while the Diffie-Hellman Key exchange is still a part of the process, the secret key exchanged by DHKE is not used when sending the Z-Hardware profile as originally designed. The final design also features an added unix-time verification step in order to protect against Relay Attacks. This was added late into the second semester which limited the amount of integration possible before the final demo. The unix-time is sent over by the application and is a part of the hashing process, but the unix-time is not verified by the PCB acting as the vehicle due to limitations with the Wi-fi capabilities of the device when also working with Bluetooth.

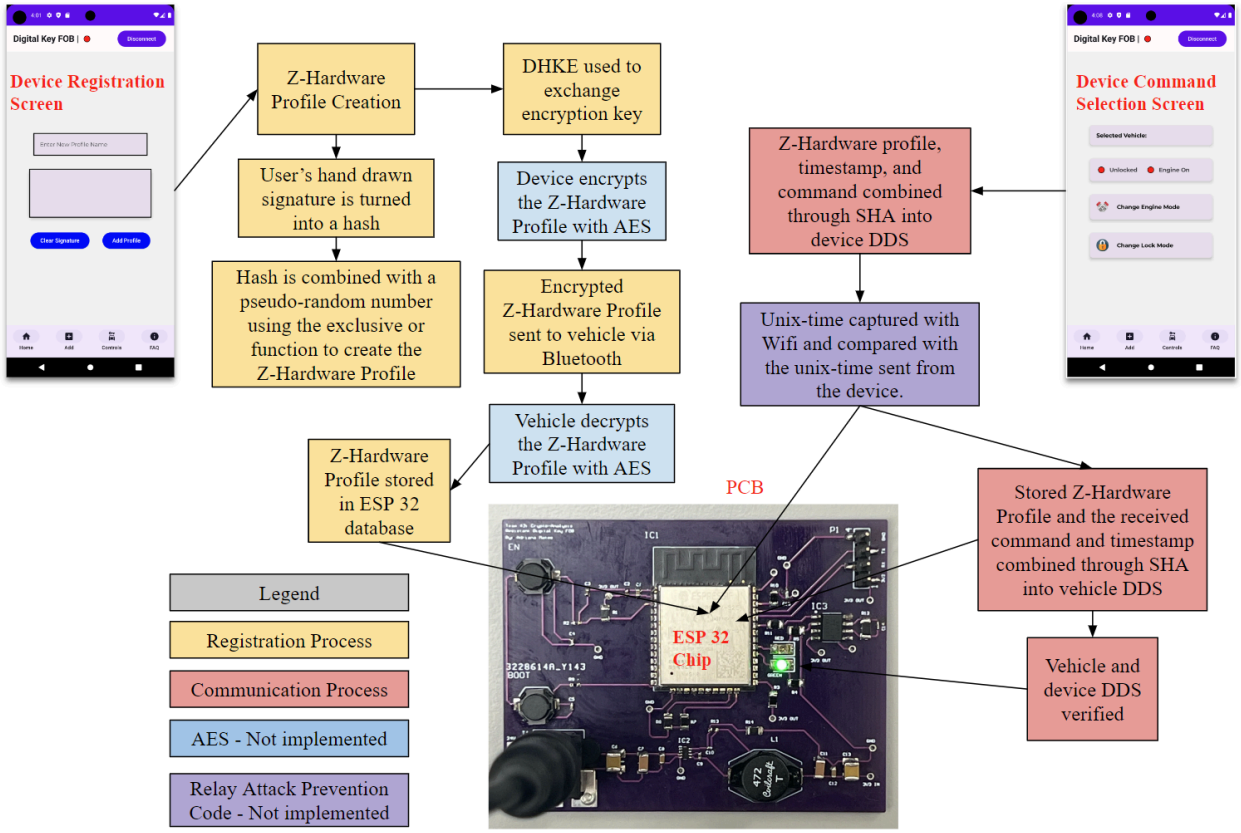


Figure 3: Final system design

3. Validation Plan

The two main components of system validation are the Registration Process and the Communication Process. An important facet of this validation is the way both the Android application and the PCB handle potential errors when either Bluetooth process is running. The Android application uses multiple types of error messages and error handling to alert the user of connection issues or process failures when communicating or registering. These messages are relayed to the user using toast messages.

Type of Message	Cause	Alert Text
Error	Registration Process incomplete/fails	Registration Failed
Success	Registration Process successful	Registration Successful
Error	The Communication process connection ends unexpectedly	Connection was interrupted
Error	The Communication process	Connection timed out

	times out	
Success	The Communication process successfully connects	You're connected!

