# 🔐 Security Overview – Secure File Sharing System Task By Bhaskar Pagadala | Tools: VS Code | Languages: Python, HTML, CSS, JS (optional frontend) | Dev Library: Flask | Encryption Library: pycryptodome or cryptography | Tools: Git, GitHub, VS Code | Secure File Sharing (Flask + AES)

# 1. Purpose

This document explains the security design choices in the Secure File Sharing System, including encryption methods, key management, and best practices followed during development.

## **2. Encryption Method**

### **AES (Advanced Encryption Standard)**

* **Mode Used:** AES EAX Mode (for both encryption and integrity verification)
* **Key Size:** 256-bit (32 bytes) symmetric key
* **Why AES?**
  + Strong, widely adopted standard for symmetric encryption
  + Fast performance for large files
  + Resistant to common brute-force attacks when used with a secure key

## **3. Encryption Process**

1. **Key Generation**
   1. A unique 256-bit key is generated using a secure random generator (get\_random\_bytes(32)).
   2. This key is stored locally as key.key (for this demo project).
2. **File Encryption**
   1. A **nonce** (random number used once) is generated for each encryption session.
   2. The file’s binary data is encrypted using AES-EAX with the generated nonce.
   3. A **tag** is produced for message authentication, ensuring the file hasn’t been tampered with.
   4. Encrypted output format:

css

CopyEdit

[nonce][tag][ciphertext]

1. **File Decryption**
   1. Reads the nonce, tag, and ciphertext from the encrypted file.
   2. Uses the same AES key to decrypt and verify file authenticity.
   3. If verification fails, the system flags the file as corrupted or altered.

## **4. Key Handling**

⚠ **Demo Project Note:** For this learning project, the AES key is stored locally (key.key) and committed to the repository **only for demonstration purposes**.

### **Best Practice (Production)**

* Never commit encryption keys to public repositories.
* Store keys in secure vaults like:
  + **AWS Secrets Manager**
  + **Azure Key Vault**
  + **HashiCorp Vault**
* Restrict key access to only the services or individuals that require it.
* Rotate keys regularly to minimize exposure risk.

## **5. Security Benefits**

✅ **Confidentiality:** AES encryption ensures files can’t be read without the key.  
 ✅ **Integrity:** AES-EAX mode includes authentication to prevent undetected modifications.  
 ✅ **Local Control:** Users manage encryption keys locally, limiting exposure to external threats.

## **6. Limitations**

* If the AES key is leaked, all encrypted files can be decrypted.
* Key rotation is not implemented in this demo.
* No user authentication/authorization features (future enhancement).

## **7. Future Enhancements**

* Implement **secure key storage** in a vault
* Add **multi-user authentication** and **access control**
* Enable **automatic key rotation**
* Support for **asymmetric encryption** for key exchange

📌 **Conclusion:**  
 This system demonstrates the **core principles of secure file storage and transfer** using modern encryption standards. While suitable for learning and testing, real-world deployment would require robust key management, authentication, and access control mechanisms.