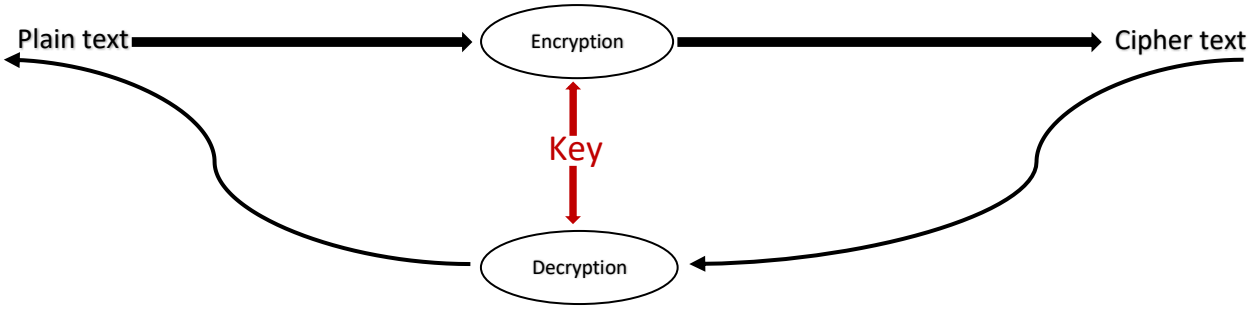
4.2. Encryption/Decryption Algorithm

4.2. Encryption Algorithm

Is a security technique used to protect data and communications by converting plaintext information into ciphertext using mathematical algorithms. Nobody can read or reuse of ciphertext until convert it into plaintext. The receiver can easily convert this into plaintext doing decryption process, when anyone other do attack for conversion and it isn't easy depend on the strength of algorithm. Is used to ensure data confidentiality, integrity, and privacy.



4.2.1. Introduction

At FOTA, data transmit over air, network, and always susceptible to theft and exploration of what is inside. Did you remember our application? We transmit code file, very sensitive data. It can be stolen or edited and retransmit. In case of edit may cause crisis. Imagine the scenario of code editing and retransmission at industrial environment, same thing and may kill people. Therefore, we go to create our own encryption code using AES Algorithm in C.

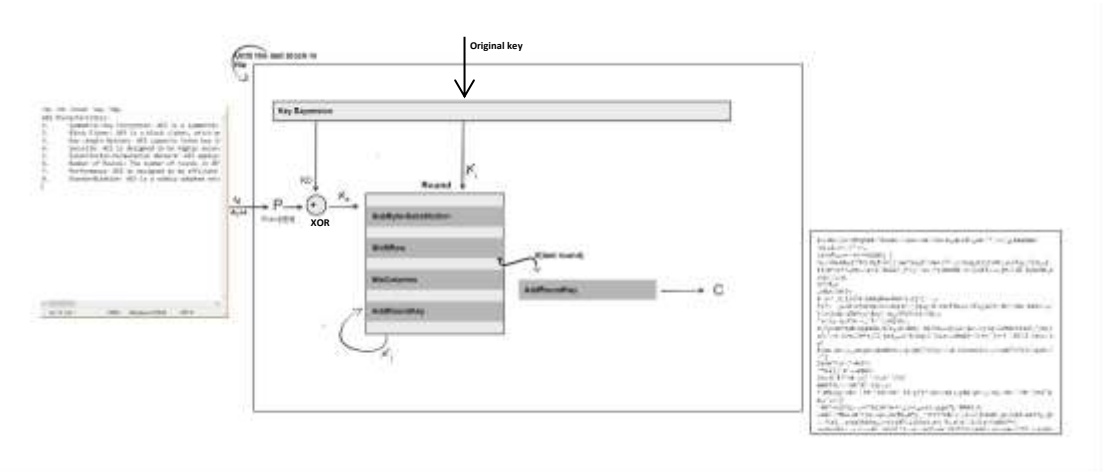
Algorithm Characteristics:

