Design Document for Parkway Spark!

Eric Lipsutz, Samuel Lipsutz, Shaswath Arunshankar, Garrick Schinkel

Table of Contents

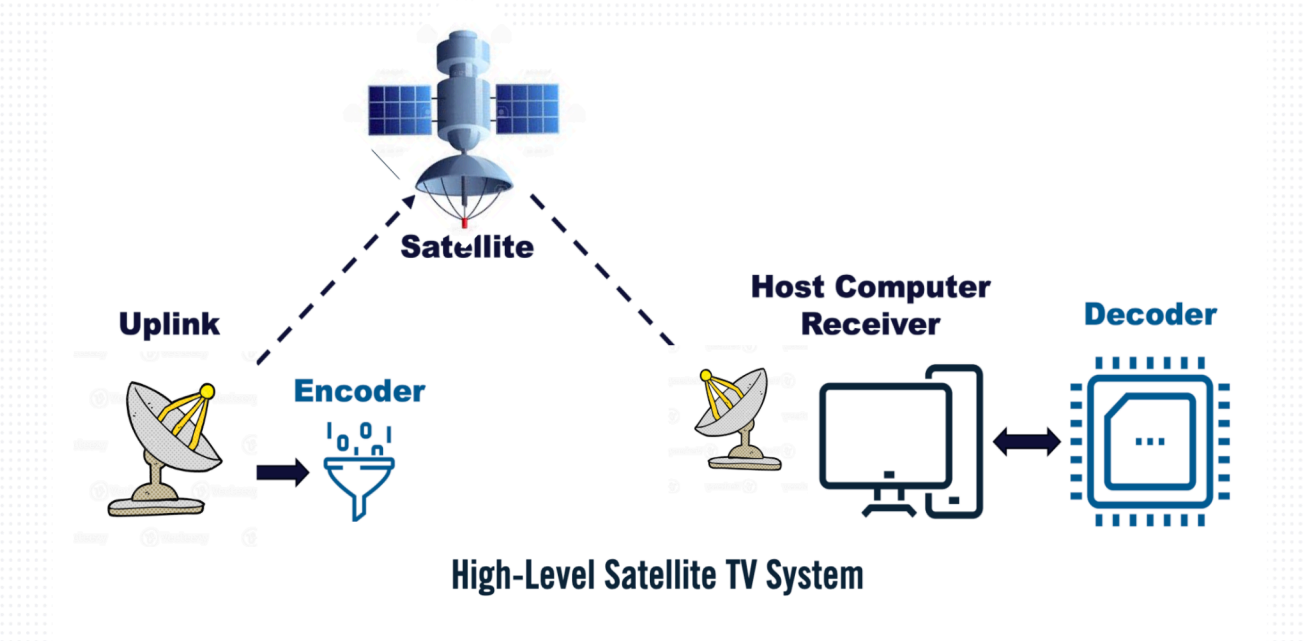
Terminology and Symbols

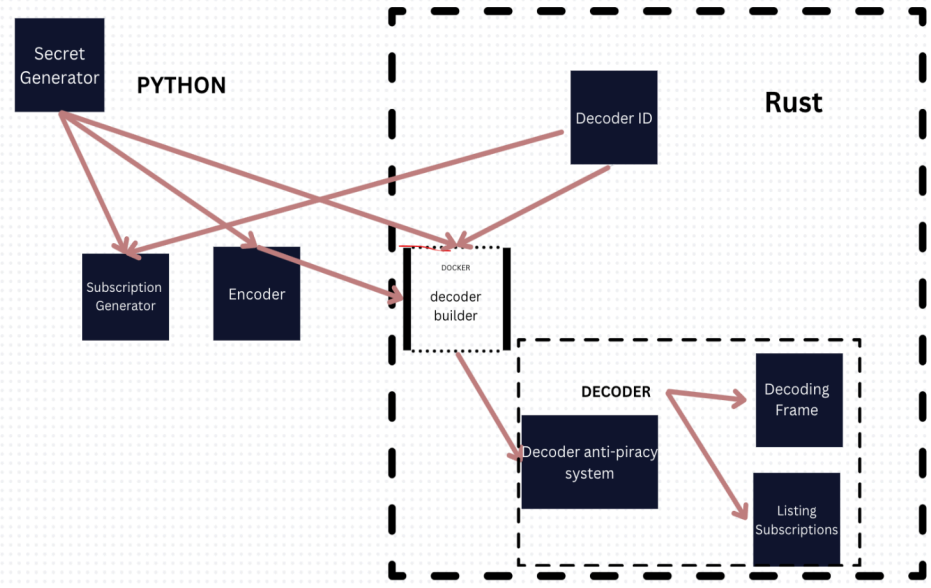
- Backward key, referred to as B_x: One of the two keys that allows a frame at the timestamp x to be decoded, the other one being the forward key. B_x may be converted into B_{x-1} far more easily than into B_{x+1}.
- Forward key, referred to as F_x: One of the two keys that allows a frame at the timestamp x to be decoded, the other one being the backward key. F_x may be converted into F_{x+1} far more easily than into F_{x-1}.
- Intermediate: a precomputed F_x or B_x that is provided to the decoder
- System secret: a precomputed 64-bit integer used to hinder piracy
- ID: The Decoder's ID
- C_{ID}: The channel ID
- $\oplus$: The rotating XOR function
- Ed25519: An elliptic curve signature scheme using SHA-512 hashes and Curve25519.
- Blake3: a modern, efficient hashing algorithm used for moving strictly in one direction through the forwards and backwards key sequences.
- AES: a modern symmetric encryption algorithm, of which the 128-bits output-feedback mode is used for preventing subscription piracy.

