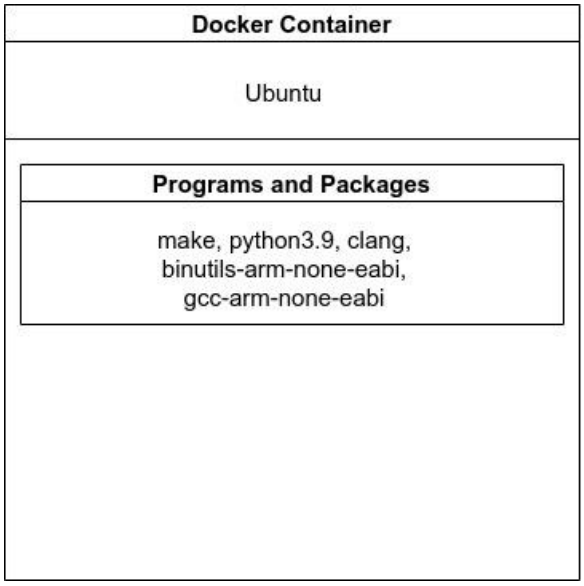
IITM Design Document

Security Requirements (SR)

1. A car should only unlock and start when the user has an authentic fob that is paired with the car
2. Revoking an attacker's physical access to a fob should also revoke their ability to unlock the associated car
3. Observing the communications between a fob and a car while unlocking should not allow an attacker to unlock the car in the future
4. Having an unpaired fob should not allow an attacker to unlock a car without a corresponding paired fob and pairing PIN
5. A car owner should not be able to add new features to a fob that did not get packaged by the manufacturer
6. Access to a feature packaged for one car should not allow an attacker to enable the same feature on another car

Environment (/docker_env)

Docker environment as described in the following figure is used,



Deployment (/deployment)

No deployment-wide secrets are developed because any secret developed would be visible to the attacker through Car 0. Hence we have avoided implementing any deployment-wide secrets.
All the secret keys required for encryption are generated during the development of the firmware.

Host tools (/host_tools)

unlock_tool.py - receives unlock message from the paired fob
pair_tool.py - paires unpaired fob to the car given the correct pin
package_tool.py - used to generate the encrypted package
package_feature.c - helper to generate encrypted package

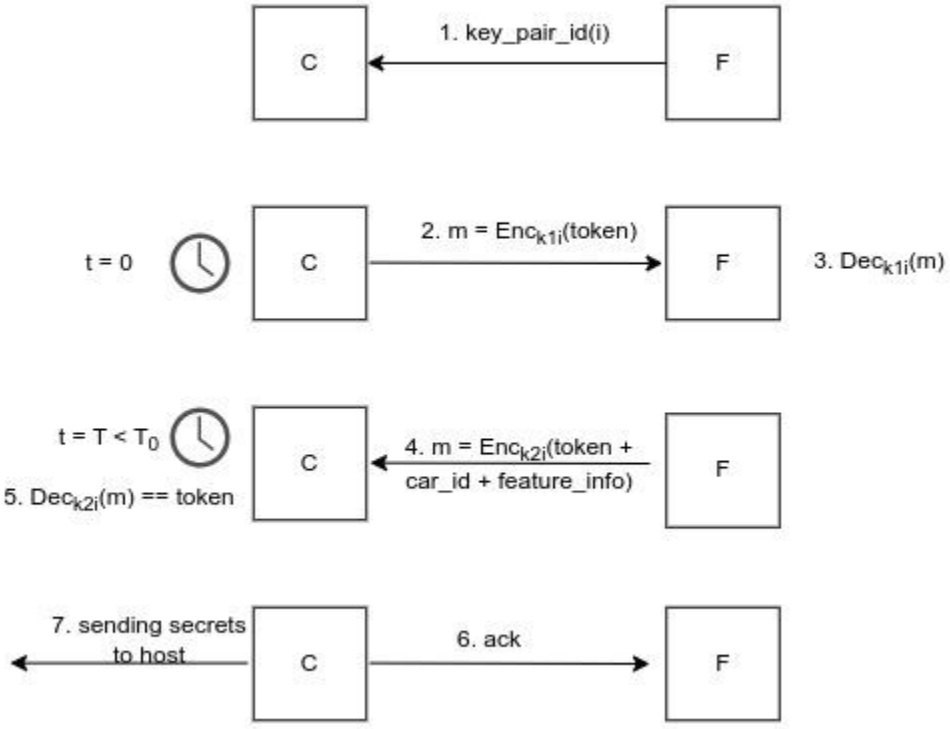
Protocols

The below-given description of the protocols is brief. Refer to the comments in the code for the detailed implementation.

Unlock

- Fob sends the key-id along with the unlock byte to the car, indicating car to start the unlock process.
- Car receives the key-id & fetches the corresponding key from EEPROM. A random token is generated and encrypted using the fetched key and sent to the fob. The validity of token is limited upto threshold time..
    - Since for every unlock process, a random number is generated, replay attacks can be prevented. Since the validity of token is limited only to half-a-second, rolljam attacks can also be prevented (SR2 & SR3)
- The fob decrypts the received cipher message to get the token. It sends the token back along with the car-id and feature information after encrypting the whole message using another key.
    - Use of AEAD encryption and decryption, ensures the authenticity, integrity and confidentiality of the entire process. This ensures only car & paired fob are involved in the process. (SR1)
    - The encrypted token must be sent to the car within threshold time.

