Security Overview Document

Overview

This document explains the security architecture, encryption methods, and key management practices used in the Secure File Upload and Download System built with Flask and PyCryptodome. The system ensures that all uploaded files are encrypted before being stored on the server, maintaining confidentiality, integrity, and authenticity of data throughout its lifecycle.

Encryption Methodology

Algorithm - The system uses AES (Advanced Encryption Standard) in CBC (Cipher Block Chaining) mode for symmetric encryption. Each encryption operation uses a unique random Initialization Vector (IV) to ensure that even identical files produce different ciphertexts.

Process Flow

Upload Phase - When a user uploads a file, the server generates a random IV (16 bytes).
The plaintext is padded using PKCS#7 padding and encrypted with AES-CBC.
The resulting ciphertext is combined with the IV and authenticated using HMAC-SHA256.

Storage Phase - The server stores only the encrypted file. The original plaintext file is never stored on disk.

Download Phase - When a user downloads a file, the system reads the .enc file, separates the IV, ciphertext, and HMAC tag, and verifies the tag.
If HMAC verification succeeds, the ciphertext is decrypted and returned to the user.
If verification fails, decryption is aborted — ensuring that tampered or corrupted files are never returned.

Data Integrity and Authentication

To protect against file tampering and corruption, the system uses HMAC (Hash-based Message Authentication Code) with SHA-256. During encryption, an HMAC tag is generated over the IV and ciphertext using the same secret key.
During decryption, the tag is verified before decrypting.
If any byte of the stored file is altered, the verification fails and an integrity error is raised.
This ensures data integrity and authentication.

Key Management

Key Generation and Storage - The AES key is not hard-coded in the source code. It is securely stored in an environment variable named FILE_KEY_BASE64 as a base64-encoded value. This allows secure deployment without exposing the key in version control. A single symmetric key is used for both encryption and HMAC operations.

Security Benefits

Threat	**Mitigation**
Source code leakage:	Key not embedded in code
Unauthorized access:	Key stored outside the web root and source repository
Key rotation:	Possible by updating environment variable without changing code
Cross-environment isolation:	Each environment (dev, test, prod) can use its own key

Conclusion

This implementation adheres to secure file-handling practices by ensuring: Confidentiality through AES encryption, Integrity through HMAC verification, Basic key management through environment-based key loading. These mechanisms collectively provide a strong foundation for secure file transmission and storage in web applications.