# 0xDACC

## Delaware Area Career Center

**Dillon D, Taylor Q, Diego N, Tony I, Cooper P**
**Advised by: Eli Cochran**

## Design Overview

A secure SAFFIRe design must ensure the following for all sensitive data:

- Confidentiality
- Integrity
- Authenticity

Our design accomplishes this with two main design elements

- Authentication passwords/signatures
- AES-128 encryption using CBC mode

## Defensive Highlight

One important feature we implemented was our method for protecting a configuration file, which goes as follows:

- A secret authentication password/signature is appended to the beginning and end of the configuration data
- This whole package is encrypted at once
- The passwords/signatures will be checked for authenticity/integrity
- Only hosts who provide a protected file which passes these test may be allowed to decrypt said file

Originally our design handled data protection differently, but this was changed due to timing requirements. Differences in our original design include:

- Configuration data was split into 1KB chunks with each chunk containing a single authentication password at the end
- Chunks were decrypted and verified by the bootloader without any host intervention

The current method for configuration protection is much faster, but less secure. Possible extra security measures we would have liked to include would be:

- Hashing to confirm the integrity of data
- Asymmetric key encryption to provide better security of signatures and hashes

## Offensive Highlight

We entered the attack phase at the very end of the competition, but had some ideas on potential attacks relating to large amounts of readback data being requested

- Readback overreach compromising data
- Readback overflow causing instability

### References

1. William F. Ehrsam, Carl H. W. Meyer, John L. Smith, Walter L. Tuchman, "Message verification and transmission error detection by block chaining", US Patent 4074066, 1976.

