Team Project – SS2020

Embedded security showcase on PSoC64 (ESSOP)

Project report

Author: Aadarsh Kumar Singh, 766499

Hari Krishna Yelchuri, 766518

Vaishnavi Sankaranarayanan, 766486

Date : 16th Oct 2020

Status: released

Contents

1. Objective

1.1 Background

- During our Embedded Architecture and Application lecture series, a remote-controlled student car with assisted driving was developed. The remote control used PSoC5 MCU for reading the joystick data provided by the driver and transmitted the Joystick data in the form of a protocol to the student car via ZigBee. The student car used the PSoC5 MCU to calculate the control speed of the engine motors based on the protocol values and the ultrasound sensor (obstacle detector).

- The driver controls speed and direction of the car using the joystick in the remote control and additionally, the ultrasound sensor was used a safety mechanism to detect the obstacles.

- The car receives the joystick data via ZigBee. Although car checks the integrity of the received protocol using CRC, but the communication between the remote control and the car is not secure because the received data is neither encrypted nor authenticated.
  - Lack of authentication results in unauthorized remote control taking control over the car or remote controller controlling a car that is not truly their car.
  - Lack of encryption can result in man in middle attack unauthorized actor can eavesdrop the protocol data that is transmitted via unsecure BLE channel. This allows them to tamper/modify the protocol.

1.2 Use-Case

The aim of our project is to secure the communication between the remote control (sender) and the car (receiver). This would enable:

1.2.1 Confidentiality:

Any unauthorized source (e.g. Hacker) will not be able to gain access to the joystick data/car control signals transmitted over unsecure BLE channel.

1.2.2 Authenticity:

The car needs to verify that the remote control going to control it is authorized and the remote control needs to validate if it is the true master of the car.

1.2.3 Message Integrity:

Ensures that the joystick data/car control signals protocol has not been modified /tampered.

1.2.4 Non- repudiation:

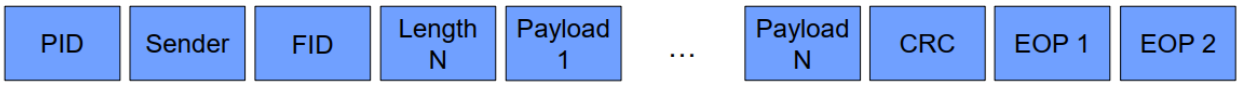
The remote controller (driver) is held credible for the joystick data provided for controlling speed and direction.

2. Threats based Analysis for Embedded Security Use Case – CIA Model

- The CIA Model stands for confidentiality, integrity and authenticity.
- We identify the data and system assets of the product which we want to secure.
- We identify all sources of threats to these data and system assets
- Each threat is assigned a data security objective (confidentiality, integrity and authenticity).

2.1 Protocol Definition

The data that has to be transmitted is composed into a protocol, the protocol looks like this:



where,

- PID stands for protocol Identifier: (Joystick data / Control Signal)
- Sender represents Identifier of the sender (CAR/Remote controller)
- FID stands for feature Identifier: (Connect, Disconnect, Feature, Stop)
- Length : Number of bytes (N)to be transmitted
- Payload : N bytes to be transmitted
- CRC : cyclic redundancy check to ensure integrity of message
- EOP stands for End of payload

2.2 Use case Analysis: secure the communication between remote control and car.

2.2.1 Data asset of remote control

Data Asset	**Description**
Remote Control Identifier	Unique Identifier for the remote control
Remote Control Firmware	Defines the behavior/functionality of the remote control
Cryptographic Credentials	Keys used for Cryptographic procedures
Feature Data	Joystick Data/ Car Control (connect, disconnect, stop) Signals

2.2.2 Data asset of student Car

Data Asset	**Description**
Car Identifier	Unique Identifier for the student car
Car Firmware	Defines the behavior/functionality of the student car
Cryptographic Credentials	Keys used for Cryptographic procedures
Feature Data	Acknowledgement and feature available data

2.2.3 Secure properties associated with data assets

Data Asset	Security Properties	Description
Identifier(ID)	Integrity	ID is a unique proof of identity for the device hence, one should not be able to tamper/modify the ID associated with the device.
Firmware	Confidentiality	Any unauthorized source should not be able to gain access to the firmware, otherwise they can copy it and use in duplicate products.
	Integrity	Any unauthorized actor should not be able to gain access otherwise, they can modify and affect the quality of software leading to uncontrolled behavior.
	Authenticity	The firmware running on the remote control/car should be authentic and the remote control/car on which the firmware is running should be genuine.
Cryptographic Credentials	Confidentiality	Unauthorized source should not be able to access the keys that is used for cryptographic operations like encryption, signature.
	Integrity	Unauthorized actors should not be able to modify/tamper the keys used for Cryptographic services Like Encryption, Signature
Feature Data	Confidentiality	Unauthorized source should not be able to gain access to the Joystick data / car control data otherwise, they can tamper the Joystick data and provide wrong values instead.
	Integrity	Unauthorized actors should not be able to modify/tamper the joystick data and car control signal.

2.2.4 Threats on the identified data assets

Threats	Target data assets	Description
Impersonation	Cryptographic credentials (Keys)	Keys form the cryptographic basis to secure the communication between the remote control and car via BLE. If the confidentiality of keys is compromised, then they can be used by unauthorized actors can gain access to the data that is being communicated via BLE.
Man in the Middle	Cryptographic credentials (Keys)	Secret keys used for cryptographic services like encryption should not be exposed via the unsecure channel, there is chance that unauthorized sources can eavesdrop and tamper/modify the Joystick data or control signals that is being transmitted.
	Feature Data	If the Joystick data/ control signals is not encrypted and authenticated than it can be easily replayed by an unauthorized actor, leading to unapproved sources driving the car.
Firmware Abuse	Firmware	The firmware flashed into MCU of the car and remote control should be genuine, one provided by authorized OEM. If unauthorized firmware is flashed and remote control/car starts, it can lead to undefined/uncontrollable behavior.
Tamper	Cryptographic credentials (keys)	Keys should be unique to the user and it should be stored in a secure environment. Any form of key exchange should be confidential and have a cryptographic basis.
	Firmware	The remote control and the car should always run authorized firmware. Modification of firmware by the unauthorized sources, it must be prohibited.

	Feature Data	The Joystick data or the control signals communicated via the BLE should be secure, any unauthorized sources should not be able to read and tamper these data.
	Unique Identifier	The identifier associated with the remote control and the car are unique. It should not be possible by unauthorized sources to tamper the identifier associated with the device.

2.3 Security Objectives

- To secure the communication between the remote control and the car, we have identified the data assets and its associated security properties.
- Based on this information we have enumerated all possible threats linked with each of the data assets.
- We need to define the security objectives based on the enumerated threats. The table below

Threats →

Security Objectives ↓

	Impersonation	**Man in the middle**	**Firmware Abuse**	**Tamper**
Access Control	X		X	
Secure Storage				x
Firmware Authenticity			X	
Communication		X		
Secure State			X	X

2.3.1 Access Control

- The remote control and the car must allow only the authentic firmware to be flashed.
- The cryptographic keys must be unique for the car and the remote control and must be confidentially inserted/generated in them.
- Before communication is established between the remote control and the car, the remote control should verify if it is communicating with the genuine car and the car needs to verify if it is being controlled by authentic remote control.

