

# **MITRE Embedded Capture the Flag Competition**

**Design Document | Initial Draft**

**Johns Hopkins University**

**Version: R001**

**January 31, 2025**

## Table of Contents

|  |           |
|--|-----------|
| <b>MITRE Embedded Capture the Flag Competition .....</b> | <b>1</b>  |
| <b>Introduction .....</b>                                | <b>3</b>  |
| <b>Overview of Requirements .....</b>                    | <b>3</b>  |
| Functional Requirements .....                            | 3         |
| Security Requirements .....                              | 5         |
| <b>Functional Design .....</b>                           | <b>5</b>  |
| Design Overview .....                                    | 5         |
| Secret Generation .....                                  | 6         |
| Subscription Generation .....                            | 7         |
| Encoder .....  | 7         |
| Decoder .....  | 8         |
| <b>Security Design .....</b>                             | <b>9</b>  |
| Overall Security Design .....                            | 9         |
| Security Controls Overview .....                         | 9         |
| Global Secret Generation .....                           | 11        |
| Subscription Generation .....                            | 12        |
| Frame Encoding .....                                     | 12        |
| Subscription Update Validation .....                     | 12        |
| Frame Decoding .....                                     | 12        |
| Keys and Selected Algorithm .....                        | 13        |
| Encryption .....   | 13        |
| Subscription Checks .....                                | 14        |
| Limitations of the Current Design .....                  | 15        |
| <b>Next Steps .....</b>                                  | <b>16</b> |

## Introduction

This document outlines the initial draft of the Johns Hopkins University secure satellite system design for the 2025 MITRE eCTF competition. The purpose of this document is to:

- Describe the functional and security requirements for the satellite TV system
- Outline functional design architecture, components, and information flow
- Identify key security threats
- Outline security controls included to respond to identified threats

Note that this document will be updated as the design develops. This draft serves to capture our team's initial ideas, goals, and objectives.

## Overview of Requirements

Our design objective is to at minimum meet both the functional and security requirements as outlined by MITRE. This section summarizes these requirements which serve as the basis of design.

### Functional Requirements

Our design must include a Generate Secrets function, a Generate Subscription function, an encoder, and a decoder. Below outlines high level requirements for each:

*Generate Secrets (Function):*

- Take a list of channels that will be valid in the system and return system-wide secrets that will be passed to the Encoder, Generate Subscription, and the build process of the decoder.
- Attackers will not have access to the output of this function.

*Generate Subscription (Function):*

- Take secrets generated by Generate Secrets and the Decoder ID, the start and end timestamps, and the channel number the subscription will be valid for. Return the contents of a subscription, which will be passed to the Decoder via TV.Subscribe.
- Attackers will not have access to the output of this function, although they may have subscriptions in certain scenarios.

*Encoder (Class):*

- Initialized with the raw value of the secrets file generated by Generate Secrets.
- Must implement the “encode” function, which accepts the channel the frame will be sent on, the raw frame, and the timestamp; return the encoded frame data as bytes that will be sent to the Decoder. This function will be called for every frame before being transmitted by the satellite to all listening TVs.

*Decoder (Firmware): There are numerous requirements for the decoder. Please refer to [MITRE docs](#) for complete details. Below summarizes key requirements.*

- Communications between Decoder and host computer will take place over UART at a baud rate of 115200.
- The decoder must be able to list all subscribed channels with the start and end timestamps that define the active lifetime of the subscription.
- The decoder must be able to update subscribed channels.
- The decoder must be able to decode TV frame data if the frame is from a channel the decoder has a valid and active subscription for or is sent on the emergency broadcast channel (channel 0).
- There will also be timing constraints for operation completion and throughput requirements for the encoder as well as the decoder.

| Operations          | Maximum Time for Completion |
|---------------------|-----------------------------|
| Device Wake         | 1 second                    |
| List Channels       | 500 milliseconds            |
| Update Subscription | 500 milliseconds            |
| Decode Frame        | 150 milliseconds            |

| Operation | Minimum Throughput         |
|-----------|----------------------------|
| Encoder   | 1000 64B frames per second |
| Decoder   | 10 64B frames per second   |

## Security Requirements

There are three primary security requirements our design aims to address:

- An attacker should not be able to decode TV frames without a Decoder that has a valid, active subscription to that channel.
- The Decoder should only decode valid TV frames generated by the Satellite System the Decoder was provisioned for.
- The Decoder should only decode frames with strictly monotonically increasing timestamps.

