Encrypted Cloud File Storage System (Google Drive + AES) — IDE Setup & Execution Guide

> **What we'll build:** A Python application that *encrypts files locally with AES*, uploads them to *Google Drive* using OAuth 2.0, and can download & decrypt them back.

1) Prerequisites & Requirements

Component	Minimum Requirement	Notes
Python	3.7+	Check with `python --version`.
IDE	Any IDE like VSCode, PyCharm	We will create a new project with a virtual environment (venv).
Google Account	Personal Gmail or Workspace	Needed to enable APIs and create OAuth credentials.
APIs	Google Drive API	Must be **enabled** in the *same project* as your OAuth client.
Libraries	`google-api-python-client`, `google-auth-httplib2`, `google-auth-oauthlib`, `pycryptodome`, `tqdm`	Installed inside your venv.
Credentials	`credentials.json`	OAuth 2.0 (Desktop App) JSON downloaded to your project folder.

2) Create the Google Cloud Project & Enable Drive API

1. Go to *Google Cloud Console* (console.cloud.google.com) and **Create** or **Select** a project.
2. Open **APIs & Services** → **Library** → search for **Google Drive API** → **Enable**.

3. Open **APIs & Services** → **OAuth consent screen** and configure:
    - **User Type:** Usually *External* for personal Gmail. For Google Workspace, you may use *Internal*.
    - **App name**, **Support email**, and at least one **Developer contact email**.
    - Add scope later via code; `drive.file` will be requested at runtime.
    - If External and in Testing mode, add yourself in **Test users**.
4. Go to **APIs & Services** → **Credentials** → **Create Credentials** → **OAuth client ID**:
    - **Application type:** Desktop app
    - **Name:** e.g., *DriveEncryptor*
    - **Download** the JSON and save as `credentials.json` in your PyCharm project root.

3) Start a New IDE Project

1. Open IDE → **New Project**.
2. Choose a location (e.g., `C:\Users\<you>\PycharmProjects\P1`).
3. Select **New Virtual Environment** (venv) and create the project.
4. Open **Terminal** in PyCharm (bottom panel) and install dependencies:

```
pip install google-api-python-client google-auth-httplib2 google-auth-oauthlib
pycryptodome tqdm
```

4) Project Structure & Files

```
Project_Name/
├─ credentials.json          # OAuth client (downloaded from Google Cloud Console)
├─ token.pickle              # Generated after first successful OAuth;
├─ auth.py                   # Authentication helper
├─ crypto_utils.py           # AES encryption/decryption helpers
├─ main.py                   # Orchestrates encrypt → upload → download → decrypt
└─ (venv)/                   # Your virtual environment
```

4.1 auth.py

```
import os
import pickle
from googleapiclient.discovery import build
from google_auth_oauthlib.flow import InstalledAppFlow
from google.auth.transport.requests import Request

SCOPES = ['https://www.googleapis.com/auth/drive.file']

def get_gdrive_service():
    creds = None
    if os.path.exists('token.pickle'):
        with open('token.pickle', 'rb') as token:
            creds = pickle.load(token)
    if not creds or not creds.valid:
        if creds and creds.expired and creds.refresh_token:
            creds.refresh(Request())
        else:
            flow = InstalledAppFlow.from_client_secrets_file('credentials.json',
SCOPES)
            creds = flow.run_local_server(port=0)
        with open('token.pickle', 'wb') as token:
            pickle.dump(creds, token)
    return build('drive', 'v3', credentials=creds)
```

4.2 crypto_utils.py

```
from Crypto.Cipher import AES
from Crypto.Random import get_random_bytes

def encrypt_file(input_file, output_file, key):
    cipher = AES.new(key, AES.MODE_EAX)
    with open(input_file, 'rb') as f:
        data = f.read()
    ciphertext, tag = cipher.encrypt_and_digest(data)
    with open(output_file, 'wb') as f:
        for x in (cipher.nonce, tag, ciphertext):
            f.write(x)

def decrypt_file(input_file, output_file, key):
    with open(input_file, 'rb') as f:
        nonce, tag, ciphertext = [f.read(x) for x in (16, 16, -1)]
    cipher = AES.new(key, AES.MODE_EAX, nonce)
    data = cipher.decrypt_and_verify(ciphertext, tag)
    with open(output_file, 'wb') as f:
        f.write(data)

def generate_key():
    return get_random_bytes(16)  # 128-bit AES key; store securely!
```

