Design Document

Purdue 2: Hammer Industries

MITRE eCTF 2025

Contents

Rust

Due to its well-known advances in static code analysis for memory-safety, Rust has been chosen for the development of the satellite TV system. However, due to the nascency of the MAX78000 processor, there is no published, public, known Rust programming resource from the microcontroller designer for our team to utilize. As such, the design utilizes a hardware abstraction layer (HAL) that we developed in conjunction with this project. Our HAL was extended from the open-source Rust HAL developed by UIUC's sigpwny team.

Functional Design

Build Satellite TV System

Build Environment

Our design will be developed using the Rust language due to the additional security native to the language over C. To program and compile the code, we have implemented the rust tool chain for compilation. While compiling the main program, the starting address within the linker file was modified to 0x1000E20C to match the reference design.

Build Deployment

Our design uses 256-bit AES key/128-bit AES initial value (IV) pairs for secrets. Two types of AES secrets are generated, a master secret and per-channel secrets. The per-channel secrets are generated for channel 0 and for any additional channels passed into the channels parameter of the gen_secrets function. A master secret is then generated in the same way. These will then be encoded into a json object, with the key being either master or the channel number and the value being an array. The array contains a pair of base64 encoded strings representing the secret pair data. The json object will then be encoded as a UTF-8 string into raw bytes.

Subscription update packages are built in a way that ensures subscriptions will only be valid for the decoder they are intended for (see Figure 1 below). This is done by using the per-channel and master AES secrets to encode the update package. For any subscription update package, the first piece of information processed is the decoder ID. This ID is zero padded and processed as little-endian bytes into a 128-bit block of data. This is then encrypted using the per-channel secret for the channel input into the function. If this channel is not valid for this deployment, i.e. there is no secret associated with that channel number or the channel number is 0, the function will fail. If the channel number is valid, the start timestamp, end timestamp, and channel number will be appended to the encrypted device ID, and the resulting string will be encrypted using the master secret. This constitutes a valid subscription update package.

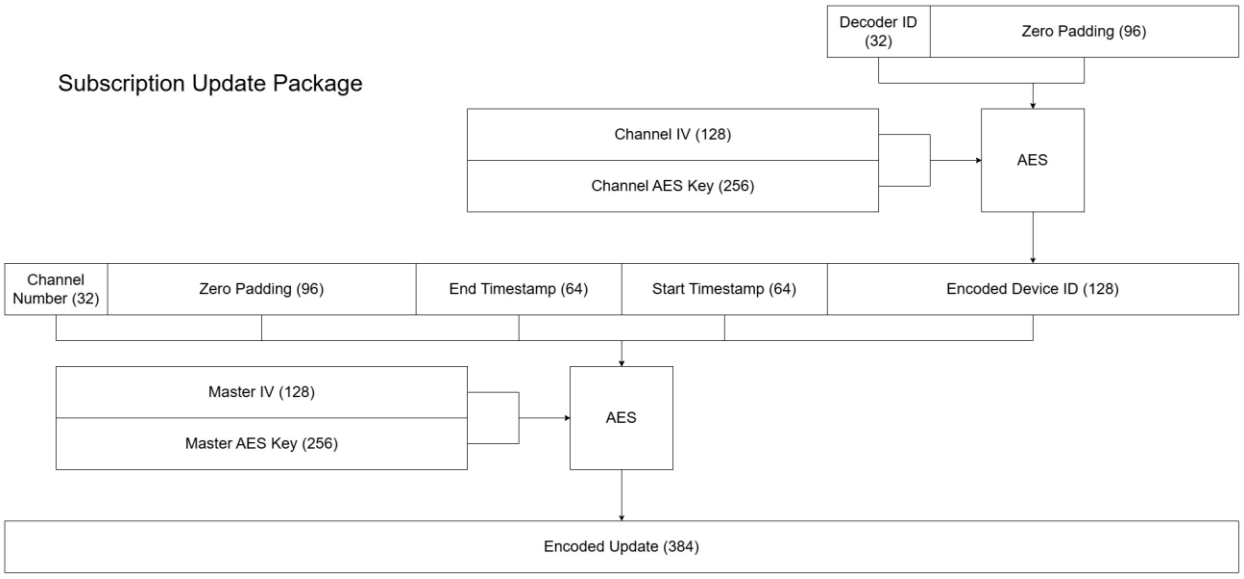

Figure 1: Subscription Update Encoding Process

Build Decoder

Once the AES secrets have been generated, the data will be passed to the docker toolchain when compiling the binary. The toolchain will dynamically insert the data into the binary, such that this information is baked into the program running on the decoder. The toolchain will not complete successfully unless valid secrets and a device ID are present.