Additionally, there are 5 specific attack scenarios that our design aims to defend against:

- Expired Subscription – Attacker can read frames from a channel that they have an expired subscription for
- Pirated Subscription – Attacker can read frames from a channel they pirated the subscription for
- No Subscription – Attacker can read frames they do not have a subscription for
- Recording Playback – Attacker can read frames from a recorded channel that they currently have a subscription for, but did not at the time of the recording.
- Pesky Neighbor – Attacker spoofs the signal of the satellite causing the attacker's chosen signal to be sent to the neighbor and decoded instead of the normal signal from the satellite.

## Functional Design

### Design Overview

To address the functional and security requirements, our design builds on the provided reference implementation to include four primary components:

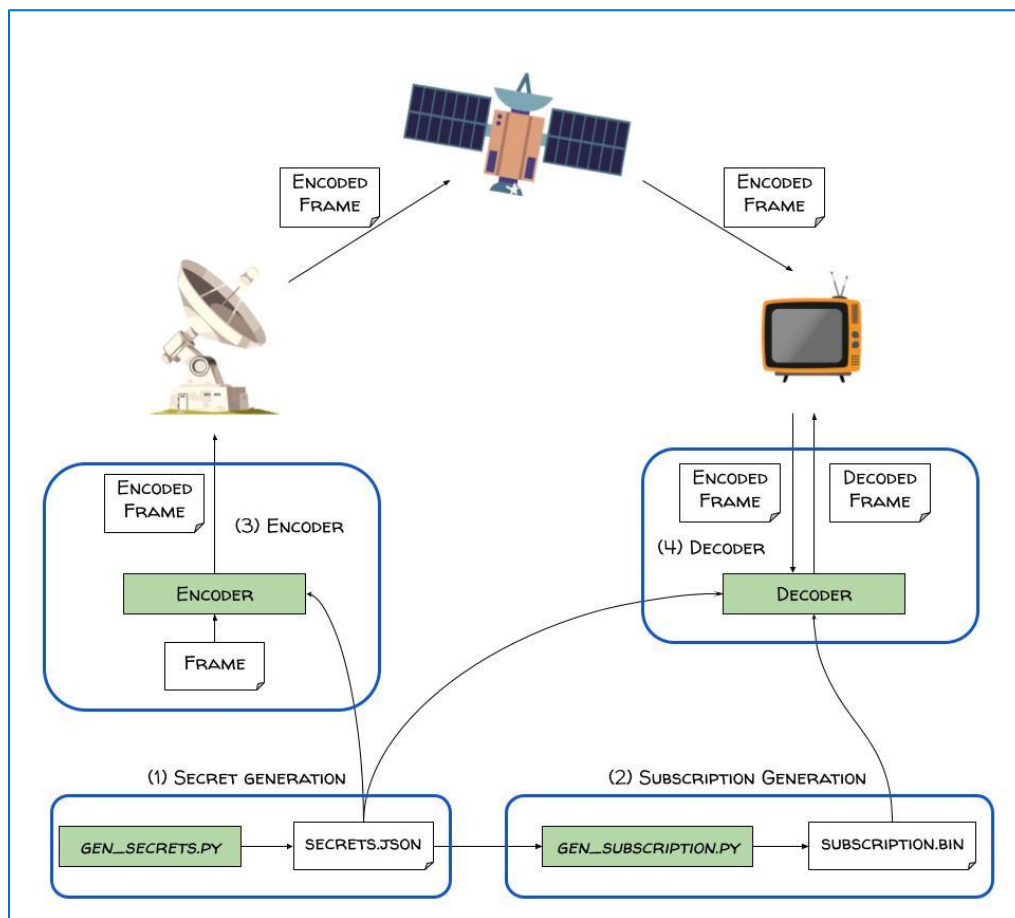
- (1) *gen\_secrets.py* to generate secrets used to securely transmit frames;
- (2) *gen\_subscriptions.py* to securely subscribe TVs and their corresponding decoders to broadcast channels;

(3) *encoder.py* to secure the frames to be transmitted;

(4) *decoder.c* and its dependencies to decode frames sent only from its provisioned satellite system.

Secrets produced from *gen\_secrets.py* are used by the encoder, decoder, and the subscription generation tool to ensure that only the decoders with valid subscriptions can decode the transmitted frames. As a result, all information that is broadcast or accessible to the public, including data frames and the subscription data, are encrypted and/or encoded to prevent unauthorized parties from reading data.

The diagram below summarizes the flow of data in our satellite system.