2.3.2 Security Storage

- The cryptographic keys must be stored in a secure area inside memory.
- The MCU must not allow overwriting the unique keys, which is already present in the key storage area.
- The secret keys must never be exposed to non-secure environment.

2.3.3 Firmware Authenticity

- The MCU must verify the firmware authenticity before booting and before upgrading based on digital signatures.
- It should also check if the authentic firmware is modified/tampered by using mechanisms like CRC checks.

2.3.4 Communication

- The communication between the remote control and the receiver must be encrypted, so that unauthorized sources are not able to read or modify the protocol that is being transmitted.

2.3.5 Secure State

- The car and the remote control firmware should be able to maintain a secure state in case verification of firmware integrity and authenticity fails during boot up / upgrading.

2.4 Security Requirements

2.4.1 Security Requirements table

Security Objectives	Threats	Methods to avoid threats
Firmware authenticity	Firmware Abuse	Digital Signature (Flashing digitally signed firmware and verifying the authenticity of the signed firmware prior to boot or upgrade) using asymmetric cryptographic algorithm.
Access Control	Impersonation	Digital Signature (Signing the unique credentials exclusive to the car/remote control and verifying the signed credentials of the partner before starting communication with it) using asymmetric cryptographic algorithm.
Secure key Storage and key exchange	Tamper	Once the cryptographic keys are written in key storage area, the firmware must not allow it to be over-written. This can be implemented using e-fuses present in MCU Cryptographic algorithms such as diffie-hellman Algorithm must be used for key exchange between car and remote control.
Communication	Man in middle	Encryption (Encoding the protocol before transmitting in such a way that only authorized person can decode it) using symmetric cryptographic algorithm.

2.4.2 Avoiding firmware abuse threat

- The authenticity of the firmware has to be verified prior to boot/upgrade. This can be done by signing the firmware using digital signature. The MCU must boot only if the digital signature of the flashed firmware is authentic also known as secure boot.
- The firmware can be signed digitally using keys generated by cryptographic algorithms. The cryptographic algorithm used for generating signature must be asymmetric to ensure non-repudiation, i.e. the actors (OEMs) signing the firmware (or a message) are held credible for the firmware that is flashed.
- The authorized source signs the firmware using a key. This key must be exclusively associated with the source and it must be kept confidential.
- The cryptographic key used to verify the authenticity of the flashed firmware must be injected in the MCU of remote control/car in a secure and confidential manner.

2.4.3 Avoiding impersonation threat

- The car and remote control will have a unique identification number (credentials) associated with them.
- Before starting communication between car and remote control, the car needs to verify that the remote control going to control it is authorized and the remote control needs to validate if it is the true master of the car.
- This can be achieved using Digital signatures. Both car and remote control should sign the credentials that are uniquely linked with them. This digitally signed credentials acts as an exclusive proof of identity.
- Remote control sends its signed credentials and car should be able to verify it and then the car sends its signed credentials and remote control should be able to verify it.
- The keys used for verifying the signed credentials should be exchanged between the car and the remote control in a confidential and a secure manner.

2.4.4 Avoiding man in middle threat

- We need to encrypt the protocol that is being transmitted via unsecure BLE channel so that if the unauthorized actors eavesdrop they are not able to read/tamper the original protocol data.
- Encryption can be done using symmetric cryptographic algorithm, since the credibility of the actor/source is verified during authentication. It is only required to encode the original protocol such that only the authorized device is able to decode it.
- We need to exchange a secret key between the car and remote control so that using this key encryption can be performed. This key exchange should be done in a secure and confidential manner.

2.4.5 Avoiding tampering data

- The cryptographic keys are linked exclusively to the device (remote control/car). Hence, it must be stored in a secure and confidential manner. It is vital for the MCU to have hardware-based isolation between secure and unsecure execution environments.

- Once the cryptographic keys are written in key storage area, the firmware must not allow it to be over-written. This can be implemented using e-fuses present in MCU.

- The key used for signing the firmware (secure boot to avoid firmware abuse), the keys used for the signing the credentials (to avoid impersonation) and keys used for encryption must be different. This is necessary because if same key is used to avoid the threats, compromising the confidentiality of the keys can lead to multiple/all security threats.

- The key exchange between the car and remote control via unsecure BLE channel should be performed using cryptographic algorithms such as diffie-hellman so that the confidentiality of the keys are not compromised.

3. Hardware Design Decisions

3.1 Hardware features essential in a MCU to fulfill the specified security requirements

The MCU of the car and remote control should have all the required security capabilities for supporting the software to establish secure communication between them. The required security capabilities are listed below:

- **E-Fuses**:

  - The MCU should support e-fuses for secure storage of data assets like unique identification numbers (UID), manufacturer numbers that are exclusive to the device and never change throughout lifetime of the product.

  - An example use case where e-fuses are required is described below:
    For verifying the firmware authenticity, we use keys to sign the firmware. These keys are unique for the student car/remote and it has to remain the same throughout the lifetime. Unauthorized actors must not be allowed to overwrite them. This can be ensured by using e-fuses. Once the keys are written to the secure key storage section e-fuses are blown off, nobody can now change the keys that are flashed into this memory region.

- **Hardware Accelerator for cryptographic algorithms:**

  - The cryptographic algorithms are extensive and require complex mathematical computation. MCU must have hardware accelerator to run these algorithms for high performance in terms of speed.

- **Isolated execution environment for trusted applications:**

  - It is necessary that the MCU offers a strong hardware based isolation between unsecure and secure environment to protect the crucial data assets like protocol data, cryptographic keys, and firmware credentials from being tampered. They must be stored in a secure manner such that it is only accessible to the authentic actors.

  - The best way to isolate the secure environment from non-secure environment is to use multicore MCU with a dedicated core for performing all the cryptographic procedures, storing cryptographic keys and the other core used for performing other non-secure operations. It has to have a secure inter core communication. Only authorized actors would have permission to execute operations/functionalities on the secure core.

3.2 Choosing PSoC64 MCU for student car and remote control

The student car and the remote control uses a PSOC5 MCU. The PSoC5 lacks the following security capabilities:

- PSoC5 is a single core MCU that does not have dedicated hardware accelerators necessary to run cryptographic algorithms like ECDSA, AES etc.
- There is no isolation between the secure and non-secure environment. Hence, it does not have hardware support to allow only the authorized actors to execute processes on the device.
- E-fuses are not present, hence immutable data assets that are uniquely associated with the device can be overwritten by unauthorized sources.