Figure 4: Alert Text Table

The next part of validation is ensuring the Z-hardware profile is created on the Android application and stored within the database of the device. **Figure 5** shows the names of the vehicle profiles and the corresponding MAC address and Z-hardware profile. The Z-hardware profile is under the “sigid” table identifier.

	id	name	locked	engine	address	sigid
1	1	Howdy	0	0	D4:8A:FC:C9:CB:7.	612707963496892
2	2	porsche	0	0	E0:5A:1B:DF:87:A2	162538084997934
3	3	ferrari	0	0	D4:8A:FC:C9:CB:7.	604822962398667
4	4	lotus	0	0	D4:8A:FC:C9:CB:7.	414748570724114
5	5	hundai	0	0	D4:8A:FC:C9:CB:7.	749486285336183

Figure 5: Database Table

Diffie-Hellman Key Exchange validation is done in the backend of the application and the console output of the PCB console. The following figure shows the monitor output for the PCB that indicates the DHKE secret key created and the messages received and sent by the microcontroller.

```
data_received string: DKE7
D
DK
DKE
process string:DKE
new data_received string after removing process:7
starting if statement selection
Entered DKE process to start DiffieHellman
b value is: 6
Bprime: Public Key:12
Secret key: 12
Done
Done
Done
```

Figure 6: Microcontroller DHKE Console Output

Z-Hardware profile exchange validation is done by viewing the console output, as shown below in **Figure 7**, of the PCB console to ensure the values match between both the Android application and the microcontroller databases.

```
data_received string: AES4147485707241145366266823291288745145602702251267531642910856700419592051776
4|lotus
A
AE
AES
process string:AES
new data_received string after removing process:414748570724114536626682329128874514560270225126753164291085670041959205
17764|lotus
starting if statement selection
Entered AES
z-hardware:4147485707241145366266823291288745145602702251267531642910856700419592051776 4
handle name:lotus
Done
Done
Done
```

Figure 7: Microcontroller Z-hardware Exchange Console Output

The following terminal view shows the console output of what happens when the user activates a command within the app, the user can visually confirm the DDS is created from the Z-Hardware Profile and command by looking at the received string on the PCB console terminal.

```
data_received string: COM28A5FDC25392C70B6E860CBBF4BFEEA6F3A15AA1788AAB71FA934B71CCF32CF9|Mode10|1714345140497|lotus
C
CO
COM
process string:COM
new data_received string after removing process:28A5FDC25392C70B6E860CBBF4BFEEA6F3A15AA1788AAB71FA934B71CCF32CF9|Mode10
|1714345140497|lotus
starting if statement selection
received_hash: 28A5FDC25392C70B6E860CBBF4BFEEA6F3A15AA1788AAB71FA934B71CCF32CF9
new data_received string after removing received_hash: Mode10|1714345140497|lotus
command: Mode10
new data_received string after removing command: 1714345140497|lotus
unix_time: 1714345140497
new data_received string after removing unix_time: lotus
handle_name: lotus
Done
Done
z_hardware: 4147485707241145366266823291288745145602702251267531642910856700419592051776 4
string for SHA: 4147485707241145366266823291288745145602702251267531642910856700419592051776 4Mode101714345140497
ending string DDS: 28A5FDC25392C70B6E860CBBF4BFEEA6F3A15AA1788AAB71FA934B71CCF32CF9
received_hash from app: 28A5FDC25392C70B6E860CBBF4BFEEA6F3A15AA1788AAB71FA934B71CCF32CF9
received_hash size: 64
```

Figure 8: DDS Exchange Console Output

As shown in **Figure 8** above, the console output shows the steps the microcontroller takes to separate the received signal, read the Z-hardware profile from the Non-Volatile Storage in the ESP32, and create a string to put through SHA-256 for command validation. Once the string has been put through SHA-256, the received hash is compared to the created hash and if they match, the ESP32 accepts the command and turns on/off the corresponding LED, if they do not match the ESP32 ignores the command.

To validate these processes working, we constantly sent signals over and monitored the console output of the PCB and the application to ensure the signal sent was also the same signal received and that all the code was working as expected. If everything worked correctly, we would see a physical validation from the specified LED turning on depending on the command sent.

4. Overall Performance of System

With everything integrated, the system worked as expected. Beginning with the Registration Process and the Digital Key FOB application, a user would create a profile for the vehicle they are registering and sign their signature that then gets created into a unique Z-hardware profile. After prompting the user to connect to the “vehicle’s” Bluetooth, this unique

Z-hardware profile is then sent over to the “vehicle” via the Registration Process to get stored in the microcontroller's Non-Volatile Storage. After the Registration Process is complete and the Z-hardware profile is stored using the profile name, the user can connect to the profile anytime needed. This project is a proof-of-concept demonstration so the PCB design shows the basic functionality of “unlocking/locking the vehicle” and “starting/stopping the engine” using red and green LEDs to indicate communication functionality. Once the user connects to the profile created the Communication Process is then started and they can then navigate to the controls page of the application to “unlock/lock” their “vehicle” or “start/stop” their “vehicle”. Once these commands are sent through Bluetooth, a hash is created on the application’s end using the unique Z-hardware profile of the profile selected and is sent over to the PCB. This signal includes the process the application is initiating, the hash created, the command, the unix-time the command was sent, and the profile’s name. Using this information, the “vehicle” can then obtain the Z-hardware profile from its stored data and create its own hash. If these hashes match, the “vehicle” accepts the command and turns on the corresponding LED.

