Satellite TV System Implementation

BYU Cyberia

February 28, 2025

1 Overview

This Design Document describes BYU's implementation of the Satellite TV System as defined by MITRE for eCTF 2025.

Contents

2 Functional Requirements

2.1 Overview

The Satellite TV system includes several components:

- The **encoder**;
- The **uplink**;
- The **satellite**;
- The **host**;
- The **decoder** (or **device**).

Each component communicates according to the arrows show in Figure 1. The design requires implementation of various functions performed by the encoder, decoder, and host, as detailed in the following subsections.

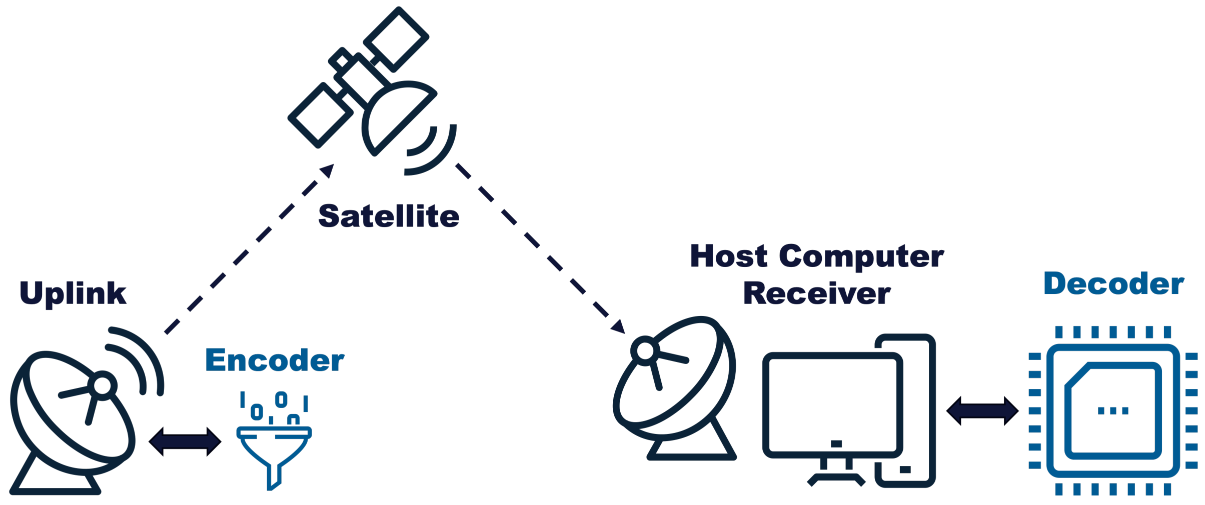

Figure 1: Communication paths within the Satellite TV system

2.2 Encoder

The encoder must:

1. Using the global secrets, create a TV frame packet which includes the following information:
   - Channel ID
   - Timestamp
   - TV Frame

   The encoding process is described in Section 6.

2.3 Decoder

The decoder (otherwise known as the device) must:

1. Initialize various global structs and variables to store channel-related metadata, including subscription validity, timestamps, and encryption keys (see Section 7)
2. Decrypt/decode TV frames packets received from the host (see Section 8)
3. Update subscription information (stored in flash) when a new subscription file is received (see Section 11)
4. List channel subscription information when a request is received from the host (see Section 12)

2.4 Host

The host must:

1. Send new subscription files over UART to the decoder (see Section 10)
2. Receive and display decrypted TV frames from the decoder

3 Security Requirements

3.1 Subscription Enforcement

The system ensures that only decoders with valid, active subscriptions can decode TV frames for a given channel. This is achieved through the following mechanisms:

- **Subscription Storage and Management**: The decoder maintains subscription information, including start and end timestamps and channel keys, in flash memory, as detailed in Section 7 and updates it when new subscription files are received and validated, as described in Section 11.
- **Subscription Verification During Decryption**: During the TV frame packet decryption process, outlined in Section 8, the decoder checks whether the timestamp in the TV frame packet falls within the subscription period for the specified channel. If the subscription is invalid or has expired, the TV frame packet is rejected.

