Password Manager Seguro con DNIe y Criptografía

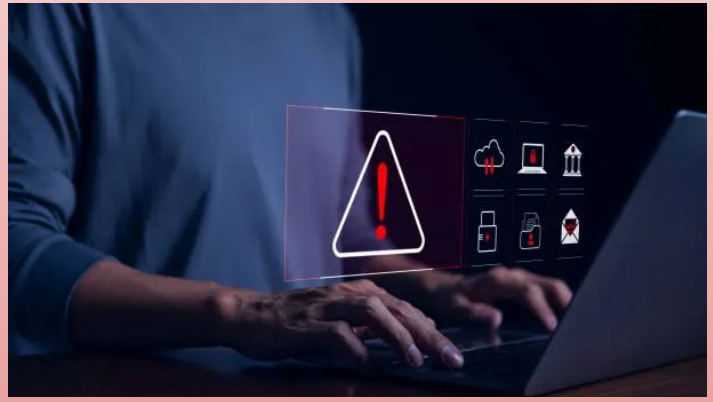
Python + RSA-OAEP + AES-Fernet + PKCS#11

Edgar González Romea
Jorge Pérez Sarasa

Problema VS Solución

PROBLEMA

- Demasiadas contraseñas, poca seguridad.
- Guardarlas en el navegador → inseguro.



SOLUCIÓN

- Gestor de contraseñas.
- Cifrado local + DNIe.
- Sin DNI físico + PIN → no accedes.

Arquitectura del Sistema

Usuario CLI

↓

Main.py → Comandos, DNIe

↓

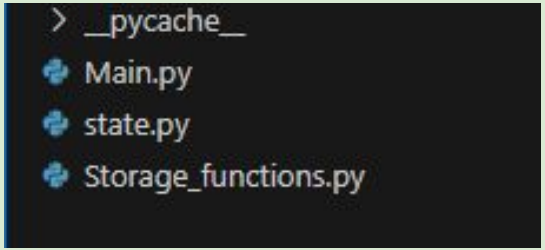
Storage_functions.py → Cifrado AES

↓

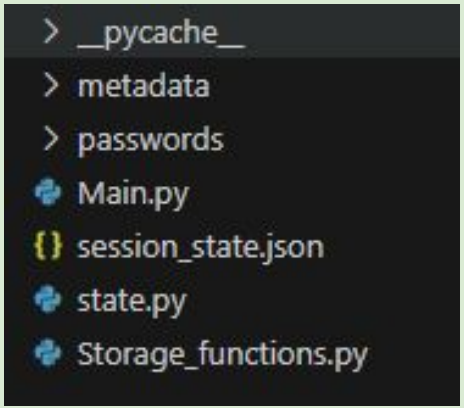
state.py → Variables compartidas “unlocked” y “k_db”

↓

Archivos: metadata.json + passwords.db + session_state.json



Tras ejecutarse



Tecnologías

TECNOLOGÍA	FUNCIÓN
Fernet (AES-128 + HMAC)	Cifra toda la base de datos.
RSA-OAEP	Protege la clave k_db con el DNIe.
PKCS#11	Para comunicar con el chip.
JSON	Almacenamiento de metadatos y entradas.

Flujo del programa

- INIT
- WRAP
- UNLOCK
- STATUS
- ADD / LIST / GET / EDIT
- LOCK

```
Password Manager con DNIe

options:
  -h, --help            show this help message and exit

Comandos disponibles:
  {init,wrap,unlock,lock,status,add,list,get,edit}
    init                Inicializar base de datos
    wrap                Cifrar DB con DNIe (AES + challenge)
    unlock              Desbloquear DB usando PIN del DNIe
    lock                Bloquear DB en memoria
    status              Estado DB
    add                 Añadir entrada
    list                Listar entradas
    get                 Obtener entrada por ID
    edit                Editar entrada por ID
```

