New York University (NYUSEC) - eCTF 2025 Design Document

This document serves as the design implementation for team NYUSEC. The document is divided as follows:

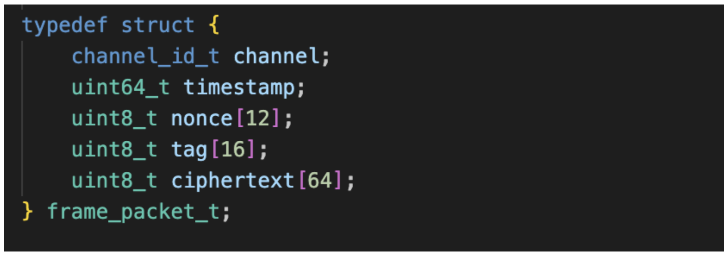
1. Basic Implementation
    - Cryptographic Algorithms
    - Data Structures
2. Decoder Functional Requirements
3. Encoder Functional Requirements
4. Security Requirements

Basic Implementation Details

In the build process, we create channel-specific **256-bit ChaCha20-Poly1305** keys that are placed in our global secrets. This serves to secure communication of packets transferred between the encoder and decoder via symmetric encryption. This will help ensure confidentiality of packets while also maintaining authentication of data.

The *frame_packet_t* data structure (building upon the MITRE reference design) adds elements required by **ChaCha20-Poly1305 encryption:** a 12 byte nonce, along with a 16 byte TAG. There are 64 bytes reserved (the max frame size) to hold the encrypted frame.

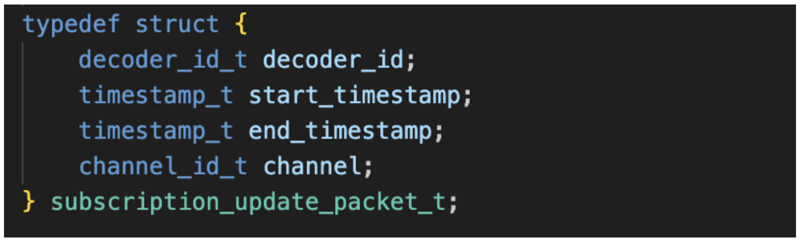
Note that the Additional Authenticated Data (AAD) is composed of the channel and timestamp data. This information is concatenated and authenticated via the provided TAG.



```
typedef struct {
    channel_id_t channel;
    uint64_t timestamp;
    uint8_t nonce[12];
    uint8_t tag[16];
    uint8_t ciphertext[64];
} frame_packet_t;
```

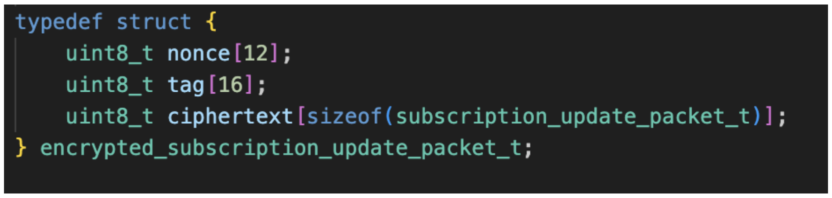
frame_packet_t struct

In regards to subscription updates, these will also make use of a separate **256-bit ChaCha20-Poly1305 key.** The encoder produces a subscription update in the form of the *encrypted_subscription_update_packet_t* struct. It holds the required nonce and TAG, along with the ciphertext of the encrypted subscription update. Once decrypted, it produces the subscription update in the form of the subscription_update_packet_t struct which includes the decoder_id, timestamps, and channel ID.



```
typedef struct {
    decoder_id_t decoder_id;
    timestamp_t start_timestamp;
    timestamp_t end_timestamp;
    channel_id_t channel;
} subscription_update_packet_t;
```

subscription_update_packet_t struct



```
typedef struct {
    uint8_t nonce[12];
    uint8_t tag[16];
    uint8_t ciphertext[sizeof(subscription_update_packet_t)];
} encrypted_subscription_update_packet_t;
```

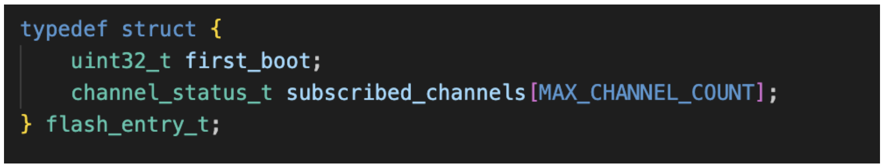
encrypted_subscription_update_packet_t struct

Additionally, we store a monotonic counter to ensure that the timestamp of any packet received is compared to this counter to verify.