3.2 Origin Verification

To ensure that only TV frames generated by the legitimate Satellite System are decoded, the system employs authenticated encryption:

- **Authenticated Encryption**: The TV frame packets are encrypted using ChaCha20-Poly1305, providing both confidentiality and integrity. As explained in Section 8, the decoder verifies the Poly1305 tag using the channel-specific key before decrypting the packet. If the tag verification fails, the packet is discarded, preventing the decoding of forged or tampered frames.

3.3 Timestamp Monotonicity

The system enforces that TV frames are decoded only if their timestamps are strictly monotonically increasing:

- **Timestamp Tracking**: The decoder maintains a global variable, `next_time_allowed`, which is initialized during boot as described in Section 7 to track the minimum timestamp for the next acceptable packet.
- **Timestamp Validation**: During the decryption process in Section 8, the decoder checks that the timestamp in the TV frame packet is greater than or equal to `next_time_allowed`. If the timestamp is older, the packet is discarded. After successful decoding, `next_time_allowed` is updated to the next expected timestamp.

3.4 Attacker Mitigation

To forge a subscription and gain unauthorized access to TV frames, an attacker would need to:

- Obtain the **ChaCha20-Poly1305 subscription key** used to encrypt and authenticate subscription files, as described in Section 10. Without this key, the attacker cannot create a valid subscription file that the decoder would accept, as the Poly1305 tag verification in Section 11 would fail.

4 Secrets Generation

4.1 Secrets Overview

The design implements the security requirements using an Authenticated Encryption with Associated Data (AEAD) protocol, specifically **ChaCha20-Poly1305**, which requires generation of several cryptographic keys. The generate secrets script must generate and store the keys required for encryption, decryption, and subscription validation.

4.2 Key Generation

As input, the script receives a list of valid channels for the current deployment. The script must:

1. For each channel in the provided list:
    - **Generate one ChaCha20-Poly1305 key** (**32 bytes**) to encrypt/decrypt TV frames for that channel.
    - **Store** the generated key in the `chacha_keys` dictionary, indexed by channel ID.
2. **Generate one ChaCha20-Poly1305 key** (**32 bytes**) to encrypt/decrypt channel 0 TV frames.
    - **Store** the generated key in the `chacha_keys` dictionary at index 0.
3. **Generate one ChaCha20-Poly1305 key** (**32 bytes**) to encrypt/decrypt subscription files sent to decoders.
    - **Store** the generated key to the variable `subscription_hex`.
4. Return the following as JSON-encoded bytes:

```
{"channel_keys": chacha_keys, "subscription_key": subscription_hex,}
```

4.3

4.4 Error Handling

The script must:

1. Handle errors, including:
    - Failure to generate encryption keys for ChaCha20-Poly1305.
    - Invalid channel IDs (size exceeds the maximum value for the `uint32_t` data type)
    - Issues with key storage, such as write failures or permission errors.
    - Missing or corrupted key data.

5 Firmware Creation

5.1 Firmware Overview

The device firmware includes functions for:

- Decoding TV frame packets
- Updating device subscriptions
- Listing current channels

To properly validate and decrypt subscription files, the device requires access to the **ChaCha20-Poly1305 encryption/decryption key for subscription files**. Additionally, since **channel 0** is assumed to always be valid, a **ChaCha20-Poly1305 encryption/decryption key for channel 0** must be included in the device's firmware.

5.2 Firmware Generation Process

The process of generating firmware for the device involves the following steps:

1. **Random Device ID Assignment**
    - The system includes the provided 32-bit **device ID** in the device firmware.
2. **Generate Secret Keys**
    - The system runs the generate secrets script as detailed in Section 4.
    - The secret keys are mounted at `/global.secrets/secrets.json`.
3. **Load Secret Keys**
    - The system loads the following secret keys from `secrets.json` into `secrets.h` for inclusion in the firmware:
        - **ChaCha20-Poly1305 secret key** for channel 0 TV frame decryption and authentication.
        - **ChaCha20-Poly1305 secret key** for subscription decryption and authentication.
4. **Compile Firmware**
    - The system compiles the firmware with the loaded keys, ensuring that the device is ready to decode TV frames, update subscriptions, and list available channels.