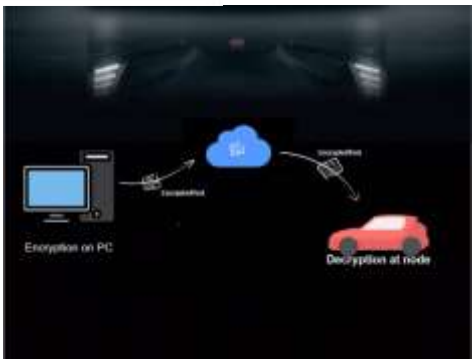
- Symmetric-Key Encryption: is a symmetric-key encryption algorithm, which means it uses the same secret key for both encryption and decryption of data. This simplicity allows for faster processing and is suitable for applications that require high-speed encryption and decryption.
- Block Cipher: is a block cipher, which means it processes data in fixed-size blocks. The block size for is 128 bits.
- Key Length Options/Fixable key length: supports three key lengths: 128 bits, 192 bits, and 256 bits. The key length directly affects the strength of the encryption. This key length contribute against brute force attack.
- Security: is designed to be highly secure and resistant to various cryptographic attacks, including brute-force attacks, differential cryptanalysis, and linear cryptanalysis.
- Substitution-Permutation Network: employs a Substitution-Permutation Network (SPN) structure, which involves multiple rounds of substitution and permutation operations on the data. These rounds contribute to the confusion and diffusion properties, increasing its resistance against attacks.
- Performance: is designed to be efficient and fast on a wide range of devices, including both software and hardware implementations.

You know, we have a problem:
We must design algorithm which suitable for our application, we have small memory and small storage for decryption on the node, we decrypt using STM32 Microcontroller.

4.2.2. Encryption process

Here, we describe the encryption process and decryption will be in the reverse direction.





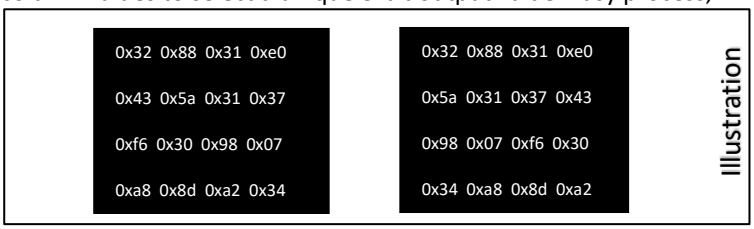
Here we have a box called a Round and
is repeated box based on the key length and because we have 128 bit original key length, the number of round is 10. Each round has its own key derived from Original Key. It's easy to develop code to receive 256 original key length. In the last round we skip MixColumns() and do only AddRoundKey() then generate block of output. Back again to file and generate other block of plain. You must know that we take only a block of plain and decrypt it separated than file, then out it to new file, then back and take another block.

The black draw tells you how encryption applied in our application, we encrypt at PC and decrypt at node, at Microcontroller.

- **Key Expansion**
  It takes the original encryption key and generates a set of round keys, which are used in subsequent rounds of the AES encryption algorithm.

1- SubByte (Byte substitution): an S-box substitution step.

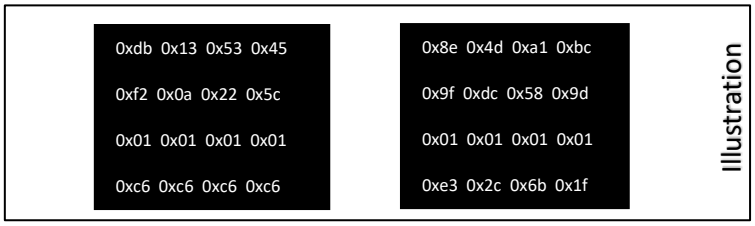
- Defines a 16 * 16 matrix of byte values, called an S-box that contains a permutation of all possible 256 input values. (Hint) As we know from ASCII we have 8 bit to represent the keyboard input, so we have 2 power 8 available values.
- Each individual byte of State, plain, is mapped into a new byte in the following way: The leftmost 4 bits of the byte are used as a row value and the rightmost 4 bits are used as a column values to select a unique 8-bit output value. Easy process, TRYIT.



2- ShiftRows: a Permutation step.
The bytes in each row of the state matrix are shifted to the left. The number of positions each byte is shifted depends on its row number in the matrix. First row (Row0) isn't shifted. Row1 shift by 1, Row2 shift by 2, and Row3 shift by 3. Easy process, TRYIT.

3- **MixColumns:** a Matrix multiplication step.