## Secret Generation

The function "gen\_secrets.py" will generate the global secrets needed by encoder, decoders, and subscription function. These secrets will be stored as secrets.json, and will contain the following information:

- Channels list: indicating valid channels will be used in future subscription
- Private keys (for encoder) and public keys (for decoder) for authentication purposes
- Subscription key for encoder to encrypt subscription messages and related channel keys

## Subscription Generation

The function "gen\_subscription.py" will generate a binary file which will be passed to the decoder to provide decoder with data it needs to subscribe to a new channel.

The subscription file will include the following information:

- Meta data: device id, start time, end time, channel
- N Encrypted keys (Symmetric encryption using encoder's subscription key): say if we subscribe channel 1 and channel 3, after generate key 1 and key 3
  - Encrypted key 1 = Enc (value = key 1, key=subscription key)
  - Encrypted key 3 = Enc (value = key 3, key=subscription key)

## Encoder

Encoder is responsible for generating encoded TV frames for transmission. It is implemented in the "encoder.py" file. It takes four inputs: channel number, timestamp, TV frame, and global secrets.

- Channel number: The channel that encoded frame belongs to.
- Timestamp: The time at which the encoded frame is generated by uplink.
- TV frame: Contents of frame to encode.
- Global secrets: Channel keys and private keys are needed.

The encoder is called by the uplink which is responsible for sending encoded frames to satellite. It first signs the TV frame and generates a digital signature. Then it selects the appropriate channel key from the global secrets based on the target channel number and encrypts the TV frame using symmetric encryption. Finally, it combines an encoded TV frame and digital signature as output.

## Decoder

Decoder is responsible for checking subscriptions and decoding TV frames for subscribed channels. To build a decoder, we need three inputs: decoder id, decoder image, and global secrets. The decoder has three primary functions: List Subscribed Channels, Update Subscribed Channels and Decode TV Frame Data. Below outlines key functionality of the decoder:

- List Subscribed Channels: Host Computer requests the list of active subscriptions from the Decoder. The decoder lists all the valid subscription with: Channel ID and Subscription validity period (start & end timestamps)
- Update Subscribed Channels: The decoder reads a new subscription.bin and verifies the subscription with public key, which ensures the channel id, expiration date is correct. After successfully validating, the decoder updates the subscription list.
- Decode TV Frame Data: The Decoder continuously listens for incoming encoded TV frames from the satellite feed. It verifies subscriptions and decodes TV frame data.
- Subscription Validation: The Decoder checks whether it has an active and valid subscription for the channel. This functionality checks three things:
  - If this is an emergency broadcast message
  - If the current time falls within the subscription's active period.
  - If the subscribed channel id is which the encoded frame comes from and it's valid.
- Clock Management: If the subscription is valid, the Decoder increments the timestamp counter for the subscribed channel to track frame sequencing.
- Decryption: The Decoder retrieves the encrypted channel ciphertext associated with the channel and the subscription key from global secret.
  - It decrypts encrypted channel ciphertext by using subscription key and get channel key.
  - It decrypts the encoded frame by using channel key to reconstruct the original TV frame data.
- Signature Verification: The Decoder verifies the digital signature attached to the frame using the public key. This step ensures that the frame has not been altered or tampered with and comes from a trusted source.



- Output the TV Frame: If the subscription is valid, decryption is successful, and the signature is verified, the Decoder outputs the original TV frame for display.
- If any check above fails, the frame is discarded.

## Security Design

### Overall Security Design

#### *Attacks Summary*

The security design must defend against FIVE attacks that are brought into place during the attack phase.



- **Expired Subscription Attack:** Attackers have a valid subscription for **channel 2** but already expired and try to use it to decode live frames from **channel 2**.
- **Pirated Subscription Attack:** Attackers have a currently valid subscription for **channel 3** but issued for another decoder (different decoder ID).
- **No Subscription:** Attackers have no subscription but a tested decoder and try to somehow decode frames of **channel 4**.
- **Recording Playback:** Attackers have a currently valid subscription for **channel 1** but try to decode recorded frames of **channel 1** out of their subscription time frame.
- **Pesky Neighbor:** Attackers have a currently valid subscription for **channel 1** but try to spoof same decoder (have same subscription) deployed in the cloud to decode any frames other than **channel 0 and 1**.

• Valid subscription means officially issued from encoder.

