SECURE FILE SHARING SYSTEM SECURITY OVERVIEW

Name: Odita Chukwudumebi Tobechukwu

Task 3: Secure file sharing system(FLASK & AES)

Program: Future Interns Cybersecurity Internship

Date: September 2025

Executive Summary

This document provides a comprehensive security overview of the Secure File Sharing System developed as the final task for the Cyber Security Internship at Future Interns. The system implements military-grade encryption to protect files both at rest and during transfer, demonstrating enterprise-level security practices in a web application environment.

Tools Used

- VScode – Visual Studio Code editor
- Python(Flask)

1. Encryption Methodology

Encryption Algorithm: AES-256-CBC

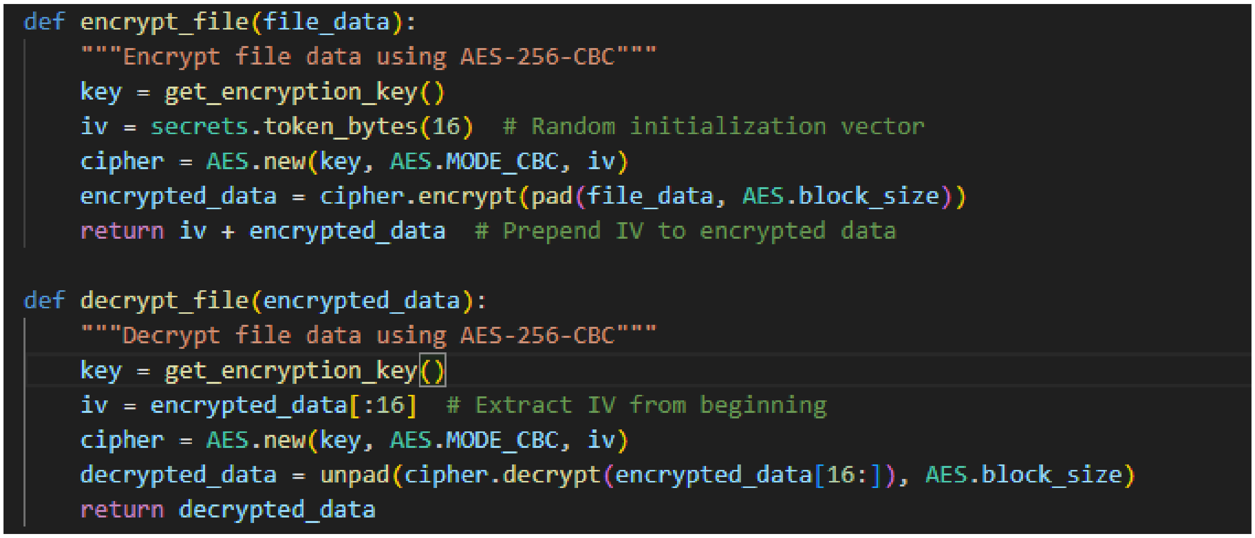
- **Algorithm**: Advanced Encryption Standard (AES)
- **Key Size**: 256-bit (Military Grade)
- **Mode**: Cipher Block Chaining (CBC)
- **Padding**: PKCS7 Padding

Encryption Process Flow

1. **File Upload**:
    - User selects file through web interface
    - File read into memory as binary data
    - Generate random 16-byte Initialization Vector (IV)
    - Encrypt file data using AES-256-CBC with secret key
    - Prepend IV to encrypted data
    - Save encrypted file with `.enc` extension
2. **File Download**:
    - Read encrypted file from storage
    - Extract IV from first 16 bytes
    - Decrypt remaining data using AES-256-CBC

- Remove PKCS7 padding
- Serve decrypted file to user

2. Key Technical Implementation



```python
def encrypt_file(file_data):
    """Encrypt file data using AES-256-CBC"""
    key = get_encryption_key()
    iv = secrets.token_bytes(16)  # Random initialization vector
    cipher = AES.new(key, AES.MODE_CBC, iv)
    encrypted_data = cipher.encrypt(pad(file_data, AES.block_size))
    return iv + encrypted_data  # Prepend IV to encrypted data

def decrypt_file(encrypted_data):
    """Decrypt file data using AES-256-CBC"""
    key = get_encryption_key()
    iv = encrypted_data[:16]  # Extract IV from beginning
    cipher = AES.new(key, AES.MODE_CBC, iv)
    decrypted_data = unpad(cipher.decrypt(encrypted_data[16:]), AES.block_size)
    return decrypted_data
```

Key Management

Secret Key Generation

- **Method**: Cryptographically secure random generation
- **Format**: Base64-encoded 32-byte key
- **Storage**: Hardcoded in application (for demonstration)
- **Production Recommendation**: Environment variables/Key Management Service

Key Security Features

- **Random IV**: Unique IV generated for each file
- **Key Isolation**: Encryption key separate from application logic
- **No Plaintext Storage**: Files never stored unencrypted

3. Security Measures

Data Protection

- **Encryption at Rest**: All files encrypted before storage
- **Secure Transmission**: Encryption maintained during download
- **IV Best Practices**: Unique IV per file prevents pattern analysis

Application Security

- **File Validation**: Basic file type checking
- **Size Limits**: 16MB maximum file size
- **Secure Deletion**: Encrypted files properly deleted
- **Error Handling**: Graceful failure without information leakage

Security Considerations

Strengths

- Military-grade AES-256 encryption
- Proper CBC mode implementation
- Cryptographically secure random number generation
- Secure IV management

Areas for Improvement (Production Environment)

- Implement proper key rotation
- Add user authentication
- Use HTTPS in production
- Implement key management service
- Add file integrity verification
- Implement access controls

4. Threat Model

Protected Against

- Unauthorized file access
- Storage medium compromise
- Network eavesdropping
- Data tampering at rest

Not Protected Against (In Current Implementation)

- Key compromise
- Server-side attacks
- User authentication bypass
- Side-channel attacks

Compliance & Standards

- **NIST Approved**: AES-256 (FIPS 197)
- **Best Practices**: CBC mode with random IV
- **Cryptographic Standards**: PKCS7 padding