Implementation of AES-128 Encryption and Decryption

1 Key Expansion

AES-128 encryption requires the generation of 10 round keys from an initial 128-bit secret key. This process is referred to as *Key Expansion*, which is fundamental to ensuring security in AES.

1.1 Included Libraries

The program utilizes two key libraries:

- `stdio.h`: Used for input-output functions.
- `stdint.h`: Provides fixed-width integer types, such as `uint8_t`, which defines an 8-bit unsigned integer.

1.2 User-defined Datatype: word

The code defines a custom datatype, `word`, as a structure consisting of an array of four 8-bit unsigned integers.

1.3 AES S-box (Substitution Box)

In this code, the Substitution Box (S-box) is a matrix of order 16 of hexadecimal values. It can be taken as a one-dimensional array of 256 elements.

1.4 Round Constant Array: Rcon

The AES key expansion uses a set of round constants, denoted as `Rcon`. These constants ensure that each round of encryption has a unique key, introducing non-linearity into the key expansion process.

1.5 Function: rotword

The `rotword` function performs a cyclic left shift on the bytes of a word:

- The first byte of the word is moved to the last position, and the other bytes are shifted left.

1.6 Function: sbox_substitute

This function substitutes each byte of the input word with a corresponding byte from the S-box:

- The function splits each byte into two 4-bit halves, using the higher 4 bits to index the row and the lower 4 bits to index the column of the S-box.
- The byte is replaced by the S-box value located at that position.

1.7 Function: RCons

The `RCons` function modifies the first byte of a word by XORing it with a round constant:

- The `Rcon` array provides a constant for each round, which is XORed with the first byte of the word.

1.8 Function: key_expand

Within this function, a matrix `w` of order 4x11 is created, where `w` consists of 11 round keys corresponding to each column. Each column contains 4 words (32x4 bits = 128 bits) for that round. The original key is stored in the first column, and the remaining 10 columns are used for the 10 round operations.

The process follows these steps:

1. The first four words (128 bits) are initialized with the original key.
2. For each subsequent round, the last word of the previous round undergoes:
   - **Rotation** with the `rotword` function.
   - **Substitution** using the `sbox_substitute` function.
   - The round constant is applied using the `RCons` function.
3. The transformed word is XORed with the first word of the previous round to create the first word of the new round.
4. This process continues for the remaining words in the round, ensuring that each word is XORed with the previous one.

2 State Transformation

Following the key expansion, three transformations are applied to the state matrix during the encryption process. The operations are respectively subbytes, shiftrows and mixcolumns. These are performed in “cipher.h”.

2.1 Function: subbytes

Each byte in the 4x4 state matrix is substituted using the AES S-box(as we have done in section 1.6).

2.2 Function: single_byte_rot

The `single_byte_rot` function performs a cyclic left shift on a 4-byte row:

- The first byte is moved to the last position, and the other bytes are shifted left.
- This operation is used in the `shiftrows` function.

2.3 Function: shiftrows

The *shiftrows* operation cyclically shifts the rows of the state matrix:

- The first row remains unchanged.
- The second row is rotated left by 1 byte. It has been done by calling "single_byte_rot" function only once.
- The third row is rotated left by 2 bytes. It has been done by calling "single_byte_rot" function twice.
- The fourth row is rotated left by 3 bytes. It has been done by calling "single_byte_rot" function thrice.

2.4 Functions: mult_by_2 and mult_by_3

These functions implement multiplication in the Galois Field GF(2^8) using the irreducible polynomial $x^8 + x^4 + x^3 + x + 1$:

- **Multiplication by 2**: The byte is shifted left by 1 bit. Since we are working in GF(2^8), the result should be confined to 8 bits, so we mask out the lower 8 bits of the shifted result. If a carry bit is generated, the masked out result is XORed with the AES irreducible polynomial (0x1b).
- **Multiplication by 3**: The byte is multiplied by 2(i.e., shifted left by 1 bit) and XORed with the original byte. The carry bit is handeled as it was done in Multiplication by 2.

2.5 Function: mixcolumns

The `mixcolumns` function performs column-by-column transformation of the state matrix using fixed polynomials in GF(2^8).

- Each column of the state matrix is multiplied by a fixed matrix.