Hence, we need to change the MCU so that it has the required security capabilities. We have chosen PSoC64 MCU as it has:

- **Isolated execution environment:**
  - PSoC64 is a dual core controller with a dedicated core (HSM) for performing all secure /cryptographic operations and cortex-M4 for performing unsecure/high performance operations.
  - Inter-Processor Communication (IPC) channels between the Arm Cortex-M4 and Cortex-M0+ cores are provided to support isolated API-based interaction.
  - An example is secure boot, once the firmware is flashed, cortex M0+ (secure HSM core) verifies the authenticity of the firmware then allows it to boot. If the attacker flashes a false firmware into the device, the HSM core will not allow it to boot. Hence, only authorized actors will be able to run the applications on secure HSM core. Only when the secure core allows it can run applications on the cortex-M4 core.
- **Integrated secure element functionality:**
  - It has Memory protection units (MPU), peripheral protection units (PPU) that allows cores to access only the permitted memory regions/peripherals. In addition, PSoC64 provides a framework to add secure services such as key storage, cryptography, etc.
- **Hardware accelerator for cryptographic algorithms**
  - The hardware accelerator are used for running cryptographic algorithms like AES, 3DES, RSA, ECC, SHA-256, SHA-512 and True Random Number Generator (TNRG) to improve the performance of the system in terms of speed.
- **E-fuses**
  - It has e-fuses support for secure storage of immutable data assets.

4 Software Design Decisions

4.1 Selecting Mbed OS library

As per the hardware design analysis, the car and the remote will use PSoC64 as the MCU. To develop an application for establishing secure communication between, we require a software library that contains all the necessary device drivers of PSoC64. In addition, it should contain a security library which is compatible with the hardware accelerators present in PSoC64 for running cryptographic algorithm and provide device driver for BLE communication.

- Mbed-OS is an open-source embedded operating system, which supports PSoC64 MCU. As mbed –OS is suitable for PSoC64 , we had evaluated the software library based on our requirements:
  - Modular:
    - Device driver for all the supported PSoC64 peripherals such as digital and analog IO, interrupts, port and bus IO, PWM, I2C, SPI,DMA and serial are present. It is compatible with wide range of software libraries for RTOS, sensors and thread safety support.
    - Mbed OS contains all the supported toolchain libraries of PSoC64. It is easy to integrate Mbed-OS into a C/C++ project running in eclipse IDE.
  - Secure :
    - Supports mbed-TLS security library that is compatible with hardware accelerators present in the PSoC64 for running the cryptographic algorithms.
    - Supports ARM Platform Security Architecture (PSA) framework.
  - Connected:
    - Driver support for Bluetooth Low Energy (BLE) present, since the communication between the car and the remote control takes place via BLE.

4.2 Cryptographic Algorithms to be employed for the given use case

- The key used for signing the firmware (secure boot to avoid firmware abuse), the keys used for the signing the credentials (to avoid impersonation) and keys used for encryption must be different. This is necessary because if same key is used to avoid the threats, compromising the confidentiality of the keys can lead to multiple/all security threats.
- Asymmetric cryptographic algorithms must be used for generating digital signatures whereas symmetric cryptographic algorithm for encryption/decryption. The reason is:
  - The cryptographic algorithm used for generating signature must be asymmetric to ensure non-repudiation, i.e. the actors (OEMs) signing the firmware (or a message) are held credible for the firmware that is flashed.

- Encryption can be done using symmetric cryptographic algorithm, since the credibility of the actor/source is verified during authentication. It is only required to encode the original protocol such that only the authorized device is able to decode it.

4.2.1 Digital Signatures

- Digital signatures has to be used for validating the authenticity and integrity of the
  - firmware prior to boot/upgrade
  - protocol data in- transit from the sender to the receiver
- Elliptic Curve Digital Signature Algorithm (ECDSA) algorithm is used for generating digital signature. Before generating the digital signature of a data (firmware/protocol), it is hashed to reduce the size of the generated digital signatures. SHA-256 is used as hashing algorithm in the application.
- The size of the cryptographic keys are 32 bytes and the size of generated signatures are of 64 bytes.
- ECDSA algorithm was employed because :
  - The crypto hardware accelerator present in the PSoC64 MCU supports it.
  - It is asymmetric and offers same level of security in comparison with other algorithms like RSA but with smaller key lengths. The algorithm for generating the signature becomes faster as the computations involves smaller keys. Small public keys require less data to pass around to establish secure connections. This implies faster connection rate.

4.2.2 Encryption/Decryption

- Application will use encryption to avoid man in middle attack aimed at tampering the in-transit data.
- To encrypt the protocol data we need to use AES encryption algorithm.
  - The main advantage is its speed. AES is symmetric key algorithm hence it requires less computational power than asymmetric one making it faster and more efficient to run.
  - The crypto hardware accelerator present in the PSoC64 MCU supports it.
- AES algorithm has three types: AES-128, AES-256, AES-192 based on the key size. We use AES-128 for encryption. This implies the size of the key will be 16 bytes. The reason to use AES-128 is that larger key size results in more computations which makes encryption process slower. AES-128 has the smallest key size and it offers the optimal security level for our use case.
- There are two cipher types namely block ciphers and stream ciphers. In case of block ciphers, encryption is performed block by block, whereas stream ciphers encrypt one bit/byte at a time. The advantages and disadvantage of each cipher mode types are tabulated below:

Cipher Mode Types	Advantages	Disadvantages
Stream Cipher	Faster than block ciphers	It offers low diffusion since all information of a plaintext symbol is contained in a single cipher text symbol.
	Low memory requirement than block ciphers.	It is vulnerable to modifications as any interceptor who breaks the algorithm might insert spurious text that looks authentic.
Block Cipher	It offers high diffusion since information from one plaintext symbol is diffused into several cipher text symbols.	Slower than stream ciphers as computations are more complex
	It has stronger immunity to tampering.	Requires more memory

For the application, we will use AES in CBC block cipher mode for performing encryption/decryption with block size of 16 bytes for AES-128/ AES-192/ AES-256 encryption. Block cipher were chosen as it had high diffusion and strong immunity to tampering as compared to stream ciphers. To speed up the rate of encryption we will use hardware accelerator present in PSoC64.

- Main advantage of CBC mode is that decryption can be performed in parallel. It uses Initialization vector (IV) to hide data patterns while encrypting the message. IV is randomly generated number having same size as the block size, its role is to make each block of encrypted message unique.

4.2.3 Secure key exchange

- As we use symmetric encryption, the receiver has to use the same key which was used for encryption to decrypt the ciphered text to plain text. In addition, Initialization vector has to be sent to the receiver in order to decrypt each block without any mismatch of original plain text and decrypted text.