5.3 Security Considerations

- Each of the secret keys will reside in flash memory on the device after the firmware has been built. On device boot, the keys are loaded into volatile memory.

6 Encoding TV Frames

6.1 Encoder Overview

The encoder processes TV frame data by encrypting it along with metadata using a secure channel key and a nonce mechanism. This ensures the integrity and confidentiality of transmitted video data.

6.2 Input Handling

The encoder must:

1. Accept the following inputs:
   - **Channel ID** (unsigned integer, 4 bytes): Identifies the TV channel.
   - **Timestamp** (long long integer, 8 bytes): Represents the time of the frame capture.
   - **TV Frame** (TV data, 64 bytes): Raw frame data to be encoded.
2. Validate the input values:
   - **Channel ID** must be expressible as an unsigned 32-bit integer.
   - **Timestamp** must follow a valid format and be a value of 8 bytes.
   - **TV Frame** must meet size and format constraints of 64 bytes or less.

6.3 Key & Nonce Handling

The encoder must:

1. **Retrieve the channel encryption key** from the `secrets.json` based on the provided **Channel ID**.
   - Ensure secure access to the key storage.
   - Handle missing or corrupted keys appropriately.
2. **Randomly generate the nonce** for the packet (a random sequence of 96 bits or 12 bytes).

6.4 Encryption Process

The encoder must:

1. **Create the following structures** in memory:
   - **Additional Authenticated Data**: Channel ID (4 bytes), Nonce (12 bytes)
   - **TV Frame Data**: Timestamp (8 bytes), TV Frame (up to 64 bytes)
2. Use the **ChaCha20-Poly1305 key** for the appropriate channel ID to encrypt the TV Frame data.
3. Create a Poly1305 MAC tag using the **Additional Authenticated Data** and encrypted **TV Frame Data** as inputs to the MAC.
4. **Return the secure packet** to then be processed by the Uplink Device.

See Figure 2 for a diagram of the packet sent to the Uplink Device.

6.5 Error Handling

The encoder must:

1. Handle errors, including:
   - Missing or invalid input data.
   - Failed key retrieval.
   - Encryption failures.
   - Failed file read/write operations.

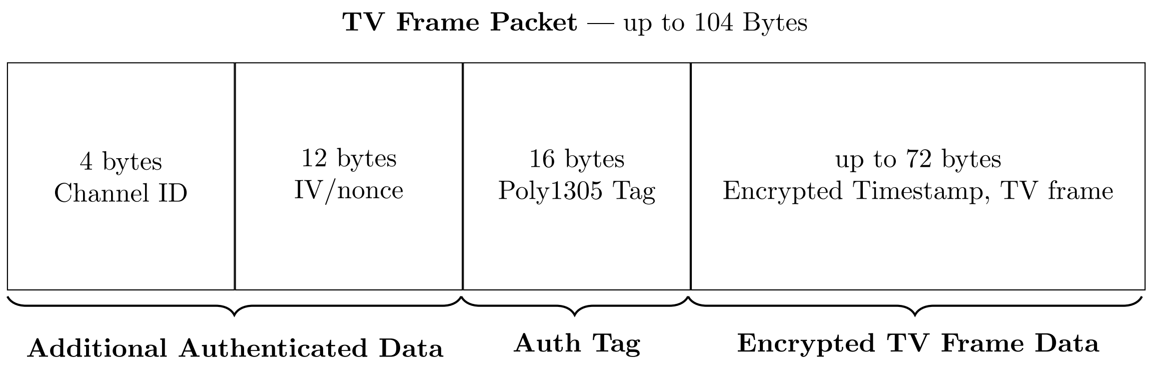
TV Frame Packet — up to 104 Bytes
4 bytes Channel ID
12 bytes IV/nonce
16 bytes Poly1305 Tag
up to 72 bytes Encrypted Timestamp, TV frame
Additional Authenticated Data
Auth Tag
Encrypted TV Frame Data

Figure 2: A diagram of the assembled TV frame packet

7 Decoder Initialization

7.1 Persistent Data Storage

The decoder stores a persistent copy of channel subscription information in flash storage, which follows the format specified by the `decoder_status` struct. A diagram is provided in Figure 3.

