Spartans

Michigan State University

Adrian Self, Udbhav Saxena, Felipe Allevato, Michael Umanskiy
Advised by: Dr. Qiben Yan, PhD

April 24, 2023

Design Overview

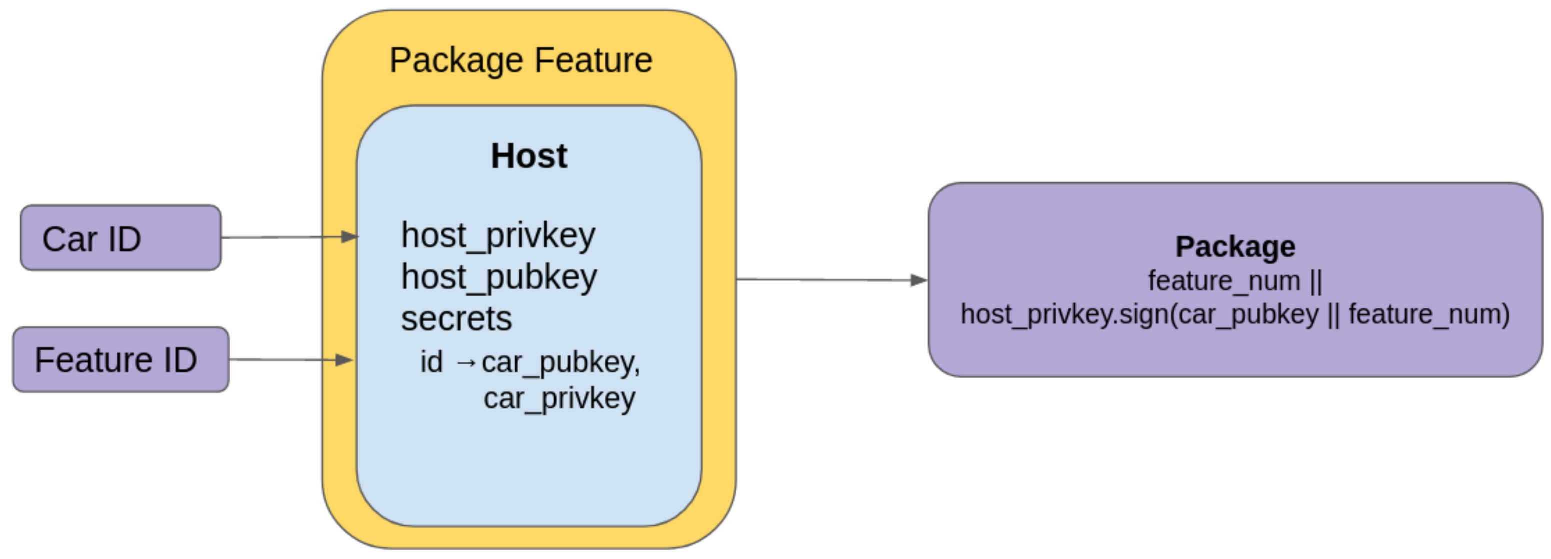
Our design uses a public-key model. Each car has a public key in EEPROM, while the corresponding private key and pairing PIN are stored in every fob authorized for the car. The car uses its public key to verify that unlock requests come from an authentic paired fob device.

Additionally, the manufacturer also has a private key used to sign feature packages. Every car has the manufacturer public key, which it uses to verify the authenticity and integrity of requested features when the car is started.

Defensive Highlight

One defensive measure we would like to highlight in our design is the use of digital signatures with elliptic curve cryptography. The goal is to establish authenticity and integrity, so that attackers can neither forge nor modify protected data.

One way we used signatures was to securely package features for a car. This is represented by the following process, where the `host_privkey` is the manufacturer private key:



Package Feature

Host

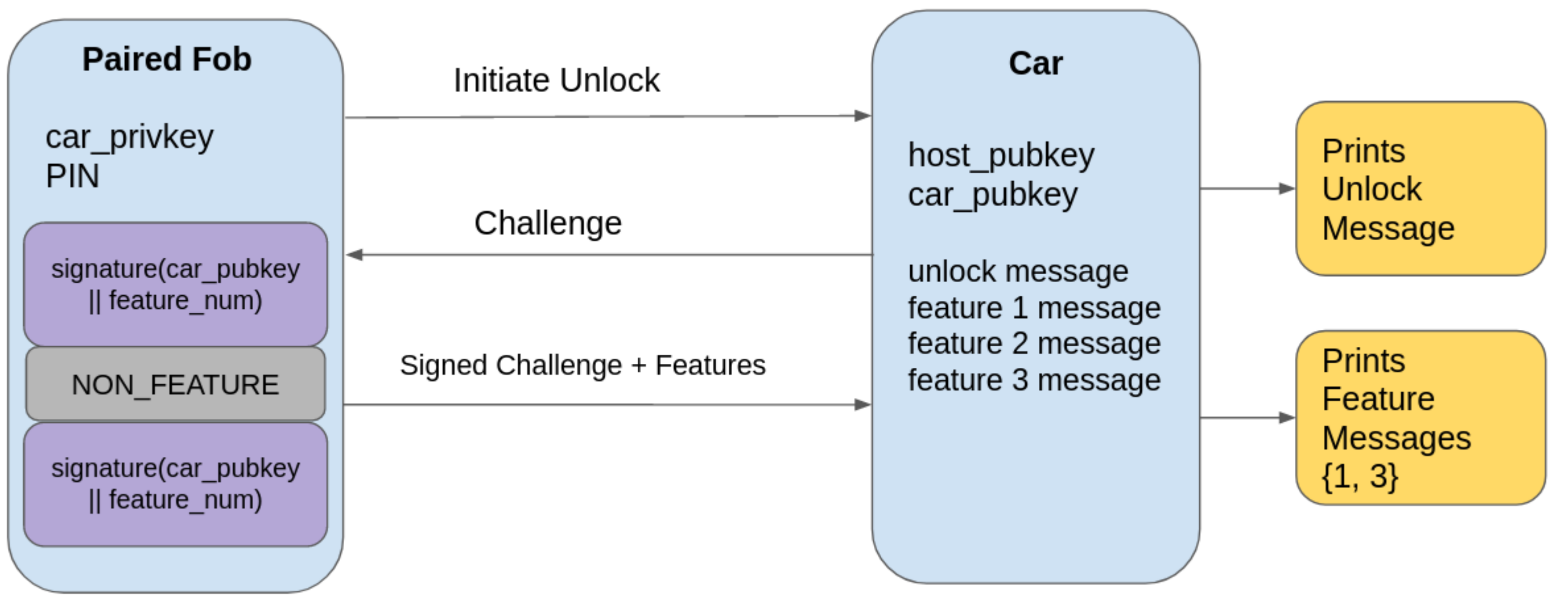
host_privkey
host_pubkey
secrets
id → car_pubkey, car_privkey

Car ID

Feature ID

Package
feature_num ||
host_privkey.sign(car_pubkey || feature_num)

We also used ECDSA for the fob to sign the unlock challenge issued by the car in order to validate unlock requests:



Paired Fob

car_privkey
PIN

signature(car_pubkey || feature_num)

NON_FEATURE

signature(car_pubkey || feature_num)