Fig. 1 Current Design

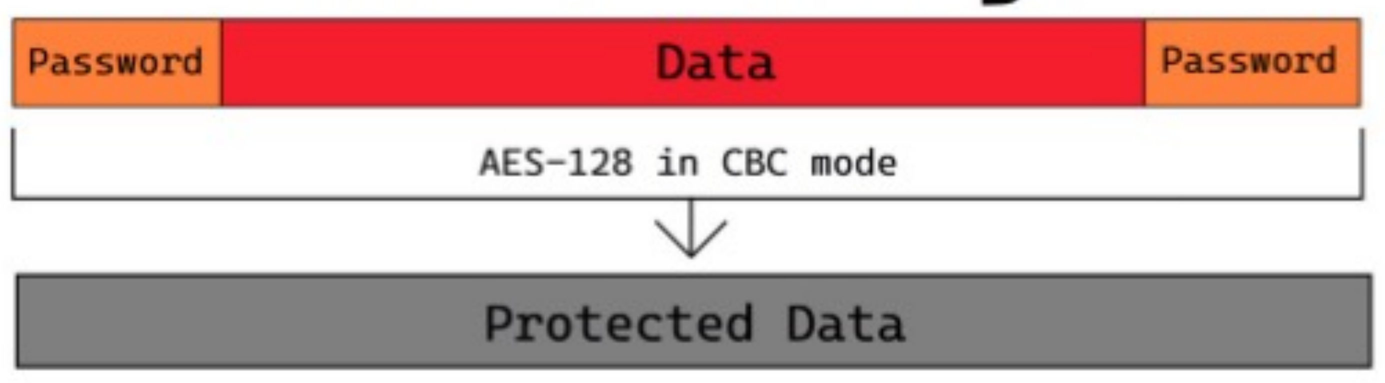


Fig. 2 Original Design