- It is designed to provide further diffusion and confusion, enhancing the overall security of the cipher. Each column of the state array is processed separately to produce a new column. The new column replaces the old one. The processing called Matrix Multiplication.
- In the MixColumns process, each column of the State matrix is treated as a polynomial, and a mathematical transformation called "polynomial multiplication" is performed on each column. Polynomial multiplication is done modulo a fixed irreducible polynomial in the Galois Field.
- This process helps to prevent patterns and correlations in the data from persisting through multiple rounds of encryption. It increases the resistance of the cipher against various cryptanalytic attacks.



4- **AddRoundKey:** an XOR step with a round key, K_i.
The 128 bits of State are bitwise XORed with the 128 bits of the round key.

4.2.3. Decryption process

In the previous section at encryption, the input is plaintext, clear text. At decryption the input is ciphertext and we go in reverse operation. The process aims to recover the original plaintext from the ciphertext using the same encryption key that was used during the encryption process. S-box be RS-box, ShiftRow be InverseShiftRow, and MixColumns be InverseMixColumns.

1- **KeyExpansion:** The decryption process starts by expanding the original encryption key into a set of round keys. These round keys are used in reverse order during decryption, beginning with the last round key used during encryption.
2- **AddRoundKey(Last Round):** The first decryption step is to apply the AddRoundKey operation using the last round key used during encryption. This step is identical to the AddRoundKey step in encryption.
3- **Inverse ShiftRows:** The Inverse ShiftRows step is applied to the State matrix (cipher), reversing the shifting of rows performed during the encryption process. This step moves the bytes in each row to the right instead of left.
4- **Inverse SubBytes:** The Inverse SubBytes step involves replacing each byte in the State matrix with its original value from the Substitution Box (S-box). This step reverses the substitution applied during encryption.
5- **Rounds:** In each round, the following steps are performed:
    - **Inverse MixColumns:** The Inverse MixColumns step is applied to the State matrix, undoing the mixing done during encryption. It involves polynomial multiplication in the Galois Field.
    - **Inverse ShiftRows:** The Inverse ShiftRows step is applied again to undo the shifting of rows performed in the corresponding encryption round.
    - **Inverse SubBytes:** The Inverse SubBytes step is performed once more to revert the substitution of bytes from the Substitution Box.
    - **AddRoundKey:** In each round, the State matrix is XORed with the corresponding round key in reverse order to undo the key mixing done during encryption.

6- **Final Round (Inverse AddRoundKey):** The last decryption round involves applying the AddRoundKey operation using the first round key used during encryption. This undoes the initial AddRoundKey operation in the first encryption round.

