Project 1: Encrypted File System Report

Name: Hemantha Krishna Challa
NetID: HXC230046

1. Design Explanation

1.1 Meta-Data Design

- **Location**: Metadata is stored in the first block (/file_name/0).
- **Structure**:
  - Bytes 0–127: Padded username (UTF-8, 128 bytes).
  - Bytes 128–143: 16-byte salt for PBKDF2.
  - Bytes 144–175: 32-byte encrypted keys (FEK + MK) using AES-ECB.
  - Bytes 176–187: 12-byte nonce for AES-CTR.
  - Bytes 188–191: 4-byte file length
  - Bytes 192–223: 32-byte HMAC-SHA256 of the preceding metadata.
- **Padding**: The entire block is padded to 1024 bytes.

1.2 User Authentication

- **Password handling**:
  - **PBKDF2-HMAC-SHA256** with 100,000 iterations generates a 16-byte KEK from the password and salt.
  - **FEK** (File Encryption Key) and **MK** (MAC Key) are encrypted using AES-ECB with KEK and stored in metadata.
- **Validation**:
  - The KEK is used to decrypt the concatenated FEK and MK stored in the metadata. An HMAC computed over the metadata (excluding the stored HMAC) is then compared to the stored HMAC. A mismatch results in a PasswordIncorrectException, thus enforcing that only the correct password allows decryption and subsequent access to file operations.

1.3 Encryption Design

- **File Data Encryption:**
  File data is stored in subsequent blocks (named /abc.txt/1, /abc.txt/2, ...). Each block is exactly 1024 bytes, where the first 32 bytes store the HMAC for that block and the remaining bytes (DATA_PER_BLOCK) store encrypted file data.
- **Mode and Key Usage:**
  AES in Counter (CTR) mode is used to encrypt file data. A unique initialization vector (IV) is constructed for each block by concatenating the fixed nonce (from the metadata) with the block number. This allows random access decryption and avoids IV reuse.
- **Key Encryption:**
  The FEK (used for file data encryption) and MK (used for HMAC computation) are generated randomly for each file. They are encrypted together with AES in ECB mode and stored in metadata.
- **Security Justification:**
  Using per-file random keys limits exposure in case of a key compromise. CTR mode provides efficient random-access encryption and decryption, while HMAC on each block ensures that any unauthorized modifications are detected. Even if the same plaintext is written multiple times, ciphertexts differ due to unique IVs.

1.4 File Length Hiding

- **File Length** is stored in metadata but padded to the nearest DATA_PER_BLOCK (992 bytes).

- **Blocks**: Files are split into 992-byte chunks. The actual length is hidden by padding plaintext within blocks.

- **Technique:**
  The file length is recorded as a 4-byte integer in the metadata block. Data blocks are always stored as fixed-size blocks (1024 bytes), even if the file content does not fill the last block.
- **Privacy Considerations:**
  Although the number of blocks used is revealed, no additional length information is leaked beyond what is inherent in the fixed-block structure.

1.5 Message Authentication

- **Implementation:**
  For every block (including the metadata block), an HMAC is computed:
  - Metadata: HMAC is computed over the concatenated fields (username, salt, encrypted keys, nonce, and file length) using the MK.
  - Data Blocks: HMAC is computed over the concatenation of the block number (as a 4-byte integer) and the ciphertext of the block.
- **Verification**:
  - Metadata HMAC ensures the integrity of keys, nonce, and length.
  - Per-block HMACs detect tampering with ciphertext or block numbers.

This ensures that tampering with file data or metadata can be reliably detected.

1.6 Efficiency

- **Storage**:
  - Each block stores 992 plaintext bytes (1024 bytes total with HMAC).
  - Minimal metadata (1 block) with no redundancy.
  - Additional blocks are only allocated as needed.
- **Speed**:
  - **Read/Write**: Only affected blocks are accessed (e.g., writing 1 byte modifies one block). This minimizes disk I/O by limiting operations only to the blocks that contain the data being read or modified.
  - **Cut**: Truncation deletes unnecessary blocks and updates only the last block.

