MITRE eCTF Design Document

Oklahoma Christian University Engineering/Computer Science

March 28, 2025

Executive Summary

This document outlines the design and implementation strategy for the Embedded Capture the Flag (eCTF) Team's encryption system. The team is tasked with creating an encryption system that securely transmits and decodes satellite TV frames. The project consists of two parts: first, developing a secure software encryption system to handle frame transmission, and second, attempting unauthorized access to other teams' systems during the competition's attack phase.

The document covers the project's scope, including the functional and security requirements, the appropriate hardware and designing both the software and hardware components. Key functional requirements include the use of Docker for the development of a secure Decoder for decoding TV frames and subscription-based access control for frame decoding. Security requirements focus on preventing unauthorized access, ensuring proper frame timestamp sequencing and secure key exchanges.

The system will run on the Arm Cortex-M4 MAX78000FTHR processor. The architecture design encompasses a satellite-based system that transmits frames to a decoder, with a focus on security and proper data handling. The software design incorporates a secure build process, encryption solutions, and a subscription update tool to manage and validate access to TV frames.

Verification and validation are integral to the project, helping to keep track of the team's progress with proof of completion for all tasks across the two phases of the project. These phases ensure that the system meets the functional and security requirements outlined by MITRE.

In summary, the eCTF team is developing an innovative and secure encryption system for satellite TV transmissions, with a strong emphasis on secure data handling, encryption, and access control, ensuring the integrity and confidentiality of the transmitted content.

Table of Contents

1.0 Introduction

The vision of the Embedded Capture the Flag (eCTF) Team is to collaborate effectively to create a productive and innovative learning environment. Our mission is to develop an encryption system that meets all the functional and security requirements specified by the eCTF administration. The project is twofold. First, our team must use python to develop a software encryption system that will send frames to a TV satellite that can be downloaded to other devices and the frames can be decoded and the unauthorized information can be read by only authorized users. Second, our team must take the software developed by other teams, download it to our attack boards that are provided by the eCTF administration, and attempt to gain unauthorized access to the teams information. This is a recreational competition to help promote the development of software encryption skills among highschool and college students.

2.0 Team Charter

The eCTF Team is dedicated to delivering a high-quality, innovative product by collaborating effectively, utilizing our technical hardware expertise, and adhering to established design standards administered by the eCTF competition while prioritizing clear communication, timely decision-making, and continuous improvement throughout the development process.

3.0 Requirements

This section presents both the Functional and Security requirements for the Satellite TV System (STS). These requirements were set by the MITRE organization.

Functional Requirements

1. **Build Environment Setup:** The STS shall establish a build environment where all project dependencies, including compilers, packages, and build tools, are installed utilizing Docker.
2. **Secure Deployment Generation:** The STS shall generate a deployment representing a complete fabrication run of components created by the satellite TV provider. This process shall produce Global Secrets, including cryptographic key material, seeds, or other necessary data for the STS.
3. **Decoder Construction and Operation:** The STS shall build the Decoder component, which is responsible for securely decoding the TV frame data streamed over a satellite's unidirectional data stream. The Decoder shall maintain active subscriptions and correctly decode data frames on subscribed channels.
4. **Subscription Listing:** The Decoder shall list the channel numbers for which it has a valid subscription.
5. **Subscription Updates Handling:** The Decoder shall update its channel subscriptions upon receiving a valid update package. If a subscription update for a channel is received for a channel that already has a subscription, the Decoder shall only use the latest subscription update. There is no subscription update associated with the emergency broadcast channel, as frames received on this channel must always be properly decoded.
6. **TV Frame Decoding:** The Decoder shall properly decode the TV frame data from any channel for which it has a valid and active subscription installed or for any emergency broadcast frames.

Security Requirements

1. **Subscription Validation:** The STS shall ensure that an attacker cannot decode TV frames without a Decoder processing a valid, active subscription to the respective channel.
2. **Prevention of Cross-Frame Decoding:** The STS shall ensure that knowledge of the content of one encoded frame shall not enable an attacker to decode other frames.
3. **Subscription Validity Criteria:** A valid subscription refers to one installed through a subscription update generated at the same Secure Facility matching the device's ID and selected channel. An active subscription refers to a subscription for a channel where the current time falls within the active subscription window.
4. **Decoder Provisioning Validation:** The Decoder shall only decode valid TV frames generated by the STS for which the Decoder was provisioned.
5. **Single STS Per Team During Attack Phase:** During the Attack Phase, there shall be only one STS created by the organizers per team, meaning all the systems are built using the same generated secrets.
6. **Timestamp Order Enforcement:** The Decoder shall only decode frames with strictly monotonically increasing timestamps. Timestamps do not need to be sequential, but the STS shall reject misordered frames if this would violate the security requirement.

4.0 **Processor Selection/Details**

The processor used for this project was provided by MITRE. The processor selected by the organizers of the eCTF competition was the **Arm Cortex-M4** on the **MAX78000FTHR** microcontroller (*Figure 1*).

Processor Details

1. **Memory:** 512KB Flash Memory, 128KB SRAM, 16KB instruction cache.
2. **Neural Network Accelerator:** Dedicated CNN accelerator for deep learning at ultra-low power.
3. **Power Efficiency:** Optimized for AI at the edge with minimal power consumption.
4. **Connectivity:** USB, UART, I2C, SPI, and GPIO for flexible interfacing.
5. **Camera and Audio Support:** Ideal for vision and speech-based applications.
6. **Security Features:**
    - **Secure Boot:** Ensures only authenticated firmware is executed, preventing unauthorized code execution.
    - **AES Hardware Acceleration:** Supports AES-129, AES-192, and AES-256 encryption/decryption.
    - **True random Number Generator (TRNG):** Generates high-quality random numbers for cryptographic security.
    - **Secure Bootloader:** Allows encrypted firmware loading and the ability to lock the SWD port.
    - **Memory Protection:** Secure execution environment to prevent code tampering.
    - **Low Power Security Operations:** Enable cryptographic functions with minimal power consumption.