4.3 main.py

```
from auth import get_gdrive_service
from crypto_utils import encrypt_file, decrypt_file, generate_key
from googleapiclient.http import MediaFileUpload, MediaIoBaseDownload
import io

def upload_file(service, filepath, filename):
    file_metadata = { 'name': filename }
    media = MediaFileUpload(filepath, resumable=True)
    file = service.files().create(body=file_metadata, media_body=media,
fields='id').execute()
    print("Uploaded file ID:", file.get('id'))
    return file.get('id')

def download_file(service, file_id, output_path):
    request = service.files().get_media(fileId=file_id)
    fh = io.FileIO(output_path, 'wb')
    downloader = MediaIoBaseDownload(fh, request)
    done = False
    while not done:
        status, done = downloader.next_chunk()
        print("Download %d%%." % int(status.progress() * 100))

if __name__ == "__main__":
    service = get_gdrive_service()

    key = generate_key()
    print("Generated AES Key:", key)

    # Prepare a sample file
    with open("mysecret.txt", "w", encoding="utf-8") as f:
        f.write("This is a confidential test file.")

    # Encrypt → Upload → Download → Decrypt
    encrypt_file("mysecret.txt", "mysecret.enc", key)
    print("File encrypted.")
    file_id = upload_file(service, "mysecret.enc", "mysecret.enc")
    download_file(service, file_id, "mysecret_downloaded.enc")
    decrypt_file("mysecret_downloaded.enc", "mysecret_decrypted.txt", key)
    print("Decryption complete. Check mysecret_decrypted.txt")
```

5) Permissions, Scopes & Security Settings

- **OAuth Consent Screen**:
  - Set *App name*, *Support email*, and *Developer contact email*.
  - If *External* and in Testing mode, add your Google account under **Test users**.
- **Scope Used:** `drive.file`
  - Grants access to files created or opened by this app — *least privilege* vs full Drive.
  - Shown during the Google login consent screen.
- **Tokens:** `token.pickle` stores OAuth tokens locally.

    - Delete it to force re-consent or when rotating credentials.
    - Keep it private; treat as a secret.
- **AES Keys:** store the generated key securely:
    - Do *not* hardcode in source code or commit to git.
    - Options: environment variables, OS secrets manager, encrypted keyfile, or cloud KMS.

6) Running the Project in PyCharm

1. Place `credentials.json` in your project root.
2. Open `main.py` → **Run** (green play button or right-click → Run).
3. Your browser opens for Google sign-in → **Allow** requested permissions.
4. First run will create `token.pickle`.
5. Console output should show:

```
6. Generated AES Key: b'...'
7. File encrypted.
8. Uploaded file ID: 1AbCDEfgHiJKlmnOPQ...
9. Download 100%.
```

```
Decryption complete. Check mysecret_decrypted.txt
```

10. Verify: open `mysecret.txt` vs `mysecret_decrypted.txt` — contents must match.
11. In Google Drive, you should see `mysecret.enc` (encrypted binary).

7) Validation & Testing Checklist

- Compare files (Windows):

```
fc mysecret.txt mysecret_decrypted.txt
```

- Open `mysecret.enc` in a hex viewer — it must look random (no readable text).
- Try different file types: *.png, .pdf, .docx, .zip*.
- Try larger files and observe progress logs during download.

8) Troubleshooting

Symptom	Cause	Fix
`HttpError 403 accessNotConfigured` (ResumableUploadError)	Drive API not enabled for the project linked to your `credentials.json`.	Enable Drive API in the *same* project; replace `credentials.json` if needed; delete `token.pickle` and re-run.

`redirect_uri_mismatch`	Wrong OAuth client type or misconfigured credentials.	Ensure OAuth client is **Desktop app**; re-download `credentials.json`.
OAuth window shows “This app isn’t verified”	External app in Testing mode.	Add yourself as a Test user in the consent screen; proceed with the warning.
Upload fails intermittently	Network issues or expired token.	Retry; delete `token.pickle` to refresh tokens; check internet connection.
Decryption fails with `ValueError: MAC check failed`	Wrong AES key or corrupted download.	Use the exact key used for encryption; re-download; validate file integrity.

Appendix A — Quick Commands

```
# Inside your project's venv
pip install google-api-python-client google-auth-httplib2 google-auth-oauthlib
pycryptodome tqdm

# Force re-authentication
del token.pickle  # (Windows: delete the file in Explorer or use 'del
token.pickle' in cmd)
```