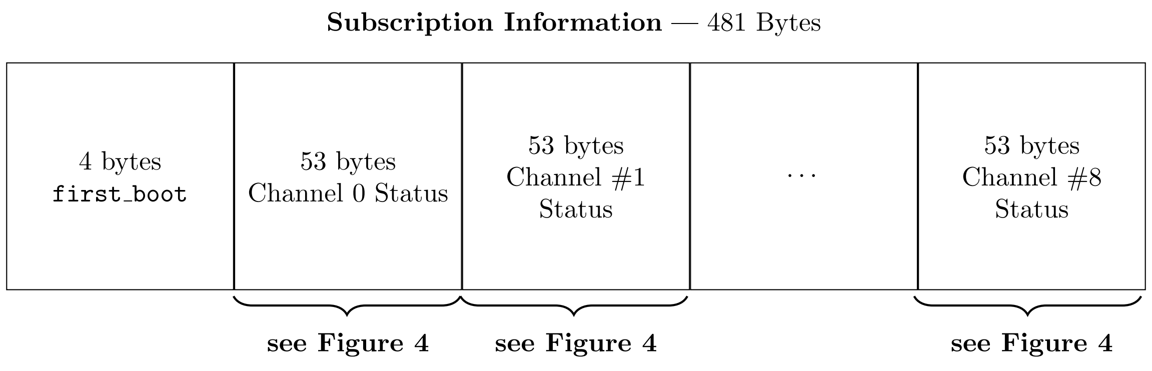
Subscription Information — 481 Bytes
4 bytes first_boot
53 bytes Channel 0 Status
53 bytes Channel #1 Status
· · ·
53 bytes Channel #8 Status
see Figure 4
see Figure 4
see Figure 4

Figure 3: A diagram of the subscription information in flash

7.2 First Boot Initialization Process

On the first boot, the decoder must:

1. **Initialize** the `next_time_allowed` variable (in volatile memory) to 0. This ensures the next accepted packet will pass the timestamp check.
2. **Copy the subscription information from flash** into a memory location in volatile memory called `decoder_status`.
3. Set the `first_boot` field of `decoder_status` to a canary value so this boot sequence is executed only once.
4. For each subscription, set the following variables to their default values (see Figure 4):
   - Set `start_timestamp` to 0.
   - Set `end_timestamp` to the maximum unsigned long long integer value ($2^{64} - 1$)
   - Set `active` to `false`.
   - Set the `channel_key` to zeros.
5. Set channel 0's `active` attribute to `true` and copy the channel 0 TV frame decryption key into channel 0's `channel_key` attribute. This ensures a persistent subscription to channel 0.
6. Write the contents of `decoder_status` back to flash memory.

Channel Status — 53 Bytes

1 byte `active`	4 bytes `id`	8 bytes `start_timestamp`	8 bytes `end_timestamp`	32 bytes `channel_key`

Figure 4: A diagram of the subscription information in flash (for a particular channel)

7.3 General Initialization Process

On any boot other than the first, the decoder must:

1. **Initialize** the `next_time_allowed` variable (in volatile memory) to 0. This ensures the next accepted packet will pass the timestamp check.
2. **Copy the subscription information from flash** into a memory location in volatile memory called `decoder_status`.

7.4 Error Handling

On boot, the decoder must:

1. Handle errors, including:
    - Failure to write and erase flash storage.
    - UART initialization failure.
    - Corrupt or missing subscription data.
    - Issues with key retrieval.
    - File read/write errors.

7.5 Security Considerations

The `next_time_allowed` variable maintains compliance with the third security requirement, as explained further in Section 8. Because this variable does not need to persist across power cycles, we may reinitialize it upon each boot.

