Project Specification for AES Implementation

Objective

The goal is to implement the **Advanced Encryption Standard (AES)** algorithm in **C/C++** using a **Linux-based compiler (gcc/g++)**. The implementation must use direct binary operations and bitwise manipulations, adhering strictly to the AES standard (FIPS PUB 197). It should handle **encryption and decryption** through command-line arguments, process files for input/output, and use a specified key format. The project must implement **AES-128** (128-bit key) and operate in **ECB mode**.

Functional Requirements

1. AES Algorithm Implementation

- **Initial Transformation**:
  - Implement the **AddRoundKey** operation by XOR-ing the plaintext with the initial round key.
- **AES Core**:
  - **SubBytes**:
    - Apply a substitution operation to each byte of the state using the AES S-box.
  - **ShiftRows**:
    - Shift rows of the state by offsets as specified in the AES standard.
  - **MixColumns**:
    - Apply a linear transformation to each column using the AES-defined matrix multiplication in GF(2^8).
  - **AddRoundKey**:
    - XOR the state with the round key for each round.
- **Key Expansion**:
  - Generate 11 round keys from the 128-bit key.
  - Include:
    - **RotWord**: Rotate bytes of the word.
    - **SubWord**: Substitute bytes using the AES S-box.
    - RCon (Round Constant): Incorporate constants in the key expansion process.
- **Final Transformation**:
  - Perform the **SubBytes**, **ShiftRows**, and **AddRoundKey** steps for the last round, skipping **MixColumns**.

2. Command-line Interface

- The program must be run via the command line with two options: **encrypt** or **decrypt**.
- It must accept the following arguments:
  - **Encryption**:

    `./aes encrypt plain_text.txt key.txt cipher_text.dat`

    * **plain_text.txt**: File containing the plaintext input.
    * **key.txt**: File containing the 128-bit key in hexadecimal format (e.g., `2b7e151628aed2a6abf7158809cf4f3c`).
    * **cipher_text.dat**: Binary file where the encrypted ciphertext will be written.
  - **Decryption**:

    `./aes decrypt cipher_text.dat key.txt plain_text.txt`

    * **cipher_text.dat**: Binary file containing the ciphertext.
    * **key.txt**: File containing the 128-bit key.
    * **plain_text.txt**: File where the decrypted plaintext will be written.

3. Input/Output Files

- **Input Key**:
  - Provided in `key.txt` in **hexadecimal format**. The program must read this key and convert it to binary for processing.
- **Plaintext Input/Output**:
  - Input plaintext from `plain_text.txt` and output ciphertext to `cipher_text.dat` (binary format).
- **Ciphertext Input/Output**:
  - Input ciphertext from `cipher_text.dat` and output plaintext to `plain_text.txt`.

4. Single File Implementation

- All code must be written in a **single .cpp file** (e.g., `aes.cpp`) without external libraries or header files.
- The implementation must include:
  - Key expansion
  - AES core operations
  - File handling for input and output
  - Encryption and decryption logic

5. Binary and Bitwise Operations

- Use **direct binary representation** and **bitwise operations** for processing data.
- Avoid reliance on external cryptographic libraries (e.g., OpenSSL).

6. No Console Output

- The program must not produce any output to the console. All input and output operations must be handled exclusively through files.

Non-Functional Requirements

1. Performance

- Optimize for **32-bit platforms**, using efficient word-level operations.
- Leverage lookup tables (e.g., precomputed S-boxes and MixColumns transformations) to enhance performance.

2. Portability

- Ensure compatibility with **Linux** using `gcc` or `g++`.

3. Code Quality

- Maintain consistent formatting and clear function/variable naming.
- Ensure modularity within the single file by organizing the code into logical sections.

Deliverables

1. Single `.cpp` File

- The file must contain the entire AES implementation, including:
  - Key expansion
  - Core AES operations
  - Command-line interface
  - File input/output logic

2. Key File (`key.txt`)

- Provide a 128-bit key in hexadecimal format.

3. Input/Output Files

- Sample input files for plaintext and ciphertext.

4. Submission

- Submit the `.cpp` file to the specified LMS or repository.