4.2.3. Code flow chart

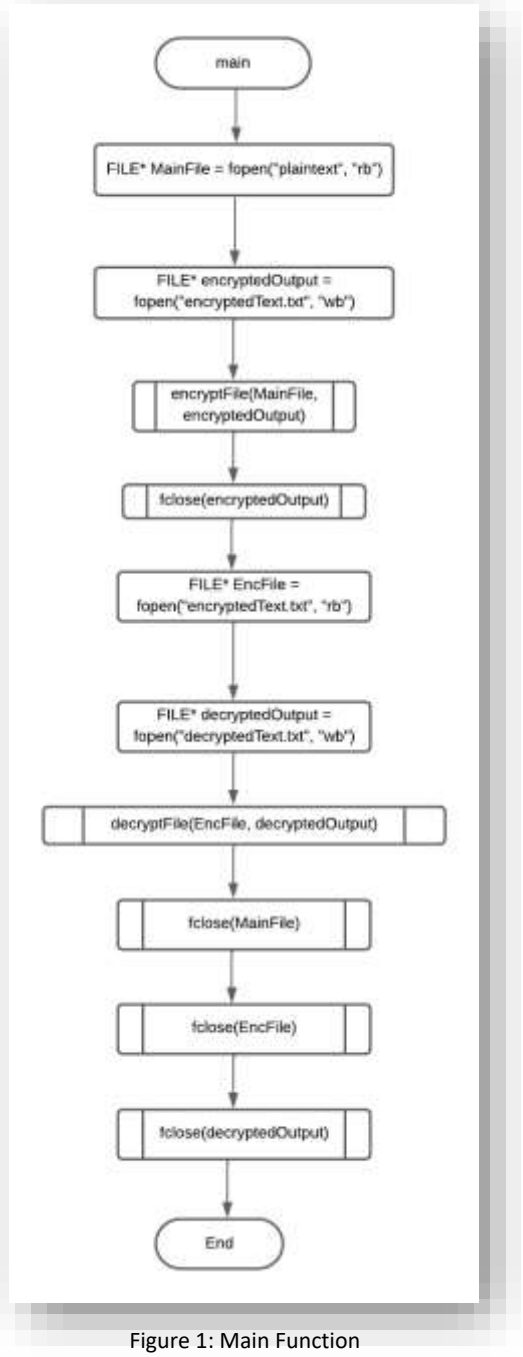


Figure 1: Main Function

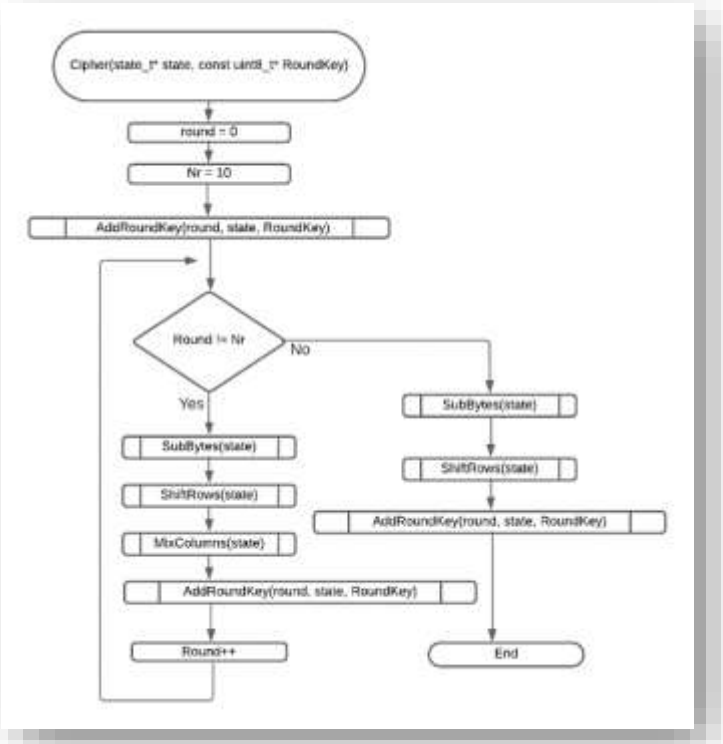


Figure 2: Cipher Function

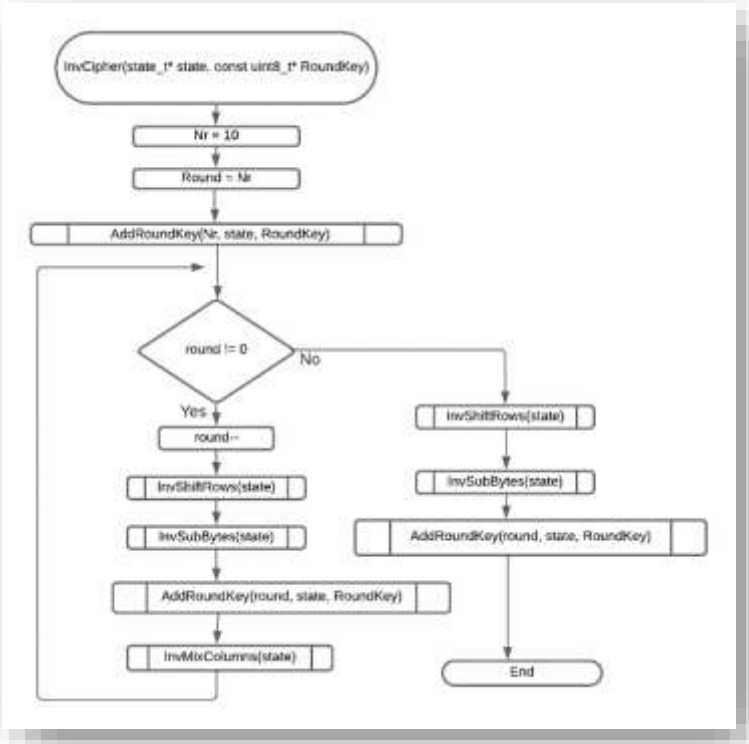


Figure 3: Inverse Cipher Function

