Design Document

System Overview

The challenge for this year's eCTF competition is to secure a system of car and fob(s). The fob's firmware run's on a board while the car's runs on the other. The car and fob(s) can communicate with each other via a UART interface. Host tools connect both directly to the car and fob and provide various functions such as unlocking the car, enabling features, packaging features and pairing unpaired fobs. Host tools communicate with the car and fob(s) using the host UART.

Each of the devices in the system (Car, paired/unpaired fob) has an EEPROM which can be programmed during build time. To ensure that the communication between the car and paired fob is secure, we decided to encrypt the conversation. The key for encrypting and decrypting messages is stored in the EEPROM of the car and the paired fob. This key is generated randomly by host tools during build time. The paired fob will have the key location in the EEPROM populated with the shared key while unpaired fobs will not.

Implementation Details For Each Component:

Car:

Generating car secrets (gen_secret.py):

- For each car (specified by its id), the script will generate a specific EEPROM txt file associated with its id called <car_id>_sec_eprom.txt. This file contains a randomly generated 16-byte key which is used to encrypt the communication between the car and the respective paired fob. When the car firmware is loaded onto the board, this key is programmed into the board's EEPROM starting at byte 0.

- The python file will also generate a header file in the car's out directory called secret.h. In this header file, car's secret location in the EEPROM will be defined as a Macro, the file also consists of car's ID as a macro named CAR_ID.

- This python script will be called during the Makefile process, when using build car-fob command in the host tool.

Car Firmware (Firmware.c, Boardlink.c, Uart.c:):

- The Uart.c establishes a communication between the car and the host, mainly for displaying the car's unlock message, which is decided by the user at build time. This file is the same as the 2023-insecure-example with no code or functions changed.

- The boardlink.c file is a function supporting file for car's communication with a fob. The setup, send_board_message, and receive_board_message stay the same. However, we modified the receive_board_message_by_time with a timeout option where it can either be a positive integer (as the timeout value) or negative (indicating no timeout needed). The file also provides functions to generate an AES encryption key structure from a 16 bytes key as well as a decryption function for decrypting any incoming messages from the fob that requests unlock.

- The firmware.c is the main firmware for the car. After setting up EEPROM, HOST UART, and BOARD UART, the program goes into its infinite while loop, periodically emitting an incremental nonce every cycle to the BOARD UART, in clear text. After emitting the nonce, the car goes into its unlock sequence by waiting for incoming fob messages with a 1 second timeout. If such a message was received, the car will decrypt this message, checking nonce's validity with its current nonce and deciding whether or not to unlock the car. Should the car be unlocked, the car will go into the startcar() function and decipher the feature information, printing it out on the console. Otherwise, the car restarts its main loop cycle.

Encryption:

- The car uses tinycrypt's AES 128 encryption algorithm. The header file is named "aes.h" with dependencies of "aes_encrypt.c", "aes.decrypt.c", "constants.h", "utils.c", and "utils.h".

Fob (Unpaired):

Encryption:

- All fobs share a key amongst themselves and the host tools which is the global host tools key. The global key is defined in the secret directory, in a file called "global_secret.txt" that will be generated during the build-deployment phase. In this file, a 16 byte AES key will be written into fob's EEPROM's starting from the

16th byte position. This key is written into all fobs built by the host tools, including unpaired fobs.

- Like the paired fob, unpaired fob will also have 16 bytes of data in its EEPROM starting at position 0x0 to 0x10. Unlike the paired fob, these 16 bytes will be stub/fake '\0' bytes acting as padding or position holders. In other words, these bytes will and should not be used to perform any required functions.

Pairing (firmware.c *pairFob(*fob_state_ram)* function) :

- The unpaired fob acts as a passive message receiver during the pairing process. Once the host tool initiates a pairing process by issuing command "pair" (encrypted) via Uart,the unpaired fob goes into a listening state to wait for the paired fob's message. Once such a message is received, the unpaired fob decrypts the message using its key and extracts both the AES key (first 16 bytes of the decrypted message) and the associated car's feature information (the 17th byte). The feature will be then written into the fob's state ram and saved onto flash whereas the AES key for fob-car link will be stored in the first 16 bytes of the EEPROM, replacing the old stub.

Fob (Paired):

Encryption:

- Paired fob has 16 bytes of data in its EEPROM starting at position from 0x00 to 0x10, which is for establishing secure communication between itself and its paired car. From 0x10 to 0x20 (next 16 bytes), the paired fob also has a global, secret key shared with all other fobs (regardless of status) for pairing. Details for this AES key can be found in the unpaired fob's encryption section.

