UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA

CARRERA DE CIENCIAS DE LA COMPUTACIÓN

Administrador de Contraseñas Seguro con AES-GCM y PBKDF2

AUTORES

Chancan Chanca, Sanders

PROFESOR

Pazos Ortiz, Jose Carlos

Lima - Perú

2025

TABLA DE CONTENIDO

Pág.

I

Introducción

El manejo seguro de contraseñas constituye un componente fundamental de la seguridad digital contemporánea. Diversos estudios establecen que más del 60 % de usuarios reutilizan contraseñas en múltiples servicios, lo cual permite que un único compromiso afecte una gran cantidad de cuentas sensibles ([1]). La existencia de ataques como *credential stuffing*, campañas de phishing automatizado y fugas masivas de datos refuerzan la necesidad de mitigar la dependencia del usuario respecto a prácticas inseguras.

En respuesta a estos riesgos, se desarrolla un **administrador de contraseñas seguro**, como parte del proyecto del curso de Criptografía. El código fuente completo de la solución se encuentra disponible públicamente en:

https://github.com/Zanderz17/-Criptograf-a-Proyecto/tree/main

El sistema implementado busca garantizar que:

- La confidencialidad de las credenciales se preserve incluso ante un compromiso del servidor.
- El usuario mantenga control exclusivo sobre el acceso a su información.
- La experiencia de uso resulte fluida y compatible con operación sin conexión.

Para ello, se aplica un enfoque *zero-knowledge backend*, donde tanto la Master Password como la clave criptográfica derivada **nunca abandonan el navegador**.

Todo el cifrado y descifrado ocurre localmente mediante WebCrypto API, lo que reduce significativamente la superficie de ataque y evita exponer secretos a terceros, incluido el propio backend. Este modelo fortalece la seguridad en escenarios adversos y promueve buenas prácticas en la protección de la identidad digital.

II

Arquitectura General del Sistema

La arquitectura del sistema (Fig. 2.1) refleja una separación estricta de responsabilidades:

1. **Frontend Web**: ejecuta todo el proceso criptográfico usando la WebCrypto API, asegurando que el backend jamás tenga acceso a información en texto plano.
2. **Backend REST**: autentica usuarios y almacena únicamente blobs cifrados junto a metadatos mínimos.
3. **Base de Datos PostgreSQL**: mantiene la persistencia del vault sin exponer contenido del usuario.

Adicionalmente, la aplicación implementa un enfoque *offline-first*, utilizando IndexedDB para permitir acceso al vault incluso sin conectividad, mejorando resiliencia y disponibilidad.