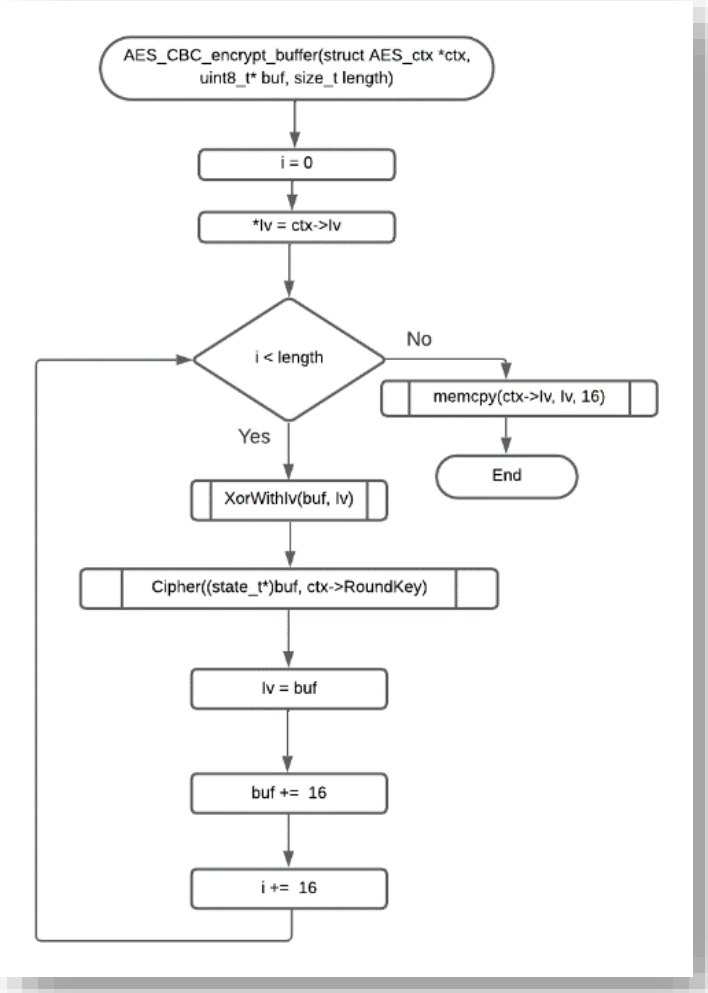


Figure 4: AES_CBC_encrypt_buffer Function

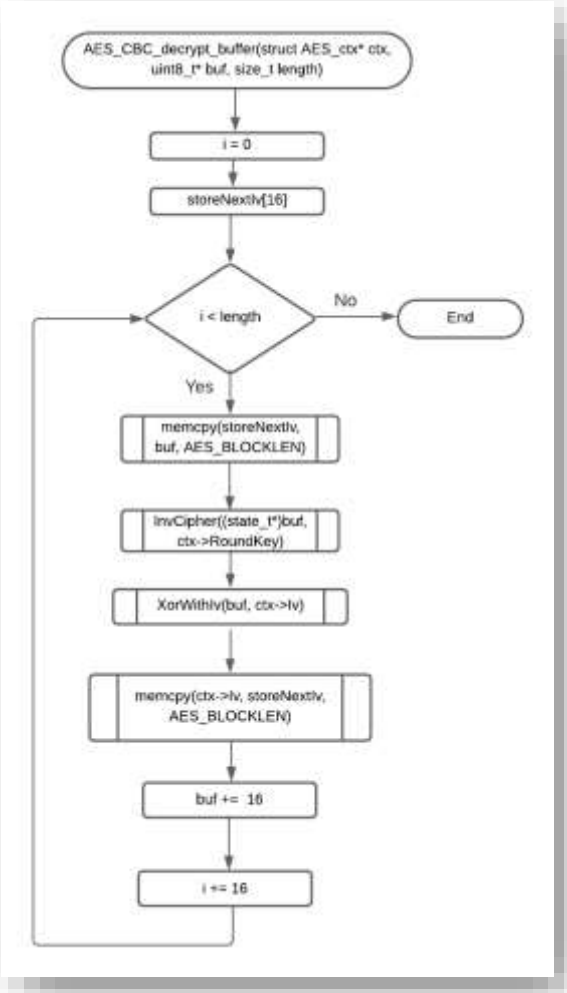


Figure 5: AES_CBC_decrypted_buffer Function

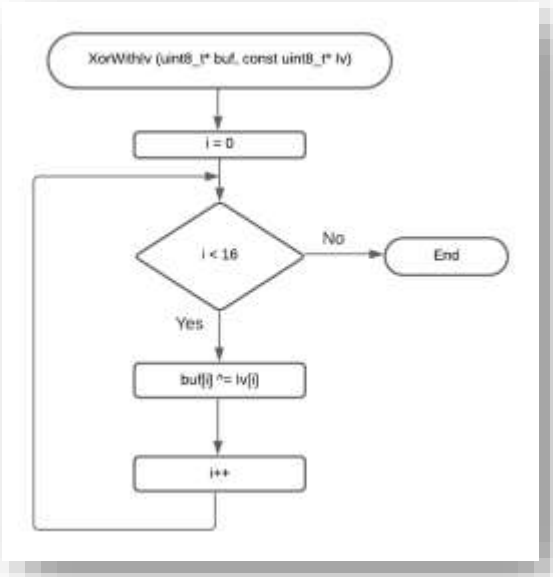