3 AES-128 Decryption

The AES-128 decryption process is designed to reverse the encryption transformations, retrieving the original plaintext from the ciphertext. It involves several key operations, each of which is detailed below (related code is provided in “decipher.h”).

3.1 Inverse S-box

The Inverse S-box is a predefined lookup table used during decryption to reverse the byte substitution applied during encryption.

3.2 Inverse MixColumns Matrix

The “mixmat” matrix is used during the Inverse MixColumns transformation. This matrix is applied to each column of the state matrix to reverse the column mixing done during encryption.

3.3 Multiplication in GF(2^8)

Arithmetic operations in AES are performed in the Galois Field GF(2^8). This field is used for operations involving bytes, where each byte is treated as a polynomial of degree 7. Multiplication in this field involves bitwise operations.

3.4 Reduction

After performing polynomial multiplication, the result is reduced modulo a specific irreducible polynomial $x^8 + x^4 + x^3 + x + 1 (i.e., 0x1b)$ such that the result fits into 8 bits.

3.5 Single Byte Rotate

The “single_byte_rotate” operation is used to cyclically shift(left) the bytes of a row within the state matrix.

3.6 Inverse ShiftRows

The “inv_shiftrows” step involves shifting rows of the state matrix to the right. Each row of the state matrix is shifted by a specific number of positions:

- Row 0: No shift.
- Row 1: Shift right by 1 byte(or left shift by 3 bytes).
- Row 2: Shift right by 2 bytes(or left shift by 2 bytes).
- Row 3: Shift right by 3 bytes(or left shift by 1 byte).

3.7 Inverse SubBytes

The “inv_subbytes” step involves substituting each byte in the state matrix with a corresponding value from the Inverse S-box (as it was done in the subbytes step).

3.8 Inverse MixColumns

The “inv_mixcolumns” step involves transforming each column of the state matrix using the “mixmat” matrix (initialilzed). This transformation reverses the column mixing that was applied during encryption.

4 AES-128 Encryption Function: `AES_ENC`

The `AES_ENC` function performs AES encryption on a 16-byte input array using a fixed 128-bit key. The encryption process involves several key steps:

4.1 Key Initialization

AES encryption begins with initializing a 128-bit key, which is used to generate round keys for the encryption process. In this case, the key is predefined as a constant 16-byte array.

4.2 State Matrix and Round Key Matrix

The state matrix is a 4x4 array used to hold the input data during the encryption process. The round key matrix is a 4x11 array used to store the round keys generated from the initial key. This matrix includes keys for all 10 rounds of encryption.

4.3 Key Expansion

The `key_expand` function is called to derive the round keys from the initial 128-bit key.

4.4 State Matrix Initialization

Each byte of the input message is placed into the state matrix in a column-major order.

4.5 Encryption Rounds

AES encryption involves 10 rounds of processing, where each round includes the following transformations:

- **Round Key Addition**: Each column of the state matrix is XORed with the corresponding column of the round key for that round.
- **SubBytes Transformation**
- **ShiftRows Transformation**
- **MixColumns Transformation**

These transformations are applied in sequence for 9 rounds.

4.6 Final Round

In the final round of encryption, the MixColumns transformation is omitted. The final round consists of:

- **Round Key Addition**
- **SubBytes Transformation**
- **ShiftRows Transformation**
- **Round Key Addition**

4.7 Output

After completing all rounds, the state matrix is converted back to the original input array format. This array now contains the encrypted data.

5 AES-128 Decryption Function: `AES_DEC`

The `AES_DEC` function performs AES decryption on a 16-byte cipher array using the same 128-bit key. The decryption process is essentially the reverse of encryption and includes the following steps:

5.1 Key Initialization

The same 128-bit key used for encryption is also used for decryption. This ensures that the decryption process correctly reverses the encryption.

5.2 State Matrix and Round Key Matrix

The state matrix and round key matrix are initialized similarly to the encryption process.

5.3 Key Expansion

The `key_expand` function is called again to generate the round keys needed for decryption. The round keys are the same as those used in the encryption process.

5.4 State Matrix Initialization

The ciphertext is converted into the state matrix format to prepare for the decryption transformations.