Encoder

Frames are encoded using a two-step encoding process (see Figure 2 below). First, the company stamp is prepended to the frame data. Then, this data is encoded using the per-channel AES secret. Then, the channel number, timestamp, and frame length are appended to the encoded frame data. Lastly, the master AES secret is used to encode the entire package. This ensures that only a decoder that shares the global secrets with the encoder can decode the package. This additionally ensures that the channel number and timestamp cannot be easily changed post encoding, as an attacker would have to break through the AES encryption using the master secret.

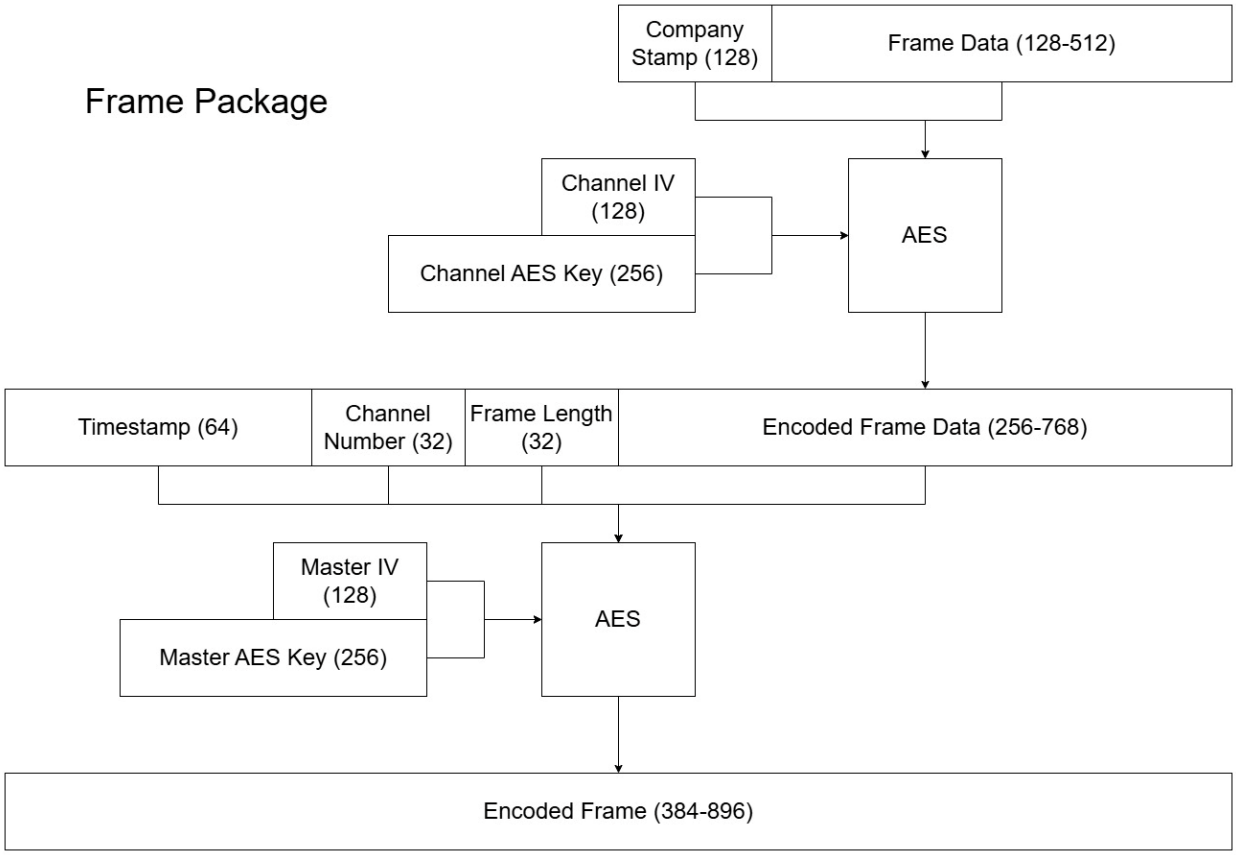

Figure 2: Frame Encoding Process

Decoder

Subscription Storage

Subscriptions are stored in an exclusive page of flash. Subscriptions to each channel are stored in an array which each element contains the following data in a struct:

- Validity of channel: 1 Byte
- Channel ID: 4 Bytes
- Start timestamp: 8 Bytes
- End timestamp: 8 Bytes

List Command

Upon receiving a list command, the microcontroller reads and decrypts the subscription page. It then sends the array over UART to the host computer. It is sent in an equivalent manner to how it is stored in flash with invalid subscriptions removed.