Initiate Unlock

Challenge

Signed Challenge + Features

Car

host_pubkey
car_pubkey

unlock message
feature 1 message
feature 2 message
feature 3 message

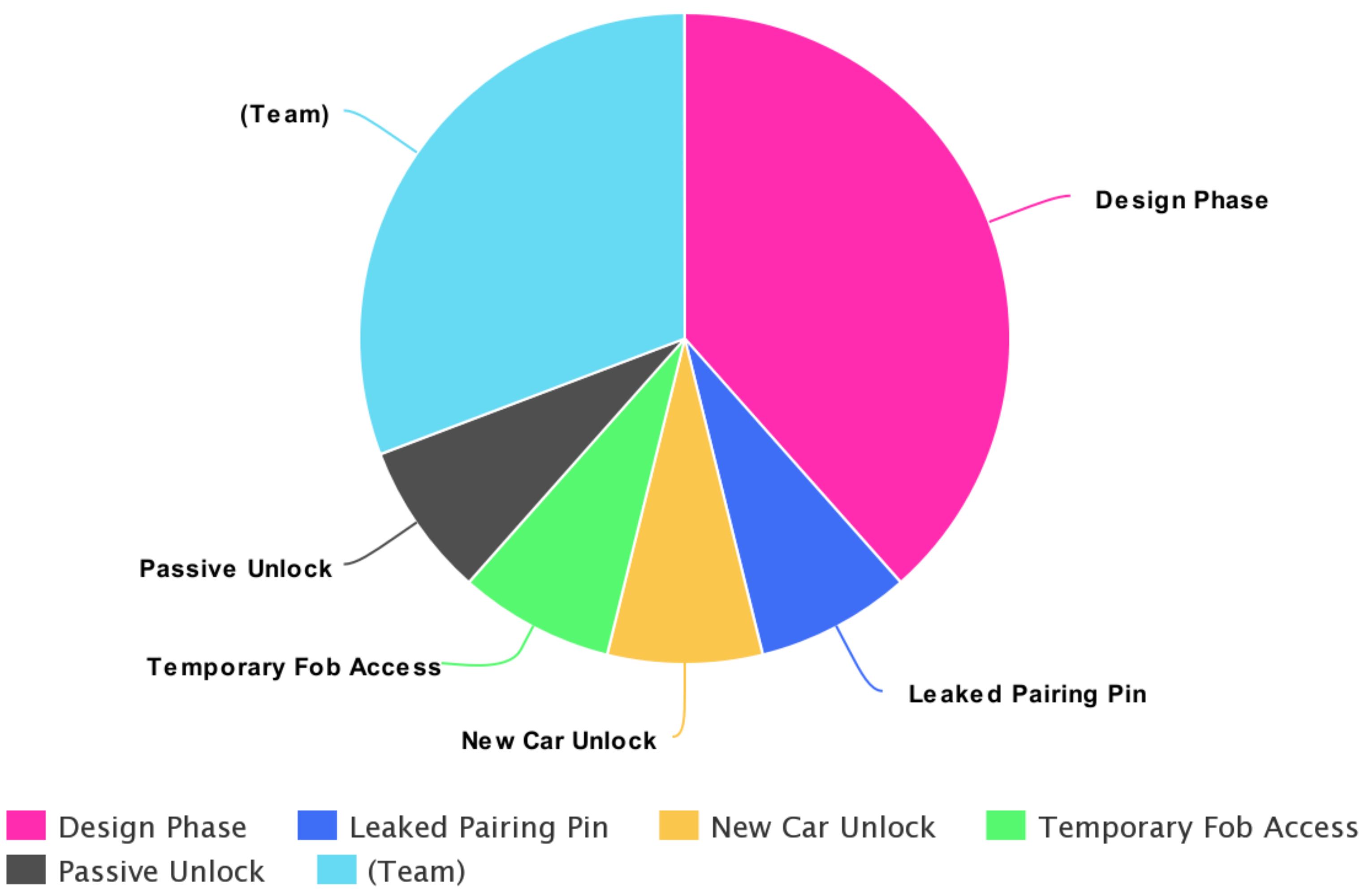
Prints Unlock Message

Prints Feature Messages {1, 3}

This defensive measure proved extremely effective, preventing other teams from enabling unauthorized features or unlocking cars without an authorized key fob. As a result, we earned many defensive points, and ended with one of the highest defensive scores out of all the teams.

In the future, we would improve the entropy source of the challenge generation to further decrease the possibility of replay attacks.

Figure 1: Solved Challenges Breakdown



Offensive Highlight

During the attack phase, we utilized a combination of automated vulnerability detection, analysis of design documents, and verification of implementation.

Some teams had a vulnerable design even on paper, such as having a deployment-wide secret which we were able to extract from the `fob0` or `car0` devices.

Other designs looked good on paper, but had flaws in their implementation code, introducing vulnerabilities. For instance, we identified a buffer overflow vulnerability caused by reading unbounded data over UART. An example vulnerable function is shown below:

```
uint32_t uart_readline(uint32_t uart, uint8_t *buf) {
    uint32_t read = 0; uint8_t c;
    do {
        c = (uint8_t)uart_readb(uart);
        if ((c != '\r') && (c != '\n') && (c != 0xD)) {buf[read++] = c;}
    } while ((c != '\n') && (c != 0xD));
    buf[read] = '\0'; return read;
}
```

This function can be exploited by sending more data than the size of the buffer, overflowing the end of the buffer and compromising security. The impact includes attacker code execution by overwriting the link register, with other impacts including PIN bypass and leaking sensitive data.

In order to prevent this vulnerability, a secure device would only accept as much data as it has room in its buffer to store. This can be accomplished by defining a fixed-width struct to represent the format of the data, such as the structs used in our team's design, and only reading data up to `sizeof(struct)`. By implementing this fix, the buffer overflow vulnerability can be eliminated.

References

1. Code Repository
2. Design Document
3. Process & Entity Diagrams
4. "The Fundamentals Of An Ecdsa Authentication System" by Analog Devices
5. "ECDSA: Elliptic Curve Signatures" by Dr. Svetlin Nakov, PhD
6. "Stack Buffer Overflows" by Azeria Labs