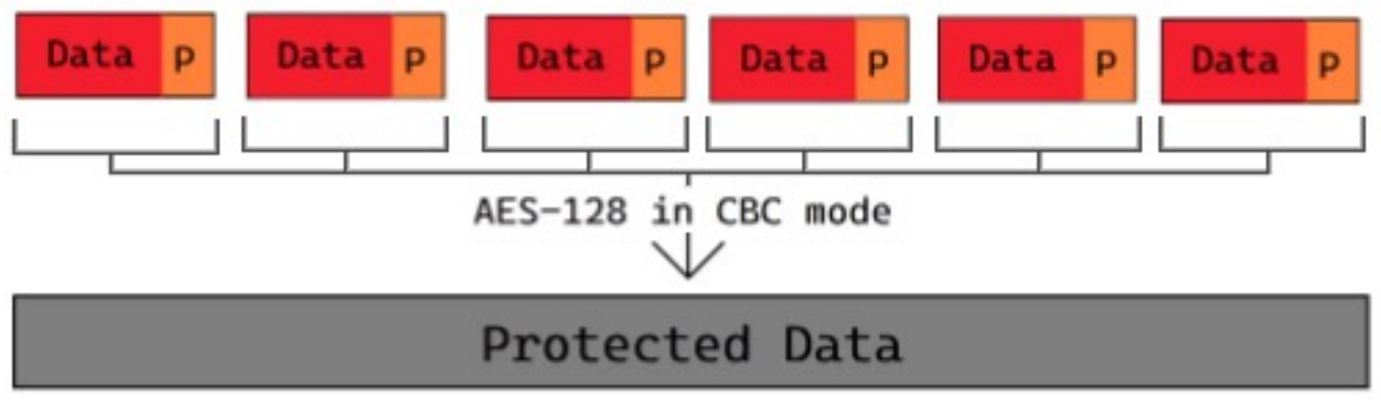


Fig. 3 Ideal Design

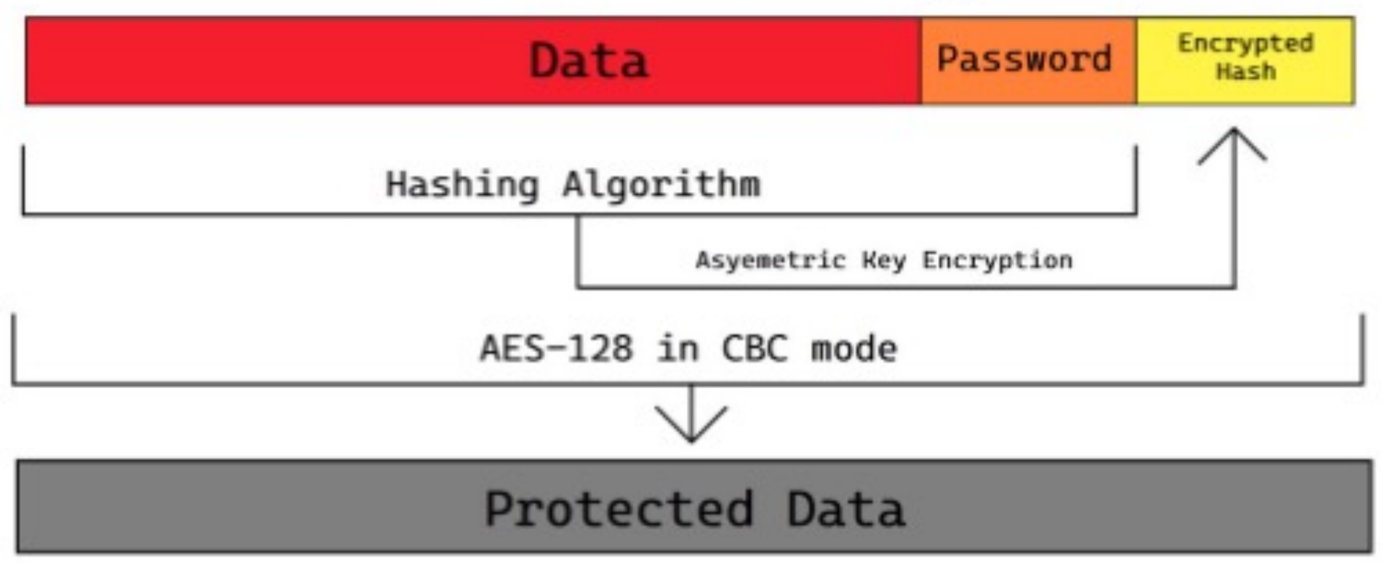


Fig. 4 Potential