Secure File Sharing System

1. Purpose of Security Design

The Secure File Sharing System is designed to protect sensitive files during storage and transfer. The primary security objectives are:

- **Confidentiality** – Prevent unauthorized access to files
- **Integrity** – Ensure files are not modified during storage
- **Controlled Access** – Only intended users can retrieve decrypted files

This is achieved through strong encryption, secure key handling, and controlled file processing.

2. Encryption Method Used

2.1 Algorithm: Advanced Encryption Standard (AES)

The system uses **AES (Advanced Encryption Standard)**, which is:

- A symmetric encryption algorithm
- Widely adopted by governments and enterprises
- Approved by NIST (FIPS 197)

2.2 Key Size

- **AES-256** (256-bit key length)
- Provides high resistance against brute-force attacks
- Considered secure for long-term data protection

Key Length: 256 bits

Block Size: 128 bits

3. Encryption Mode

AES Mode: CBC (Cipher Block Chaining)

The system uses AES-CBC mode with the following characteristics:

- Each plaintext block is XORed with the previous ciphertext block
- Prevents identical plaintext blocks from producing identical ciphertext
- Requires an **Initialization Vector (IV)**

Security Properties:

- Ensures confidentiality
- Protects against pattern analysis

Implementation Detail:

- A random IV is generated for each file encryption
- IV is prepended to the encrypted file for use during decryption

4. Encryption Workflow

File Upload Process

1. User uploads a file via the web interface
2. File is temporarily stored in the server memory
3. File contents are encrypted using AES-256-CBC
4. Encrypted file is saved with .enc extension
5. Original plain text file is securely deleted

File Download Process

1. User requests file download
2. Encrypted file is retrieved
3. IV is extracted from encrypted data
4. File is decrypted in memory
5. Decrypted file is sent to the user
6. Temporary decrypted file is removed after transfer

5. Padding Mechanism

Padding Scheme: PKCS7

- AES operates on fixed-size blocks (128 bits)
- PKCS7 padding ensures plain text fits block size
- Padding is safely removed during decryption

This prevents data corruption and padding oracle issues in basic implementations.

6. Encryption Key Handling

6.1 Key Generation

- Encryption key is generated using:

Crypto.Random.get_random_bytes(32)

- Cryptographically secure random number generator
- Generated at application runtime

6.2 Key Storage (Current Implementation)

- Key is stored **in application memory**
- Key exists only while the application is running
- Key is **never written to disk**

6.3 Security Implications

Advantages

- Reduces risk of key leakage from filesystem
- Simple and safe for demonstration purposes

Limitations

- Files become unreadable if server restarts
- Not suitable for production environments

8. Transport Security

- Application runs on HTTP (development)
- In production, HTTPS should be enforced
- TLS protects encryption keys and file data in transit

9. Access Control

- File access is limited through application routes
- Direct access to encrypted files is not exposed
- Download requires correct file reference

Future enhancements may include:

- Authentication (login system)
- Role-based access control
- Download authorization tokens

10. Security Best Practices Followed

✓ Strong AES-256 encryption
✓ Random IV per encryption
✓ No plaintext file storage
✓ Secure padding handling
✓ Separation of encrypted and decrypted files
✓ Minimal attack surface

12. Conclusion

The Secure File Sharing System demonstrates practical implementation of modern encryption techniques and secure file handling practices. It mirrors real-world scenarios where data confidentiality is critical, while maintaining simplicity for learning and demonstration purposes.