Gestor de Contraseñas

Proyecto Ciber 1

Seguridad, cifrado y control total de tus credenciales

Francisco Bosqued Lahoz y Daniel Pérez Mancebón

El Problema: Contraseñas en Riesgo

Contraseñas débiles

Fácilmente predecibles o reutilizadas

Ataques dirigidos

Fuerza bruta y phishing masivo

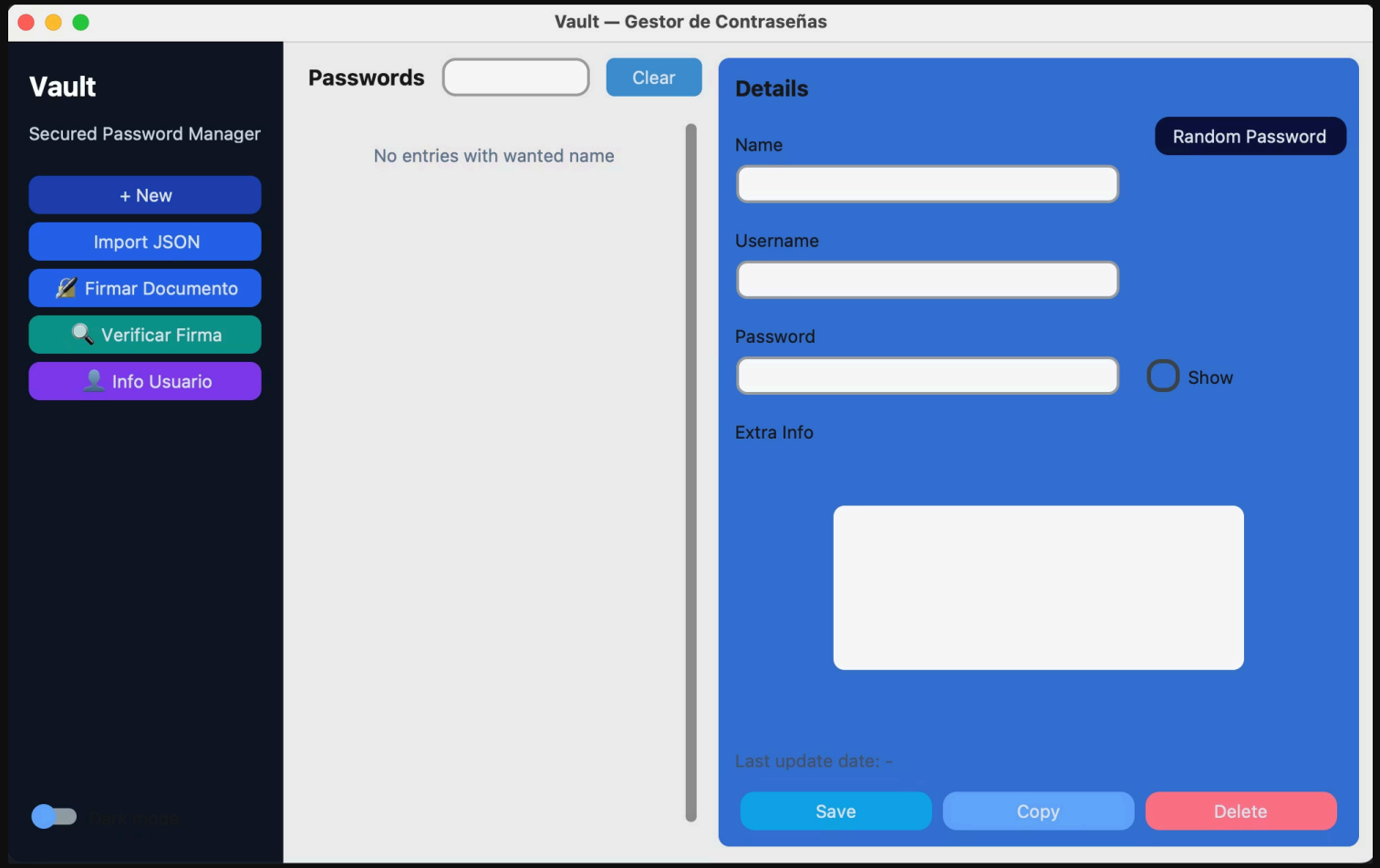
Brechas de datos

Acceso no autorizado a credenciales

Características Principales

1. Cifrado Fernet (AES-128)
2. Autenticación con DNIe + OTP (2FA)
3. Protección contra ataques
4. Firma y verificación 2 en 1