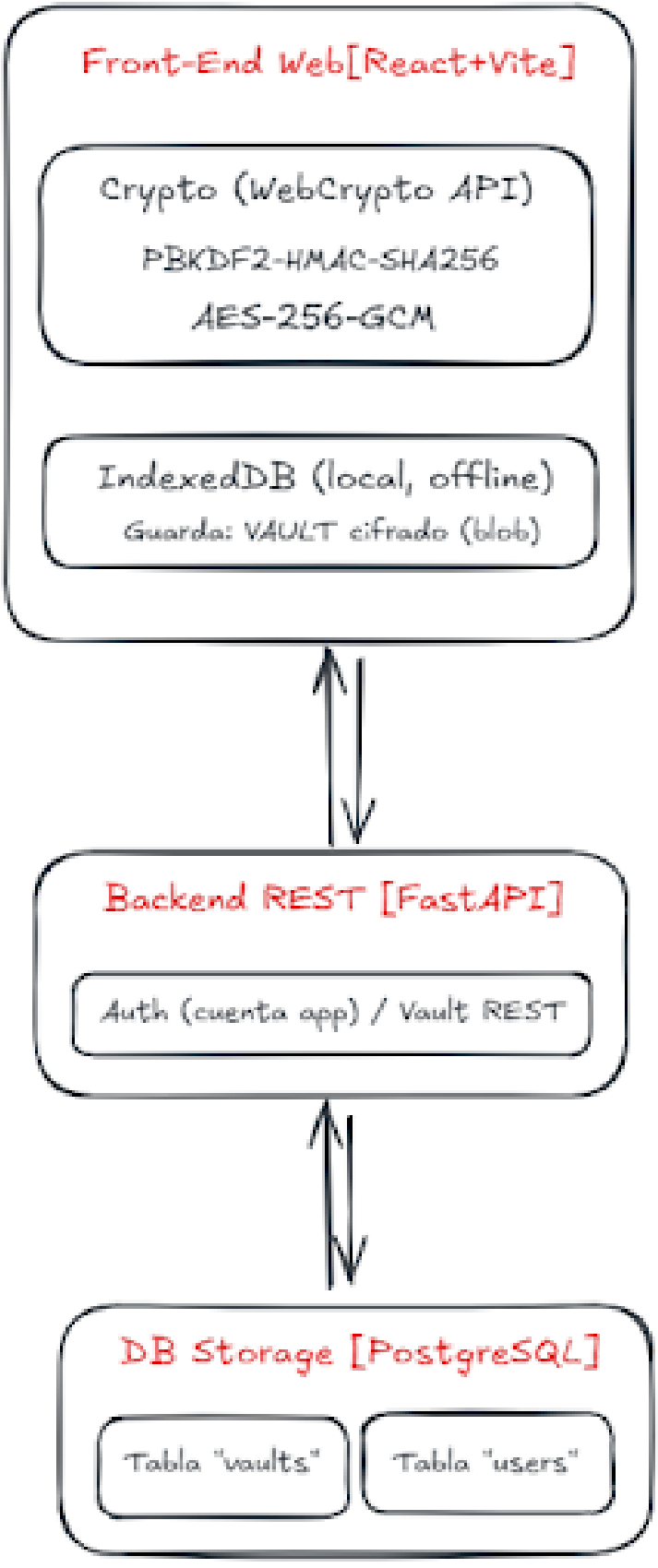

FIGURA 2.1: Arquitectura general del sistema, destacando su enfoque zero-knowledge.

III

Diseño Criptográfico

La seguridad del sistema se basa en tres pilares: **derivación de clave segura**, **cifrado autenticado** y **control de versiones seguro**.

Derivación de Clave

Dado que la Master Password puede ser débil desde el punto de vista criptográfico, se transforma en una clave segura de 256 bits mediante:

- PBKDF2-HMAC-SHA256 [2]
- 300 000 iteraciones (valor ajustado para navegadores modernos)
- Salt único de 128 bits generado con CSPRNG

Este proceso incrementa el costo computacional de ataques de fuerza bruta tanto en cliente como servidor. El salt evita ataques mediante tablas precalculadas o reutilización de hashes.

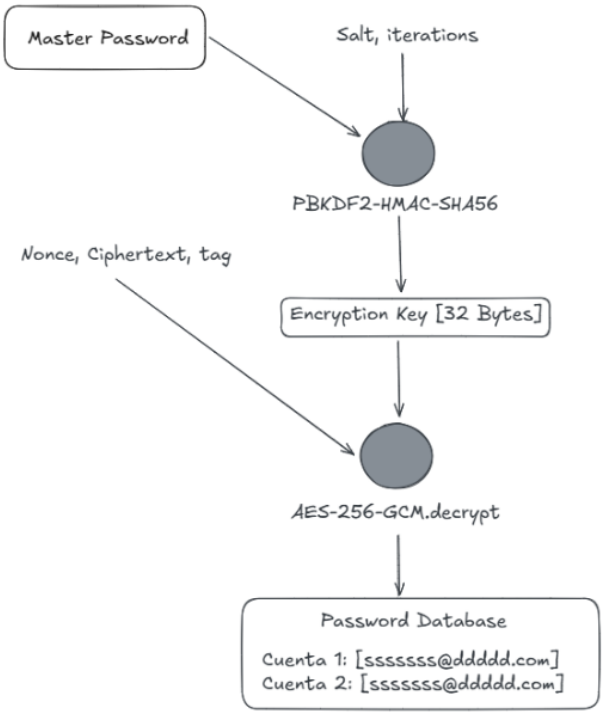

FIGURA 3.1: Derivación de clave con PBKDF2 en el cliente.

