DESCRIPCIÓN TÉCNICA

Whitepaper de Arquitectura Criptográfica: MPLock & mplock-core

Análisis técnico detallado, modelo de amenazas, derivación de claves bajo demanda y trazabilidad técnica con el código fuente abierto de `mplock-core`.

VERSIÓN DE DOCUMENTO
v1.5 (14 de Octubre de 2026)

VERSIÓN SOFTWARE CUBIERTA
MPLock Desktop v1.4.0 (Windows x64)

MOTOR ABIERTO
`mplock-core` v0.1.1 en `main`

LICENCIA DEL CORE
MIT / Apache-2.0 Dual

---

01. Resumen Ejecutivo (ES / EN)
02. Alcance y Modelo de Amenazas
03. Arquitectura de la Bóveda (DEK Dual)
04. Aleatoriedad y Muestreo sin Sesgo
05. Gestión de Secretos en Memoria
06. Motor 2FA / TOTP (RFC 6238)
07. Licenciamiento Offline (Ed25519)
08. Extensión y Host Nativo
09. Delimitación de Código Abierto
10. Pruebas Automatizadas y Reproducibilidad
11. Límites y Estado de Auditoría
12. Historial y Matriz de Trazabilidad

01. Resumen Ejecutivo / Executive Summary

Resumen Ejecutivo (Español)

MPLock es un gestor de credenciales y autenticador de doble factor (2FA/TOTP) concebido para Microsoft Windows bajo el principio de control soberano de los datos y procesamiento local (Local-First). A diferencia de las plataformas dependientes de nubes centralizadas, en MPLock la base de datos reside de forma exclusiva en el almacenamiento local del equipo del usuario y no es transferida a servidores remotos ni sincronizada en la nube.

El núcleo criptográfico subyacente, implementado en la biblioteca abierta `mplock-core` desarrollada en Rust, implementa un esquema de clave de sobre cerrado (*envelope encryption*) con una Clave de Cifrado de Datos (DEK, *Data Encryption Key*) de 256 bits generada con el generador criptográfico del sistema operativo (`rand::rngs::OsRng`). El archivo de credenciales se cifra con AES-256-GCM (nonce aleatorio de 96 bits con probabilidad de colisión estadísticamente despreciable), mientras que la DEK se almacena protegida independientemente por dos factores: la contraseña maestra y un código de recuperación alfanumérico legible de 24 caracteres (alfabeto Base32 de 32 símbolos sin caracteres ambiguos (O, 0, I, 1), equivalente a 120 bits de entropía). Ambas envolturas utilizan la función de derivación PBKDF2-HMAC-SHA256 con 600.000 iteraciones y sales aleatorias de 16 bytes.