### Security Controls Overview

This section provides an overview of key security controls selected to defend against attack types mentioned above. Each security control described in the table below is outlined in detail in subsequent sections.

| Defends Against             | Threat Model                    | Control           |
|-----------------------------|---------------------------------|-------------------|
| Expired Subscription Attack | Attacker tries to decode frame; | Frame encryption; |

|                             |   |   |
|-----------------------------|---|---|
|                             | Attacker modifies timestamp in the subscription;  | Sign Subscriptions;   |
|                             | Attacker reuses expired subscription ;            | Subscription Timeframe Check in Decoder;  |
|                             | Attacker extracts channel key from subscription;  | Channel Key Encryption;   |
|                             | Attacker extracts subscription key from decoder;  |    |
| Pirated Subscription Attack | Attacker tries to decode frame;                   | Frame encryption;   |
|                             | Attacker modifies decoder ID in the subscription; | Sign subscriptions;   |
|                             | Attacker uses pirated subscription ;              | Invalid subscription rejection;   |
|                             | Attacker extracts channel key from subscription;  | Channel Key Encryption;   |
|                             | Attacker extracts subscription key from decoder;  |  |

|                    |  |  |
|--------------------|--|--|
| No Subscription    | Attacker tries to decode frame;  | Frame encryption;                        |
| Recording Playback | Attacker tries to decode frame;  | Frame encryption;                        |
|                    | Attacker uses current subscription to decode recorded frame;               | Subscription Timeframe Check in Decoder; |
| Pesky Neighbor     | Attacker spoofs satellite arbitrary data as channel 0 to neighbor decoder; | Sign encrypted/encoded frames            |

### *Current Security Design*

Currently, our security design is plotted as below:

#### **Global Secret Generation**

There will be three kinds of global secrets being generated during deployment,

- **Channel Keys:** which are **4** symmetric encryption keys used to encrypt the channels from 1-4 (channel 0 is emergency channel subscribed natively by all decoders).
- **Encoder public-private key pair:** which is **a** public-private key pair for producing digital signature of encoder during the subscription generation and encoded frames sending. The **private key** is used for signing **subscription** binary and **encoded frame** before sent out to satellite; The **public key** will be provided to and stored in decoder firmware during the deployment and used to verify if **subscription** binary or **encoded frame** they receive were **issued from encoder**, i.e. protect from pirated subscription attack, Pesky Neighbor attack and Recording Playback (e.g. tampering the timestamp)
- **Subscription Key:** which is **a** symmetric encryption key used to encrypt the **channel keys** and put them into the subscription. Decoder will later extract the **channel keys** from the subscription to decode frames. The subscription key aims at protecting against **channel key extraction** from valid subscriptions but is potentially vulnerable to reverse engineering attack.

The global secrets will be fed into encoder and decoder during deployment. The encoder will keep the copy of **channel keys**, **encoder private key** and **subscription key**, and is not accessible to attackers. The decoder will keep the copy of **encoder public key** and **subscription key**, and is accessible to attackers. The decoder must have somewhere to store these two keys either in certain hardware unit OR memory, which, however, is potentially vulnerable to reverse engineering attack.

### **Subscription Generation**

The encoder tool generates subscription which should include the encrypted subscribed channel keys.

### **Frame Encoding**

The encoder should first encrypt the raw channel frame data with corresponding channel key, then sign it with the encoder private key.

### **Subscription Update Validation**

According to the functional requirement, the decoder is mandated to accept any **valid subscriptions** (made for the decoder ID same as the decoder) including the expired subscriptions and must use the latest installed subscription. So, we are not able to reject the expired subscription directly and must allow it to be installed in the decoder. Thus, there should be two mechanisms separately handling expired subscriptions and invalid subscriptions.

For invalid subscriptions that are made for another decoder ID, the decoder should reject them during the subscription update due to failure of digital signature verification.

For expired subscriptions that are valid (made for the current decoder), the decoder should allow it to be installed and specially treat them in the frame decoding process.

### **Frame Decoding**

The decoder should check if incoming encoded frame is within the scope of current subscription (for example, time frame) before verification and decrypting. If, for example, the encoded frame is outside the time frame of current subscription, ignoring the following process.

The decoder must only decode frames with strictly monotonically increasing timestamps [across different channel frames](#). (security requirement 3; more of functional requirement)

## Keys and Selected Algorithm

- **Channel Keys:** symmetric keys used for encrypting the channel frames with **AES-256** encryption scheme
- **Encoder public-private key pair:** asymmetric keys used for providing digital signatures of encoder with **Ed25519 signature algorithm**. The private key will be used by encoder to sign encrypted/encoded frame data and encrypted subscription. The public key will be used by decoder to verify incoming subscriptions and encoded frames before decrypting them. Ed25519 has better performance than RSA.
- **Subscription Key:** symmetric keys used for encrypting channel keys in the subscription with **AES-256-GCM** encryption scheme

## Authentication

Authentication is provided for both the subscription updates and encoded frames to ensure authenticity:

- Authentication for subscriptions file: Encoder private key signing the subscription which can be proved issued by encoder by verifying it with public key.
- Authentication for Frames: Encoder private key signing the encoded frames which can be proved issued by encoder by verifying it with public key.