Building the Decoder

1. Clone the `xnossisx/spark-ectf` repository
2. Download Docker and Python (if they are not already installed)
3. Setup the virtual environment through `python -m venv .venv --prompt spark-ectf` and activate it with `. ./.venv/bin/activate`
4. Install the design and tools packages with `python -m pip install ./design` and `python -m pip install ./tools`
5. Create a set of secrets (preferably in `./global.secrets`)
6. Run `docker build -t build-decoder ./decoder` to create the Docker image
7. Run `docker run --rm -v ./build_out:/out -v ./decoder:/decoder -v ./global.secrets:/global.secrets -e DECODER_ID=0xdeadbeef build-decoder` to build a decoder with ID 0xDEADBEEF
8. Upload build_out/max78000.bin to a MAX78000FTHR through the normal method (unplug, press and hold SW1, plug in, flash)

Technical Architecture Diagram

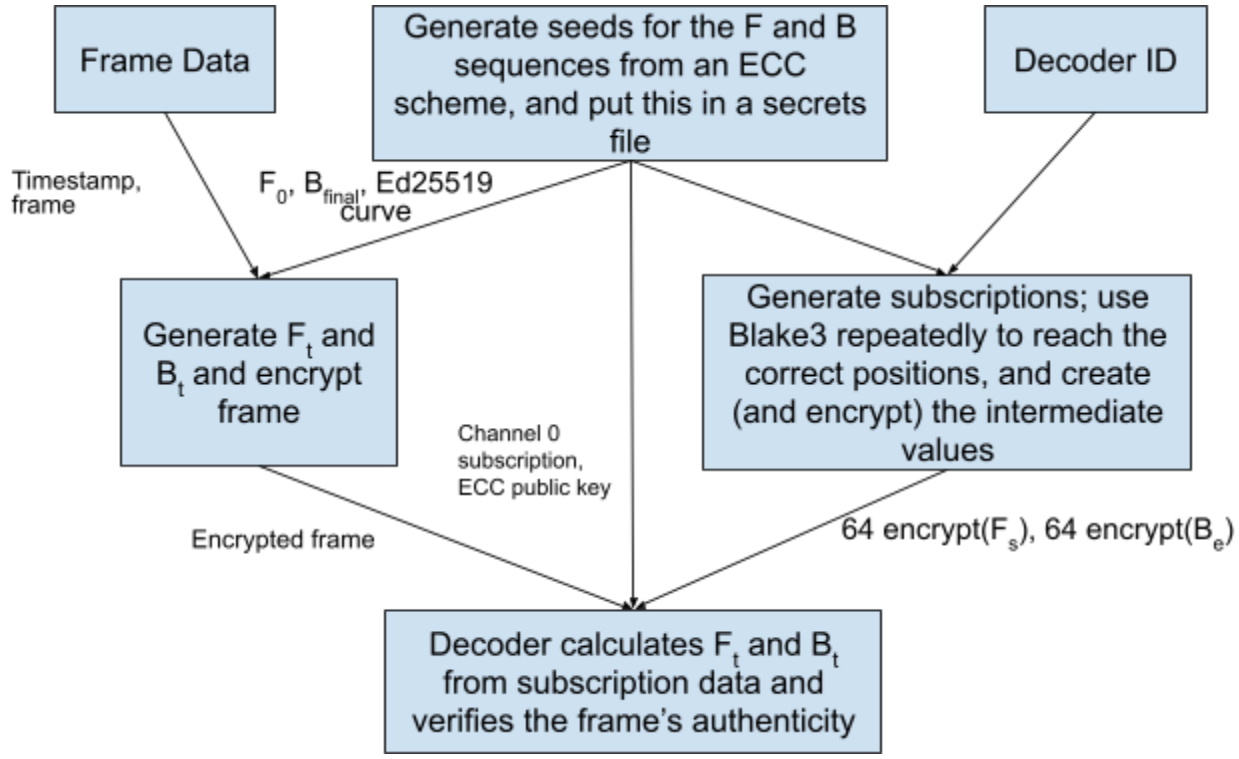




Frame Processing

Frame Data

Each decrypted and encrypted frame is simply represented by a 64-byte array. Each signal is additionally preceded by a byte identifying the channel.



Secrets

One set of secrets is created for every single channel passed into the secret generation function, including those for channel 0. A subscription for channel 0 is automatically generated, as described in Subscription Generation, which is passed into the the decoder build.

Initially, the elliptic curve Ed25519 is used to generate a random elliptic-curve cryptography (ECC) key. A 64-bit integer made to secretly and uniquely identify the satellite system is also randomly generated (known as the system secret). Finally, two 128-bit integers are created using pseudo-RNG: $\mathbf{F_0}$ and $\mathbf{B_{final}}$ for every single channel.

Encoding

Each subscription provides two keys that can decode frames within an interval. In order to encrypt a frame $\mathbf{O_n}$ at timestamp **n** into the encoded version $\mathbf{X_n}$, the encoder performs this operation, where $\oplus$ is a typical bitwise XOR:

$$X_n = O_n \oplus B_n \oplus F_n$$

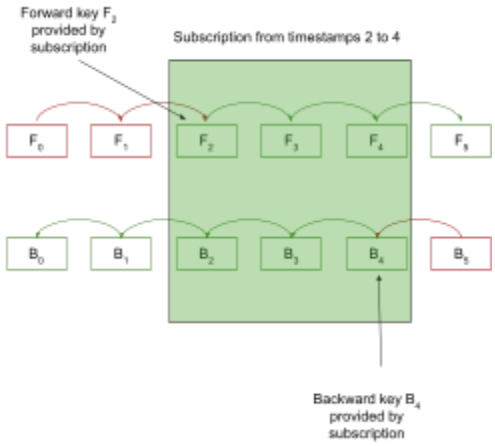
F_n and B_n are specially constructed sequences such that it is very simple to increment the bits in a left-to-right (big-endian) order. For example, in F_n, for any positive integers n (between 0 and 2^{64}-1) and m (between 0 and 63), if n & 2^m is not one:

$$Blake3(n, F_n) = F_{n+2^m} \,\&\, (2^{128} - 1)$$

Specifically, the hashing algorithm Blake3 is used to make a digest of both the index and the previous number in the sequence, making for an effective irreversible, high-entropy operation.

B_n works in a very similar way, except that it works backwards; one turns *off* the bits in a left-to-right direction.

This effectively means that one can obtain a large number of forward or backward keys just from a single one, as long as the hashing operations are performed in order.



The frame data sent to the Decoder is preceded by a byte indicating the channel ID, and it is also followed by a digital signature generated by the Ed25519 algorithm on the original frame.

Subscription Generation

A subscription includes intermediate values of the forwards and backwards sequences so that the decoder can use at least one of them to get the correct keys in the right order. These intermediates are also matched with their timestamps to aid in this process. The subscription also contains the channel ID and the start and end timestamps.

To prevent the subscription from being directly pirated, each of the intermediate values are encrypted using a symmetric key built into the decoder. This uses AES-128 in OFB (output feedback) mode. The keys for this operation are taken from the first output of PyCryptodrome's random number generation, seeded with a number constructed from the decoder ID, the channel number, and the system secret itself. With the system secret being directly built into the random number generation, it makes it infeasible for an attacker to determine how to generate the symmetric keys for another pirated subscription.

Decoder Building

The decoder is built with certain pieces of data that are essential to its operation and the security of the satellite system under this particular scenario (given that attackers will only have access to one properly provisioned decoder in the attack package).

Each decoder is built with:

- An emergency subscription, which works over every single timestamp.
- A set of AES keys for decrypting each channel's intermediate forward and backward keys.
- The channel IDs associated with each AES key (up to 16 non-emergency channels).
- The satellite system's Ed25519 public key for conducting frame verification.