Figure 6: XorWithIv Function

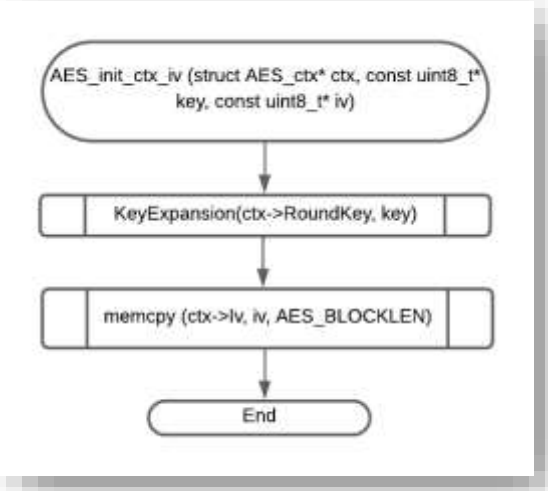


Figure 7: AES_init_ctx_iv Function

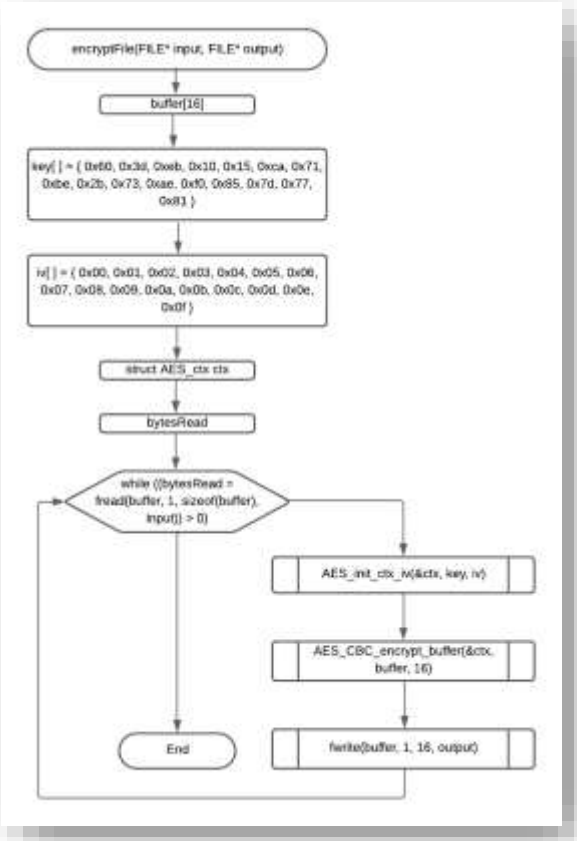


Figure 8: encryptFile Function

Figure 8: decryptFile Function