The **MAX78000FTHR** board, equipped with the **ARM Cortex-M4** processor, is a low-power AI development board with **secure boot**, **AES encryption**, and a **True Random Number Generator**, ensuring strong cryptographic protection. Its **secure bootloader** prevents unauthorized firmware modifications, making it ideal for the eCTF MITRE competition. With **firmware integrity**, **encryption**, and **low-power security processing**, it provides a strong foundation for developing and defending secure embedded systems.

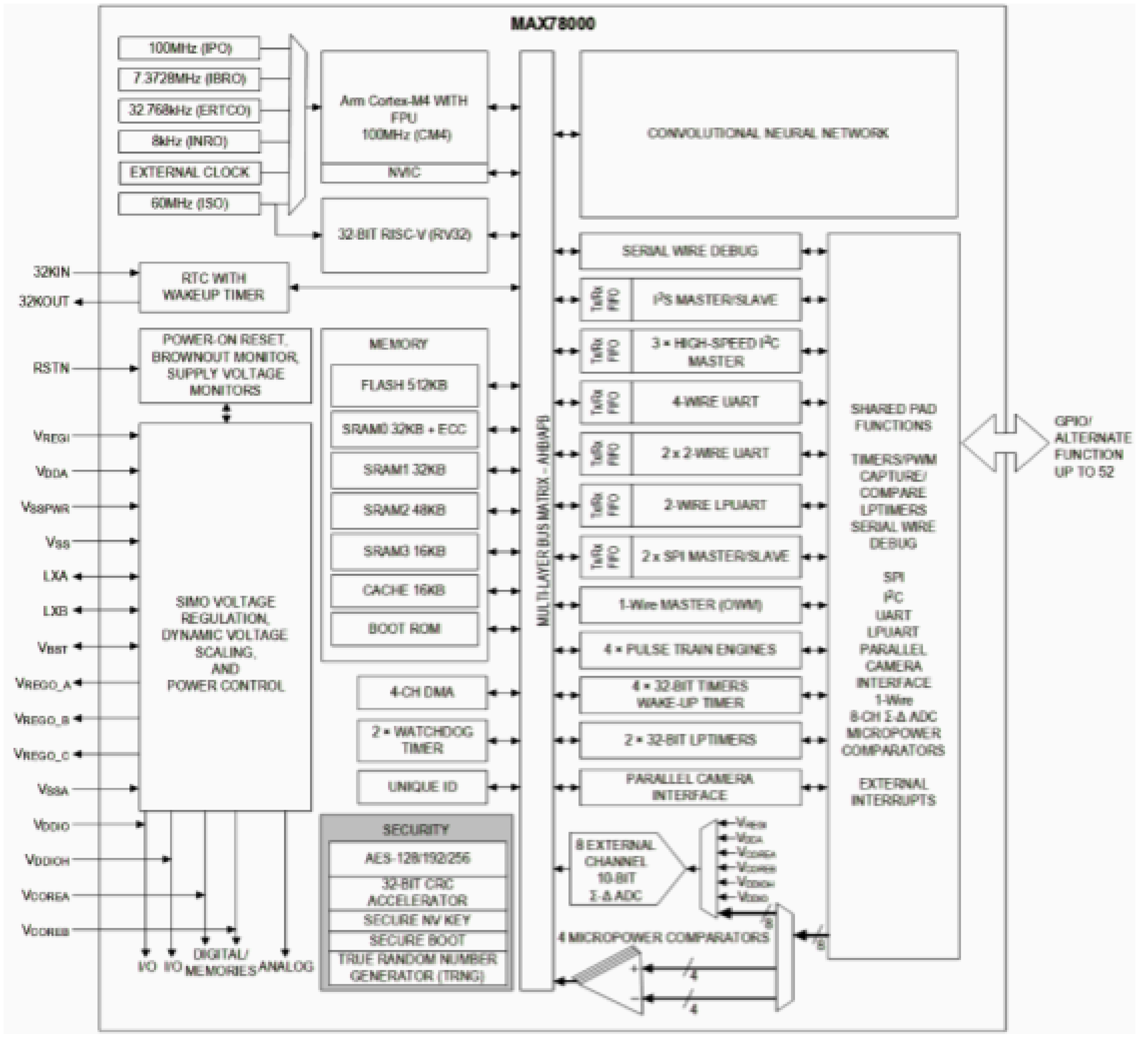


Figure 1 - This figure shows a low-level overview of the MAX78000FTHR microcontroller, including the Arm Cortex-M4 processor.

5.0 Architecture Design

The eCTF MITRE competition provides the device architecture, comprising six main subsystems: **Uplink**, **Encoder**, **Satellite**, **Host Computer**, **TV**, and **Decoder**. The Uplink sends frames to the Encoder, then forwards them to the Satellite, which broadcasts them to TVs on Host Computers. The Host Computer connects to the Decoder, which is responsible for decoding live-streamed data over multiple channels. The Decoder is the primary focus, ensuring accurate data frame decoding and subscription management. The architecture described is shown in *Figure 2*.