Pairing:

- The pairing process starts when the host tool initiates the "pairing" command alone with the pairing pin, sent via UART. The paired fob will check if the message is valid by checking the pairing pin, and if it is it encrypts the feature information, followed by the car-fob AES key using its global Host-fob AES key and sends it out to the unpaired fob via UART1.

Unlock:

- The fob state ram stores the fob's state in flash. With the fob state ram, we can check if the fob is a paired fob. If the fob is paired, the fob starts requesting unlock. The program first extracts the feature information from the fob state ram.

It uses the receive_board_message_by_type function to receive a nonce from the car. Should the nonce not be received, the fob will go to DDOS and has to be manually reset by pressing the reset button. Otherwise, the fob encodes the nonce using its shared AES key with the car and sends the message back to the car (letting the car check the nonce's validity and decide whether to unlock).

Feature Information:

- Unlike the 2023-insecure-example, we simplified the feature information for a car-fob pair. The information now takes up only 1 byte and should be read in a purely binary fashion. Each bit corresponds to a feature where 0 means that feature is disabled and 1 means that feature is enabled. For example, a car with feature 1, 3 enabled should have a feature information of 00000101 or 0x05.

Host Tools:

Constant.h:

- Tinycrypt interface to constants, mainly just Header Macros for AES library. This file should not be modified or removed otherwise could result in failure of host-tool compilation of the Ctype library.

Encrypt.c:

- This is the library function that can be called by the python module Encryption.py. It serves to encrypt and decrypt a byte string at C level.

Encryptlib.so:

- A shared library generated/compiled from encrypt.c. The compilation process will be done in the makefile. Hence, it's operating system independent.

Encryption.py:

- This is an AES encryption tool for encrypt and decrypt messages. The encryption algorithm is imported from encryptlib.so using ctypes. The key is saved in a JSON file, and will be read when Encryption class is called. The encryption function first encodes the message into plain text and encrypts it by calling encryption in encryption lib.

Pair_tool.py:

- This tool will prompt the user to enter the 6 digit pairing pin. The tool will then compute the hash using python's default hash() function. During build time, the hash of this pin is already computed and stored in a secret file only accessible to host tools. The set of hashes for all car-paired fob pairs are stored in that file. If the hash exists in that secret file, the tool relays an encrypted message to the paired fob, allowing it to share secrets and features with the unpaired fob.

Enable_tool.py:

- The enable tool ensures a packed feature package is transferred to the fob (paired) via UART0. This tool reads the packed binary file and sends all its data to the fob's board. It stays the same as the original enable tool, except all the message that it transfers to the fob is encrypted, during the package process.

Package_tool.py:

- The package tool generally stays the same with 2023-insecure-example. To accommodate our new feature structure, as referred to Feature Information section in the paired fob documentation, we expanded the argument to the following structure:

```
python3 -m ectf_tools run.package
--name <SYSTEM_NAME> # Tag name of the ectf Docker image
--deployment <DEPL> # Name of the deployment
--package-out <PACKAGE_OUT> # Path to output directory
--package-name <PACKAGE_NAME> # Name of the packaged feature binary file
--car-id <CAR_ID> # 32b unsigned ID number for the car
--feature-number1 # 1 or 0 to enable or disable feature 1
--feature-number2 # 1 or 0 to enable or disable feature 2
--feature-number3 # 1 or 0 to enable or disable feature 3
```

- The user now has to manually enter their decision for what feature to enable or disable. In the python file call, these 0 and 1 indicators will be converted to the byte as discussed in the Feature Information section. Subsequently, the python script will create a packaged binary file with car id and feature information.
- Package feature accepts 3 different arguments. feature 1, feature 2, and feature 3 are passed as three arguments; if 0 is passed for any of the features, the feature is not enabled. If a non-zero number is passed, the feature is enabled. The package binary is encrypted by (host-fob or car-fob key) before being transmitted over UART
- The package feature tool will then encrypt the whole message and store them into a binary file. The encryption will be done by calling Encryption.py. Since package_tool has access to the car's AES key, It will use a specific car's (identified by its ID) AES key to encrypt the message.

Unlock_tool.py:

- The unlock tool remains the same as how it's implemented in 2023-insecure-example. During a car unlocking process, the host computer will be connected to a car via UART_HOST to issue commands for both car and a valid paired fob. The process then is transferred to the car-fob pair as described in the previous sections. When a car successfully unlocks, the message sent from the unlocked car will be printed on the host computer's terminal.