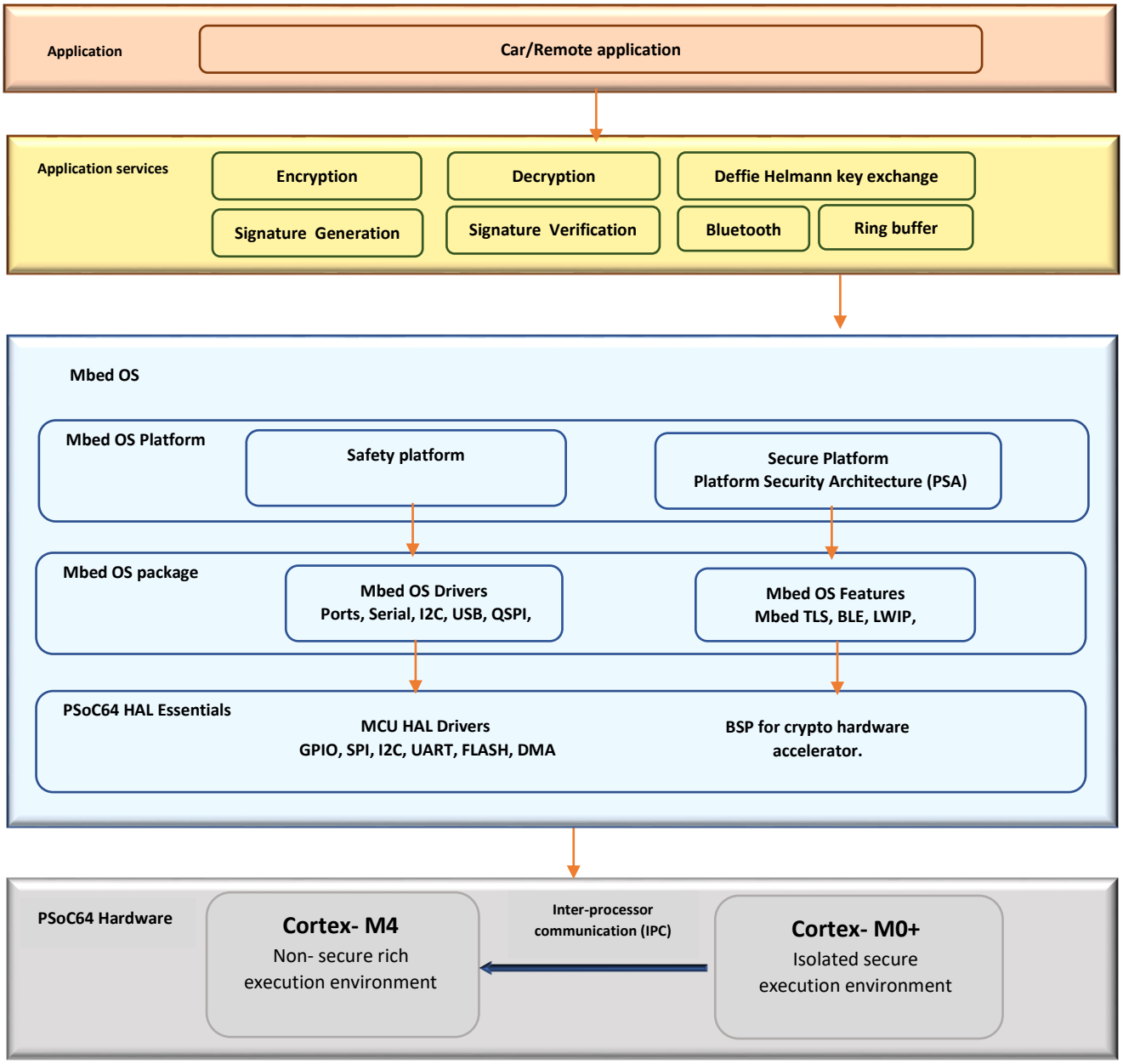
- The key and the IV has to have a cryptographic basis for exchange between the car and remote control. Hence, we require a secure key exchange algorithm. This application uses elliptic-curve Deffie–Hellman (ECDH) key exchange algorithm.

- The crypto hardware accelerator present in the PSoC64 MCU is compatible with Elliptic-curve Deffie–Hellman (ECDH).

- The car and the remote generates a 32-byte Deffie–Hellman key using Elliptic Curve 25519. The generated key has to be exchanged with the corresponding communication partner. The partner will use the exchanged key to compute a secret key of 32 bytes. First 16 bytes will be used as encryption keys and the next 16 bytes could be used as Initialization vector.

5 Software Architecture

To develop the application on PSoC64 MCU we used Mbed OS library. The pictorial representation of the software framework is provided below:



- The Mbed OS library encapsulates the BSP for PSoC64 HAL library to access the peripherals of PSoC64 MCU like GPIO, UART, crypto accelerator etc. The PSoC HAL essentials are accessed via the Mbed OS package. The Mbed OS package contains driver for communication protocols, GPIO ports etc. It also contains features, which has BLE stack, LORA-WAN stack, and security library stacks like MbedTLS. Mbed OS is a safe and secure OS hence it has safety and security platform layer on top of its package layer.

- The application services have been developed to perform security services like encryption, decryption and supports BLE communication. The APIs present in the security services make use of PSA platform to use the mbedTLS library present in the Mbed OS feature package. This internally calls the PSoC64 BSP which uses the crypto hardware peripheral present.

- Since we make use of PSA library to implement the security services our software is PSA complaint.