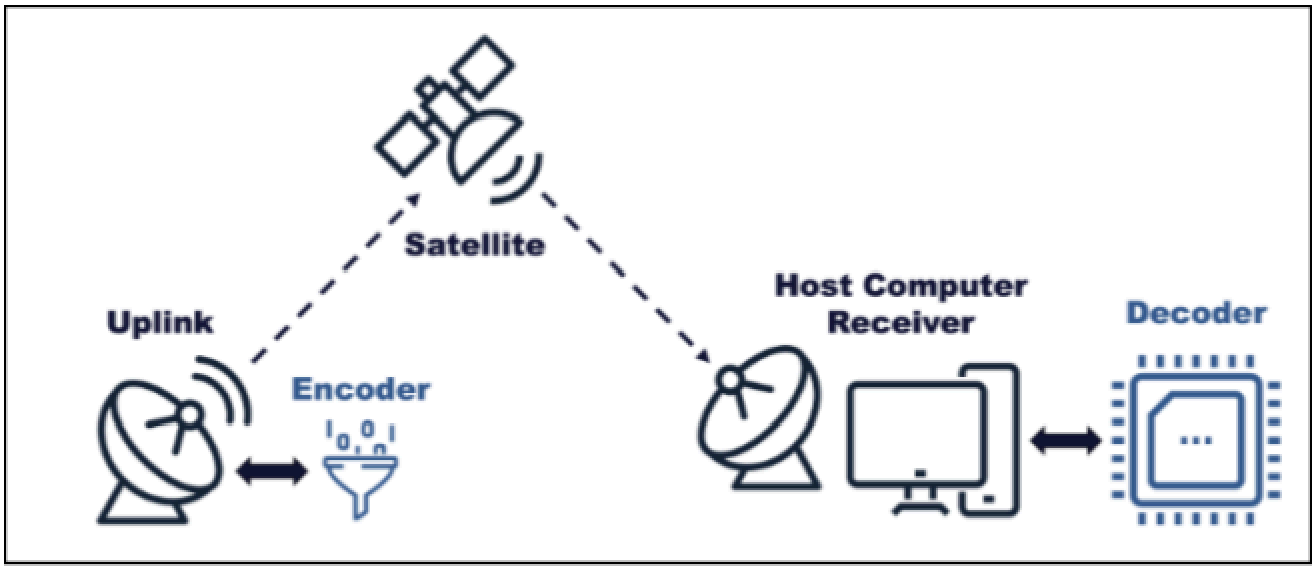


Figure 2 - This figure, provided by eCTF shows a high-level overview of the functional requirements.

6.0 Hardware Design

This project will use the MAX78000FTHR board to implement the MAX78000 processor. It is standardized across all eCTF teams.

Architecture

The development resources provided by the MITRE organizations include **3 unkeyed MAX78000FTHR** boards for the development of the design. These devices will not be able to run the Attack Phase designs provisioned by the eCTF organizers. However, the development microcontrollers can be used to practice attacks against designs in the Attack Phase that are compiled locally by the team from source. **3 keyed MAX78000FTHR** boards will also be provided for the Attack Phase to secure the load of other teams' designs that will be provided by the eCTF organizers. These devices will be unusable during the Design Phase.

