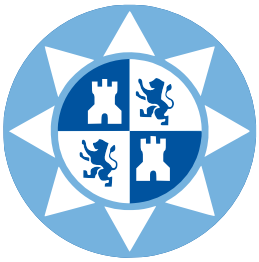

Universidad Politécnica de Cartagena

Universidad Politécnica de Cartagena

Key Management and Encryption System in eCTF

Authors
TrustLab Team

February 25, 2025

Contents

Chapter 1

Final proposal

This section describes our encryption and authentication scheme, based on *AES-CTR, AES-CMAC and Threshold Signature Scheme (TSS)*, designed to secure transmissions in a one-way environment without the need for complex responses or a heavy synchronization mechanism.

1.1 Main Elements

1.1.1 Algorithms and Keys Used

- **AES-CTR (Counter Mode)**: It is used to encrypt the data frames sent from the Encoder to the Decoder.
- **AES-CMAC (Cipher-based Message Authentication Code)**: It is used to derive a primary key from the master key and sign the subscription codes.
- **K_master**: Master key immutable during the whole process. Used for generating primary keys with AES-CMAC and ciphering subscription codes.

1.1.2 Identifiers

- **TS (Timestamp)**: Timestamp to verify the validity of the frames. We will use T_start and T_end to specify a range of time.
- **#SEQ (Sequence Number)**: Avoids replay attacks and ensures the order of packets.
- **$\mathbf{CH}_{ID}$ (Channel ID)**: Identifier of the TV Channel.
- **ENCODER_ID**: Unique identifier of the encoder.
- **DECODER_ID**: Unique identifier of the decoder.
- **FRAME**: Content of the TV frame to be transmitted.

1.2 Process in the Encoder

The Encoder is responsible for generating keys, encrypting subscriptions, and preparing protected frames for transmission. This process is divided into three main steps:

1. **Key Generation (genSecrets)**: The Encoder executes the function `genSecrets()` with all available channels to derive primary keys:
   - We pass the list of channels to encode `[1,2,3]` as input.
   - Using **AES–CMAC**, the funcion will generate an specific key for each channel, being $K_{CH_{ID}}$.
   - Also, a key for authentication purposes (`K_mac`) is generated.
   - The output will be a list of keys `[`K_1`, `K_2`, `K_3`, K_mac]`.

   This output will be exported into our `global_secrets.json`, which will be accessed in both the Encoder and Decoder.

2. **Subscription Code Generation (GenSubscription)**:

   To manage secure subscriptions, a subscription code `C_SUBS` is generated with the following structure:
   - We pass as input the necessary information: `(T_end, T_start, [CH_ID], DECODER_ID, ENCODER_ID)`.
   - The encoder will access to `global_secrets.json` to get the previously generated keys.
   - A **HMAC** code is generated for each channel and decoder, using $K_{CH_{ID}}$ from `global_secrets.json` and the rest of the input data.
   - The whole subscription is exported not only with the HMAC code, but also with unencrypted information such as the start and end date, the channel ID, and the decoder ID. This will help the decoder to validate the subscription.

   The final structure of a valid subscription is:

$$\{CH_{ID} \mid \text{DECODER\_ID} \mid TS\_\text{start} \mid TS\_\text{end} \mid HMAC\}$$

3. **Frame Encryption**:

   Each transmitted frame will consist of a validation code, the encrypted frame and unencrypted information such as the channel ID and timestamp. The encrypted frame will be generated using first the **AES-CTR** algorithm with the key $K_{CH_{ID}}$ and a nonce made from the CH_{ID} and the timestamp. This and then this will be **XORed** with the TV frame and TS to generate the final encrypted frame.

$$\text{AES–CTR}(K_{CH_{ID}}, TS \parallel CH_{ID}) \oplus (\text{FRAME} \parallel TS) \rightarrow \text{ENCRYPTED\_FRAME}$$

   An authentication code **HMAC** will also be generated only from **K_MAC**, the channel ID, and the timestamp to make the decoder be able to quickly validate a frame.

   Finally, the packet sent to the satellite has the following structure:

$$\{CH_{ID} \mid TS \mid \text{ENCRYPTED\_FRAME} \mid HMAC\}$$

1.3 Process in the Decoder

The Decoder is responsible for validating a subscription, receiving the encrypted frames, and decrypting the data.