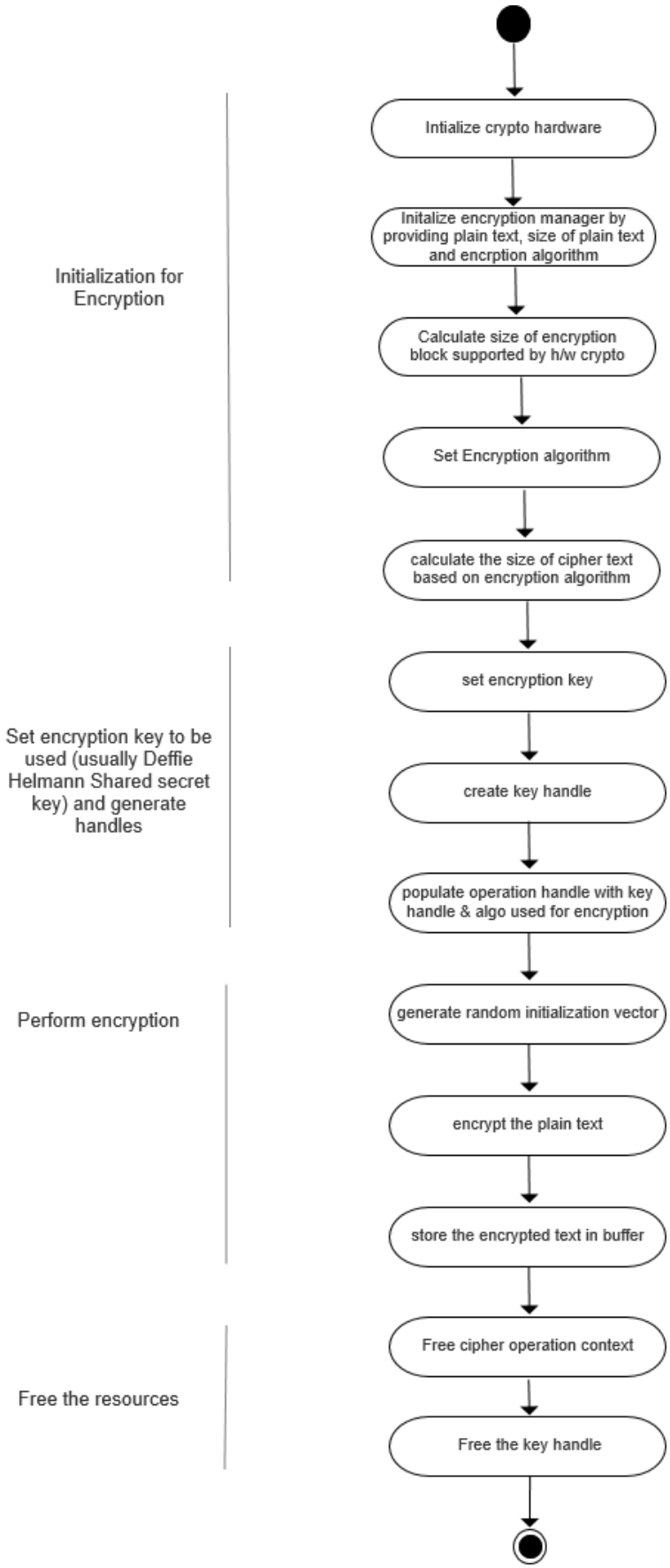
6 Security Library and their version

- We use Mbed-OS 5 software library with version Mbed-OS 5.14.
- It has mbed-TLS security library with the version mbedtls-2.21.0.
- The architecture of PSoC64 MCU is complaint to Arm Platform Security Architecture (PSA). The mbed OS uses PSA Cryptography library, which internally uses the mbed-TLS library to builds services such as secure boot, secure storage and secure inter- core communication compliant to Arm Platform Security Architecture (PSA).
- The PSA library 1.0.0 will be used for developing services for encryption, decryption, diffie-hellman key exchange signature generation and verification.

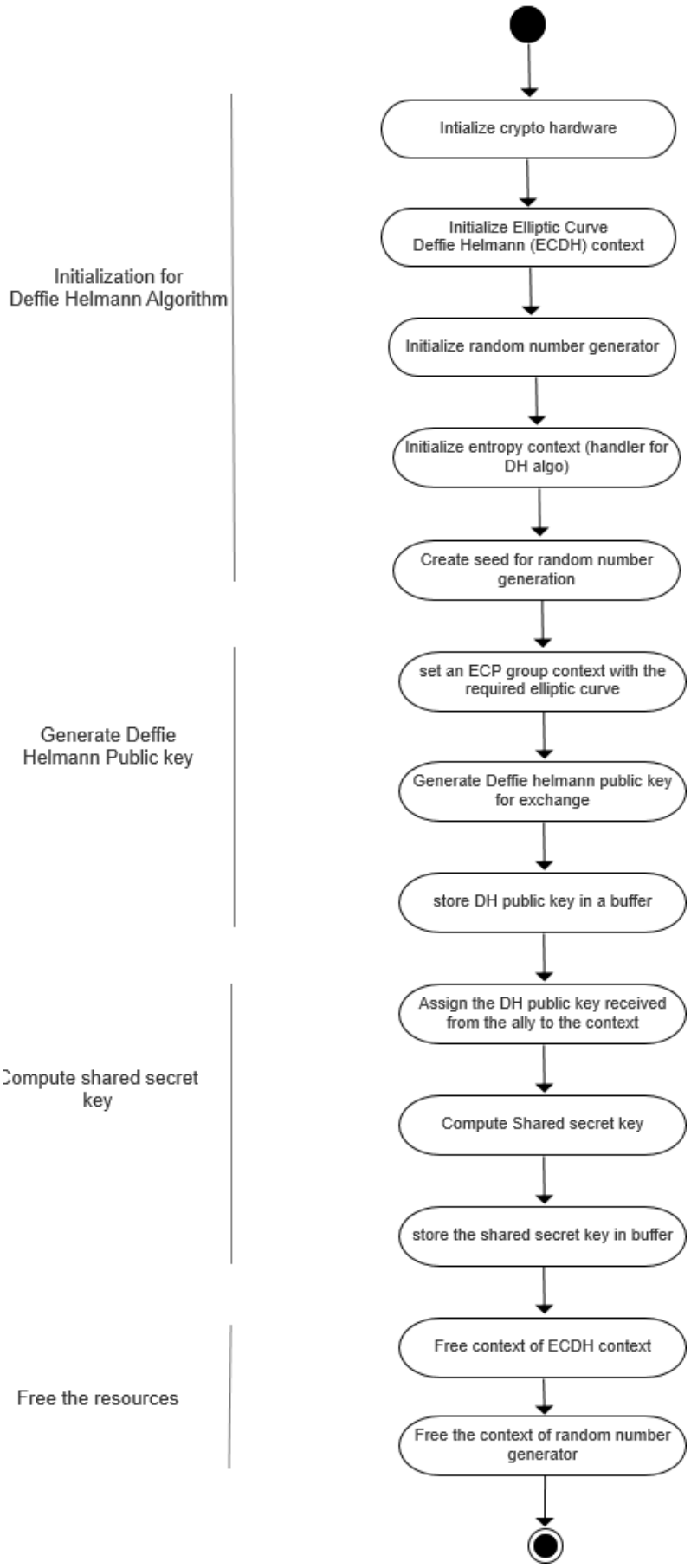
7 Implementation of application services

- The security services for encryption, decryption, diffie-hellman key exchange, signature generation, signature verification, ringbuffer, Bluetooth communication is implemented in the application services module. The APIs for each of these services is present in their respective manager layers. The activity diagram representing functionality of each of the services are specified below.