## Encryption

The subscription file (subscription.bin) contains encrypted channel keys, which the decoder will later use to decrypt and authenticate TV frames. To protect the confidentiality and integrity of the subscription data, we use AES-256-GCM from WolfSSL.

Each channel key is encrypted using the Subscription Key via AES-256-GCM before being stored in the subscription file. The encryption process is as follows:

1. Generate a random IV (Initialization Vector) using `wc_RNG_GenerateBlock()`.
2. Encrypt each channel key using AES-256-GCM with the Subscription Key via `wc_AesGcmEncrypt()`.
3. Store the IV, encrypted channel key, and authentication tag.

The following WolfSSL API functions are used in this process:

- `wc_RNG_GenerateBlock()` – Generates a secure random IV.
- `wc_AesGcmSetKey()` – Initializes the AES-GCM context with the Subscription Key.

- `wc_AesGcmEncrypt()` – Encrypts channel keys in GCM mode.
- `wc_AesGcmDecrypt()` – Used later by the decoder to decrypt channel keys.

TV frames are also encrypted before transmission using AES-256-GCM, ensuring that only authorized decoders with valid subscriptions can decrypt them. Each TV frame consists of:

- Channel Number: Identifies the channel.
- Timestamp: Ensures frames are processed in order.
- TV Frame Data: The actual encrypted video content.
- Digital Signature: Ensures authenticity.

The following outlines the frame encryption process:

1. Retrieve the corresponding channel key.
2. Generate a random IV using `wc_RNG_GenerateBlock()`.
3. Encrypt the TV frame using AES-256-GCM via `wc_AesGcmEncrypt()`.
4. Sign the encrypted frame using Ed25519.

The following WolfSSL API functions are used in this process:

- `wc_AesGcmSetKey()` – Initializes AES-GCM with the channel key.
- `wc_AesGcmEncrypt()` – Encrypts the TV frame data.
- `wc_AesGcmDecrypt()` – Used later by the decoder to decrypt the frame.
- `wc_ed25519_sign_msg()` – Signs the encrypted frame.
- `wc_ed25519_verify_msg()` – Used later by the decoder to verify the signature.

## Subscription Checks

Each subscription file contains a signed decoder ID, ensuring that only the correct decoder can use it. This is verified by:

1. Extracting the signed Decoder ID from the subscription file.
2. Verifying the signature using Ed25519 via `wc_ed25519_verify_msg()`.

3. If verification fails, the decoder rejects the subscription.

The following WolfSSL API functions are used in this process:

- `wc_ed25519_verify_msg()` – Verifies the decoder ID signature.
- `wc_ed25519_sign_msg()` – Originally used to sign the subscription.

The decoder verifies if the subscription is still valid based on timestamps. This process is verified by:

1. Extracting start time and end time from the subscription.
2. Comparing the current system time with the timestamps.
3. If expired, the decoder rejects frames from that channel.

## Limitations of the Current Design

This section outlines current limitations of the design. These items will be addressed throughout the design phase.

The current security design largely depends on the security of **subscription key**. Ideally it will effectively defend against all FIVE attacks.

However, once **subscription key** is leaked, **Expired Subscription** (target CH2), **Pirated Subscription** (target CH3) and **Recording Playback** (target CH1) attacks are undefendable, since attackers own these channels' subscription, thus channel keys are decryptable and so the encoded frames.

While, fortunately, the security against **No Subscription** and **Pesky Neighbor** attacks will be guaranteed. Since, **No Subscription** attack requires to decode **channel 4**, however attackers never own valid subscription for channel 4 thus against channel key extraction; **Pesky Neighbor** attack requires attacker to spoof an exact same decoder to decode channels except 0-1 they have valid subscriptions for. To do so, attackers must be able to encrypt frames with channel key 1 to be compliant with neighbor decoder, which is viable by extracting the key from subscription with leaked subscription key. However, attackers have no access to **encoder private key** at all, thus neighbor decoder will reject to decode invalidly signed frames anyway.

## Next Steps

### Functionality Requirements

- While implementing functionality requirements, we'd like to utilize static analysis tools that guarantee safety and correctness.
- Begin implementing elements of the design in a step-by-step manner.

### Security Requirements

- In general, the security design needs to be updated to prevent some existing security holes, as outlined in the Limitations of the Current Design section above.

**END OF REPORT**