readback overflow attack

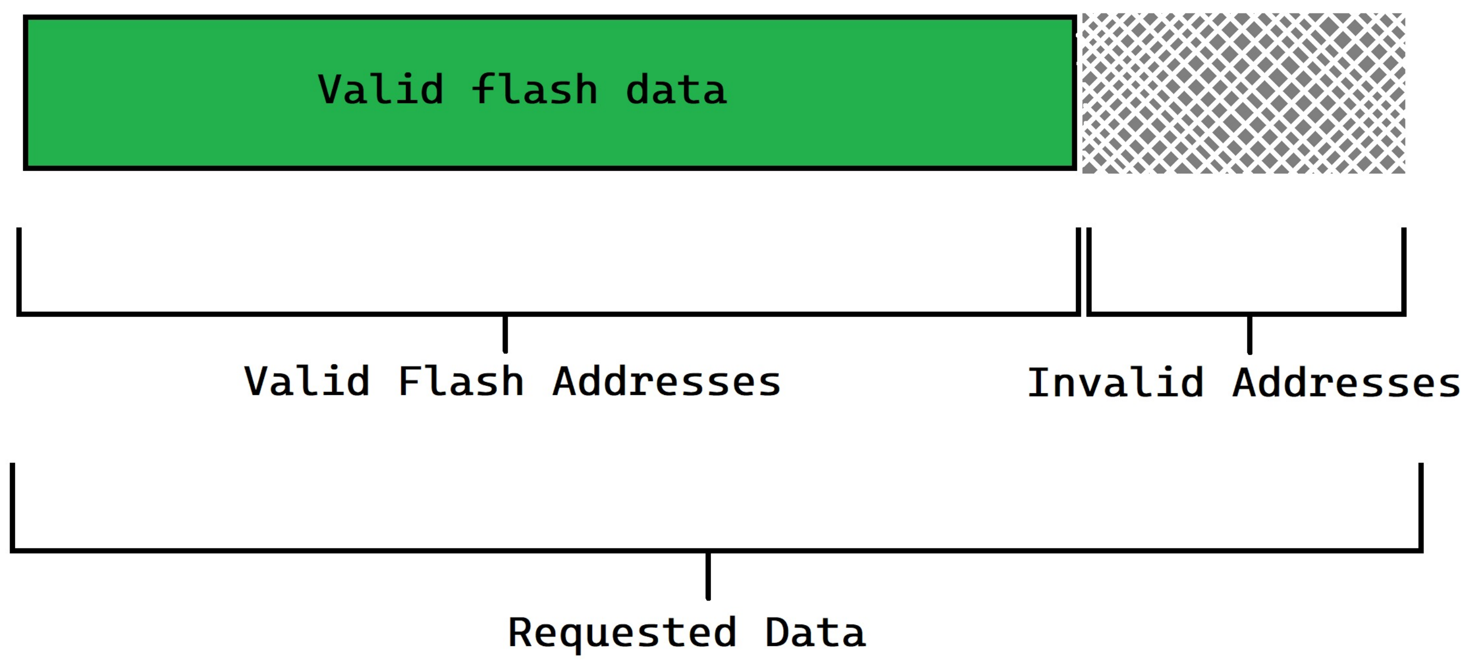
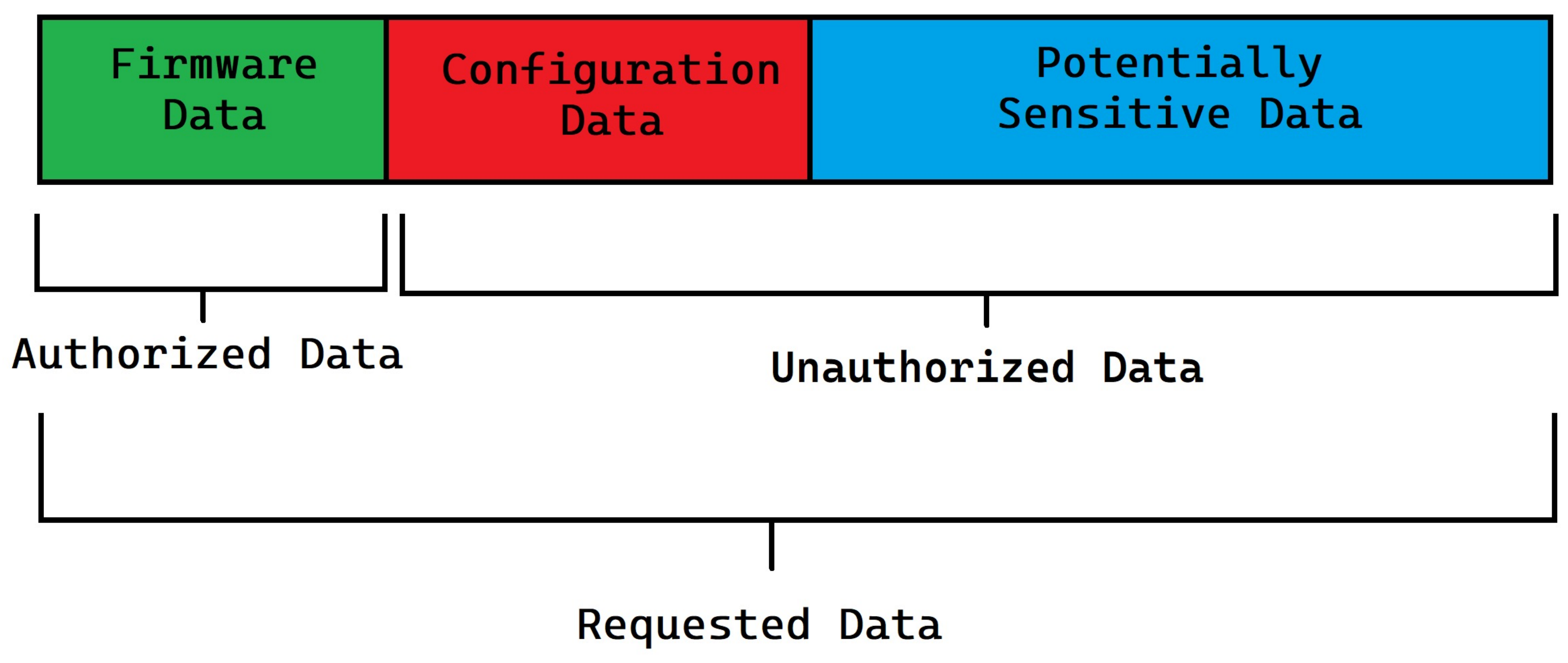