5.5 Initial Round Key Addition

The final round key is XORed with the state matrix to prepare the state for the decryption process.

5.6 Decryption Rounds

The decryption process involves 10 rounds of transformations, which are the inverses of the encryption operations:

- **Inverse SubBytes Transformation**: Each byte in the state matrix is substituted using the inverse of the AES S-box.
- **Inverse ShiftRows Transformation**: The rows of the state matrix are cyclically shifted right by varying offsets. This reverses the ShiftRows transformation applied during encryption.
- **Round Key Addition**: Each column of the state matrix is XORed with the corresponding column of the round key for that round.
- **Inverse MixColumns Transformation**: Each column of the state matrix is mixed using the inverse of the fixed matrix used during encryption. This step reverses the MixColumns transformation.

These transformations are applied in reverse order for 9 rounds.

5.7 Final Round

In the final round of decryption, the Inverse MixColumns transformation is omitted. The final round consists of:

- **Inverse ShiftRows Transformation**
- **Inverse SubBytes Transformation**
- **Round Key Addition**

5.8 Output

After completing all rounds, the state matrix is converted back to the original input array format. This array now contains the decrypted data.

6 AES Encryption/Decryption using ECB Mode of Operation

In this section, we discuss the process of reading data from a file, applying padding to the input data, performing AES encryption, and writing the encrypted data to an output file. We also cover the decryption process, which reverses these steps to retrieve the original data.

6.1 Libraries and Header Files

- `AES.h`: Contains functions for AES encryption and decryption, including functions like `AES_ENC` and `AES_DEC`.

6.2 Padding the Input Data

AES encryption requires that the input data be processed in 16-byte blocks. If the data does not exactly fit this block size, padding is added to the final block:

- Padding is added by filling the remaining space in the block with the number of bytes that need to be padded.
- For example, if the final block has 10 bytes, 6 padding bytes with the value `0x06` (indicating 6 bytes of padding) will be added to make the block size 16 bytes.

6.3 AES Encryption Process

The AES encryption process consists of several key steps:

1. **File Reading:** The input file is opened, and its contents are read in 16-byte chunks.
2. **Padding:** If a chunk contains fewer than 16 bytes, padding is applied to ensure the block meets the required size for AES encryption.
3. **AES Encryption:** The padded 16-byte block is encrypted using the AES-128 encryption algorithm.
4. **File Writing:** The encrypted blocks are written to a new file.

6.4 AES Decryption Process

The decryption process mirrors the encryption process, reversing the transformations to recover the original data:

1. **File Reading:** The encrypted file is opened, and its contents are read in 16-byte blocks.

2. **AES Decryption:** Each 16-byte block is decrypted using the AES-128 decryption algorithm.
3. **Padding Removal:** After decryption, padding is removed from the final block by checking the value of the last byte. This byte indicates how many padding bytes were added during encryption, and those bytes are discarded.
4. **File Writing:** The decrypted data is written to a new file, effectively restoring the original content.

7 AES File Encryption/Decryption Using CBC Mode of Operation

This document describes the functionality of a C program that encrypts and decrypts a file using AES with Cipher Block Chaining (CBC) mode. An initialization vector (IV) is used in encryption and decryption processes.

7.1 Encryption Process

The encryption process involves reading the input file block by block, encrypting each block using AES in CBC mode, and writing the ciphertext to an output file.

1. The IV is written as the first block to the output file, so it can be used during decryption.
2. Each 16-byte block of data is read from the input file.
3. The plaintext block is XORed with the previous ciphertext block (or the IV for the first block) as per CBC mode.
4. The resulting block is encrypted using AES (`AES_ENC(temp)`).
5. The ciphertext is written to the output file.
6. If the final block has less than 16 bytes, padding is applied to ensure it is exactly 16 bytes.

7.2 Decryption Process

The decryption process reads the encrypted file block by block, decrypts each block using AES, and writes the resulting plaintext to an output file.

1. The first 16 bytes of the encrypted file, which contain the IV, are read.
2. Each subsequent 16-byte block of ciphertext is read, decrypted using AES (`AES_DEC(buffer)`), and XORed with the previous ciphertext block (or the IV for the first block) to retrieve the plaintext.