Decoder Functional Requirements

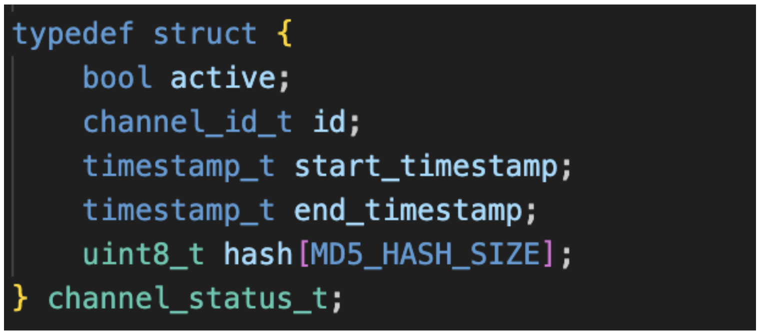
The decoder will receive packets from the Satellite. These packets will be received over UART. The Decoder serves a few key functionalities:

1. List Channels
    - The Decoder lists the channel numbers that the Decoder has a valid subscription for. It maintains a global list of subscriptions in the following structure (from the reference implementation):



```
typedef struct {
    uint32_t first_boot;
    channel_status_t subscribed_channels[MAX_CHANNEL_COUNT];
} flash_entry_t;
```

    - subscribed_channels is of type channel_status_t which is a struct that tracks whether a channel is actively subscribed to along with the start and end timestamps (modified from the reference implementation to include a hash for integrity):



```c
typedef struct {
    bool active;
    channel_id_t id;
    timestamp_t start_timestamp;
    timestamp_t end_timestamp;
    uint8_t hash[MD5_HASH_SIZE];
} channel_status_t;
```

- To list channels the list_channels function iterates over the global variable "decoder_status" which is of type flash_entry_t. It iterates over this to list out each actively subscribed channel.

2. Update Channel Subscriptions
    - The Decoder has an update_subscription function – once a channel subscription update is received, this function validates that the encrypted subscription update is legitimate (using the ChaCha key, nonce, and TAG). Additionally, it checks whether the timestamps are valid (that the end time is greater than the start time). It then accesses the global decoder_status which contains the valid subscriptions and updates it to reflect this change.
3. Decode TV Frames
    - Frames received by the encoder are set in the data structure as described in 1.) Basic Implementation Details. The decode process works as follows
        1. Calculates frame size (Frames up to 64B are considered valid)
        2. Verifies timestamp (using the monotonic counter)
        3. Validates subscription (ensures that the frame received has a valid, active subscription.) It also performs a quick integrity check using an MD5 hash.
        4. Decrypt message using the channel-specific key, nonce, tag, and ciphertext

Encoder Functional Requirements

The encoder will take in the following components to return encoded packets:

- Channel Number
- Timestamp
- TV Frame
- Global Secrets

Packets will be encoded to match the data structure described in 1.) Basic implementation details. The encoder is responsible for encrypting the TV frames using the 256-bit ChaCha20-Poly1305 channel-specific keys provided in global.secrets. The Encoder makes use of the PyCryptoDome ChaCha20_Poly1305 implementation to produce random nonces for each frame that is sent.

Security Requirements

Security Requirement 1

Requirement: An attacker should not be able to decode TV frames without a Decoder that has a valid, active subscription to that channel.

The system protects TV frames using channel-specific 256-bit ChaCha20-Poly1305 encryption with unique nonces and authentication tags. Only decoders with valid, active subscriptions can decrypt content, preventing unauthorized access and ensuring both confidentiality and data integrity for TV frames.

Security Requirement 2

Requirement: The Decoder should only decode valid TV frames generated by the Satellite System the Decoder was provisioned for.

The decoder is designed to only process legitimate frames from its provisioned Satellite System through a multi-layered authentication approach. Subscription updates include the decoder's unique device ID that is both encrypted and authenticated with ChaCha20-Poly1305, ensuring that subscription updates can only be applied to their intended decoder. When the decoder receives a subscription update packet, it must first successfully decrypt it with the subscription key and verify the authentication tag before processing the contained decoder_id. Frames are also encrypted and authenticated. When these frames are received, they are checked against the active subscriptions array held by the decoder.

Security Requirement 3

Requirement: The Decoder should only decode frames with strictly monotonically increasing timestamps.

A monotonic counter is utilized to track the highest-seen timestamp for each channel. Before processing a frame, the decoder compares its timestamp to the stored value, rejecting any packet that is older or out of order. Upon successful decoding of the frame, this counter is then updated.