Diagrams

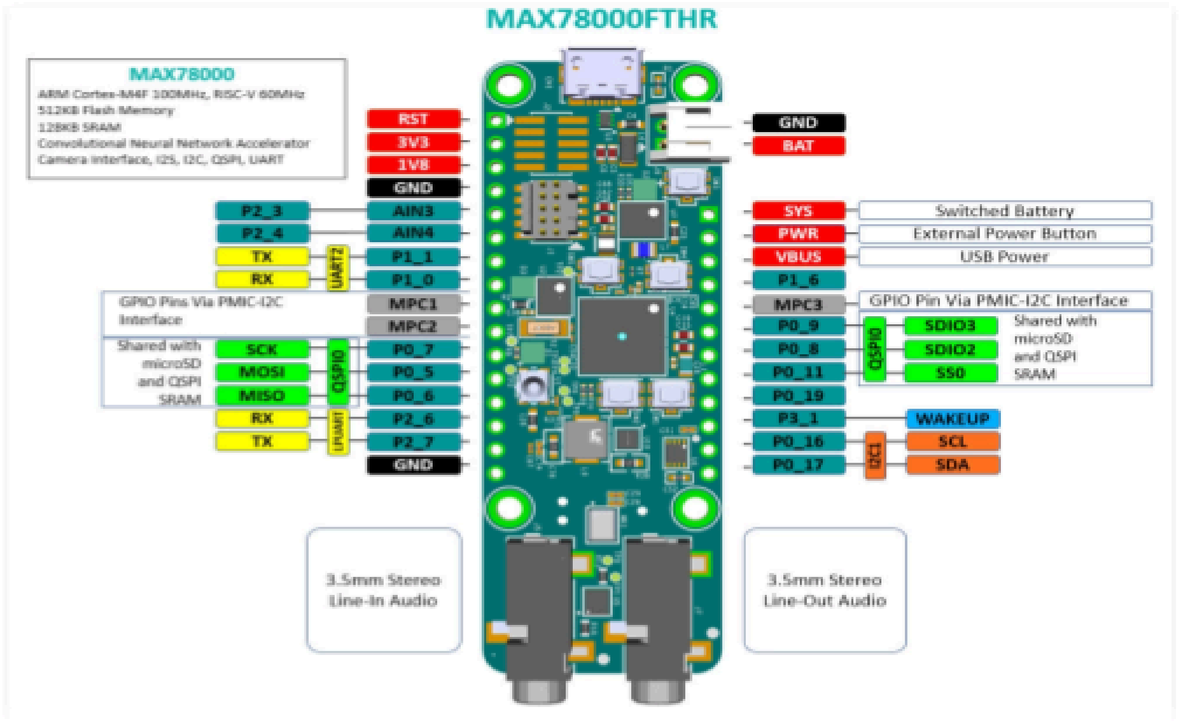

MAX78000FTHR
MAX78000
ARM Cortex-M4F 100MHz, RISC-V 60MHz
512KB Flash Memory
128KB SRAM
Convolutional Neural Network Accelerator
Camera Interface, I2S, I2C, QSPI, UART
RST
3V3
1V8
GND
P2_3
AIN3
P2_4
AIN4
TX
RX
UART2
P1_1
P1_0
GPIO Pins Via PMIC-I2C Interface
MPC1
MPC2
Shared with microSD and QSPI SRAM
SCK
MOSI
MISO
QSPI0
P0_7
P0_5
P0_6
RX
TX
LPUART
P2_6
P2_7
GND
GND
BAT
SYS
Switched Battery
PWR
External Power Button
VBUS
USB Power
P1_6
MPC3
GPIO Pin Via PMIC-I2C Interface
P0_9
P0_8
P0_11
QSPI0
SDIO3
SDIO2
SS0
Shared with microSD and QSPI SRAM
P0_19
P3_1
WAKEUP
P0_16
P0_17
I2C1
SCL
SDA
3.5mm Stereo Line-In Audio
3.5mm Stereo Line-Out Audio

Figure 3 - Pinout for the MAX78000FTHR board.

MAX78000FTHR Application Platform Diagram

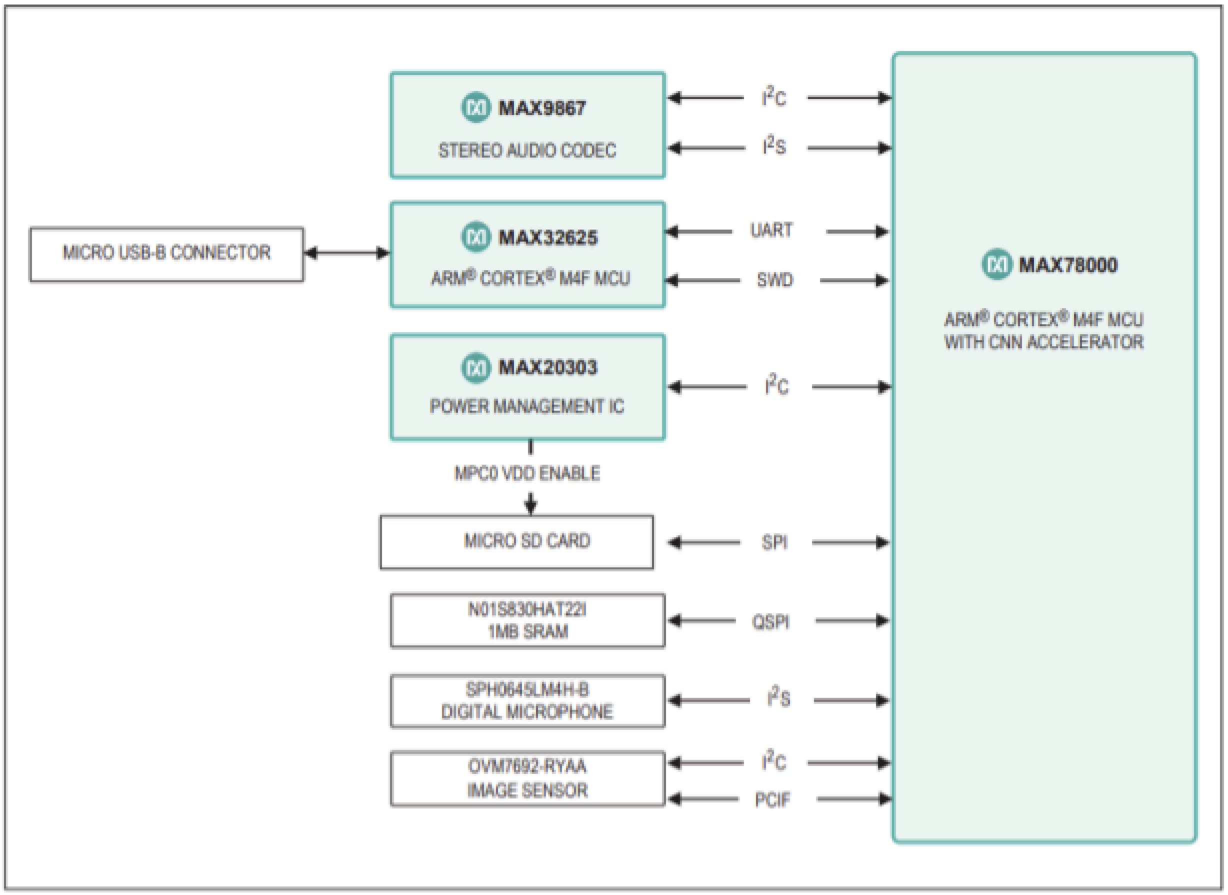


Figure 4 - Diagram showing the external connectors to the MAX7800FTHR board.

