J1939 secure communication

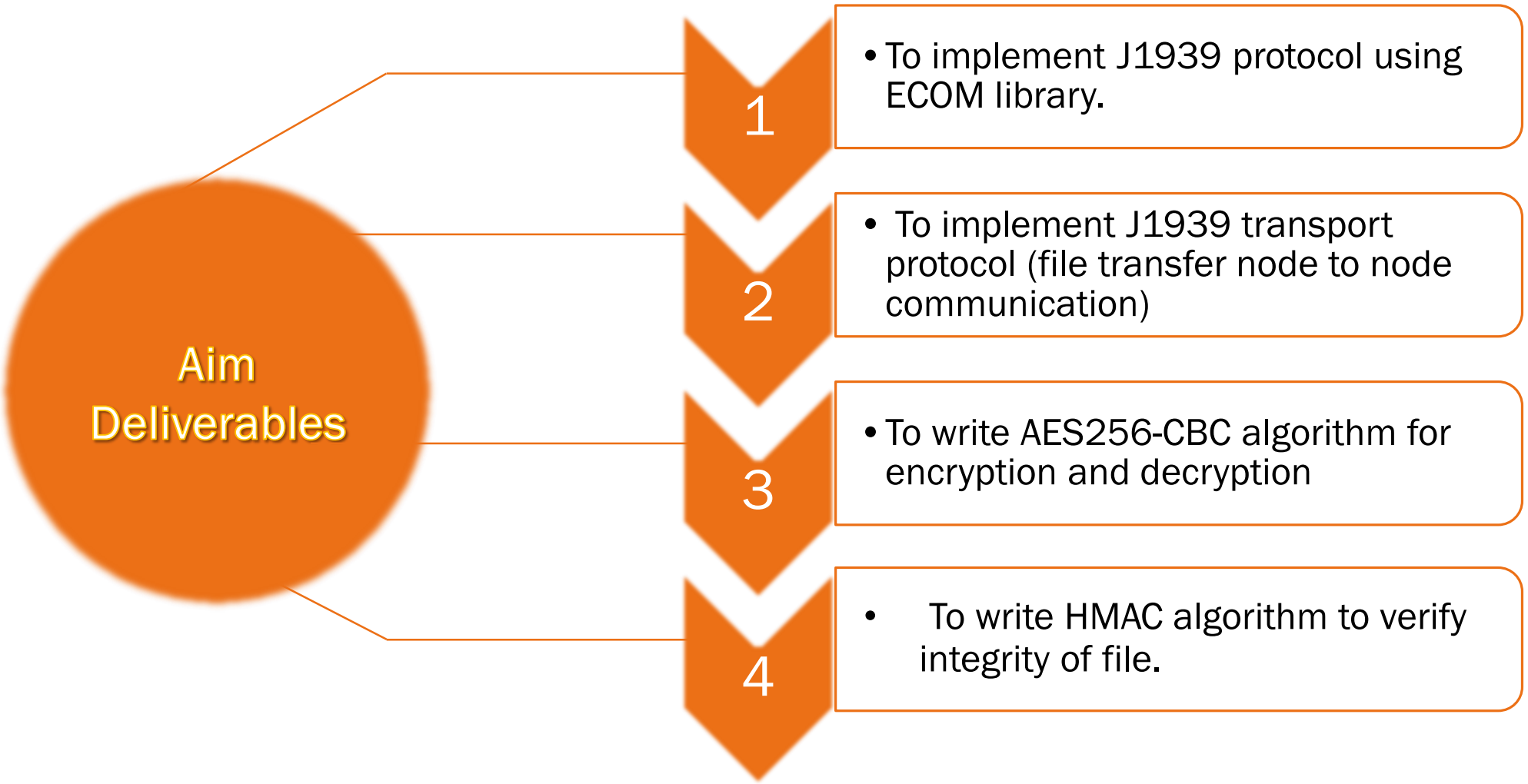
- SockSurge

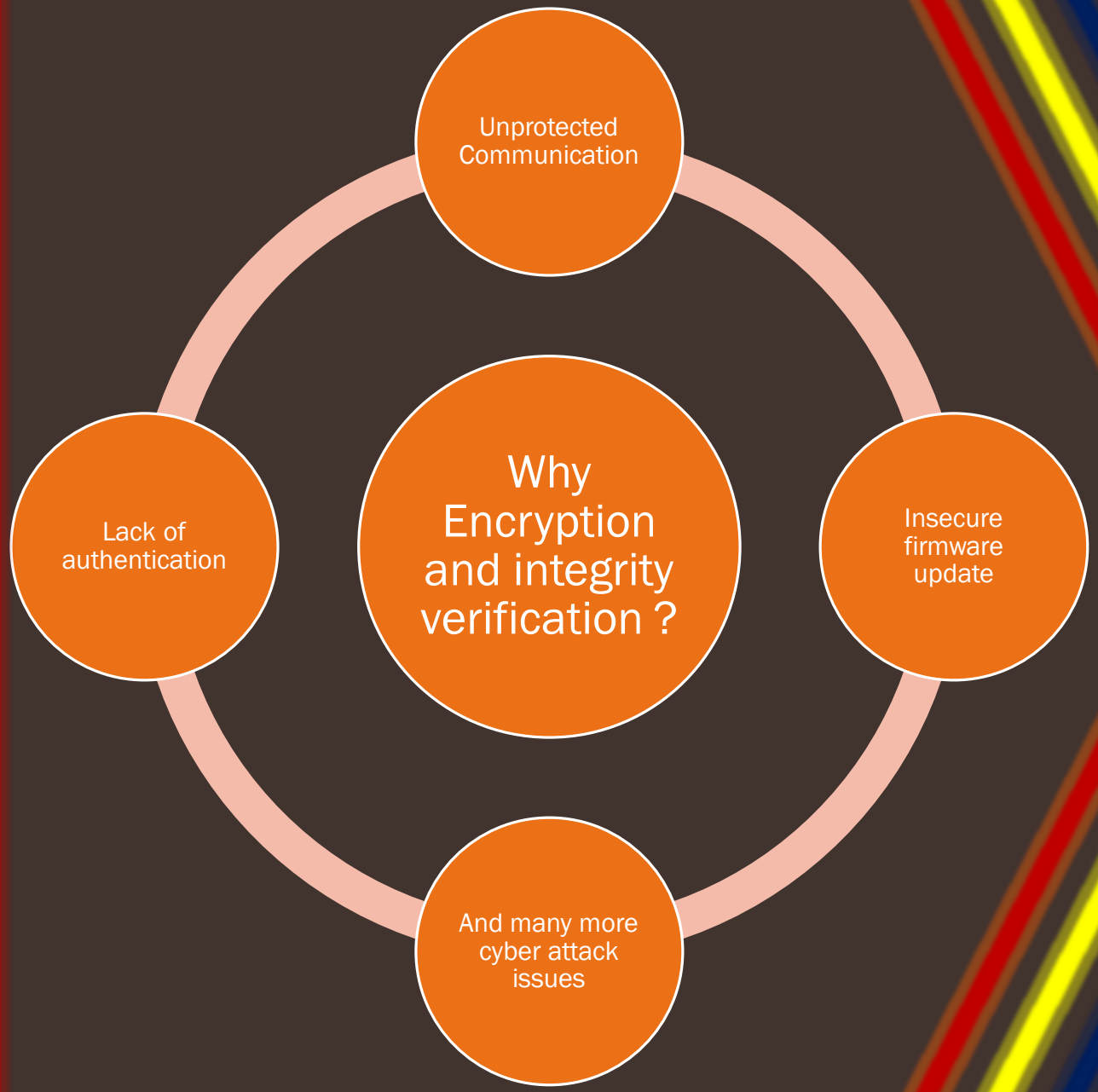
Background

Vapor Bus International Organization is a Transit Business operating from US, Buffalo Grove, Illinois 60089. Vapor Bus Organization is involved in design, manufacture and market door system for buses and special vehicles.