Overall the application worked well with the cryptography code added and the microcontroller, ensuring a secure Bluetooth connection which was the overall goal of the project.

5. Conclusions

5.1 Key Decisions and Limitations

There were several key moments within the project that changed the way we designed specific areas of the project. These were caused by limitations in cryptography libraries and the development time required to implement the changes while providing enough time for validation before the final demo. These limitations are briefly explained below and more details regarding these issues can be found in the Communication and Cryptography subsystem report.

The first of these was the removal of AES-256 from the registration process. This occurred due to the inability of cryptography libraries to effectively communicate with different coding languages. As a result, when the encrypted text was being created on both ends of the project in Kotlin and C++, the resulting cipher texts were different. This is likely due to the different initialization vectors and other parameters that may have been set differently between the coding libraries.

Additionally, around a month before the final demo, we were asked to add unix-time verification to the Communication Process. This added an extra level of randomization to the DDS creation scheme and also prevented Relay Attacks by verifying the time stamp to be within 1-2 seconds of the initial signal creation. This prevents hackers from simply listening to the intermediate signals between the Digital Key FOB application and the PCB vehicle and then repeating the messages later on to steal the vehicle. We were unable to fully implement the verification as the ESP32 chip did not hold its own unix-time and required a separate Wi-fi connection to pull the unix-time from an online server. This required a significant amount of code development in the MCU and was not feasible within the given time frame.

5.2 Learnings

This project was a new experience, exposing the team members to things they have never done before but are needed in the industry. Sandia National Laboratories tasked the team with a National Security proof-of-concept project that included application development and design, Cryptographic code, and PCB design and development as well as microcontroller programming.

Application development was an entirely new area of project development for Jeremy Hein. Having only taken two courses in C++ and Python, learning to use an entirely new IDE, and coding language, and create a functional app was a huge challenge. To overcome this, extensive research was done using online resources to learn the basic building blocks of application development and Kotlin. One of the largest challenges when building the application was the inner workings of the Bluetooth functionality required to maintain a connection with another device and properly handle errors and edge cases. Designing the UI was also a unique challenge as it required multiple revisions and constant updates to feedback from testers. This contributed to understanding the importance of user feedback and how to effectively incorporate it into app updates. Overall, this project was a complete step out of our comfort zone and challenged our abilities to quickly pivot when deadends or difficulties arose within the project.

Going into this project, Courtney Eaton had no prior experience or knowledge in cryptography so this project was a huge learning experience. Throughout this project, knowledge was gained of the various cryptography algorithms, their backgrounds, and advantages and disadvantages compared to each other. There was also a large amount of learning in the various cryptography libraries as it was chosen which libraries would be the best to utilize and how to call the needed functions and algorithms in order to use the necessary cryptography protocols for this project.

PCB design and development was a completely new topic touched on in the development of the project. Previously, the only experience introduced to students was with the use of breadboards in laboratories, but with the project, Adriana Matos-Almodovar was able to learn the basics of PCB design and development using the Altium design software. After the PCB was printed, the components needed to be soldered onto the PCB, so that was also a new experience that exposed the students to a new skill. Since the design included a microcontroller, coding was also necessary to program the ESP32. C++ was used, which had been introduced to the students previously, but ESP-IDF was also used for programming. Even though C++ was a familiar coding language, there was still room for learning since ESP-IDF was unfamiliar and the example code and Espressif documentation provided the needed insight to complete the project.

Overall, the project provided many learning opportunities for the team, and although difficult, these new skills allowed for a working project that met the specifications given by Sandia National Laboratories.

5.3 Future Work

If more time had been allotted to complete the project, the team may have been able to solve the problem regarding the miscommunication between languages for the AES-256

code and implement it. Along with AES-256, the team could have also implemented the Relay Prevention code by figuring out how to get the microcontroller to connect to Wi-fi while simultaneously having it connected to the Digital Key FOB application through Bluetooth. This connection to an internet server would be to obtain the unix-time when the signal arrived at the microcontroller. Both of these additions would add to the security of the Bluetooth communication between the Digital Key FOB application and the vehicle and help further prevent automotive theft and hacking insecurities.