INTERFAZ Y DEMO DEL PROYECTO



Crypto Manager

```python
def initialize_with_pin(self, pin: str) -> bool:
    """Inicializar con PIN y mantener sesión abierta"""
    try:
        if self.multi_user:
            user_id = self.get_user_id_from_dnie(pin)
            if not user_id:
                return False

            certificate = self.dnie_manager.get_certificate()
            if not certificate:
                return False

            key = self._derive_key_from_certificate(certificate)
            self.fernet = Fernet(key)
        else:
            self.dnie_manager = DNIeManager()
            key = self.dnie_manager.authenticate(pin)
            self.fernet = Fernet(key)

        self.authenticated = True
        return True

    except Exception as e:
        print(f"❌ Error de autenticación DNIe: {e}")
        return False
```

```python
def get_user_id_from_dnie(self, pin: str) -> str:
    """Obtener ID único del usuario basado en el certificado del DNIe"""
    try:
        self.dnie_manager = DNIeManager()
        self.dnie_manager.authenticate(pin)
        certificate = self.dnie_manager.get_certificate()

        if not certificate:
            raise Exception("No se pudo obtener el certificado del DNIe")

        self.user_id = hashlib.sha256(certificate).hexdigest()[:32]

        if self.multi_user:
            user_vault_dir = os.path.join(self.vaults_dir, f"vault_dnie_{self.user_id}")
            os.makedirs(user_vault_dir, exist_ok=True)
            self.db_file = os.path.join(user_vault_dir, "passwords.db.enc")

        return self.user_id

    except Exception as e:
        raise Exception(f"Error obteniendo ID del DNIe: {str(e)}")
```

```python
def _derive_key_from_certificate(self, certificate: bytes) -> bytes:
    """Derivar clave Fernet del certificado del DNIe"""
    derived = hashlib.pbkdf2_hmac(
        'sha256',
        certificate,
        b'dnie_vault_salt',
        100000,
        32
    )
    return base64.urlsafe_b64encode(derived)
```

```python
def save_db(self, db_dict: dict):
    """Guardar base de datos (requiere autenticación previa)"""
    if not self.authenticated or not self.fernet:
        raise Exception("No autenticado. Llame a initialize_with_pin primero.")
    if not self.db_file:
        raise Exception("No se ha configurado archivo de base de datos")

    plaintext = json.dumps(db_dict).encode()
    ciphertext = self.fernet.encrypt(plaintext)
    with open(self.db_file, 'wb') as f:
        f.write(ciphertext)

def load_db(self) -> dict:
    """Cargar base de datos (requiere autenticación previa)"""
    if not self.authenticated or not self.fernet:
        raise Exception("No autenticado. Llame a initialize_with_pin primero.")

    try:
        with open(self.db_file, 'rb') as f:
            ciphertext = f.read()
        plaintext = self.fernet.decrypt(ciphertext)
        return json.loads(plaintext.decode())
    except FileNotFoundError:
        return {"entries": []}
```

Dnie Manager

```python
def authenticate(self, pin: str) -> bytes:
    """Authenticate with DNIe and return derived key"""
    try:
        print(f"🔍 Buscando DNIe con {self.pkcs11_lib}...")

        if self.pkcs11_lib == "pkcs11":
            # Windows/Linux con python-pkcs11
            self._lib = pkcs11.lib(self.lib_path)
            slots = self._lib.get_slots(token_present=True)

            if not slots:
                raise Exception(f"❌ No se detectó ningún DNIe.")

            token = slots[0].get_token()
            print("✅ DNIe detectado, iniciando autenticación...")

            self.session = token.open(user_pin=pin)

            # Buscar clave privada para firmar
            priv_key = self._find_private_key()

            # Create and sign challenge
            challenge = os.urandom(32)
            signature = priv_key.sign(challenge, mechanism=Mechanism.SHA256_RSA_PKCS)
```

```python
def sign_file(self, file_path: str, pin: str) -> dict:
    """Firmar un archivo y retornar paquete de firma"""
    if not Path(file_path).exists():
        raise FileNotFoundError(f"Archivo no encontrado: {file_path}")
    if not self.session:
        # En caso de que se use la función stand-alone
        self.authenticate(pin)

    # Calcular hash del archivo
    file_hash = self._calculate_file_hash(file_path)

    # Leer contenido del archivo
    with open(file_path, 'rb') as f:
        file_data = f.read()

    # Firmar los datos
    signature = self.sign_data(file_data)

    # Obtener certificado
    certificate = self.get_certificate()

    if not certificate:
        raise Exception("No se pudo obtener el certificado del DNIe")
    # Crear paquete de firma
    signature_package = {
        'file_path': str(Path(file_path).absolute()),
        'file_name': Path(file_path).name,
        'file_hash': file_hash,
        'hash_algorithm': 'sha256',
        'signature': base64.b64encode(signature).decode('utf-8'),
        'certificate': base64.b64encode(certificate).decode('utf-8'),
        'timestamp': self._get_timestamp()
    }
    return signature_package
```