Cifrado Autenticado

El vault completo se cifra usando:

- **AES-256-GCM**: proporciona confidencialidad e integridad ([3])
- Nonce aleatorio de 96 bits por operación de cifrado
- Tag de autenticación de 128 bits que evita manipulación del blob

El encabezado del vault se incluye como **AAD**, vinculado criptográficamente al contenido.

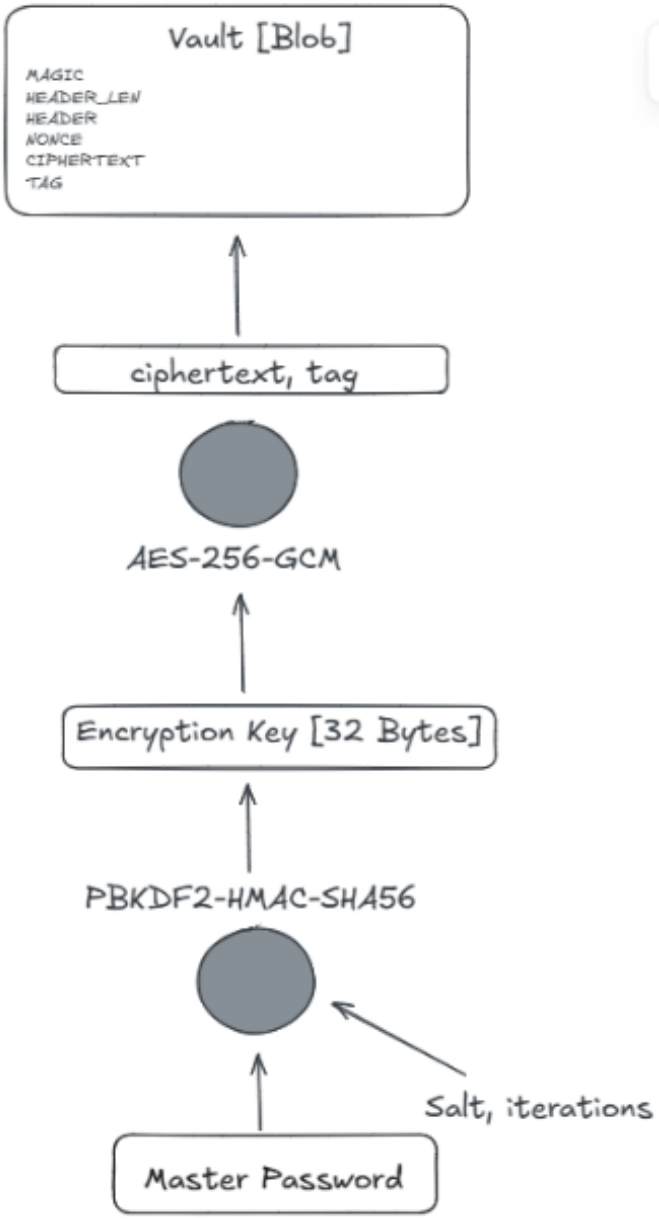

FIGURA 3.2: Formato cifrado del Vault almacenado en DB e IndexedDB.

Este esquema garantiza que ninguna modificación del vault pase desapercibida, incluso si el atacante controla el servidor.

IV

Implementación

Frontend

Se desarrolló la aplicación usando:

- React + Vite para un flujo moderno de desarrollo.
- Zustand para gestión segura del estado y control de sesión.
- IndexedDB como caché persistente y soporte *offline*.
- WebCrypto API para KDF y operaciones AES-GCM nativas del navegador.

El cifrado y descifrado se realizan exclusivamente en cliente, cumpliendo con el modelo de privacidad extremo a extremo.

