Gestor de Contraseñas

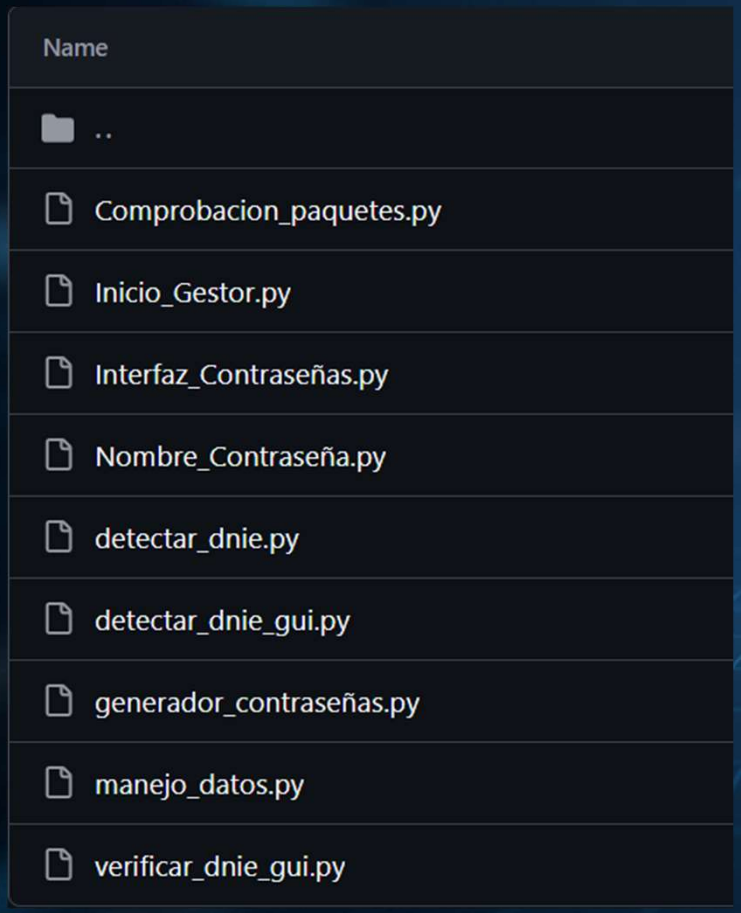
By Rubén Sanz & Enrique Landa

Finalidad y Herramientas

Funcionalidades

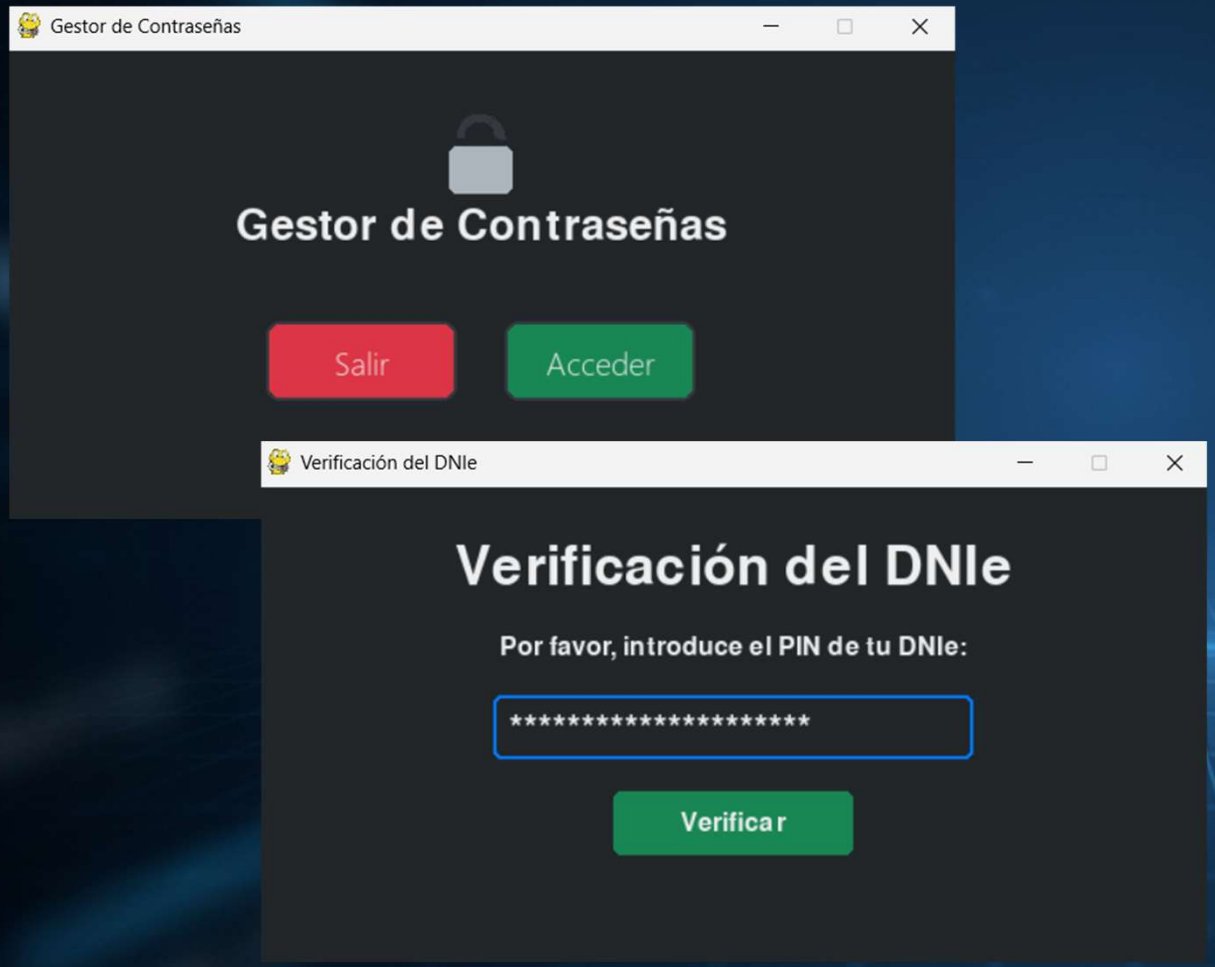
- Desbloqueo/Bloqueo
- Nuevas contraseñas
- Editar/Borrar contraseñas
- Generar contraseñas aleatorias
- Copiar contraseña
- Mostrar contraseña