For advanced control over then CAN network and configuring each node separately for specific task, the requirement is to update the firmware(in this context, the user or task specific program uploaded to the microcontroller board.) via the J1939 protocol on the top of the CAN bus.

AIM : To implement secure firmware transfer via J1939 protocol on the top of CAN buses.


Aim
Deliverables
1
• To implement J1939 protocol using ECOM library.
2
• To implement J1939 transport protocol (file transfer node to node communication)
3
• To write AES256-CBC algorithm for encryption and decryption
4
• To write HMAC algorithm to verify integrity of file.



Encryption and Integrity verification communication implies secure and authenticated communication.

These implementations are necessary for avoiding two serious complications where the second one is more rare than the first one.

Cyber-attacks and Electromagnetic field issues.

Implementation of J1939 Application protocol via ECOM library.

This includes implementation of J1939 application layer protocol via the APIs provided by some standard ECOM libraries. This includes message formatting and unformatting according to the J1939 protocol.

Once the frame is ready, this can be directly sent to CAN bus,
Note that the frame structure is defined by these ECOM libraries.

The frame contains the actual payload and other required information.

Implementation of J1939 Transport protocol

This includes implementation of J1939 transport layer protocol which handles how the data or J1939 frame will be transported to other nodes via CAN bus.

This protocol in necessary for message transmission as well, one can Design this protocol as per needs like including “ACK” similar to TCP.

The entire frame J1939 frame or some sub frame is given to this depending on the board architecture which has to be transported.

Implementation of AES256-CBC encryption algorithm.

This implementation is necessary from security and accidential data corruption.
AES256-CBC is a symmetric key encryption algorithm with key length of 256 bits, the key has to be pre-shared among all the nodes.

The payload will be encrypted by this algorithm so that the data if tapped doesn’t make any sense to the actor.

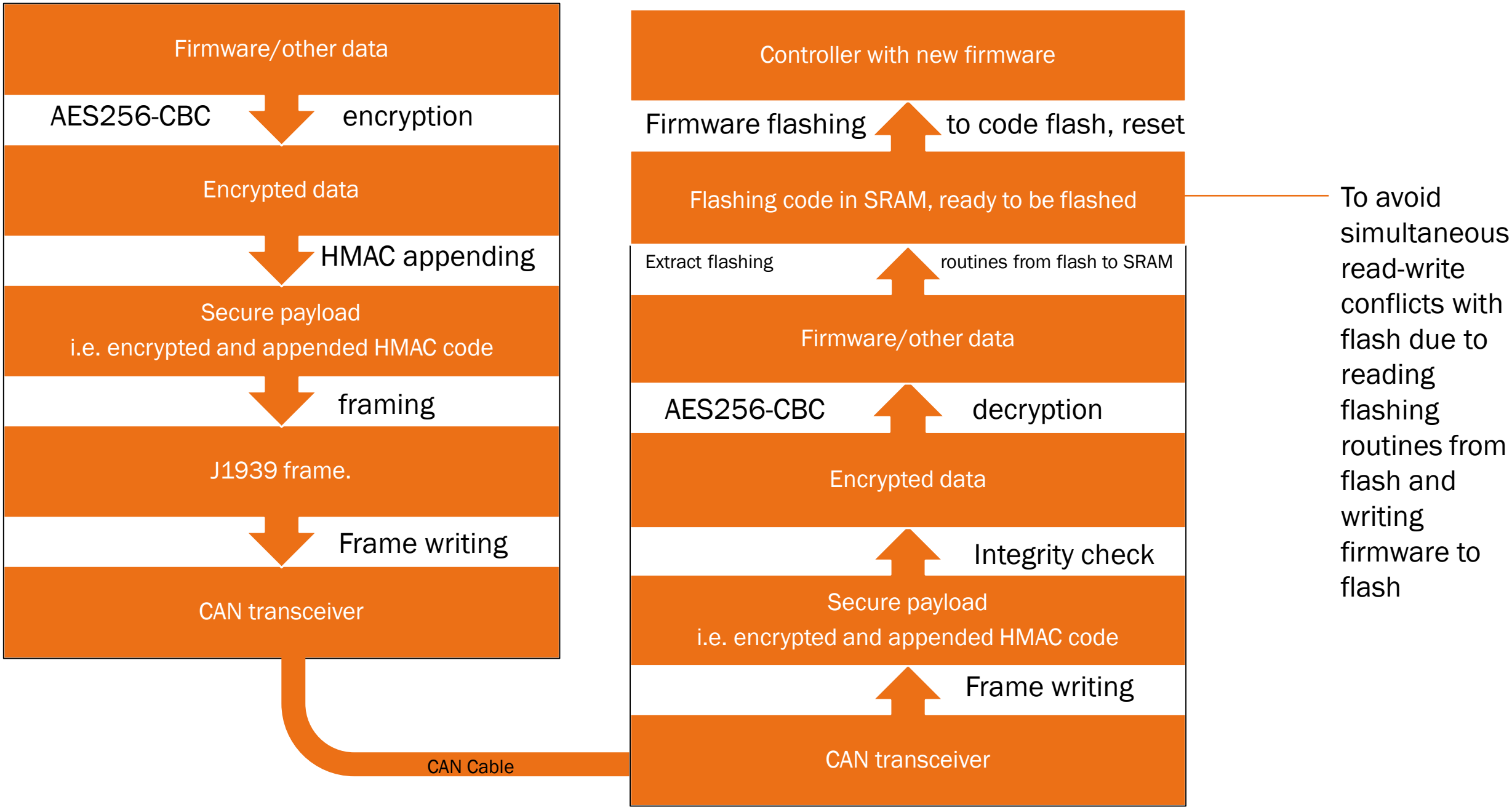
The payload will be sent after encryption which then will be processed and attached into the J1939 frame for transmission.