Fig. 5 Potential readback over reach attack



**Design Overview:**

This year's CTF is centered around a device tasked with uploading and updating flight firmware and configurations for an avionic device destined for commercial flights, dubbed the SAFFIRe (Secure Avionics Flight Firmware Installation Routine). A reference design was provided, and, while functional, it lacked any security measures. In order to assure the safety of a flight, a secure SAFFIRe design must ensure confidentiality, integrity, and authenticity of any data that the device uses. Our design accomplishes these goals using two main design elements. One being secret authentic passwords/signatures, and the second being the AES-128 algorithm. When combined properly, these two can ensure all information used by the SAFFIRe device is safe and secure.

**Defensive Highlight:**

One interesting and important feature we developed in our design was the method used to protect configuration files. Our design protects configuration files by appending an authentication password to the beginning and end of the configuration data (Fig. 1). The password is a 16 byte key which is randomly generated during the build process of each SAFFIRe device, and is thus unique to each device. Once the passwords have been added, an AES-128 algorithm running in CBC mode is used to encrypt everything. During the configuration loading process, the host would first send the initial 16 bytes of a protected configuration file to the device, which would be decrypted, and checked by the device. It is important that the first 16 bytes be sent, as these bytes, when decrypted, would contain the first authentication password, and due to the recursive nature of CBC decryption, the first 16 bytes of encrypted data are the only bytes that can be decrypted without first decrypting any other data. This is the reason for the second password, which can only be decrypted correctly if the rest of the data before it has also been decrypted properly. If any data before the second password *has* been compromised in any way, then the second password will not decrypt correctly, thus indicating a loss of data integrity [1].

Once the device has confirmed that the first password is present and correct, compared to a copy kept on an EEPROM on the device, then the device will supply the host with the cryptography keys necessary to decrypt the rest of the data, namely the encryption key and initialization vector. The host will proceed to decrypt the configuration file, and will subsequently send the second, now unencrypted, password to the device for it to confirm. If either of these password checks should fail for any reason, the device will reject the update and will not save any data previously provided to it.

This method for protecting configuration files has undergone some changes throughout the design process. The original method was slightly different in a few ways, and was changed in order to comply with timing requirements. Originally configuration data would be split into 1 KB chunks, and a single authentication password would be added to the end of each chunk (Fig 2). The chunks would then be recombined and encrypted individually before being recombined into one protected file. During the update process, the device would load each chunk into memory and would decrypt it and verify its integrity without any host intervention. This method, while more secure, is far too slow. The SAFFIRe device runs on a Texas Instruments Tiva C series LaunchPad, which has a 32 Khz clock speed. Due to this low clock speed a full 64 KB configuration file could take up to nearly 2 minutes to be fully decrypted and digested by the device, which is far longer than the 25 second requirement. By moving all decryption to a host device, which will likely be several orders of magnitude faster than the SAFFIRe device, this decryption step can be reduced to milliseconds, and adds a negligible amount of time to the upload process.

While the new method for protecting configuration files is significantly faster, it does increase the attack surface for a potential bad actor. After decryption keys are sent to the host device, it is possible for an attacker to intercept them, which could lead to a compromise of a significant amount of sensitive information. In order to address some of these risks, we would have liked to incorporate a hashing algorithm of some sort into the protection process to generate a checksum of what the unencrypted data should be, to further confirm data integrity. We also would have liked to have found a way to use an asymmetric encryption algorithm in combination with a hash to further increase confidentiality, rather than trusting the host device (Fig. 3).

**Offensive Highlight:**

Due to our entry into the attack phase on only the very last day of the competition, we were unable to see how our design would perform against other teams. Likewise, we were unable to perform any attacks of our own against other teams' designs. Despite this, however, we did have several ideas about possible vulnerabilities that could be exploited.

One such vulnerability relates to the SAFFIRe system’s firmware/configuration readback functionality, which would expose, in plain text, the contents of the system's flash memory. If this functionality was improperly implemented, it is possible for an attacker to request to see a significant amount of data, which could lead to security compromises. For example, even though a firmware binary can only be up to 16 KB, an attacker could request to see however much data they want, and without the proper checks, the attacker could request for, and receive, far more than 16 KB of data on the flash. This has the potential to expose possibly sensitive and confidential information (Fig. 5). A similar attack could request to see far more data than the device is physically capable of handling. If, for example, an attacker requested several GB of data, the device may encounter overflow errors or instability due to the large amount of data it is trying to handle. These attacks have been dubbed the readback overreach, and readback overflow attacks, respectively (Fig. 4).

These attacks would only be possible if the readback functionality did not filter the size of a request properly. A firmware binary can only be a maximum of up to 16 KB and a configuration binary can only be up to 64 KB. A design could prevent instability in the SAFFIRe device by storing the number of bytes to read back in a 16 bit value, which would limit the readback data to a maximum of 64 KB. This is less than ideal, though, because this would still leave firmware readback vulnerable to overreach. In order to fully alleviate these vulnerabilities, additional logic would be required to limit the amount of data that could be read back for certain operations.