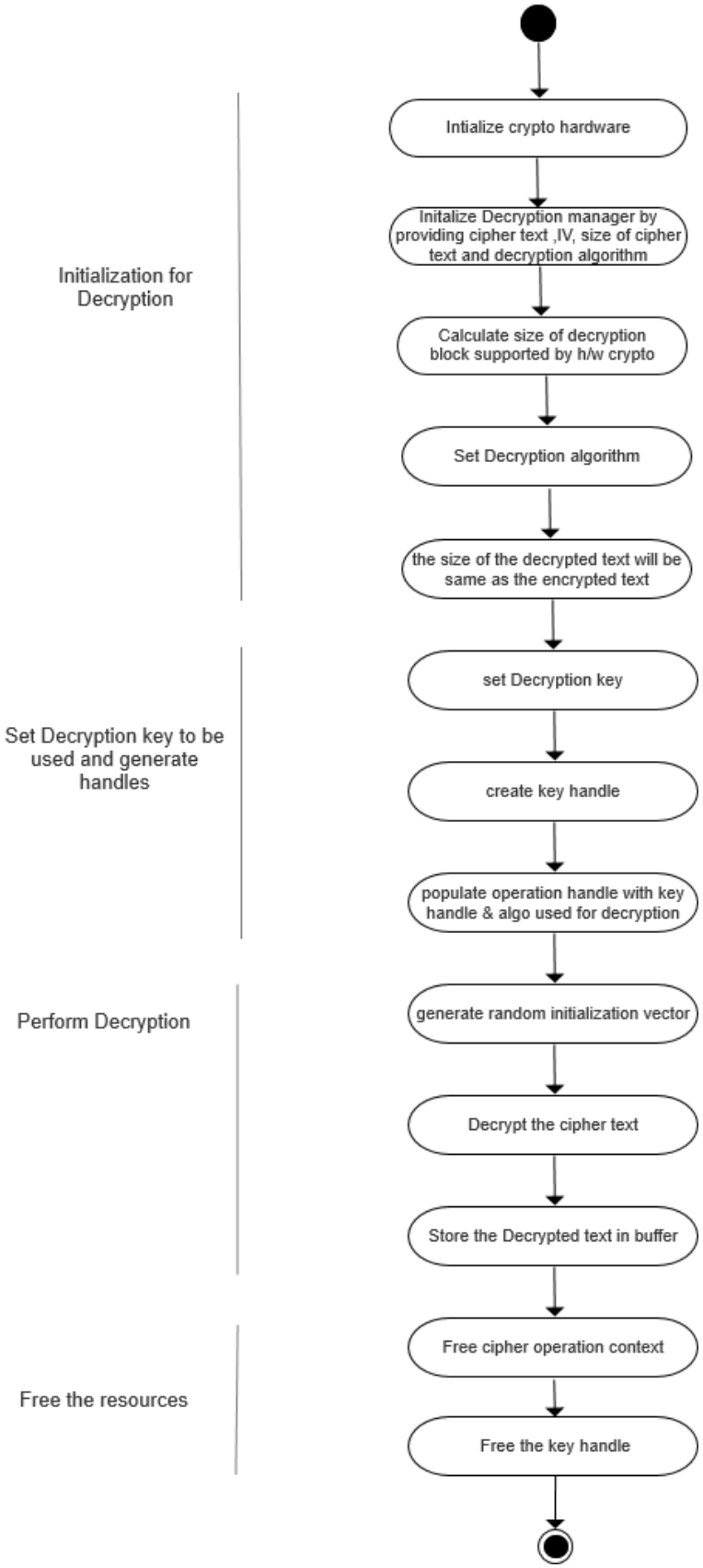
7.1 Activity Diagram of Encryption



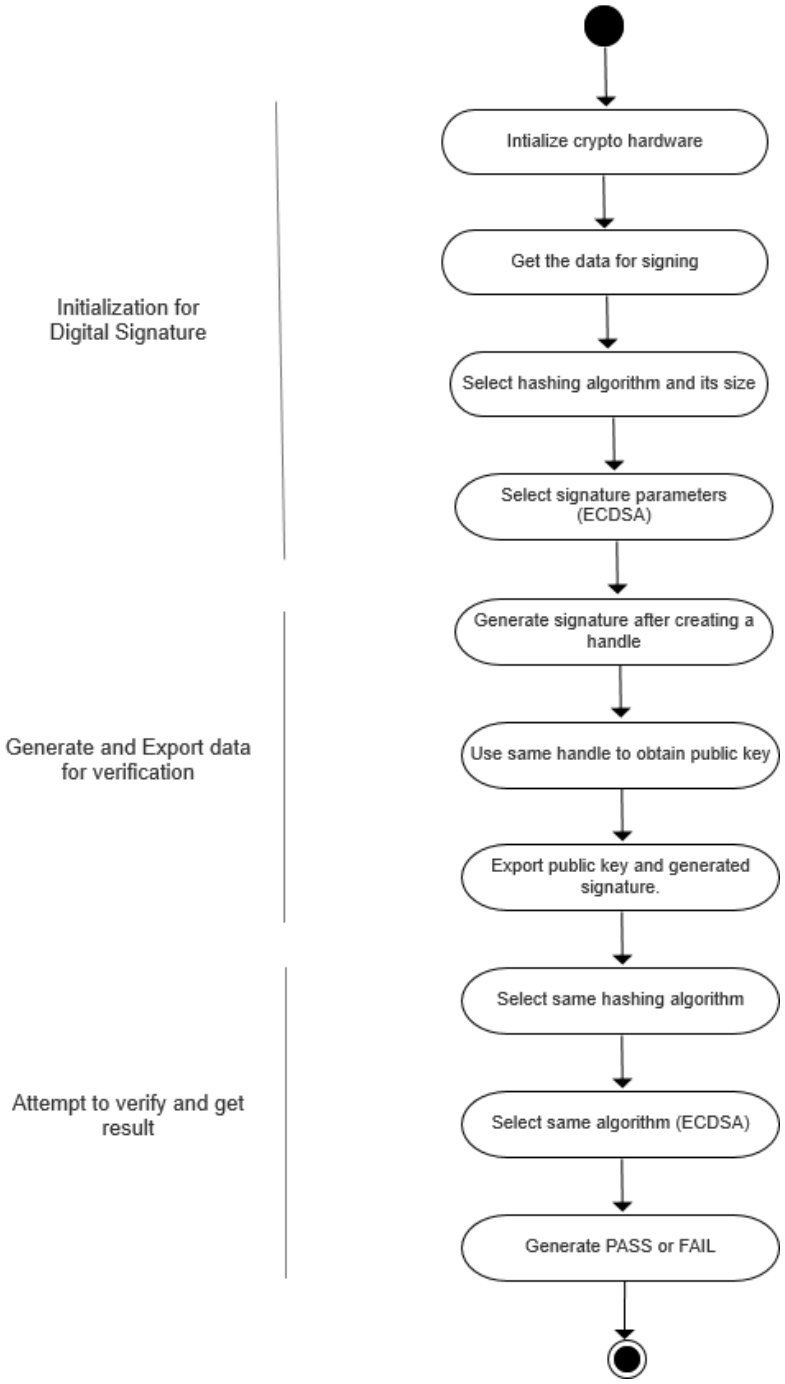
7.2 Activity Diagram of Diffie-Hellman Key Exchange