7.0 Software Design

Build Process

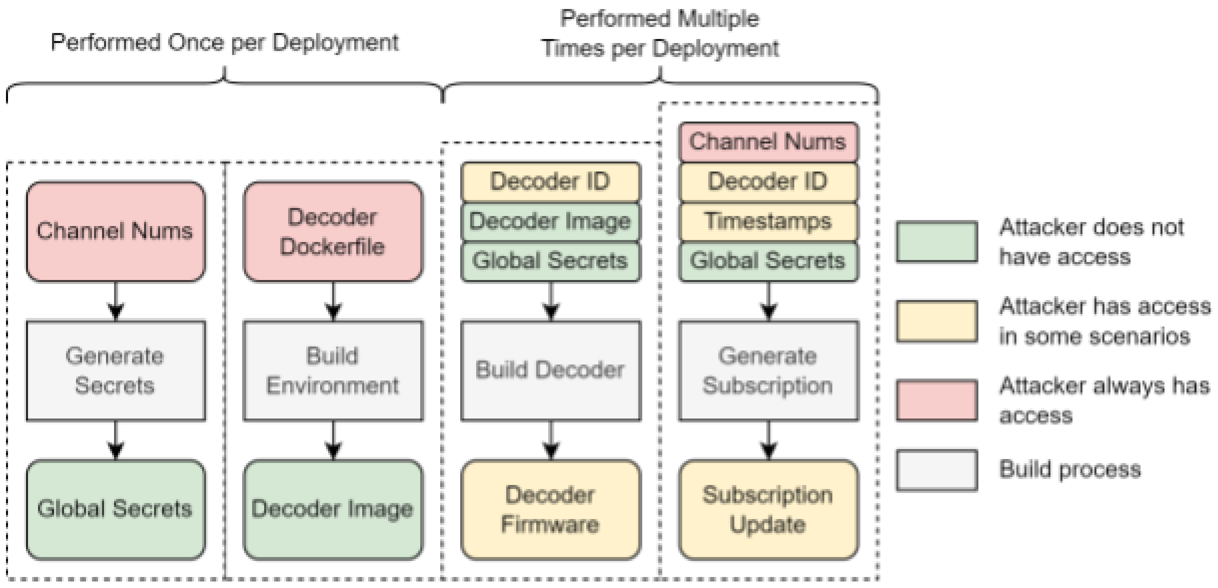


Figure 5 - This figure, provided by eCTF, shows a high-level flow chart of the functional software requirements.

1. **Build Environment:** All dependencies will be installed using Docker.
    a. **Compiler:** GCC (GNU Compiler Collection)
    b. **Packages:** Docker Client
2. **Build Development:**
    a. **Generate Secrets:**
        - Holds cryptographic keys, seeds and any other build design data.
        - Should be treated as read only files.
    b. **Secure Secrets:**
        - Attackers will not have access to global secrets.
        - After the secrets are generated, no new secrets will be added.
        - There will be only one deployment for each scenario.

Subscription and Decoding

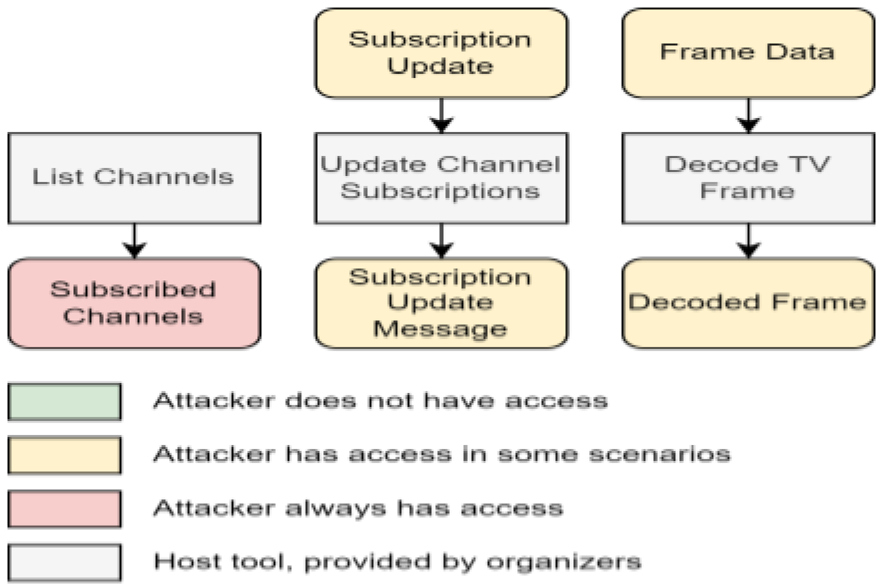
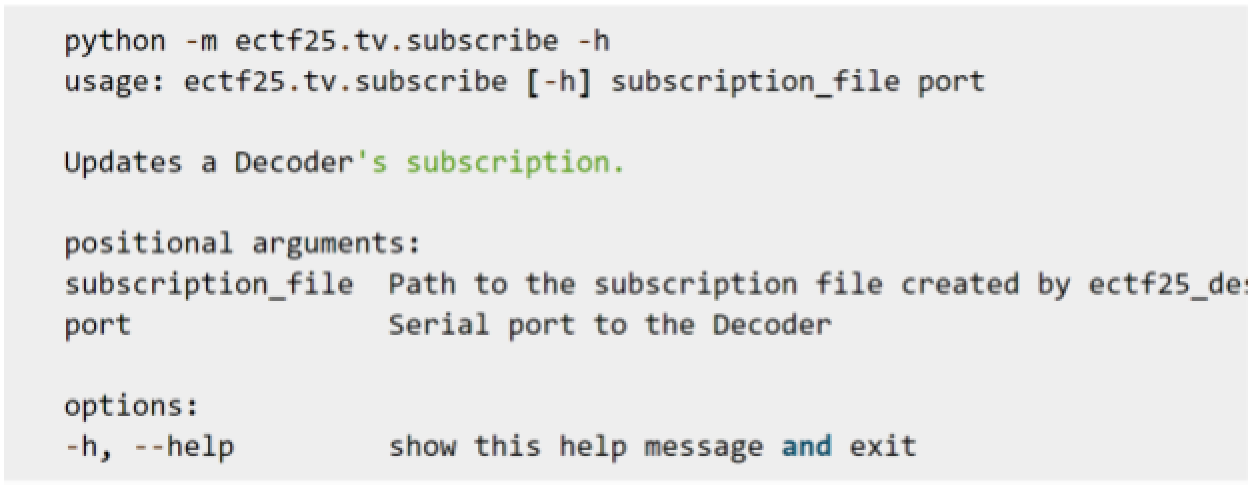