3. The plaintext is written to the output file.
4. If the last block is detected (i.e., the size of the block is less than 16 bytes), the padding is removed from the decrypted data.

8 AES Encryption and Decryption in OFB Mode

This document provides an overview of a C program that encrypts and decrypts a file using the AES (Advanced Encryption Standard) algorithm in Output Feedback (OFB) mode. OFB is a stream cipher mode where the block cipher's output is used to generate keystream blocks that are XORed with plaintext blocks to produce ciphertext.

8.1 Encryption Process

In OFB mode, encryption proceeds as follows:

1. The IV is written to the output file, so it can be used during decryption.
2. For each block of data:
   - A keystream block is generated by encrypting the previous keystream block (initially the IV).
   - The plaintext block is XORed with the keystream block to produce the ciphertext.
   - The ciphertext block is written to the output file.
3. If the final block has fewer than 16 bytes, padding is applied to ensure the block size is exactly 16 bytes.

8.2 Decryption Process

The decryption process mirrors the encryption process in OFB mode, as encryption and decryption in this mode are symmetric.

1. The IV is read from the input file to initialize the keystream.
2. For each block of ciphertext:
   - The next keystream block is generated by encrypting the previous keystream block (or IV for the first block).
   - The ciphertext block is XORed with the keystream block to recover the plaintext.
   - The plaintext block is written to the output file.
3. If the last block is detected (i.e., its size is less than 16 bytes), the padding is removed from the plaintext before writing it to the output file.

9 AES Encryption and Decryption in CFB Mode

This document describes a C program that performs encryption and decryption using the AES algorithm in Cipher Feedback (CFB) mode. CFB mode allows the AES block cipher to be used as a stream cipher, where each block of plaintext is XORed with an encrypted version of the previous ciphertext block (or an Initialization Vector, IV, for the first block).

9.1 Encryption Process

In CFB mode, encryption is done as follows:

1. The IV is written to the output file so it can be used during decryption.
2. For each block of plaintext:
    - The IV (or the previous ciphertext block) is encrypted using AES to generate an intermediate value.
    - The plaintext block is XORed with this intermediate value to generate the ciphertext.
    - The ciphertext block is written to the output file.
    - The ciphertext is then used as the"IV" for the next block.
3. If the last block is smaller than 16 bytes, it is padded to ensure it is the correct size before encryption.

9.2 Decryption Process

Decryption in CFB mode mirrors the encryption process, with the same AES encryption function being used for both encryption and decryption. This is because CFB is a self-synchronizing stream cipher.

1. The IV is read from the encrypted file to initialize the decryption process.
2. For each block of ciphertext:
    - The previous ciphertext block (or IV for the first block) is encrypted using AES to generate an intermediate value.
    - The ciphertext block is XORed with the intermediate value to recover the plaintext.
    - The plaintext block is written to the output file.
    - The ciphertext block is used as the"IV" for the next block.
3. When the final block is detected, padding is removed before writing the last block of plaintext.

10 AES Encryption and Decryption in Counter Mode

10.1 Counter Management

A critical aspect of AES-CTR mode is the counter, which is 16 bytes long and is incremented after processing each block. The counter ensures that each block of plaintext is encrypted with a unique value. The counter operates as follows:

- The initial counter value is predefined and written to the encrypted file to ensure decryption can reverse the process.
- The last byte of the counter is incremented for each block, and if it overflows, the previous byte is incremented (similar to a multi-byte counter overflow).
- This counter is used as the input to the AES encryption algorithm for each block.

10.2 AES Encryption Process

The encryption process follows these steps:

1. For each block of plaintext, the counter is encrypted using the AES algorithm.
2. The encrypted counter is XORed with the plaintext block to produce the ciphertext.
3. The resulting ciphertext is written to the output file.
4. The counter is then incremented for the next block.

10.3 AES Decryption Process

The decryption process mirrors the encryption steps:

1. The encrypted counter for each block is recovered and passed through the AES encryption function.
2. The ciphertext block is XORed with the encrypted counter to produce the original plaintext.
3. The recovered plaintext is written to the output file.
4. The counter is incremented similarly as during encryption.

By using the same counter values during decryption, the original plaintext can be successfully restored.