Spartans
Design Document
Version 2.0

2023 MITRE Collegiate Embedded Capture The Flag Competition

General Features

EEPROM

We have 0x700 bytes of EEPROM available to us, which we use to store data for both fob and car devices. The last 0x100 bytes of EEPROM is reserved for an unlock message and a start message for each of 3 features.

Entropy

Our firmware will use entropy that is initially programmed into the flash by the host tools at build time. On booting, it uses the established entropy to regenerate new bytes of entropy using a Cryptographically Secure Random Number Generator.

Public Key Cryptography

We use public key cryptography, both for signature-verification to ensure integrity and authenticity of packages, and for challenge-response to ensure authorization for unlock.

Each car has a public key in EEPROM. The corresponding private key is stored in every fob authorized to unlock the car. If it is a fob that was created as a paired fob at build-time, then the private key is stored in the generated EEPROM. When an unpaired fob completes the pairing process, it is given the private key and stores it in its flash memory.

Additionally, the manufacturer has a private key used to sign feature packages, and each car is given the corresponding manufacturer public key to verify them.

For our public key cryptography implementation, we use Elliptic Curve Cryptography, specifically the Elliptic Curve Digital Signature Algorithm with SHA-256 as a hash function.

Build and Key Management

There is a manufacturer keypair `(host_privkey, host_pubkey)` stored as PEM files in the secrets volume in the system. For each car, there is a car keypair `(secrets[n].car_privkey, secrets[n].car_pubkey)`, where `n` is the car id. These keypairs are stored in PEM format in a JSON file called “car_secrets.json” in the secrets volume.

During the build process for a car, the car is given the manufacturer public key and its own car public key. Thus car `n` is given (`host_pubkey, secrets[n].car_pubkey`).

During the build process for a paired fob, the fob (corresponding to car `n`) is given the car private key `car_keypairs[n].priv`.

During the pairing process for an unpaired fob, the paired fob shares the car private key `secrets[n].car_privkey` if the correct PIN is entered by the host.

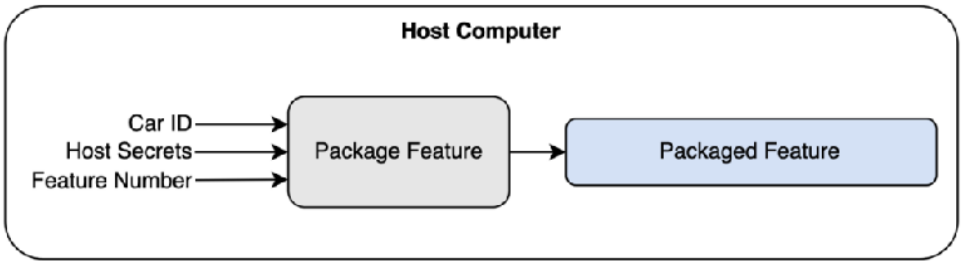
Key Processes

Packaging Features

Features are packaged for a car by the manufacturer. Thus, it uses the manufacturer's private key `host_privkey` to sign the package. The signature is run on the concatenated bytes of the car's public key as well as the feature number, but the package only contains the feature number as a single byte followed by the signature. The car public key is not included to optimize for size and simplicity, as the car can fill in its own public key on the other end to verify the signature.

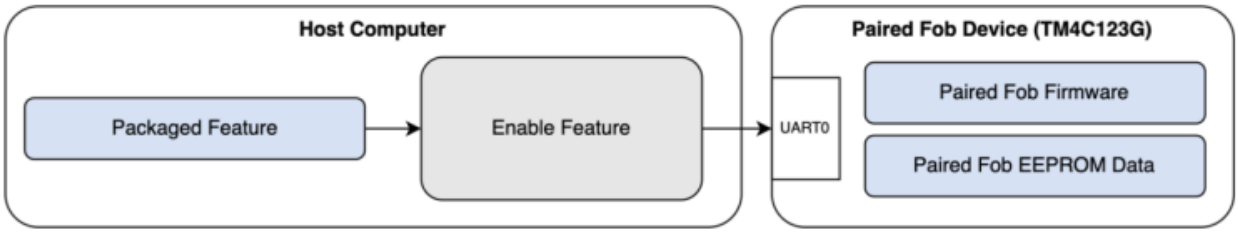
The following pseudocode represents the packaging process:

```
signature = host_priv.sign(secrets[n].car_pubkey || feature_num)
package = feature_num || signature
```



Enabling Features

To enable a feature on a fob, we store the the package in the fob's FLASH if its feature number corresponds to a feature message: in other words, if the feature number is either 1, 2, or 3. We store only the signature portion of the package, as like the car public key, the feature number is implied from context. When the fob later requests this feature on the car, the car will be able to verify it by reconstructing the package with its public key and the implied feature number from the index.