Funcionamiento del Main

init()
Inicializa la base de datos y genera la clave `k_db`.
→ Crea el archivo vacío `passwords.db`, genera una clave aleatoria (`os.urandom(32)`) y guarda los metadatos iniciales.

wrap()
Cifra `k_db` y la firma con el DNIe.
→ Usa AES-GCM para cifrar la clave, firma un *challenge* con el DNIe y guarda `nonce`, `aes_key` y `signature`.

unlock()
Desbloquea la base verificando el DNIe.
→ Comprueba la firma con el certificado del DNIe, descifra `k_db` y la carga en memoria (`state.k_db`).

status()
Muestra el estado actual.
→ Indica si la base de datos está bloqueada o desbloqueada.

add()
Añade una nueva contraseña a la base.
→ Lee `temp_db.json`, crea una nueva entrada con `id`, `name`, `username`, `password` y guarda el JSON actualizado.

list()
Muestra todas las contraseñas guardadas.
→ Carga `temp_db.json`, recorre las entradas y las muestra ordenadas por `id`.

get()
Consulta una contraseña concreta.
→ Busca la entrada por `id`, muestra sus datos y guarda una copia `*_get.json`.

edit()
Modifica una contraseña existente.
→ Pide nuevos valores al usuario, actualiza la entrada correspondiente y guarda los cambios.

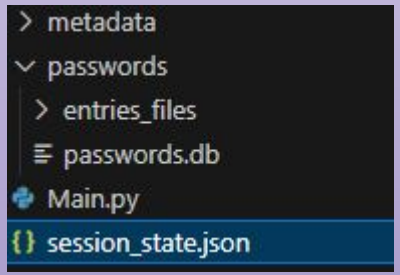
lock()
Bloquea la base de datos y borra la clave.
→ Vacía `state.k_db` y elimina el archivo temporal en JSON.

Demostración de funcionamiento

1. Ejecutamos init, wrap y unlock. Comprobando que status responde correctamente. Simultáneamente vemos que se crean los directorios metadata.json + passwords.db

Además de guardar en todo momento las variables compartidas “unlocked” y “k_db”, que se guardan es session_state.json.

```
PS C:\Users\Edgar\Desktop\Teleco 2025-2026\Cuatri 1\Seguridad R y S\Trabajos\PASSWORD MANAGER\PASSWORD_MANAGER_DNIe\Pm> python Main.py init
Inicializando base de datos...
🔑 Clave inicializada y desbloqueada.
✅ DB inicializada en RAW_BASE64 y desbloqueada en memoria.
PS C:\Users\Edgar\Desktop\Teleco 2025-2026\Cuatri 1\Seguridad R y S\Trabajos\PASSWORD MANAGER\PASSWORD_MANAGER_DNIe\Pm> python Main.py status
DB desbloqueada
PS C:\Users\Edgar\Desktop\Teleco 2025-2026\Cuatri 1\Seguridad R y S\Trabajos\PASSWORD MANAGER\PASSWORD_MANAGER_DNIe\Pm> python Main.py wrap
Introduce PIN del DNIe:
🔒 DB bloqueada en memoria.
✅ DB cifrada con AES+DNIe y bloqueada en memoria.
PS C:\Users\Edgar\Desktop\Teleco 2025-2026\Cuatri 1\Seguridad R y S\Trabajos\PASSWORD MANAGER\PASSWORD_MANAGER_DNIe\Pm> python Main.py unlock
Introduce PIN del DNIe:
🔓 DB desbloqueada y guardada en disco.
```



session_state.json > ...

```
1   {"unlocked": false, "k_db": "QVhFUXJxUjB1SUlwTUVMYWtoOXdjTW11c1VZT2prcVJSdFJCRjZqbzFlQT0="}
```

Ejemplo tras el wrap clave bloqueada y clave cifrada

metadata > {} metadata.json > ...

```
1   {"enc_k_db": "tDZmnqfGsD2xHl+Mcb63W/0d1kEhYj0ybHrDbdCsDMnDT3mxqdCKGUXO6gfmv6iE0vGq61770TBrmApw", "aes_nonce": "A5JQAq0g7La9qdwt", "aes_key_b64": "6uDqjl8m7ALpl3ifGg9iDZUwa4mGTsH5
```