7.3 Activity Diagram of decryption



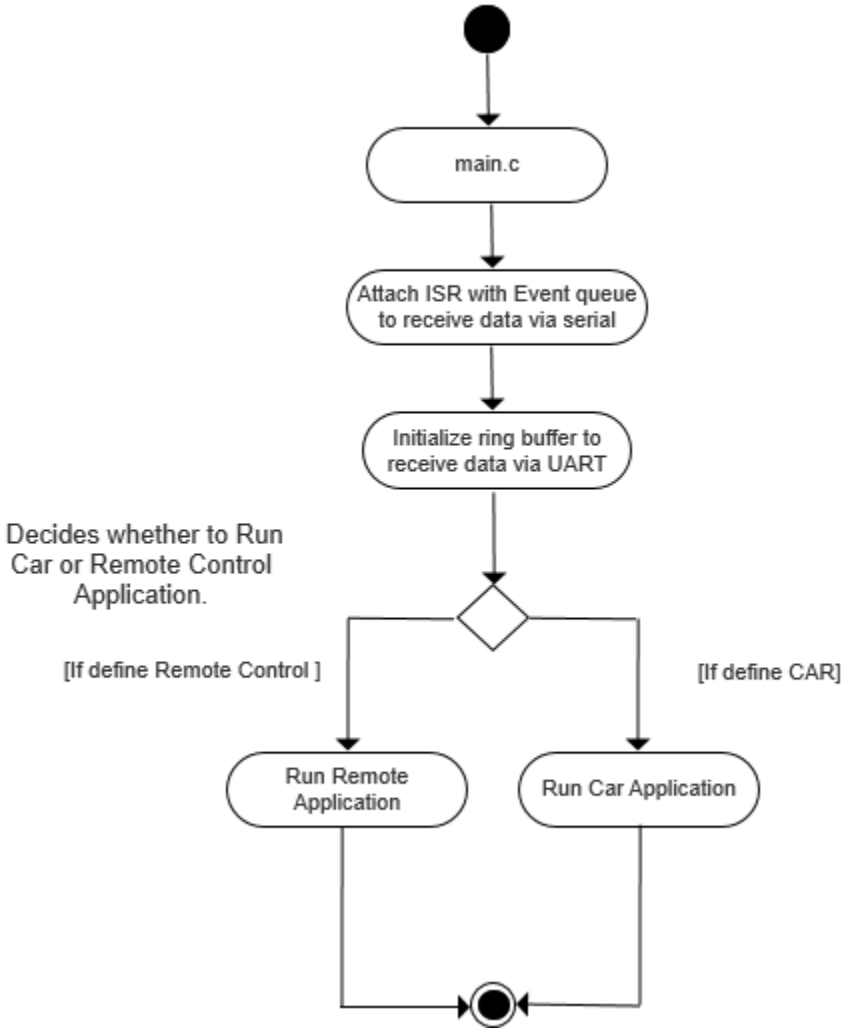
7.4. Activity Diagram of signature generation and verification (Authentication)



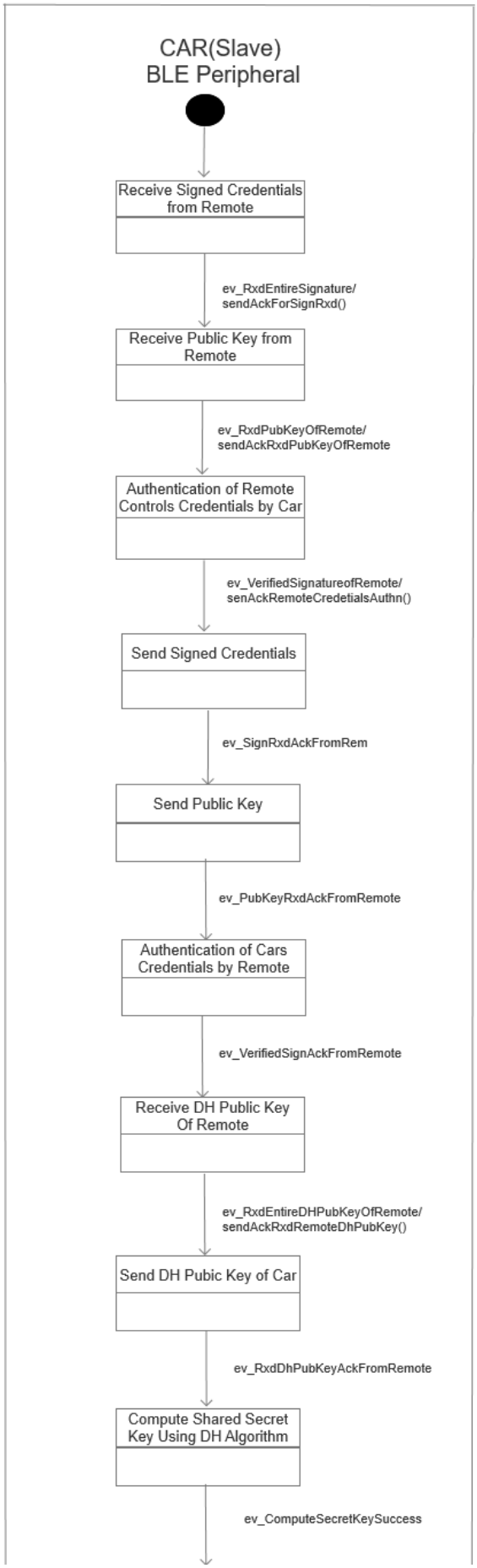
8 Implementation of Application

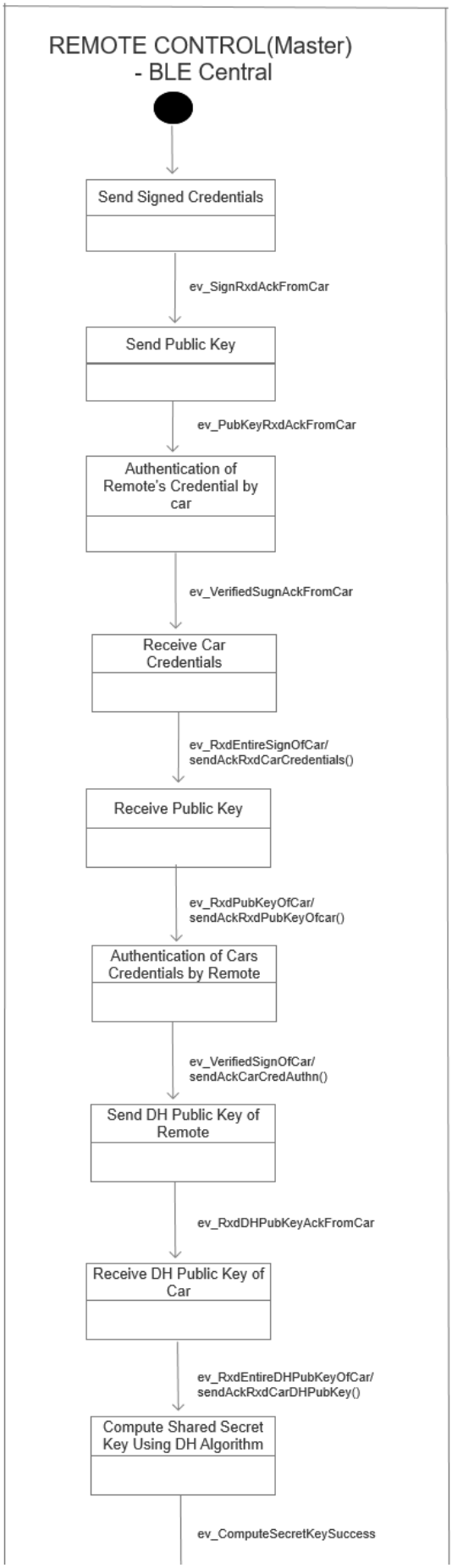
- The use case of securing the communication between the car and the remote controller is implemented in the application software (asw) module.
- The main application attaches event queue with ISR to receive the serial protocol data via interrupt and decides whether to run remote application or the car application.
- The car and remote control application uses a state machine implementation for establishing secure communication.
- The activity diagram for the main application and the state-machine for the car and remote application is demonstrated below