8 Decoding TV Frames

The decoder is responsible for processing encrypted TV frame packet by decrypting it using the appropriate channel key (stored securely on the device) and nonce (randomly generated for each packet and included as authenticated plaintext data). This ensures the integrity and authenticity of received video data before it is further processed by the system.

8.1 Input Handling

The decoder must:

1. Accept the following inputs:
    - **TV Frame Packet** (binary data): contains the encrypted TV frame packet and authenticated metadata (channel ID and nonce), as well as the authentication tag (see Figure 2).
2. Validate the input values:
    - **TV Frame Packet** must be no more than 104 bytes in length (see Figure 2).

8.2 Key & Nonce Handling

The decoder must:

1. **Retrieve the channel decryption key** from a secure storage location based on the provided **Channel ID**.
    - Ensure access to key in RAM.
    - Handle missing or corrupted keys appropriately.

8.3 Decryption Process

The decoder must:

1. **Extract the encrypted payload** from the received packet, which includes:
    - **Timestamp** (encrypted)
    - **TV Frame** (encrypted)
2. **Verify the integrity of the packet** by computing the Poly1305 tag associated with the encrypted data and additional encrypted data.
    - Use the ChaCha20-Poly1305 key associated with the channel ID, as well as the provided nonce, to regenerate the tag.
    - Compare the computed tag with the one found in the packet. If they differ, discard the packet.
3. **Decrypt the encrypted TV Frame Data** using the **ChaCha20-Poly1305 key** associated with the channel ID.
4. **Validate the packet timestamp** to ensure it is within an acceptable range.
    - If the timestamp falls outside of the valid subscription period for the channel, discard the packet.
    - If the timestamp is less than `next_time_allowed`, discard the packet.
5. **Return the decrypted TV frame** for displaying on the host.

8.4 Error Handling

The decoder must:

1. Handle errors, including:
    - Invalid or missing encrypted packet data.
    - Failed key or nonce retrieval.
    - Decryption failures.

9 Subscription Management

9.1 Subscription Management Overview

The device's current subscriptions are managed by the host, which initializes communication with the device and sends encrypted subscription update data over UART.

9.2 Subscription File Structure

Once decrypted, the subscription files sent to the device contain the following information in plaintext:

- **Device ID**: (4 bytes) Identifies the device for which the subscription is generated.
- **Start Timestamp**: (8 bytes) Represents the start time of the subscription.
- **End Timestamp**: (8 bytes) Represents the end time of the subscription.
- **Channel**: (4 bytes) The channel the device will subscribe to.
- **ChaCha20-Poly1305 Key**: (32 bytes) The encryption key for the subscribed channel.

This structure is shown in Figure 5.

9.3 Security Considerations

The key included with the subscription is not device-specific, since the encoder uses this key to encrypt all TV frames broadcast to the given channel ID. Encrypting the TV frames for each channel with a unique key offers security benefits; in particular, if the key for any one channel is compromised, the remaining channels remain secure.

Decrypted Subscription Information — 56 Bytes

4 bytes Device ID	8 bytes Start Timestamp	8 bytes End Timestamp	4 bytes Channel ID	32 bytes Channel Key

Figure 5: A diagram of the decrypted subscription information

10 Subscription Generation

The subscription generation process happens on some unspecified secure server, similar to the environment where the encoder is run. The encoder and subscription generation process may be ran on the same machine, or it may be ran on different (secure) machines, however the global secrets are available to both processes. The process of generating subscriptions can be broken into the following steps:

1. **Server Processes Subscription Data**
   - The server fetches the key for the chosen channel from `secrets.json`.
   - The server organizes the file into the format provided in Figure 5.
2. **Server Generates Random Nonce**
   - The server randomly generates a nonce to be included in the subscription file (a random sequence of 96 bits or 12 bytes).
3. **Server Generates Additional Random Bytes**
   - The server randomly generates a string of 4 bytes to be included in the subscription file.
4. **Server Encrypts Subscription Data**
   - The server encrypts the data of Figure 5 using the **ChaCha20-Poly1305 subscription key** from `secrets.json`.