1. **Subscription Validation**:

   The Decoder is built with a subscription, as we saw the structure of a valid subscription is:

   $$\{CH_{ID} \mid \text{DECODER\_ID} \mid TS\_\text{start} \mid TS\_\text{end} \mid HMAC\}$$

   The Decoder will access to `global_secrets.json`, and evaluate all subscriptions using the proper key $K_{CH_{ID}}$ of the channel and all the other public information to validate the HMAC code. If the validation is successful, the decoder will be subscribed to the channel.

   The same process will be done for updating a subscription.

2. **Frame Decryption**:

   When a frame is received, the decoder will start the decryption process:

   - First of all, the decoder checks the subscription list. As the frame has the channel ID visible, a frame can be rejected if the decoder is not subscribed to that channel.
   - If the channel is valid, the decoder validates the HMAC code using the key K_{MAC} from `global_secrets.json`.
   - Depending on the validation of the HMAC, the decoder will encrypt an **AES–CTR** code with the same data as the encoder used:

     $$\text{AES–CTR}(K_{CH_{ID}}, TS \parallel CH_{ID})$$

   - Then the frame is decrypted using the XOR operation with the encrypted frame and the AES–CTR code.

     $$\text{AES–CTR} \oplus \text{ENCRYPTED\_FRAME} = TV\_FRAME \parallel TS'$$

   - Finally, the decoder will validate if the timestamp decrypted is the same as the original one and if it has a more recent value than the previous frame received.
   - If the validation is successful, the decoder will be able to visualize the TV frame.

1.4 Transmission flow

The transmission flow is summarized as:

1. **Encoder → Uplink**:

   - Before any transmisison, we suppose the encoder already generated subscription codes and the decoder is set up.
   - The encoder will listen to raw frames.
   - The encoder encrypt the frames and transmit to the Uplink.
   - The encoder can also update subscriptions

2. **Uplink → Satellite**:
    - The Uplink retransmits the encrypted packet to the Satellite without any modifications.
3. **Satellite → TV**:
    - The Satellite broadcasts the packet to all available TVs.
4. **TV → Decoder**:
    - The TV receives the encrypted frame and hands it over to the Decoder.
5. **Decoder → TV**:
    - The decoder validates the frame.
    - If the frame is valid, the decoder decrypts the frame and transmit it on the TV.
    - The frame is shown on the TV.

This flow ensures the authenticity, integrity and confidentiality of the transmission, guaranteeing that only authorized decoders can access the content.

Chapter 2

Requirements requested

2.1 How does the design meet the requirements?

1. **Performance Requirements (Throughput Requirements)**: The use of AES-CTR (Counter Mode) to encrypt TV frames aligns with the high performance requirements described in the eCTF documentation. AES-CTR allows for fast and efficient encryption and decryption because:
    - It supports parallel operations (taking advantage of hardware or processor-specific encryption instructions).
    - It does not require padding or block reorganization, reducing overhead.
    - It is suitable for continuous data streams and large volumes of information (e.g., video streaming).
2. **Security Requirements**:
    1. **Confidentiality**:
        - Ensured by AES-CTR and the encryption of the subscription code with AES-CMAC.
        - Each decoder only receives the partial keys that correspond to it, making it difficult for unauthorized receivers to access content.
    2. **Integrity**:
        - The subscription code is encrypted and signed with AES-CMAC, ensuring that it cannot be altered without detection.
        - Timestamp (TS) and #SEQ are added to prevent replay attacks.
    3. **Authenticity**:
        - The identity of the Decoder and Encoder is verified through the fields `DECODER_ID`, `ENCODER_ID`, and subscription validation.
    4. **Resistance to Replay Attacks**:
        - The sequence number (#SEQ) and the timestamp (TS) are used to reject duplicate or delayed frames.
    5. **Use in Embedded Environments**:

- The design considers cryptographic operations with standard algorithms (AES, CMAC), which often have hardware acceleration or optimized implementations, facilitating integration into devices with limited resources.

In summary, these points meet the various eCTF requirements (confidentiality, integrity, authenticity, and protection against replay), including the performance requirement to handle the data volume characteristic of TV transmission.