Subscription Update
Frame Data
List Channels
Update Channel Subscriptions
Decode TV Frame
Subscribed Channels
Subscription Update Message
Decoded Frame
Attacker does not have access
Attacker has access in some scenarios
Attacker always has access
Host tool, provided by organizers

Figure 6 - Subscription and Decoder Process

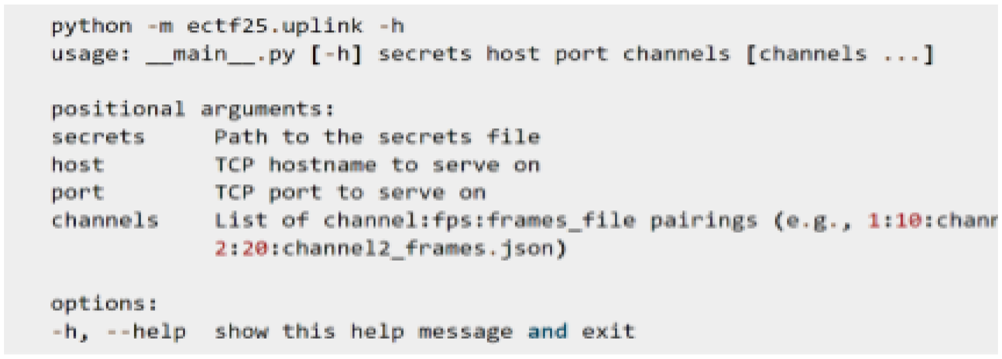
1. **Subscription Update Tool:** The Subscription Update Tool applies the required subscription update functionality from the Satellite TV Decoder System. The Subscription Update Tool takes in an encoded update packet (in the form of a *.bin* file) and sends it to the Decoder.



```
python -m ectf25.tv.subscribe -h
usage: ectf25.tv.subscribe [-h] subscription_file port

Updates a Decoder's subscription.

positional arguments:
subscription_file  Path to the subscription file created by ectf25_de
port               Serial port to the Decoder

options:
-h, --help         show this help message and exit
```

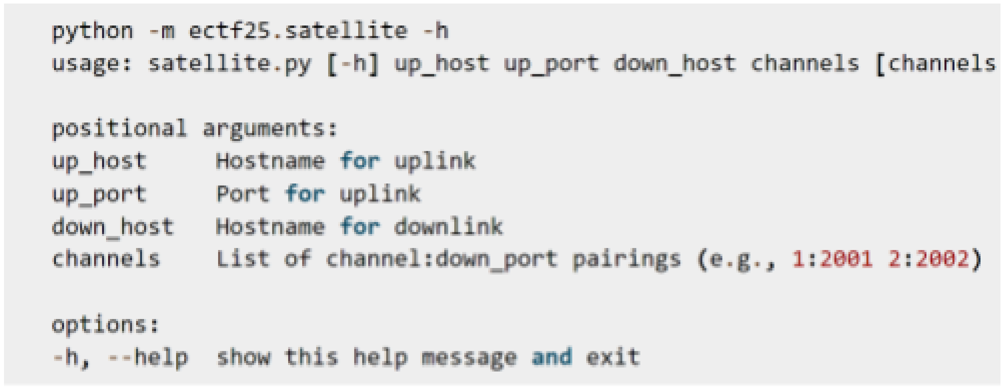
Figure 6 - Subscription Update Tool code.

2. **Running the Satellite and Encoder:** To run all of the infrastructure, the Uplink must first be started. Then in a separate terminal window, start the satellite. And finally, start the TV instance for every Decoder being tested.



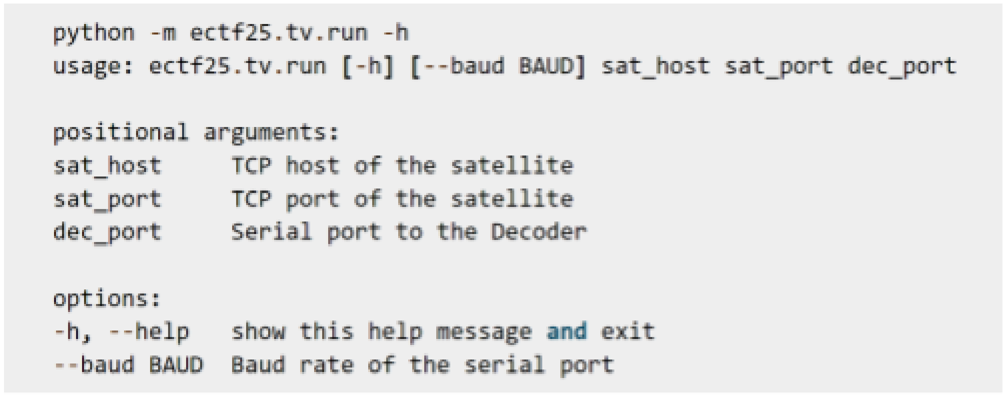
```
python -m ectf25.uplink -h
usage: __main__.py [-h] secrets host port channels [channels ...]

positional arguments:
secrets     Path to the secrets file
host        TCP hostname to serve on
port        TCP port to serve on
channels    List of channel:fps:frames_file pairings (e.g., 1:10:chann
            2:20:channel2_frames.json)

options:
-h, --help  show this help message and exit
```