Paring and PINs

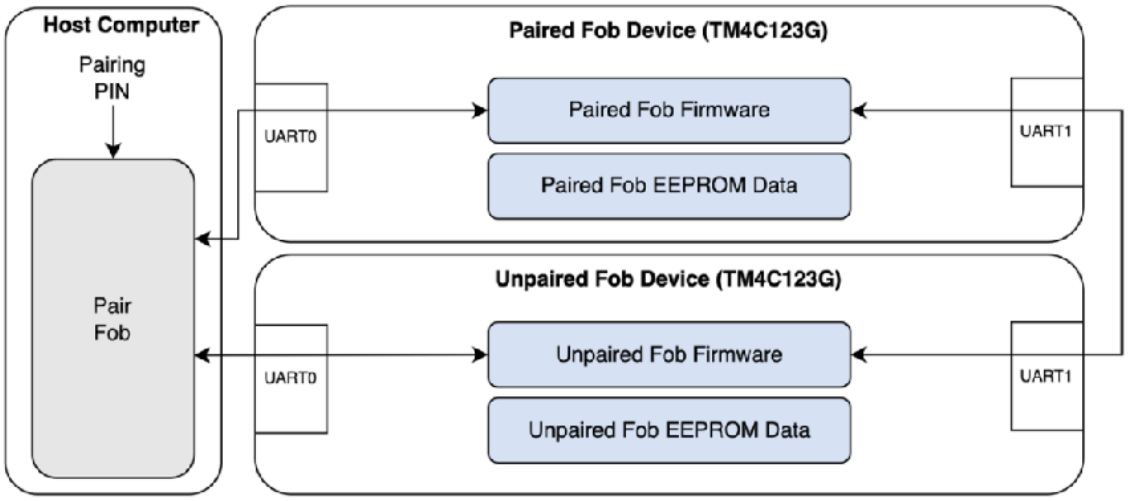
Each paired fob device (pFob) stores a pairing PIN in EEPROM, along with the car private key used to unlock the car it is paired with.

An unpaired fob device (uFob) does not have either of these bits of information.

When a fob pairing is attempted, the pFob uses integer comparison to check whether the correct PIN has been entered by the computer.

- On an incorrect pairing PIN attempt, we time out for 5 seconds to prevent brute-force attacks.
- On a correct pairing PIN attempt, the pFob sends the PIN and private key (`secrets[n].car_privkey`) to the uFob over the UART connection, and the uFob stores these in flash. The former uFob is now considered a pFob for all intents and purposes, and can no longer be paired with a different car in future.

There is no need for a paired fob to later become an unpaired fob, and thus an unpaired fob must have been originally unpaired, while a paired fob may originally have been paired or unpaired.
We make this distinction by identifying pFobs, uFobs, OG_pFobs, and OG_uFobs, where every uFob is an OG_uFob, but a pFob may be either an OG_pFob or an OG_uFob.



Unlocking

The process for a secure paired fob device unlocking a secure car device goes as follows:

- A keyfob requests the car to unlock.
- The car creates a challenge using a randomly generated nonce and sends it to the fob.
- The fob must then sign the challenge using the car private key,
  providing a valid response and sending it to the car.
    - This response also includes the set of feature packages currently held by the fob
- The car will verify the response using the car public key,
  and verify the feature packages using the manufacturer public key

If the response is determined to be valid, the car will unlock, printing the unlock message. Then, it will begin the start car process, printing the feature message corresponding to each enabled feature.

When the car has issued a challenge and is accepting responses, this step implements a fast timeout so that it will meet the functional timing requirements even if the communication is interrupted and prevent responses from being later replayed.