Implementation of HMAC integrity verification algorithm.

This implementation is necessary for data integrity.

The original message has to be appended with the HMAC code (a fixed length code derived from message itself via a hash function and key.) and then the payload is ready to be framed in J1939 frame.

The receiver must be running the same hash function in conjugation with the same HMAC key. The receiver will recalculate the HMAC code itself and compare with the HMAC code it received appended with the original message which may be further decrypted depending on security implementations.



To avoid simultaneous read-write conflicts with flash due to reading flashing routines from flash and writing firmware to flash

: States of GD based controller

: Computational processes.

Task description

AES256-CBC, HMAC based encryption methodologies should be used for the secure CAN communications to upload firmware files to the GD controller.

At Sender we should encrypt the file via AES256-CBC algorithm, generate 64 bytes HMAC code using HMAC algorithm, Append HMAC code at the end of the file. And then Send the file via J1939 transport protocol.

Receiver upon receiving should store file in data flash, retrieve HMAC code from the file, generate HMAC code of the received file and verify file integrity via HMAC algorithm,
decrypt file block by block using AES256-CBC algorithm and flash decrypted file to code flash

Self-Programming / In-Application Programming (IAP)

Self-Programming or IAP is the practice used by embedded microcontroller boards to self update their firmware code and bootloader code as well (if supported).
Generally, the sketch uploading or firmware update is done by connecting the controller to a programming board or computer, when the board is reset, it enter the bootloader mode and receives the firmware code into its SRAM via some specific communication protocols and then writes the firmware code to the flash. This flashing is happening in bootloader mode, bootloader mode has instructions or routines for flashing, once the flashing is done, control is transferred to new firmware after reset.
Note that the bootloader mode on some hardware has access to write operation to flash while on some other boards, both user program/firmware and bootloader have access to write operation to flash.
Simultaneous read and write to specific storage device must be avoided to avoid conflicts.

ARDUINO

- Firmware can be updated by bootloader only.
- Only bootloader has access to write operation to flash.
- Instructions can only be executed from flash.
- Controller reset in necessary for firmware update.

STM32 arm-cortex

- Firmware can be updated by both bootloader and firmware itself.
- Both bootloader and firmware have access to write operation to flash.
- Instructions can be executed from flash and SRAM.
- Controller reset in not necessary for firmware update.