---

2. Pseudo Code

2.1 create (file_name, user_name, password)

1. Generate salt (16B), nonce (12B), FEK (16B), MK (16B).

2. Derive KEK = PBKDF2(password, salt, 100k).

3. Encrypt FEK+MK with AES-ECB(KEK) → encryptedKeys (32B).

4. Build metadata: username || salt || encryptedKeys || nonce || file_length (0)

5. Compute HMAC (metadata, MK) → mac (32B).

6. Write padded metadata (1024B) to /file_name/0.

2.2 read (file_name, pos, len, password)

1. Validate password and decrypt FEK/MK.

2. For each block in [start_block, end_block]:

   a. Read block and verify HMAC with MK.

   b. Decrypt ciphertext with AES-CTR (FEK, nonce || block_num).

   c. Extract plaintext segment.

3. Return concatenated plaintext.

2.3 write (file_name, pos, content, password)

1. Validate password and decrypt FEK/MK.

2. For each affected block:

   a. Read existing block, decrypt, and modify plaintext.

   b. Encrypt modified plaintext with AES-CTR.

   c. Compute HMAC (block_num || ciphertext, MK).

   d. Write MAC || ciphertext to block.

3. Update file_length in metadata if expanded.

2.4 check_integrity (file_name, password)

1. Validate password and decrypt FEK/MK.

2. Verify metadata HMAC.

3. For each data block:

   a. Read block, verify HMAC (block_num || ciphertext, MK).

4. Return True if all checks pass.

2.5 cut (file_name, length, password)

1. Truncate to `length` by deleting excess blocks.

2. If the last block is partial:

   a. Decrypt, truncate plain text, re-encrypt, and update HMAC.

3. Update file_length in metadata.

---

3. Design Variations

3.1 Append-Only Writes

- **Optimization**: Append new blocks without modifying existing ones.
- **Efficiency Improvement:** The system would not need to support random-access updates. Each new append would only affect the final block or result in the allocation of a new block.
- **Metadata Update:** The file length will be updated once per append, and no in-place block updates will be needed.
- **Encryption Simplification:** Since appending does not modify existing blocks, there is no need to re-read and decrypt the previous content; only the new blocks need to be encrypted and MAC-ed.

This change reduces the number of disk accesses and computations per write, improving performance and reducing code complexity.

3.2 Single-Version Adversary

- **Simpler MAC Scheme:** It may be acceptable to compute a MAC only once when the file is created and after final writes, rather than maintaining MACs on every block.
- **Less Frequent Key Rotation:** Keys may be reused for longer durations if the adversary cannot compare multiple versions.

The overall storage and computational overhead can be reduced, though caution must be taken not to weaken security too much.

3.3 CBC Mode Feasibility

CBC (Cipher Block Chaining) mode can be used for encryption; however, it introduces challenges:

- **Random Access Issues:** CBC encryption requires sequential processing (each block depends on the previous ciphertext), which makes random-access reads and writes inefficient.
- **IV Management:** A unique IV per block would still be needed, but the chaining dependency would force the re-encryption of subsequent blocks when modifying any block.

Due to these efficiency and flexibility drawbacks, CBC mode is unsuitable for a file system that requires fast random access.

3.4 ECB Mode Feasibility

No. ECB (Electronic Codebook) mode is generally not recommended for encrypting file contents:

- **Security Concerns:** ECB encrypts identical plaintext blocks to identical ciphertext blocks, revealing patterns in the data.
- **Lack of Randomization:** Without an IV or nonce, ECB mode does not provide semantic security.

ECB mode might be used for fixed-size key encryption but must be avoided for general data encryption.

The EFS design in this project meets the specified functionality, security, and efficiency needs. The system provides strong confidentiality and integrity while minimizing storage and performance overhead.

References

1. **NIST Special Publication 800-38A** – *Recommendation for Block Cipher Modes of Operation – Methods and Techniques.*
   Available at: https://csrc.nist.gov/publications/detail/sp/800-38a/final

2. Menezes, A. J., van Oorschot, P. C., & Vanstone, S. A. (1996). *Handbook of Applied Cryptography.* CRC Press.

3. Schneier, B. (1996). *Applied Cryptography: Protocols, Algorithms, and Source Code in C.* John Wiley & Sons.

4. **Wikipedia Articles:**

   - AES (Advanced Encryption Standard) – https://en.wikipedia.org/wiki/Advanced_Encryption_Standard

   - Block cipher mode of operation -

     https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation

   - CTR Mode (Counter Mode) – https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation#CTR

   - HMAC – https://en.wikipedia.org/wiki/HMAC