🔐 Step-by-Step Guide to Implement a Secure File Storage System using Hybrid Encryption on Azure

This guide provides a **detailed step-by-step implementation** for your project, ensuring security best practices and leveraging **Azure services** effectively.

📌 Overview

This system enables users to:

1. **Upload files** → Files are encrypted with **AES**.
2. **Encrypt the AES key with RSA** (stored securely in **Azure Key Vault**).
3. **Store encrypted files in Azure Blob Storage**.
4. **Retrieve and decrypt files** → The AES key is decrypted via **Azure Key Vault**, then the file is decrypted.

🛠️ Tech Stack

- **Backend:** Python (Flask/FastAPI) or Node.js (Express)
- **Frontend:** React.js (optional)
- **Database:** Azure SQL Database / CosmosDB (for metadata)
- **Storage:** Azure Blob Storage (for encrypted files)
- **Key Management:** Azure Key Vault (for RSA key storage)
- **Authentication:** Azure Active Directory (AAD)

🚀 Step 1: Set Up Azure Cloud Services

1.1 Create an Azure Storage Account

1. **Log into Azure Portal** → Go to **Storage Accounts**.
2. Click **Create Storage Account** → Fill in:
   - **Subscription**: Select your subscription.
   - **Resource Group**: Create/select a resource group.
   - **Storage Account Name**: Choose a unique name (e.g., `securefilestorage`).
   - **Region**: Select a region close to your users.
   - **Performance**: Standard.
   - **Replication**: Locally Redundant Storage (LRS).
   - **Access tier**: Hot.
3. Click **Review + Create** → **Create**.

Create a Blob Container

1. Inside **Storage Account**, go to **Containers**.
2. Click **+ Container** → Name it `encrypted-files`.
3. Set **Access Level** to **Private**.

1.2 Create an Azure Key Vault

1. **Go to Azure Portal** → Search for **Key Vault** → Click **Create**.
2. **Set up the Key Vault**:
   - **Subscription**: Select your subscription.
   - **Resource Group**: Use the same one from Storage.
   - **Key Vault Name**: Choose a unique name (e.g., `securevault`).
   - **Region**: Same as storage.

- Click **Review + Create** → **Create**.

Generate an RSA Key Pair

1. Inside **Key Vault**, go to **Keys**.
2. Click **Generate/Import** → Select:
   - **Type**: RSA
   - **Key Size**: 2048 or 4096 (4096 is more secure).
   - **Key Name**: `rsa-encryption-key`.
3. Click **Create**.

1.3 Create an Azure SQL Database for Metadata

1. **Go to Azure Portal** → Search for **SQL Database** → Click **Create**.
2. **Database Name**: `FileMetadataDB`
3. **Server**: Create a new server (`securefile-db-server`).
4. **Authentication**: SQL Authentication (store username/password securely).
5. **Compute + Storage**: Basic (for small projects).
6. Click **Review + Create** → **Create**.

Table Structure

sql

Copy Edit

```sql
CREATE TABLE file_metadata (
    id INT PRIMARY KEY IDENTITY(1,1),
    filename NVARCHAR(255) NOT NULL,
    file_url NVARCHAR(1024) NOT NULL,
    encrypted_aes_key NVARCHAR(2048) NOT NULL,
    upload_timestamp DATETIME DEFAULT CURRENT_TIMESTAMP
);
```

1.4 Set Up Azure Active Directory (AAD) for Authentication

1. **Go to Azure Portal** → Search for **Azure Active Directory**.
2. **Create an App Registration**:
   - **Name**: `SecureFileApp`
   - **Redirect URI**: `http://localhost:3000`
3. Save the **Client ID & Tenant ID**.
4. **Assign Roles** → Give backend permissions to access Storage & Key Vault.

🚀 Step 2: Backend Implementation (Encryption & File Handling)

2.1 Install Required Libraries

For **Python (Flask) Backend**:

bash

Copy Edit

```bash
pip install flask flask-sqlalchemy azure-storage-blob azure-identity cryptography
```

For **Node.js (Express) Backend**:

bash

Copy Edit

```
npm install express azure-storage azure-keyvault-keys @azure/identity crypto
```

2.2 Encrypt and Upload Files

1. **Generate an AES Key** → Encrypt the file.
2. **Encrypt AES Key with RSA (via Azure Key Vault)**.
3. **Upload Encrypted File to Blob Storage**.
4. **Store Encrypted AES Key in Azure SQL**.

Python Code for File Encryption

python

Copy Edit

```
from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes
import os

def encrypt_file(file_data):
    aes_key = os.urandom(32)  # Generate random AES key
    iv = os.urandom(16)  # Initialization vector
    cipher = Cipher(algorithms.AES(aes_key), modes.GCM(iv))
    encryptor = cipher.encryptor()
    encrypted_data = encryptor.update(file_data) + encryptor.finalize()
    return encrypted_data, aes_key, iv
```

Encrypt AES Key with Azure Key Vault

python

Copy Edit

```
from azure.identity import DefaultAzureCredential
from azure.keyvault.keys.crypto import CryptographyClient, EncryptionAlgorithm

key_vault_name = "securevault"
key_name = "rsa-encryption-key"

credential = DefaultAzureCredential()
crypto_client = CryptographyClient(f"https://{key_vault_name}.vault.azure.net/keys/{key_name}", credential)

encrypted_aes_key = crypto_client.encrypt(EncryptionAlgorithm.rsa_oaep, aes_key).ciphertext
```

Upload File to Blob Storage

python

Copy Edit

```
from azure.storage.blob import BlobServiceClient

blob_service_client = BlobServiceClient.from_connection_string("YOUR_AZURE_CONNECTION_STRING")
blob_client = blob_service_client.get_blob_client(container="encrypted-files", blob="file.enc")

blob_client.upload_blob(encrypted_data)

# Save encrypted AES key to database
cursor.execute("INSERT INTO file_metadata (filename, file_url, encrypted_aes_key) VALUES (?, ?, ?)",
               ("file.enc", blob_client.url, encrypted_aes_key.hex()))
conn.commit()
```

2.3 Decrypt & Retrieve Files

1. **Fetch encrypted file from Blob Storage**.
2. **Retrieve Encrypted AES Key from Database**.
3. **Decrypt AES Key with Key Vault**.
4. **Decrypt File with AES Key**.

Decrypt AES Key

python
Copy Edit
```python
decrypted_aes_key = crypto_client.decrypt(EncryptionAlgorithm.rsa_oaep, encrypted_aes_key).plaintext
```

Decrypt File

python
Copy Edit
```python
def decrypt_file(encrypted_data, aes_key, iv):
    cipher = Cipher(algorithms.AES(aes_key), modes.GCM(iv))
    decryptor = cipher.decryptor()
    return decryptor.update(encrypted_data) + decryptor.finalize()
```

🚀 Step 3: Frontend (React)

1. **Implement File Upload Form** (calls Flask API).
2. **Implement File Download Page** (fetches & decrypts files).
3. **Use Azure AD for Authentication**.

✅ Step 4: Deploy to Azure

1. **Deploy Backend using Azure App Services**.
2. **Deploy Frontend (React) to Azure Static Web Apps**.
3. **Enable Azure Defender for Storage**.

🚀 Final Notes

✅ **AES Key is never stored in Blob Storage**
✅ **RSA Private Key is securely stored in Azure Key Vault**
✅ **Only authorized users can access files (via AAD)**

Would you like a **GitHub starter template** for quick setup? 🚀