5. **Generate Poly1305 Tag**
    - The server generates a **Poly1305 message authentication code (MAC) tag** from both (i) the AAD and (ii) the encrypted subscription file using the **ChaCha20-Poly1305 subscription key** from `secrets.json` and the **random nonce** generated previously.
6. **Package Subscription Data**
    - The server packages the encrypted subscription data and the MAC tag into the format specified in Figure 6.
7. **Output File**
    - The server outputs the file; the subscription is transferred to the host using some unspecified out-of-band channel.

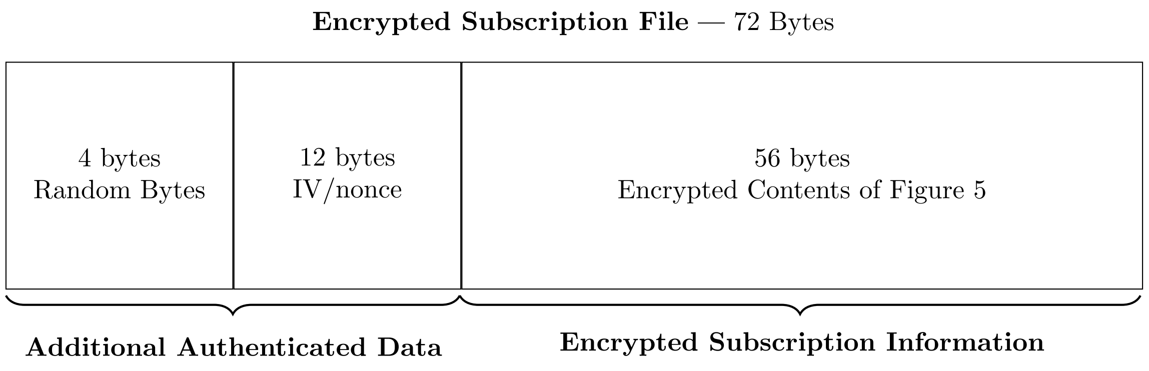
Encrypted Subscription File — 72 Bytes
4 bytes
Random Bytes
12 bytes
IV/nonce
56 bytes
Encrypted Contents of Figure 5
Additional Authenticated Data
Encrypted Subscription Information

Figure 6: A diagram of the encrypted subscription file sent to the device

11 Updating Subscription Information

The host and decoder communicate via the following protocol to update the decoder's subscriptions:

1. **Receive Subscription File**
    - The host receives the subscription file through some unspecified out-of-band channel.
2. **Send Subscription File**
    - The host sends the subscription file to the decoder over UART.
3. **Decoder Receives Subscription File**
    - The decoder receives the subscription file.
4. **Validate Subscription File**
    - The decoder validates the subscription file using the **ChaCha20-Poly1305 subscription key** stored in flash memory and the provided **nonce**.
    - The decoder regenerates the tag from these inputs. If the computed tag for the file differs from the one found in the packet, we discard the packet.
5. **Decrypt Subscription File**
    - The decoder decrypts the subscription file using the **ChaCha20-Poly1305 subscription key** stored in flash memory.
6. **Verify Subscription File**
    - The decoder verifies that the provided decoder ID matches the decoder's own ID.

- The decoder verifies that the provided start timestamp precedes the provided end timestamp.

7. **Store Channel Information**

   - The decoder stores all the provided channel information both to flash memory and the corresponding in-memory structs.

11.1 Error Handling

The decoder must:

1. Handle errors, including:

   - Invalid input data.
   - Failed key retrieval.
   - Decryption failures.
   - Invalid packet lengths.
   - Failed file read/write operations.

12 Listing Subscriptions

12.1 Listing Subscriptions Process

The List Channel Subscriptions process begins when the host sends a request and concludes when the decoder responds with a validated list of active channels for which it has subscriptions. The process is as follows:

1. The decoder **receives** a List Channel Request from the host.

2. The decoder initializes memory space to store subscription information.

3. The decoder iterates through the `decoder_status` struct. For each channel with an active subscription, the decoder:

   - Stores the channel ID
   - Stores the start timestamp
   - Stores the end timestamp

4. The decoder writes this information to a packet which is returned to the host.