Part1: Defensive Write-up

When designing our secure system, our team sought out industry best practices to keep our data safe. For the encryption of our data, we utilized AES symmetric encryption. We felt comfortable using a symmetric encryption algorithm because we understood that the registration process with the server was a hard-wired connection, and therefore the distribution of keys at registration would not be vulnerable to attack. Specifically, we used CBC (Cipher Block Chaining) AES encryption over ECB (Electronic Codebook). CBC is considered a more secure algorithm, as the encryption of one block of data influences the encryption of the following blocks of data, making it impossible to modify the encrypted data without changing the rest of the data in the message upon decryption. To authenticate our messages, we append to the encrypted data a hashed message authentication code (HMAC). We generated our HMAC by running our encrypted data through a SHA256 algorithm. In this way, a receiving device can generate its own HMAC based on the encrypted data, and compare it with the appended HMAC received, to determine if the message is authentic.

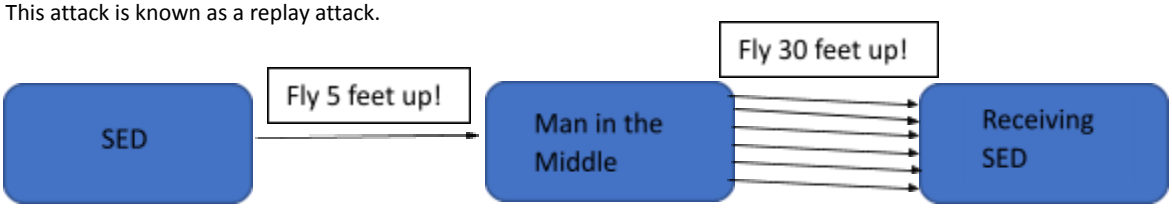
Based on our research and limited knowledge, and trusting our encryption and hashing library (tinyCrypt by Intel), these measures would seem to secure our design. However, one challenge we faced was how to prevent replay attacks. If an authentic message was sent repeatedly, how would our SED know that it was not authentic. Ultimately, our design was not successful in preventing replay attacks, but we think our approach does have merit and possibly could have been effectively implemented. After struggling on how to include a timestamp on the message using a microcontroller without an operating system or system clock, we had the idea to store the previously received HMACs in an array and compare new messages to those. We also added a message counter to each message. In theory, even the same message sent with only a different message count should generate an entirely different HMAC. We stored only authentic HMACs in an array, and upon receiving a new message, would check if the calculated HMAC was the same as any of those stored. If it was the same, the device would discard the message.

This may be an effective measure, but I think we failed to appreciate the ability of attackers to utilize the man in the middle interface to exploit our design. Our replay protection only stored 16 of the most recently received HMACs. In an attack deployment that runs for hours, it is only a matter of time before an attacker has observed and recorded 17 authenticate messages intended for a device. By spamming those messages in a loop, the receiving device will not have the previously received HMAC in memory and it will treat these replayed messages as authentic.

To make this design more secure, we could have expanded the amount of HMACs that were saved by the device. However, the same vulnerability exists in that once an attacker has recorded one more authentic message than the size of the previously received HMAC array, then spamming those messages in a loop would yield the same results. Also, there is a question of performance as the device is doing so much comparison before authenticating a message. One augmented security measure in addition to the stored HMAC array would be an array of message counters corresponding to each device ID. In that way, even if a replayed message is not detected by comparing to the HMAC array, if the message count in the message was less than or equal to one previously received, then the receiving device would know to discard that message.