- Car after decrypting the message verifies and validates the token and car-id and if found correct, the unlock and feature secrets are read from the eeprom are sent to the host machine.
- Symmetric Encryption is used with keys that are pre-shared to both car and fob during firmware build-time itself.



Pairing Fob

- The host machine sends a pairing signal to both paired and unpaired fobs. It also sends the pairing PIN to the paired fob.
- Paired fob after verifying the pairing PIN sends the pairing pin, car_id, packaging key, encryption_keys, to the unpaired fob after retrieving it from EEPROM. (SR4 & SR5)

Package Feature

- During the firmware build-time of the paired fob, the encryption key to be used in the packaging of the features is shared to both host tools and the paired fob.
    - The package-encryption key is unique for every Car-ID.
- Using the package encryption key, the package is encrypted for the given car-id and feature-id.
    - AEAD encryption is used to ensure the integrity, and authenticity of the package to ensure it is from the factory only and is not tampered. (SR6)

Enable-Feature

- The encrypted package is sent to the paired fob, where it decrypts the package using the pre-shared key and after performing checks regarding the authentication and car-id comparison enables the features if possible.

Helper Utilities

This section gives an overview of the file structure with a brief description of the implementation wherever required.

random.*

- Contains implementation of random number generator. TM4C123GH6PM doesn't have hardware RNG. This increases the challenge in implementing a RNG with high entropy.
- The seed for the srand() function is created using a combination of
    - Initial values in SRAM at the startup.
    - A random salt generated while building the firmware that is unique to each car/fob is added to increase entropy.
- The value from rand() is combined with System Tick to generate the random number. This ensures that even if the output from rand() is predictable, it is difficult to attain the same random number precisely.

ascon.* (dependencies: word.h, permutations.h**)**

- ASCON-128 with AEAD(Authenticated Encryption with Associated Data) is used as the encryption scheme in our implementation. NIST has decided to standardize the Ascon family for lightweight cryptography applications.
- ASCON is chosen because of its high cryptanalytic security, lightweight, robustness, allows interleaving (for 32bit), efficiency in software, and side-channel countermeasures.
- For details regarding the implementation of the scheme, please refer to this document.

uart.*

- handles uart communication
- timeout for uart read is implemented with the help of time.*

time.*

- implements functions for time measurement using System Tick counter

delay.*

- Has functions that can result in different delay times

brownout.*

- used for brownout prevention.
- Whenever brownout/tampering is detected, system resets itself.

Generating Secrets

Car

- Gen_secret.py generates 12 pairs of keys for unlock process and 1 more package encryption key. The keys are stored in the host secrets which will be then copied to the paired fob firmware
- The unlock keys are stored in the EEPROM file of the car firmware.

Fob

- For the paired fob, gen_secret.py retrieves keys stored in the host secrets and modifies the EEPROM file to store this data along with the car id, car pin, and package encryption key.
- For the unpaired fob, garbage data is written to the EEPROM.

A pair of keys will be used during the unlock transaction between the car and the fob. The car device will use the first key for encrypting token while sending the data to another fob and the fob after decrypting the data will encrypt and send it to the car using the second key. Car decrypts it using the second key and ensures whether it received the same token or not. This ensures that both car and fob authenticate each other.

The keys and secrets are stored in EEPROM of car and fob. The read & write accesses are protected by a password which is also generated by the gen_secret.py of both car and fob. Apart from the keys and passwords, the gen_secret.py of both car and fob generates the ‘salt’ used in the random number generator.

Attack Prevention Measures

Zeroing of stack

- Overwrites the temporary variables being stored in SRAM stack during the execution of a function, once they are not required any further.
- Overwriting will prevent the attacker from getting the actually data of the keys, even if they are able to read the SRAM.

Stack execution prevention

- SRAM is rw only (Earlier it was rwx)
- Prevents any malicious code from being executed in the process stack

Random delays added

- This introduces misalignment while measuring traces. Misaligned traces are difficult to synchronize making side channel and fault injection attacks difficult to implement.
- Timing attacks are difficult to achieve since accurate measurement of time is prevented because of random delays

Brownout protection

- Any brownout/tampering with voltage or PLL will result in system reset.

Use of Precision internal oscillator

- Fault injection through clock glitching is prevented since precision internal oscillator clock signals are not available externally.
- Use of Main oscillator would require an external oscilloscope which is prone to tampering.

EEPROM Protection

- Individual Password protection is provided to every block of the EEPROM containing key, secrets, etc. for both car and fob.
- This ensures that only corresponding function responsible for accessing an EEPROM block can access only that particular EEPROM block. This will prevent any unauthorised reading from the EEPROM

Timing attacks in array element comparison

- Instead of breaking the loop, A flag will be used to store the result of the comparison
- This will prevent attacker to guess the correct PIN using timing attack
- Adding random delays will also help in dealing with this attack

Arrays of fixed of size - preventing stack smashing attack

- Only way in which the input is provided is UART, and the implementation is done in a way in which all the communication through UART is of fixed size
- This will help in preventing any stack smashing attacks