2. Ejecutamos add, list, get y edit , los cuales modifican nuestros registros de usuarios y contraseñas.

Se crea cuando ejecutamos add un directorio nuevo con los registros añadidos: (id,name,username,password,notes,created) dentro de passwords.db llamado entries, con el comando list podemos ver todas las entradas añadidas, el get nos ayuda a seleccionar una entrada en concreto y con edit nos permite editar los registros de una entrada determinada.

```
PS C:\Users\Edgar\Desktop\Teleco 2025-2026\Cuatri 1\Seguridad R y S\Trabajos\PASSWORD MANAGER\PASSWORD_MANAGER_DNIe\Pm> python Main.py add  "Gmail" -u "usuaric@gmail.com" -p "clave123" -n "Cuent
al"
✅ Entrada 'Gmail' añadida y DB ordenada.
📄 Archivo guardado en passwords\entries_files\1_Gmail.json
PS C:\Users\Edgar\Desktop\Teleco 2025-2026\Cuatri 1\Seguridad R y S\Trabajos\PASSWORD MANAGER\PASSWORD_MANAGER_DNIe\Pm> python Main.py list
Listado de entradas:
1: Gmail (usuaric@gmail.com)
📂 Archivos individuales guardados en passwords\entries_files
PS C:\Users\Edgar\Desktop\Teleco 2025-2026\Cuatri 1\Seguridad R y S\Trabajos\PASSWORD MANAGER\PASSWORD_MANAGER_DNIe\Pm> python Main.py get 1
{
  "id": 1,
  "name": "Gmail",
  "username": "usuaric@gmail.com",
  "password": "clave123",
  "notes": "Cuenta personal",
  "created": "2025-10-24T02:07:49.379153Z"
}
📄 Archivo guardado en passwords\entries_files\1_Gmail_get.json
PS C:\Users\Edgar\Desktop\Teleco 2025-2026\Cuatri 1\Seguridad R y S\Trabajos\PASSWORD MANAGER\PASSWORD_MANAGER_DNIe\Pm> python Main.py edit 1
Editando entrada ID 1 (dejar vacío para no cambiar):
Nombre [Gmail]: pepe@gmail.com
Username [usuaric@gmail.com]: Pepe Laez
Password [clave123]: Perico654
Notas [Cuenta personal]: correo personal
✅ Entrada ID 1 actualizada correctamente.
📄 Archivo actualizado en passwords\entries_files\1_pepe@gmail.com.json
```

passwords > entries_files > {} 1_pepe@gmail.com.json > ...

```
{
  "id": 1,
  "name": "pepe@gmail.com",
  "username": "Pepe Laez",
  "password": "Perico654",
  "notes": "correo personal",
  "created": "2025-10-24T02:07:49.379153Z"
}
```

3. Ejecutamos lock para bloquear el acceso a nuestros registros además de borrar la clave hasta que sean desbloqueados otra vez con el mismo método

```
PS C:\Users\Edgar\Desktop\Teleco 2025-2026\Cuatri 1\Seguridad R y S\Trabajos\PASSWORD MANAGER\PASSWORD_MANAGER_DNIe\Pm> python Main.py lock
🧹 DB completamente bloqueada (memoria + disco).
🔒 DB bloqueada en memoria.
```

Robustez

- Todo es local.
- La clave nunca sale del DNIe.
- Cifrado AES.
- Verificación HMAC.
- Necesario PIN de usuario.

FIN

ESPERAMOS QUE OS HAYA GUSTADO NUESTRO PROYECTO

"*No hay una solución milagrosa con la ciberseguridad, una defensa en capas es la única defensa viable*". – James Scott

¿ALGUNA PREGUNTA QUE HACERNOS?