Subscription Update Format

Data	Description	Length
Channel ID		32 bits
Start Timestamp	Updated at the same time as F_s.	64 bits
End Timestamp	Updated at the same time as B_e.	64 bits
Length of Floc	The number of forward keys being transmitted.	8 bits
Length of Bloc	The number of backward keys being transmitted.	8 bits
Floc	Each transmitted forward key's location within the sequence.	64 bits each
Bloc	Each transmitted backward key's location within the sequence.	64 bits each
Encrypted forward key	Each transmitted intermediate forward key needed for the subscription's particular start and end.	128 bits each
Encrypted backward key	Each transmitted intermediate backward key needed for the subscription's particular start and end.	128 bits each

Decoding

The Decoder, as mentioned above, only has access to B_e and F_s. However, they can use the recurrence relations above to determine (with t being the timestamp of a frame) B_t and F_t, at which point the Decoder can XOR them together along with the frame. They additionally use the Ed25519 public key to verify the authenticity of the frame.

Security Techniques

RCE Management

RCE (remote code execution) attacks will be hampered by leaving the majority of the SRAM inaccessible to execution and the program data unwritable.

On-Device TRNG Usage

Many randomly-generated values (except for the positions of checksums on the device) are produced by the Encoder's RNG system and are produced at the time of secret generation. Derivatives of these values are transferred over during subscription creation and subsequent updates. However, when random values are needed on the device to detect faults, sources of entropy, such as modular arithmetic transformations of clock values and data reception times, will be used to scramble TRNG outputs into random values.

Bootloader Verification

Immediately upon the initiation of the program, a quick checksum is created from the sum modulo 2^{32} of the products of each byte and its position. A failing check will result in an error message and a refusal to decode instructions.

Timing-Based Fault Detection

If a frame takes more than the required 150 milliseconds to decode, a decoding error is returned, exiting the host tool and stopping certain hardware-based attacks.

Impossible-Code Based Fault Detection

Particularly in the sections of the Decoder software that check the checksum of the returned frame, the software will include impossible code branches (which are kept through `#[allow(unreachable_code)]`. Any of these being activated would indicate a fault injection, which would immediately cause the device to return an error and exit the host tools.

Double-Checking Fault Detection

Critical calculation sections will be hardened by a function located elsewhere in the RAM. This function pauses a randomly-generated amount of time and then returns its initial argument, possibly modified. If hardware-based fault injection occurs, the system will still be able to test if the passage of time and the conservation of critical values were maintained properly, and the attack surface will be much sparser. In addition, checksums will be located at randomized addresses for an extra layer of security.

Objective Requirements

Functional Requirement Implementations

- Decoder Requirements:
  - It is trivial to simply loop through the subscriptions, remove the default channel 0 subscription, and return the list of IDs.
  - It is similarly easy to add a subscription directly to memory.
  - Decoding will be implemented by first extracting the subscription, validating the timestamp, using modular exponentiation to derive the frame-specific forwards and backwards keys, and then performing the XOR-based decryption of the frame.
- Encoder Requirements:
  - By performing the same modular exponentiation as the Decoder, the Encoder is capable of making a new frame.
  - The Encoder uses ECC-generated secrets to provide a consistent encryption setup without needing to reveal the keys to the Satellite System.

Security Requirement Implementations

- Security Requirement #1: An attacker should not be able to decode TV frames without a Decoder that has a valid, active subscription to that channel.
  - The attacker would need the forward and backward keys provided by the subscription to the channel to decode the frame. Secondly, the attacker would need to be aware of ID, as the initial keys issued in the subscription update are protected by the XOR cipher of H_{ID}.
  - Additionally, the forward and backward keys, when combined, permit the Decoder to properly decode only an interval of timestamps, since both

keys are required to decode a frame. Without the ability to produce a backward key of a later timestamp or a forward key of an earlier timestamp from the current keys,

- Security Requirement #2: A Decoder should only decode frames from the Satellite System it was made for.
    - Given that the encoder and the subscription update function are both using the same forward and backward keys, it would be incredibly unlikely for any other system to create a frame that happened to be interpretable from both of them. A checksum from another Satellite system or spoofer would almost certainly fail.

- Security Requirement #3: The Decoder should only decode frames with strictly monotonically increasing timestamps.
    - The Decoder specifically records the most recent frame that has been decoded to produce an error message when timestamps stay the same or decrease.