Diseño



- Interfaz fluida y funcional
- Implementa la funcionalidad de forma visual
- Necesidad de importar paquetes

Interfaz



- Uso de la librería pygame
- Clara y sencilla

DNIe

- Detección con Lector
- Verificación
- Obtención token
- Obtención Certificado
- Firma para cifrado

Detección con Lector

```python
def detectar_dnie():

    #Función que detecta si un lector de tarjetas y un DNIe están conectados.
    try:
        # Obtener la lista de lectores de tarjetas disponibles
        lista_lectores = readers()

        # Si no se detecta ningún lector, muestra un mensaje y termina
        if not lista_lectores:
            return False

        # Obtener el primer lector disponible
        lector = lista_lectores[0]

        # Conectarse al lector
        conexion = lector.createConnection()

        try:
            # Intentar conectarse a la tarjeta (DNIe)
            conexion.connect()

            # Si la conexión es exitosa, se ha detectado un DNIe
            return True

        except Exception as e:
            return False

    except Exception as e:
        return False
```

Verificar/Obtener Token

```python
# Funcion para obtener el token del DNIe
def obtener_token(self):
    pkcs11 = pkcs11_lib(self.PKCS11_LIB)
    slots = pkcs11.get_slots(token_present=True)
    if not slots:
        raise RuntimeError("No se encontró token DNIe.")
    return slots[self.SLOT_INDEX].get_token()

# Funcion de verificacion del pin del DNIe
def verificar_dnie(self, pin):
    try:
        token = self.obtener_token()
        with token.open(user_pin=pin): # Verifica que el pin e
            return True
    except Exception: # Excepcion si no encuentra el token o e
        return False
```

Obtener Certificado (Nombre DataBase)

```python
# Funcion para obtener el certificado de autenticacion del DNIe
def obtener_certificado_autenticacion(self):
    with self.token.open(rw=True) as session: # Con el token abierto con permisos de lectura (No
        certificados = list(session.get_objects({Attribute.CLASS: ObjectClass.CERTIFICATE})) # A
        if not certificados:
            raise RuntimeError("No se encontró certificado en el DNIe.")
        der = certificados[0][Attribute.VALUE] # Obtiene el valor del certificado
        return x509.load_der_x509_certificate(der) # Devuelve el certificado, gracias a la libre

# Funcion para calcular el hash del numero de serie del DNIe
def obtener_hash_serial(self) -> str:
    serial = str(self.cert.serial_number).encode('utf-8') # Obtener el número de serie(utf-8 dev
    h = sha256(serial).hexdigest()[:16] # Calculamos el hash del número de serie y como este es
    return h
```