Figure 7 - Utilizing the Uplink.



```
python -m ectf25.satellite -h
usage: satellite.py [-h] up_host up_port down_host channels [channels

positional arguments:
up_host     Hostname for uplink
up_port     Port for uplink
down_host   Hostname for downlink
channels    List of channel:down_port pairings (e.g., 1:2001 2:2002)

options:
-h, --help  show this help message and exit
```

Figure 8 - Utilizing the Satellite.



```
python -m ectf25.tv.run -h
usage: ectf25.tv.run [-h] [--baud BAUD] sat_host sat_port dec_port

positional arguments:
sat_host     TCP host of the satellite
sat_port     TCP port of the satellite
dec_port     Serial port to the Decoder

options:
-h, --help   show this help message and exit
--baud BAUD  Baud rate of the serial port
```

Figure 9 - Utilizing the TV.

Encryption Implementation/Solutions:

Encryption:

- **Solution:** Encryption allows us to not only hide the data being transferred from the satellite to the decoder, but also verify that no one has tampered with the data. To implement this encryption, we first fully encrypt the packet on the server side. Everything is encrypted, from decoder ids to timestamps to frames. This encrypted data is transferred to the decoder, which proceeds to decrypt the data and serialize it back into the original data types. Finally, we can examine the message and return the decoded message if all goes well.

Buffers:

- **Solution:** We also tried to be careful when working with buffers to reduce the amount of data they could access. Overall, we attempted to use memory-safe coding techniques, although in order to be fully memory safe, we would've needed to rewrite the code using a language like rust.

Security Requirement 1:

- **Solution:** Subscription matching using the device ID and the channel numbers to ensure that only devices that are valid and active are able to decode the proper traffic. Additionally, we implemented AES GCM encryption, which fully encrypts all the subscription data sent to the decoder (including timestamps and the device ID), so that the attacker can't see any data by attempting to read the subscription file.

Security Requirement 2:

- **Solution:** AES GCM encryption ensures that knowledge of one encoded frame does not enable an attacker to decode other frames. Additionally, the GCM encryption supports integrity, preventing attackers from changing the frame during transit.

Security Requirement 3:

- **Solution:** AES GCM encryption on subscription updates ensures validity. Subscription decryption is only allowed if the request tie falls within the active subscription window. The encryption supports integrity, preventing attackers from manipulating the channel or timestamps of the subscription during transit. Though we used a static IV, which could open the door to some attacks, we think that because of the built-in data integrity of the GCM protocol, it will still be hard to spoof a subscription.

Security Requirement 4:

- **Solution:** Diffie-Hellman secure key exchange ensures that only the correct decoder can establish communication with the satellite.

Security Requirement 5:

- **Solution:** Unique identifier checks on the decoder ensure that only the designated device can decrypt frames from the satellite, ensuring a single STS per team.

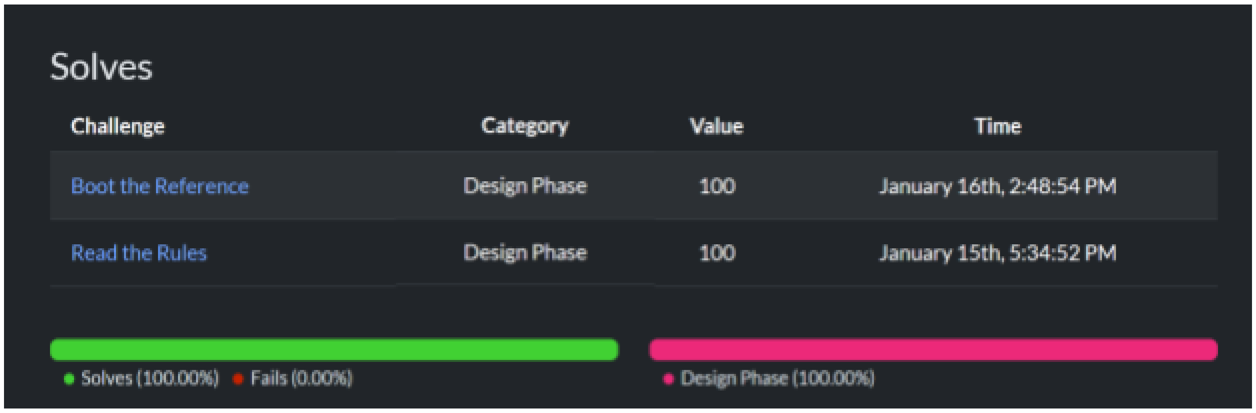
Security Requirement 6:

- **Solution:** Timestamp validation checks ensure that the decoder only processes frames with strictly increased timestamps. Encryption only begins when timestamps synchronize, preventing misordered frame processing.

