Part 1: Defensive Write-up

In order to defend against replay attacks—the sending of messages by replaying previously sent messages—we planned to implement an internal clock and timestamp system.

In our original plan, when a device was added to the network, it would be sent a global system time in milliseconds. The device would then initialize its own internal clock using that global system time and increment its internal time by one for each passing millisecond. Using that internal clock, we would timestamp each outgoing message with the device's current internal time and encrypt it. Each receiving device would store the timestamp of the last message sent from each other device. Using the stored timestamp of the most recently received message, each device would discard any incoming messages with timestamps that were sent at a time before the previous message was sent. Each message type would also have implicit time constraints that would be enforced by the receiving device. If the time delta was too short (i.e., the message was received impossibly soon) or too large (i.e., the message took too long to be received), then the message would be discarded to protect against replay attacks. This could be checked by comparing the difference in time between the timestamp in the message and the system time, as tracked by the receiving device.

We first implemented timekeeping using a timer triggered every millisecond using the emulated chip's clock frequency. After testing, however, we found that there was too much inconsistency in the device's internal clock frequency to yield accurate timing over the full duration of the test. To remedy this, we planned to modify our design to use the emulated chip's RTC (real-time clock), but we were unable to complete the modification within the project time constraints.

As a result, we were left with a system time that was prone to drift and were forced to remove the bulk of the timestamp-checking functionality from our final design, only keeping the check that the receiving device's current time was greater than the timestamp of the last recorded message from the sending device. The unsynchronized clocks are still used to generate message timestamp as a source of additional entropy in our message.

We believe that the successful implementation of this synchronized system clock would have allowed for our implementation to repel a large majority of replay attacks. While the time constraints are useful in their own right to enforce timing requirements, they would also allow us to perform checks on the number of messages received in a given timeframe. When coupled with a check for duplicate messages, this becomes a more robust means of preventing replay messages from being received. For example, if we limited our implementation to sending only a single message per timestep, then receiving multiple messages from a single device within that timestep would point to someone attempting a replay attack. Additionally, if an attacker then tried to space their messages out by that timestep, the timestamps would become outdated quickly, as there would be no way to update them within cracking the message's encryption.

Part 2: Offensive Write-up

The No-Fly Zone flags are acquired by forcing a device to fly above the maximum allowed altitude ceiling, which is most easily accomplished using a replay attack — the sending of previously sent messages. For the purposes of our attack, we only focused on when two devices had a height conflict (both flying at the same height in close proximity). If we saw two devices broadcast their position followed by a direct communication between the two devices, it was highly probable that the direct message was a “deconflict message”, telling the receiving device to fly higher to prevent a collision. By capturing this deconflict message, we were then able to replay it 100 times, tricking the receiving device into thinking that it needed to continue moving higher to avoid a collision. This would eventually cause the device to fly above the maximum allowed height, enabling us to capture the No-Fly Zone flag.

This repeated deconflict message attack did not require us to break encryption and was fast to execute because we only had to wait for the deconflict message to be sent between two launched devices. Then we could repeatedly send the target device the same deconflict message. If the devices did not have strong replay attack protection, they would be unable to know that the messages were being replayed by the attacker.

A potential defense against this attack would be a combination of strict timing enforcement, rate limiting, and duplicate message checking. This defense against replay attacks could be implemented as follows:

TIMING & RATE LIMITING

1. Maintain a precise internal clock in every device, synced across the network.
2. Each message should include a timestamp which is verified by the receiver to ensure that messages meet the timing requirements and constraints of the system.
3. Message sending should be rate limited to enforce specific timing requirements - such as limiting a single direct transmission to another device per X seconds. This limit should be tuned for the system requirements and precision of the clock.

DUPLICATE MESSAGE PROTECTION

1. Each device on the network, should maintain a list of the Y most recent message hashes from each other device.
2. After a targeted transmission is received and it has based the timing checks, the receiving device should then check the hash of the received message against the hashes of the past Y messages received from that device. Y is tunable for the system.
3. If a matching hash is found, the device would drop the message, as it must have been a duplicate. If the message is unique then its hash is added to the list.

Since the hashes would contain timestamps and the frequency of message sending would be limited, we could guarantee uniqueness of properly sent messages, more easily detect replayed messages, and defend against replay attacks. This defense could additionally be tuned to a specific system’s requirements by storing as much message history as desired and setting the message frequency to a reasonable limit for the system.