Firma con DNIe

```python
def firmar_con_dni(self, data: bytes) -> bytes:
    with self.token.open(user_pin=self.pin) as session:
        # Busca la clave privada en el DNIe
        keys = list(session.get_objects({Attribute.CLASS: ObjectClass.PRIVATE_KEY}))
        if not keys:
            raise RuntimeError("No se encontró clave privada para firmar en el token.")

        priv = keys[1]

        try:
            # Se intenta firmar ÚNICAMENTE con el mecanismo seguro especificado.
            signature = priv.sign(data, mechanism=Mechanism.SHA256_RSA_PKCS)
            return signature
        except Exception as e:
            # Si la firma con SHA256 falla por cualquier motivo, lanzamos un error claro.
            # Ya no se intenta un segundo método inseguro.
            raise RuntimeError(
                "El DNIe no pudo firmar con el mecanismo de seguridad requerido (SHA256_RS
                f"Asegúrate de que los drivers son correctos y el DNIe es compatible. Erro
            ) from e
```

Seguridad

- No guardar variables críticas (PIN) en variables globales(Instanciación)
- Cifrado con DNIe
- Varias capas de cifrado en base de datos, sin guardar en variable

Instanciación

```python
class manejo_datos:

    PKCS11_LIB = r"C:\Program Files\OpenSC Project\OpenSC\pkcs11\opensc-pkcs11.d
    SLOT_INDEX = 0

    AES_KEY_SIZE = 32  # 256 bits
    C_FILENAME = "C_value.bin"
    # Constructor de manejo de datos, solo necesita el pin, self represen al p
    def __init__(self, pin: str):
        self.pin = pin # Obtiene el valor del pin(En verificar dnie)
        self.token = self.obtener_token() # Obtiene el token del dni, el token e
        self.cert = self.obtener_certificado_autenticacion() # Obtiene el certif
        self.serial_hash = self.obtener_hash_serial()  # hash del número de seri
        # archivos:
        self.archivo_kdb = os.path.join(os.path.dirname(__file__), f"kdb_enc_{se
        self.archivo_bd = os.path.join(os.path.dirname(__file__), f"Database_{se
        self.archivo_C = os.path.join(os.path.dirname(__file__), self.C_FILENAME
        self.k_db_cache = None # Guarda la k_db cuando es necesario para no tene
        self.inicializar_C() # Crea C si no existía anteriormente
        self.inicializar_kdb() # Crea kdb si no existia anteriormente para ese usuario.
```

```python
def verify_dnie_thread_func(pin):

    # Esta función se ejecuta en un hilo separado para no bloquear la GUI
    global verification_result
    try:
        ini = md.manejo_datos(pin) # Aqui se inicializa la instancia de m
        if ini.verificar_dnie(pin):
            verification_result = {'status': 'success', 'instance': ini}
        else:
            verification_result = {'status': 'fail'}
    except Exception as e:
        print(f"Error en el hilo de verificación: {e}")
        verification_result = {'status': 'error'}
```

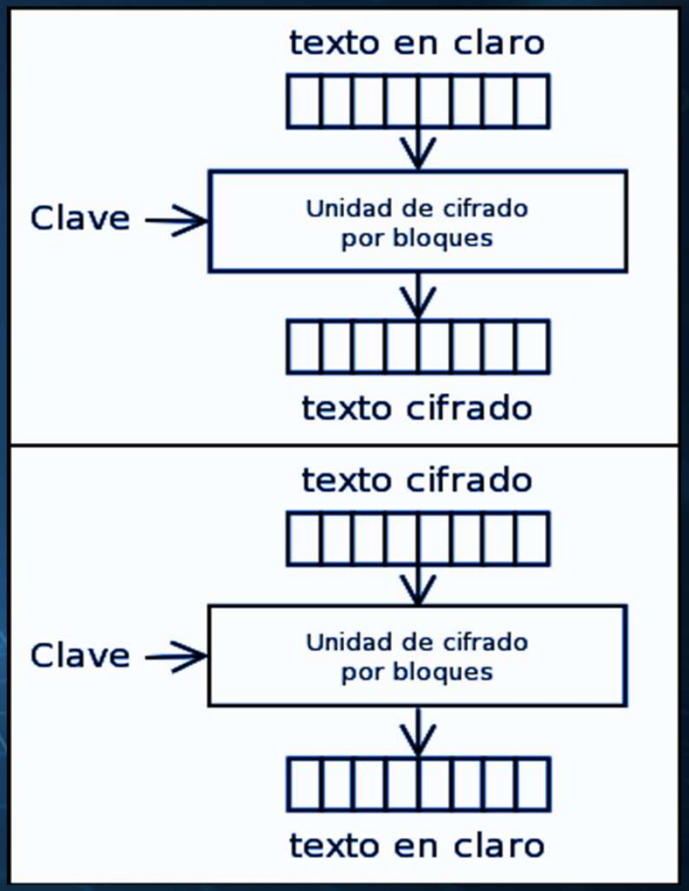
```python
dnie_instance = verification_result.get('instance')
```

```python
pygame.quit()
if dnie_instance:
    ic.interfaz_contrasenas(dnie_instance)
sys.exit()
```