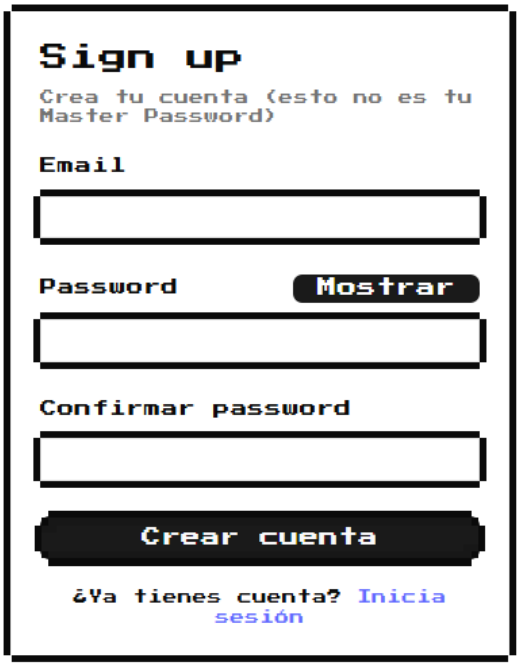

FIGURA 4.1: Interfaz de inicio de sesión con validación criptográfica local.

Backend

El backend incluye:

- FastAPI para una API confiable y validación de datos
- PostgreSQL para almacenamiento persistente
- **JWT** como esquema de autenticación
- **ETag** + `If-Match` como control optimista de concurrencia

El siguiente fragmento ejemplifica el control de versiones en la actualización del vault:

Listing IV.1: Control de ETag en escritura del vault

```
if row:
    if not if_match or if_match != row.etag:
        raise HTTPException(409, "etag␣mismatch")
    row.version += 1
    row.blob = body
    row.etag = make_etag(user_id, row.version, body)
    db.commit()
```

Este mecanismo previene sobrescritura accidental cuando múltiples dispositivos actualizan simultáneamente.

V

Seguridad Práctica

Cada componente se diseñó acorde al modelo de amenazas definido:

- **Zero-Knowledge**: el servidor jamás conoce la Master Password ni la clave derivada.
- **End-to-end encrypted**: cifrado desde el cliente hasta su almacenamiento.
- **Protección en tránsito**: uso de HTTPS para evitar ataques de intercepción.
- **Protección en reposo**: cifrado del blob tanto en BD como en IndexedDB.
- **Autenticidad de datos**: validación mediante tags de GCM.

Incluso con acceso root al servidor, el atacante solo obtendría datos cifrados sin utilidad práctica.

VI

Resultados, Fallas y Limitaciones

Resultados

- Vault cifrado funcionando correctamente en pruebas funcionales.
- Sincronización backend con validación de versiones mediante ETag.

Adicionalmente, se validó que sin la Master Password correcta es imposible descifrar el contenido, cumpliendo la privacidad esperada.

Limitaciones

- PBKDF2 tiene un nivel de protección inferior frente a hardware especializado en comparación con Argon2id.
- La sincronización multi-dispositivo requiere mayor manejo de concurrencia y resolución de conflictos.
- No se evaluaron ataques relacionados con canales laterales o tiempos de ejecución.

VII

Conclusiones

Este proyecto permitió aplicar de manera integrada múltiples aspectos de la criptografía moderna en un sistema real: derivación segura de claves, cifrado autenticado, modelos de almacenamiento resistente a ataques y protocolos de autenticación. El desarrollo evidenció en la práctica la relevancia de conceptos como:

- Selección correcta de primitivas y parámetros.
- Diseño seguro desde la arquitectura (*security-by-design*).
- Minimización del riesgo mediante modelos Zero-Knowledge.

Se comprobó que una implementación adecuada puede mantener la confidencialidad incluso bajo escenarios adversos donde un atacante controla el servidor. Así, el usuario conserva el dominio total sobre sus credenciales, cumpliendo plenamente con los objetivos del curso y fortaleciendo las bases para futuros desarrollos en seguridad aplicada.

REFERENCIAS BIBLIOGRÁFICAS

[1] J. Bonneau *et al.*, "The science of guessing: Analyzing an anonymized corpus of 70 million passwords," *IEEE Symposium on Security and Privacy*, 2012, importancia de contraseñas fuertes y problemas de reutilización.

[2] B. Kaliski, *PKCS#5: Password-Based Cryptography Specification.* RSA Laboratories, 2000, especificación del estándar PBKDF2.

[3] M. Dworkin, "Recommendation for block cipher modes of operation: Galois/counter mode (gcm) and gmac," National Institute of Standards and Technology (NIST), Tech. Rep. SP 800-38D, 2007. [Online]. Available: https://csrc.nist.gov/publications/detail/sp/800-38d/final