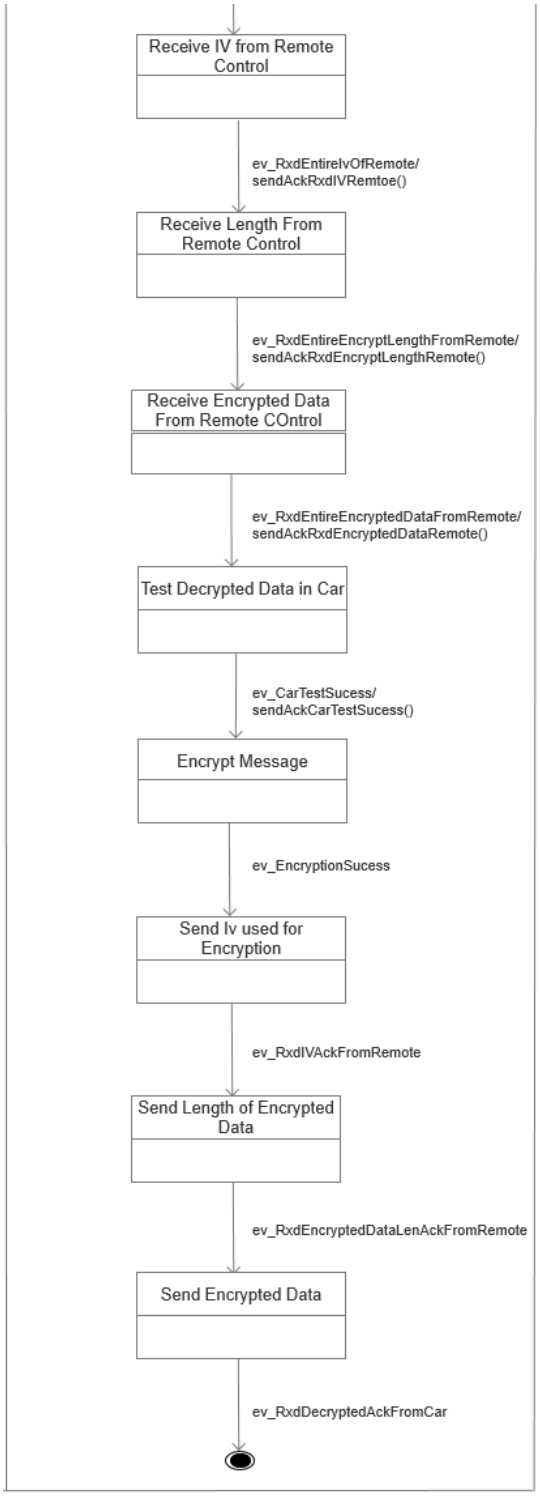
8.1 Activity Diagram of remote application

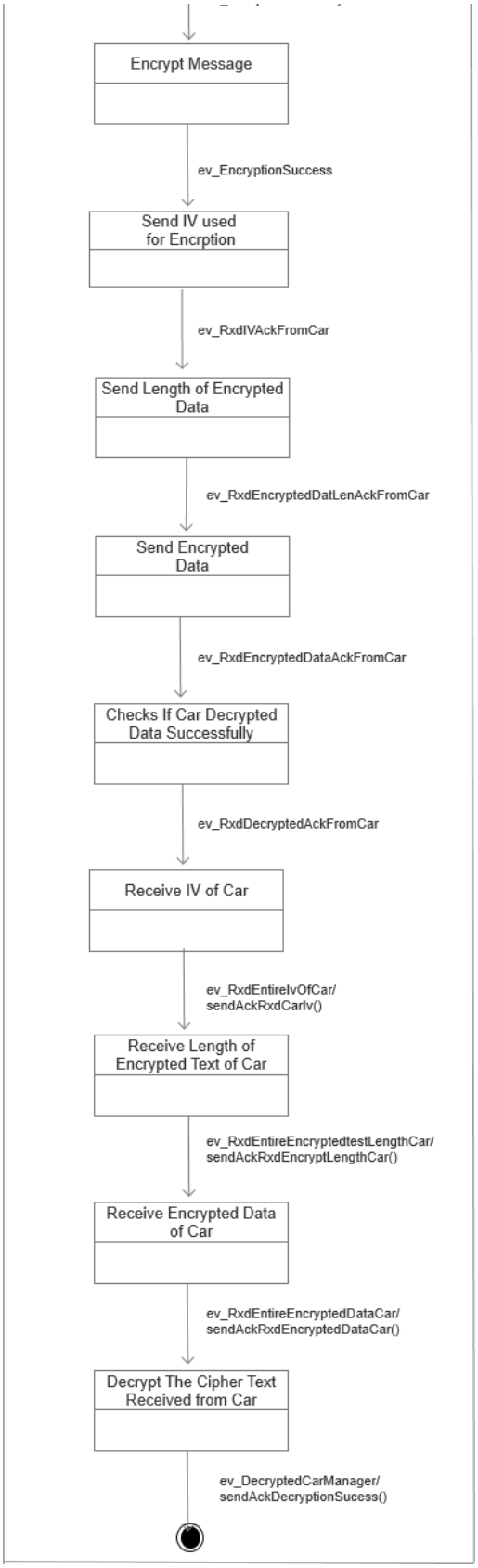


8.2 State-machine of car and remote application









9 Improvements

- The PSoC64 MCU is a new MCU, hence IDE support for PSoC64 MCU was neither available in Modus Toolbox nor in the Mbed Studio library. The essential library package for the PSoC64 MCU was exported by using a hack in the python script project.py present in the location < <root>\mbed-os\tools >. Please find the bug

report and the solution link here . When the support for the PSoC64 MCU is available in Modus Toolbox/ Mbed Studio it has to be migrated to the new IDE . This will significantly lower the size of the project.

- The Mbed OS library used is 5.14, We need to migrate to new Mbed OS library with PSoC64 MCU support when available.

- Event Queues are used to receive the serial protocol data using interrupts because when normal ISR were attached it used to throw hard fault error whenever data was received. On Fixing of this bug we need to use normal interrupt instead of event queues.

10 References

- Threat based analysis – CIA model : https://www.cypress.com/file/447056/download

- Mbed Crypto Getting started example : https://github.com/ARMmbed/mbed-crypto/blob/development/docs/getting_started.md

- PSA API specification : https://armmbed.github.io/mbed-crypto/PSA_Cryptography_API_Specification.pdf

- Security comparision between PsoC64 and PsoC62 : https://www.cypress.com/file/495821/download

- PsoC6 Programming specifications : https://www.cypress.com/file/385671/download