2.2 How does a human need to manage the project?

In practice, the human factor is essential to coordinate and maintain the system throughout its life cycle. Among the main tasks:

1. **Master Key Generation and Custody (K_master)**:
   - An operator or system administrator must ensure that the master key is created and securely stored.
   - Security policies and backup procedures for K_master.
2. **Subscription Management**:
   - Distribute, in a timely manner, subscription codes (C_SUBS) to subscribers.
   - Revoke permissions or modify subscriptions when necessary.
3. **Key Rotation Plan**:
   - Schedule key rotation (e.g., renew master key or update partial keys) periodically or when any security incident is detected.
   - Execution of the genSecrets(K_master) function in a controlled and safe manner.
4. **Supervision and Audit**:
   - Monitor activity logs and verify that there are no anomalies (failed retries, decryption of unauthorized frames, etc.).
   - Apply Encoder and Decoder firmware/OS patches or upgrades as needed.
5. **Physical/Secure Device Distribution**:
   - Deliver the decoders (hardware) to the end users with their corresponding partial key preloaded or managed under a secure channel.

In short, a human (or security/operations team) must create the keys, manage subscriptions, rotate and revoke those keys according to policy, and ensure the physical and logical security of the system.

2.3 How do your choices fit into the design and target platform?

1. **Cryptographic Suitability**:
   a. **AES-CTR** is chosen for its efficiency and speed on embedded devices, taking advantage of:
      - Low memory usage compared to more complex encryption modes.
      - Parallelization capabilities that reduce latency.
   b. **AES-CMAC** for key derivation and subscription signing:
      - CMAC is specifically designed for integrity and key derivation and is simpler to implement than HMAC with SHA in some devices.
2. **Embedded Platform Considerations**:
   a. **Memory Usage and Persistence**:
      - Only the master key `K_master` needs to be securely stored (it can be in a secure module or protected firmware).
   b. **Hardware or Software Implementation**:
      - AES-CTR and AES-CMAC are standards with support in most cryptographic libraries.
      - Many embedded platforms have AES acceleration instructions (e.g., ARMv8 Cryptography Extensions), reducing CPU load and energy consumption.
   c. **Boot Times and Performance**:
      - By performing key derivation and validation outside the encryption loop for each frame (the derivation occurs before using the key), the encryption of each packet is done with simple block operations (CTR + CMAC), aligned with the eCTF's bandwidth and latency requirements.

2.4 How does your design protect against specific attacks?

1. **Pirated Subscription Attack**:
   - **Description:** An attacker attempts to steal or clone another user's subscription code (`C_SUBS`) to gain unauthorized access to the service.
   - **Design Countermeasures:**
     - **Identity Validation (`DECODER_ID` / `ENCODER_ID`):** The decoder verifies that the `DECODER_ID` in the encrypted code matches its own identifier. If not, the frame is rejected.
     - **Encryption and Authentication of `C_SUBS`:** The subscription code is encrypted and signed with `AES-CMAC(K_master)`. Any modification or attempt to reuse it in another decoder will be detected when decrypting or validating the signature.
2. **Pesky Neighbor Attack**:
   - **Description:** An "interfering neighbor" who receives the satellite signal and tries to decrypt channels without the proper subscription.

- **Design Countermeasures:**
  - **Frame Encryption with AES-CTR:** All frames are encrypted with the derived key (`K1` or the partial keys), preventing unauthorized devices from decrypting the content.
  - **Subscription Validation:** The decoder checks that the subscription (`CH_ID`, `DECODER_ID`) corresponds to the correct channel and decoder before decrypting the frame.

3. **No Subscription Attack**:
   - **Description:** A user without any subscription tries to decrypt frames or access a restricted channel.
   - **Design Countermeasures:**
     - **Subscription Check in Each Packet:** To decrypt the frame, the decoder needs the valid partial key associated with its subscription.
     - **Encryption of `C_SUBS`:** Without a verified `C_SUBS` (and decrypted with the master key), the decoder cannot obtain its partial key, thus denying access.
4. **Expired Subscription Attack**:
   - **Description:** An attempt to use a subscription code whose validity period (`T_fin`) has expired to continue receiving the service.
   - **Design Countermeasures:**
     - **Time Fields in `C_SUBS`:** Each subscription has `T_inicio` and `T_fin`. The decoder verifies that the current date is before `T_fin`.
     - **Automatic Invalidity:** If `T_fin` is in the past, the decryption of the frame is rejected. The user cannot renew their subscription without going through the authorized process again.
5. **Recording Playback Attack**:
   - **Description:** An attacker records encrypted video frames and replays them later to gain repeated access to content or attempts to "inject" old frames into the network.
   - **Design Countermeasures:**
     - **Sequence Number (#SEQ) and Timestamp (TS):**
       - The decoder only accepts frames with #SEQ higher than the last received.
       - Additionally, TS is checked to ensure it falls within a valid time window.
     - **Replay Protection:** Old packets (with #SEQ or TS delayed) are discarded, preventing the replay of previously recorded content.