8.0 Verification/Validation

Design Phase

- **Read Rules** COMPLETED

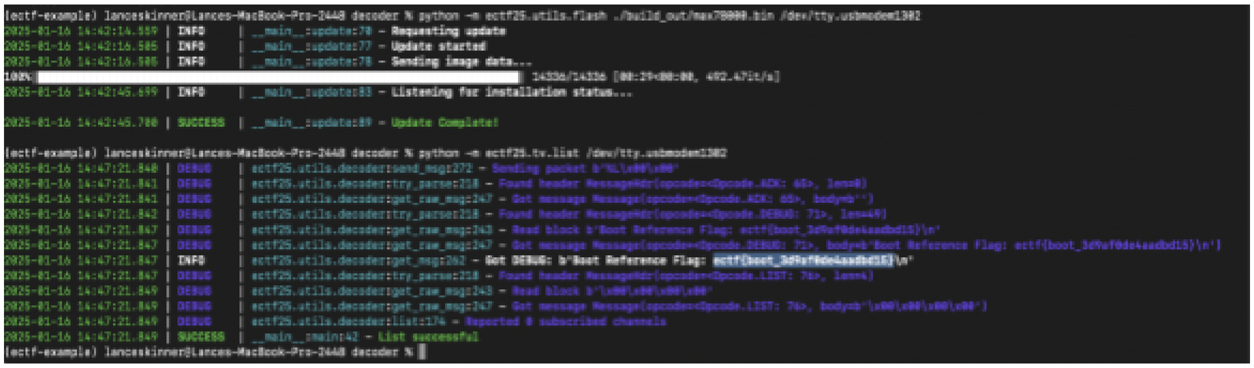


Solves

Challenge	Category	Value	Time
Boot the Reference	Design Phase	100	January 16th, 2:48:54 PM
Read the Rules	Design Phase	100	January 15th, 5:34:52 PM

Figure 10 - Completion of Read the Rules and Boot the Reference Flag

- **Boot Reference Design** COMPLETED



```
(ectf-example) lanceskinner@Lances-MacBook-Pro-2448 decoder % python -m ectf25.utils.flash ./build_out/max78000.bin /dev/tty.usbmodem1302
2025-01-16 14:42:14.559 | INFO     | __main__:update:70 - Requesting update
2025-01-16 14:42:16.505 | INFO     | __main__:update:77 - Update started
2025-01-16 14:42:16.506 | INFO     | __main__:update:78 - Sending image data...
100%|████████████████████████████████████████| 14336/14336 [00:29<00:00, 492.47it/s]
2025-01-16 14:42:45.699 | INFO     | __main__:update:83 - Listening for installation status...

2025-01-16 14:42:45.700 | SUCCESS  | __main__:update:89 - Update Complete!

(ectf-example) lanceskinner@Lances-MacBook-Pro-2448 decoder % python -m ectf25.tv.list /dev/tty.usbmodem1302
2025-01-16 14:47:21.840 | DEBUG    | ectf25.utils.decoder:send_msg:272 - Sending packet b'%L\x00\x00'
2025-01-16 14:47:21.841 | DEBUG    | ectf25.utils.decoder:try_parse:218 - Found header MessageHdr(opcode=<Opcode.ACK: 65>, len=0)
2025-01-16 14:47:21.841 | DEBUG    | ectf25.utils.decoder:get_raw_msg:247 - Got message Message(opcode=<Opcode.ACK: 65>, body=b'')
2025-01-16 14:47:21.842 | DEBUG    | ectf25.utils.decoder:try_parse:218 - Found header MessageHdr(opcode=<Opcode.DEBUG: 71>, len=49)
2025-01-16 14:47:21.847 | DEBUG    | ectf25.utils.decoder:get_raw_msg:243 - Read block b'Boot Reference Flag: ectf{boot_3d9af0de4aadbd15}\n'
2025-01-16 14:47:21.847 | DEBUG    | ectf25.utils.decoder:get_raw_msg:247 - Got message Message(opcode=<Opcode.DEBUG: 71>, body=b'Boot Reference Flag: ectf{boot_3d9af0de4aadbd15}\n')
2025-01-16 14:47:21.847 | INFO     | ectf25.utils.decoder:get_msg:262 - Got DEBUG: b'Boot Reference Flag: ectf{boot_3d9af0de4aadbd15}\n'
2025-01-16 14:47:21.847 | DEBUG    | ectf25.utils.decoder:try_parse:218 - Found header MessageHdr(opcode=<Opcode.LIST: 76>, len=4)
2025-01-16 14:47:21.849 | DEBUG    | ectf25.utils.decoder:get_raw_msg:243 - Read block b'\x00\x00\x00\x00'
2025-01-16 14:47:21.849 | DEBUG    | ectf25.utils.decoder:get_raw_msg:247 - Got message Message(opcode=<Opcode.LIST: 76>, body=b'\x00\x00\x00\x00')
2025-01-16 14:47:21.849 | DEBUG    | ectf25.utils.decoder:list:174 - Reported 0 subscribed channels
2025-01-16 14:47:21.849 | SUCCESS  | __main__:main:42 - List successful
(ectf-example) lanceskinner@Lances-MacBook-Pro-2448 decoder % 
```

Figure 11 - Boot Reference Flag

- Design Document COMPLETED
- Debugger
- Testing Service
- Attack the Reference Design
- Final Design Submission
- Bug Bounty

Attack Phase

- Expired subscription
- Pirated Subscription
- No Subscription
- Recording Playback
- Pesky Neighbor

9.0 References

MITRE. (2025). *Security requirements*. https://rules.ectf.mitre.org/2025/specs/security_reqs.html

MITRE. (2025). *Functional requirements*. https://rules.ectf.mitre.org/2025/specs/functional_reqs.html

MITRE. (2025). *Detailed specifications*. https://rules.ectf.mitre.org/2025/specs/detailed_specs.html

Elektor. (2025, January 30). *AI at the edge: Get started with the Maxim MAX78000FTHR*. Elektor. https://www.elektormagazine.com/articles/ai-edge-get-started-maxim-max78000fthr

Analog Devices. (2025, January 30). *MAX78000FTHR data sheet*. Analog Devices. https://www.analog.com/media/en/technical-documentation/data-sheets/max78000fthr.pdf

NewAE Technology Inc. (2025, February 06). *NAE-CW1101*. NewAE Technology. https://www.newae.com/products/nae-cw1101