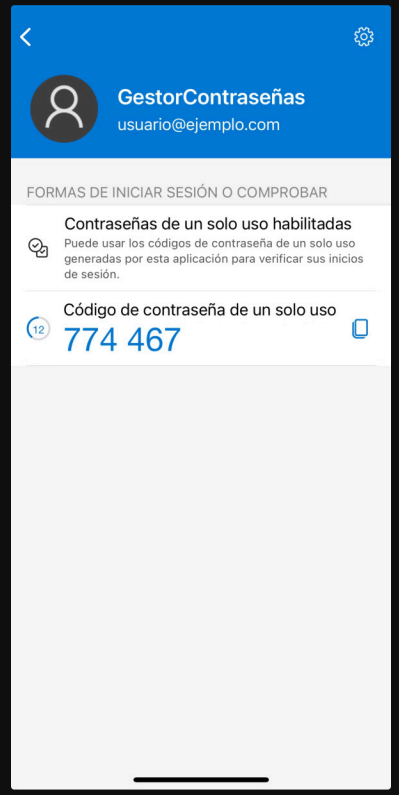
```python
def _find_private_key(self):
    """Encontrar clave privada para python-pkcs11 (Windows/Linux)"""
    keys = self.session.get_objects({
        pkcs11.Attribute.CLASS: ObjectClass.PRIVATE_KEY,
        pkcs11.Attribute.SIGN: True
    })

    # Intentar encontrar clave especifica
    for key in keys:
        try:
            label = key[pkcs11.Attribute.LABEL] if hasattr(key, '__getitem__') else None
            if label and any(auth_word in label.lower() for auth_word in ['autenticacion', 'auth', 'firma']):
                return key
        except:
            continue

    # Si no encontramos clave específica, usar la primera
    keys = list(self.session.get_objects({
        pkcs11.Attribute.CLASS: ObjectClass.PRIVATE_KEY,
        pkcs11.Attribute.SIGN: True
    }))
    if keys:
        return keys[0]

    raise Exception("No se encontró ninguna clave privada de firma en el DNIe")
```

```python
def verify_signature(self, file_path: str, signature_path: str) -> bool:
    """Verificar firma de un archivo"""
    if not Path(file_path).exists():
        raise FileNotFoundError(f"Archivo no encontrado: {file_path}")
    if not Path(signature_path).exists():
        raise FileNotFoundError(f"Archivo de firma no encontrado: {signature_path}")
    try:
        # Cargar paquete de firma
        with open(signature_path, 'r') as f:
            signature_package = json.load(f)

        # Verificar integridad del archivo
        current_hash = self._calculate_file_hash(file_path)
        if current_hash != signature_package['file_hash']:
            print("❌ El archivo ha sido modificado desde la firma!")
            return False
        # Decodificar firma y certificado
        signature = base64.b64decode(signature_package['signature'])
        certificate_data = base64.b64decode(signature_package['certificate'])
        # Cargar certificado
        cert = x509.load_der_x509_certificate(certificate_data)
        public_key = cert.public_key()
        # Leer archivo para verificación
        with open(file_path, 'rb') as f:
            file_data = f.read()
        # Verificar firma
        public_key.verify(
            signature,
            file_data,
            padding.PKCS1v15(),
            hashes.SHA256()
        )
        return True
```

Interfaz intuitiva, segura y completamente cifrada. Acceso rápido a credenciales con máxima protección.

OTP (Google Auth - 2FA)

```python
def _load_or_generate_secret():
    """Carga el secreto desde SECRET_FILE o genera uno nuevo y lo escribe.
    Devuelve (secret, newly_created_bool).
    """
    if os.path.exists(SECRET_FILE):
        try:
            with open(SECRET_FILE, "r", encoding="utf-8") as f:
                secret = f.read().strip()
            if secret:
                return secret, False
        except Exception:
            pass  # si falla la lectura, generamos nuevo
    # generar nuevo secreto y persistirlo
    secret = pyotp.random_base32()
    try:
        with open(SECRET_FILE, "w", encoding="utf-8") as f:
            f.write(secret)
    except Exception as e:
        # Si no se puede escribir, aún devolvemos el secreto generado (no persistido)
        print("Warning: no se pudo guardar SECRET_FILE:", e)
    return secret, True

def _get_totp_from_secret(secret):
    return pyotp.TOTP(secret)
```



Mejoras Futuras

Sincronización en nube

Acceso seguro desde múltiples dispositivos

Interfaz web y móvil

Plataforma multiplataforma con navegador

Autenticación biométrica

Desbloqueo con huella y reconocimiento facial

Auditoría avanzada

Logs detallados de acceso y cambios

Esperamos que os haya gustado !

Autores del Proyecto

Francisco Bosqued Lahoz

Daniel Pérez Mancebón

Información

Proyecto Ciber 1

2025

Gestión segura de credenciales mediante criptografía de confianza