Update

Subscription update packages arrive as encoded data. The first step in decoding the package is to use the master AES secret for this deployment. At this step, the start timestamp, end timestamp, and channel ID should be valid and retrievable from the package. If the package is in an unexpected format at this point, the command will error. After this happens, the encoded device ID can be retrieved. This is then decrypted using the per-channel AES secret matching the retrieved channel number. If there is no key for the channel, or the channel number is 0, the command will error. Finally, once the device ID is decoded, it is checked against the device ID stored on the decoder. If it doesn't match, either because the subscription was provisioned for a different device or the secrets weren't valid, the command will error. If all checks pass, the subscription will be 100% overwritten with the data retrieved.

Decode

Frames are decoded using a three-step process. First, the data is decrypted using the master AES secret. The decoder will then be able to retrieve the channel number, timestamp, and length of the frame. Using the subscription data on the board, the decoder will check that the user has an active subscription for the correct time for the frame. The decoder will also check the current time data on the board, ensuring that the frame's timestamp is greater than the current time on the board. It will also check that the remaining body length matches the frame length provided. If any of these checks fail, the decoder will not continue decoding the data. If all checks succeed, the decoder will proceed to decode the first block of data using the per-channel AES secret. The data resulting from this should match the company stamp. If it does not, the decoder will not proceed. The decoder will then decrypt all of the following data and check that the resulting bytes match the amount of data listed. Lastly, the decoder will first overwrite the current time on the board with the timestamp of the frame and serve the frame back to the TV.

Security Requirements

This section outlines the design's features at a high level to meet security requirements.

SR1

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

In the encoder, all packet bodies will be sent encrypted with a secret unique to each channel. In the decoder, this body will only be decrypted after checking the subscription's validity, existentially and temporally. If the decryption process fails, the decoder will take no further action and discard the packet. The secrets used to decrypt the frame are only present in the decoder flash and are never transmitted after boot.

SR2

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

All sent frames will be encrypted using a master secret only known by the encoder and the decoder. This is in addition to the encryption applied to the body of the message. If the decryption process fails, the decoding process with fail and the remainder of the packet will be discarded.

SR3

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

If a frame from a previous timestamp is detected, it will be discarded, and no further action will be taken by the decoder. *After* each decoded packet has been verified as originating from the encoder and after the timestamp has been checked to be after the previous packet, a global timestamp variable will be updated.

Side-Channel Attack Countermeasures

Side channels are a common attack method to analyze the current state of a running program. In anticipation of this attack method, we are attempting to generate 2 sources of noise to inhibit attacks through run-time hardware obfuscation: power and timing. Additionally, all functions which examine the state of any secret to be branchless during the examination of the secret. Thus, little timing information may be gleaned from execution.

Side-channel noise will be generated primarily by the execution of two processes

1. **Random Duration**: after receiving a message, the processor waits a random amount of time between 100 and 500 microseconds to throw off any timing analyzers that attempt to synchronize on the transmission or receiving of the initial command packet.
2. **Power Noise**: during boot, the processor invokes a non-standard operating voltage to confuse power analyzers that are used to the typical operating voltage of the microcontroller.

A Note on AES

Through the development process, it was decided to use a sort of Anti-AES. In all instances in the design, all AES "encryption" is done using the decryption protocol and vice-versa. Additionally, the CBC process is applied in its non-reversed form to preserve its purpose.

Timing Requirements

All timing requirements will be benchmarked prior to handoff in a variety of scenarios and tests to ensure compliance.

Using the stability test, the team was able to observe an average encoding rate of around 3000 frames per second and an average decoding rate of around 25 frames per second.

Additional Libraries Used

Python

As part of the encoder, the pycryptodome library is used to execute the AES encryption.

Rust

As part of the decoder, the following packages are used in either release or as part of the build process:

- cortex-m
- cortex-m-rt
- max7800x-hal (our fork)
- panic-halt
- embedded-alloc
- rand_chacha
- rand_core
- base64
- serde
- serde_json