Implementación del cifrado de datos con DNIe

1. Generamos C (Si no existe)
2. Firmamos C -> (S)
3. Obtenemos K -> Hash(S)
4. Generación K_db
5. Cifrado/Descifrado K_db
6. Encriptar/Desencriptar Base de datos

SHA-256, AES-GCM



Inicialización de variables

```python
def inicializar_C(self):
    if os.path.exists(self.archivo_C):
        return
    C = os.urandom(8)  # 64 bits (8 bytes)
    with open(self.archivo_C, "wb") as f:
        f.write(C)

# Funcion para leer C si ya existe
def leer_C(self) -> bytes: # bytes hace que la función devuelva la
    with open(self.archivo_C, "rb") as f:
        data = f.read()
    if len(data) != 8:
        raise RuntimeError("Valor C inválido (longitud incorrecta)
    return data
```

Se guarda sin cifrar

Se inicializa y se guarda cifrada

```python
def inicializar_kdb(self):
    if os.path.exists(self.archivo_kdb):
        return
    k_db = os.urandom(self.AES_KEY_SIZE) # Creamos k_db con
    C = self.leer_C() # Obtenemos C del documento
    S = self.firmar_con_dni(C) # Firmamos C con la clave pr
    K = sha256(S).digest() # Hacemos un hash de S, es decir
    aesgcm = AESGCM(K) # Utiliza el algoritmo AES GCM para
    nonce = os.urandom(12) # Crea un número aleatorio para
    ct = aesgcm.encrypt(nonce, k_db, associated_data=None)
    with open(self.archivo_kdb, "wb") as f:
        f.write(nonce + ct) # Escribimos la clave más el no
    self.k_db_cache = k_db # Guardamos la clave en cache pa
```

Descifrado K_db

```python
def descifrar_kdb(self) -> bytes:
    if self.k_db_cache is not None:
        return self.k_db_cache # Si el cache no esta vacío(No es la primera v
    if not os.path.exists(self.archivo_kdb):
        raise RuntimeError("No existe la clave k_db cifrada para este DNI.")
    with open(self.archivo_kdb, "rb") as f:
        contenido = f.read()
    nonce = contenido[:12] # Leemos el nonce del documento
    ct = contenido[12:] # Leemos el contendio cifrado
    C = self.leer_C()
    S = self.firmar_con_dni(C)
    K = sha256(S).digest()
    aesgcm = AESGCM(K)
    k_db = aesgcm.decrypt(nonce, ct, associated_data=None) # Desencriptado sc
    self.k_db_cache = k_db # Guarda en cache (La primera vez que entramos)
    return k_db
```

Cifrado/Descifrado Base de datos

```python
def cargar_bd(self):
    k_db = self.descifrar_kdb()
    if not os.path.exists(self.archivo_bd):
        return {"Contrasenas": []}
    with open(self.archivo_bd, "rb") as f:
        contenido = f.read()
    nonce = contenido[:12]
    ct = contenido[12:]
    aesgcm = AESGCM(k_db)
    datos_bytes = aesgcm.decrypt(nonce, ct, associated_data=None)
    return json.loads(datos_bytes.decode("utf-8"))
```

```python
def guardar_bd(self, db):
    k_db = self.descifrar_kdb()
    aesgcm = AESGCM(k_db)
    nonce = os.urandom(12)
    ct = aesgcm.encrypt(nonce, json.dumps(db, indent=4).encode("utf-8"), associated_data=None)
    with open(self.archivo_bd, "wb") as f:
        f.write(nonce + ct)
```

Limitaciones y Soluciones

- Caducidad DNIe
- Volcado de memoria (PIN)
- Pérdida de C, o otros archivos
- Portapapeles permanentes
- Migración de contraseñas
- Compromiso comodidad-Seguridad
- Backup
- Aviso usuario

DEMO