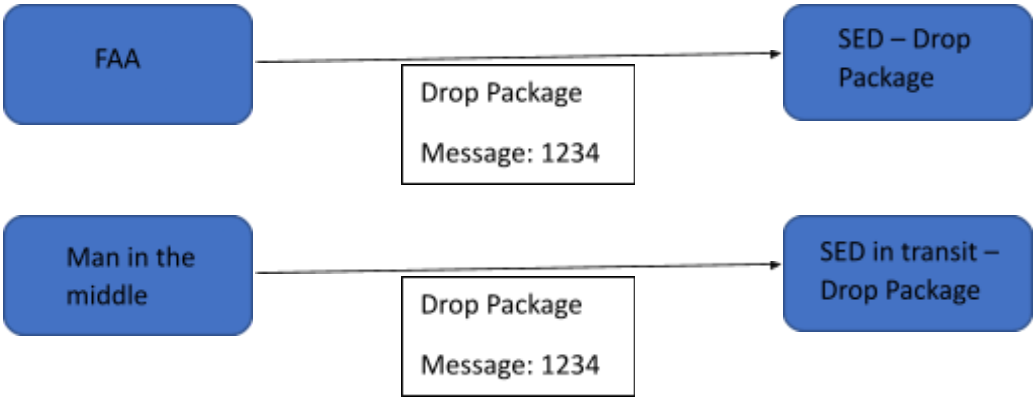
Part2: Offensive Write-up

The easiest flag that our team was able to acquire was the no-fly-zone flag which is done by forcing an SED to fly above its intended air space. We decided the best way to perform this attack was to perform our man in the middle attack and monitor the communications that are being sent out by enemy SEDs and we concluded that the only reason an SED should communicate with another would be to in the case that they are flying at the same altitude, and one has to fly up more to avoid collisions. With that thought in mind, we intercepted that exact communication and the message that was sent and forced The receiving SED to accept that same message multiple times, so for example instead of flying 5 feet above it was told to fly 5 feet countless amount of times so that it would fly around 100 feet above now and thus giving us the No-fly-zone flag. This particular attack worked on many times but not all, one of the ways to combatant this would be to check how often a message from SED to SED be sent and received and if it's more then 1 or 2 within seconds then ignore the rest of the communications after the first one. This attack is known as a replay attack.

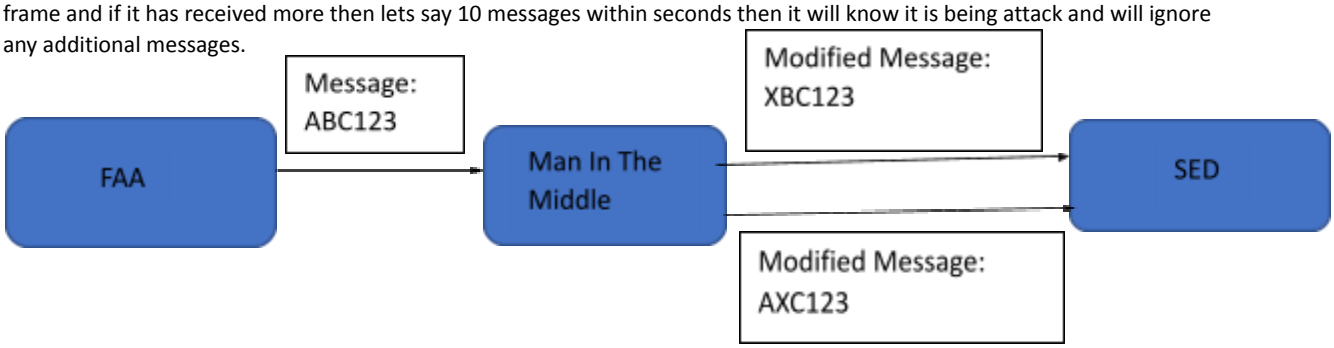


One of the unsuccessful attack attempts was to acquire the Drop Package flag which is done by making an SED think it is being told to drop whatever package it is holding by the FAA but in reality it is being told to drop it by the attack when it is still

in transit. Our understanding was this action would only normally be taken place where the SED would arrive at its destination and the FAA would send the command to drop the package so we monitored the communications to see what that message might look like so we can try and replicate it or understand it to get the SED to perform that action while in the air. Unfortunately We were not able to determine which message was the drop package or the arrive at destination one but our idea was once we figured out when or what it looks like we might be able to figure out what to do during our man in the middle attack to acquire that flag.



Our other successful attack was our Recovery flag attack, the way we manage to capture this flag a couple teams was we take any message that is communicated from the FAA to SED and we start by changing one byte at a time from the encrypted message and sending the modified message to the SED for every change we made. For example, if the message was 128 bytes in length, then we sent 128 modified messages To the SED with each changing the next byte until we received the Recovery flag. This method did not work on every team but it did on a few and one of the counter measures that could of taken place to this attack would have been for the SED to check if the messages it has been receiving all came within a certain time frame and if it has received more then lets say 10 messages within seconds then it will know it is being attack and will ignore any additional messages.



Below is the script we used to generate our no-fly zone and recovery flags. Once we capture a direct message from one device to another, we spammed it, each time changing either the sender ID, the data, or the receiving ID.

```
if src > 13 and tgt > 13 \\ if uav to uav message
for i in range (3,60):
        for j in range(0,len(data)):
        data2 = data[:len(data) - (1 + j)] + bytes(1) + data[len(data) - (0 + j):] \\ corrupt data and change src id
         mitm_hdr = struct.pack('<2sHHH', b'MM', tgt, i, len(hdr + data))
        hdr = struct.pack('<2sHHH', b'SC', tgt, i, len(data))
        sock.send(mitm_hdr + hdr + data2)
        sock.send(mitm_hdr + hdr + data)
        mitm_hdr = struct.pack('<2sHHH', b'MM', tgt, src, len(hdr + data)) \\ only corrupt data
        hdr = struct.pack('<2sHHH', b'SC', tgt, src, len(data))
        sock.send(mitm_hdr + hdr + data2)
        sock.send(mitm_hdr + hdr + data)
        mitm_hdr = struct.pack('<2sHHH', b'MM', i, src, len(hdr + data)) \\only change src id
        hdr = struct.pack('<2sHHH', b'SC', i, src, len(data))
        sock.send(mitm_hdr + hdr + data2)
        sock.send(mitm_hdr + hdr + data)
```