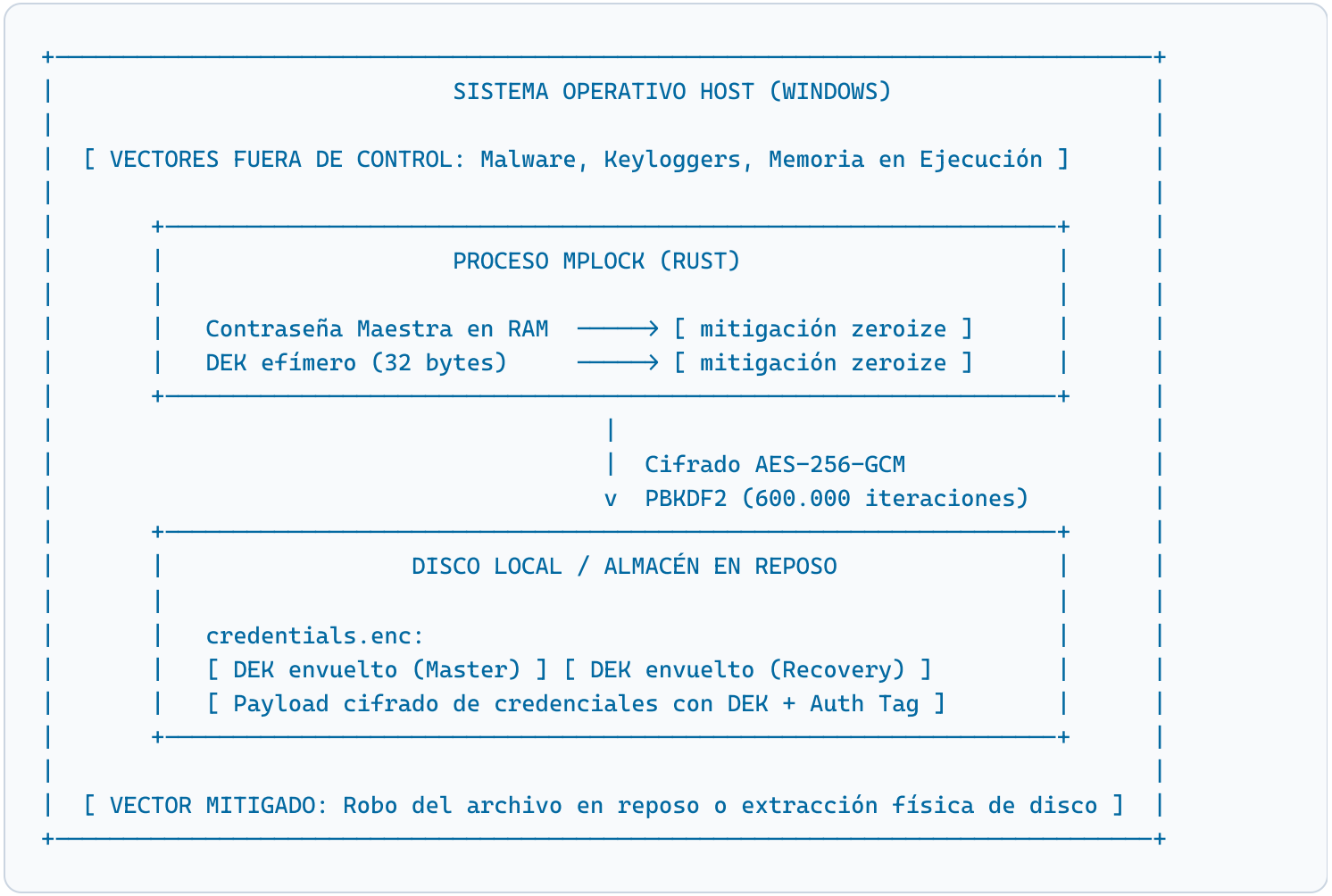
Executive Summary (English)

MPLock is a local-first password manager and time-based one-time password (TOTP/2FA) authenticator engineered for Microsoft Windows to ensure sovereign local data control. Unlike cloud-dependent alternatives, user credentials stored in MPLock reside exclusively within the local file system and are never synchronized or transmitted to remote infrastructure.

The core cryptographic engine, encapsulated in the standalone, open-source Rust library `mplock-core`, employs an envelope encryption model anchored by a 256-bit Data Encryption Key (DEK) generated via the operating system CSPRNG (`OsRng`). Vault payloads are encrypted using authenticated AES-256-GCM with a 96-bit random nonce. The DEK itself is wrapped twice independently: once under the master password and once under a 24-character human-readable recovery code (Base32 alphabet of 32 symbols without ambiguous characters (O, 0, I, 1), providing 120 bits of entropy). Both key wrapping operations utilize PBKDF2-HMAC-SHA256 with 600,000 iterations and distinct 16-byte random salts.

02. Alcance y Modelo de Amenazas

Un diseño criptográfico riguroso debe definir con precisión la frontera entre las amenazas que neutraliza y los vectores que quedan fuera del alcance del software. La aplicación de escritorio MPLock está diseñada y empaquetada exclusivamente para Microsoft Windows (x64).



2.1 Qué Protege MPLock

- **Robo del archivo de bóveda en reposo (`credentials.enc`).** Si un atacante sustrae el disco, copia la carpeta de datos locales de la aplicación (`%APPDATA%/com.passwords.app/credentials.enc`) o accede a una copia de seguridad sin autorización, los datos permanecen computacionalmente inviables de descifrar sin una contraseña maestra de alta entropía o el código de recuperación.
- **Manipulación e inyección de datos.** El modo AES-256-GCM verifica un tag de autenticación de 128 bits. Cualquier alteración accidental o deliberada en los bytes del archivo resulta en un fallo de autenticación inmediato antes del procesamiento.
- **Coste computacional ante fuerza bruta.** El cómputo de 600.000 rondas de PBKDF2-HMAC-SHA256 impone una barrera de coste temporal frente a ataques de fuerza bruta basados en CPU.

2.2 Lo que MPLock NO Protege (Límites Explícitos de Seguridad)

Ningún gestor de contraseñas de software puede sustituir las garantías fundamentales de un sistema operativo seguro. MPLock declara con honestidad que en los siguientes escenarios:

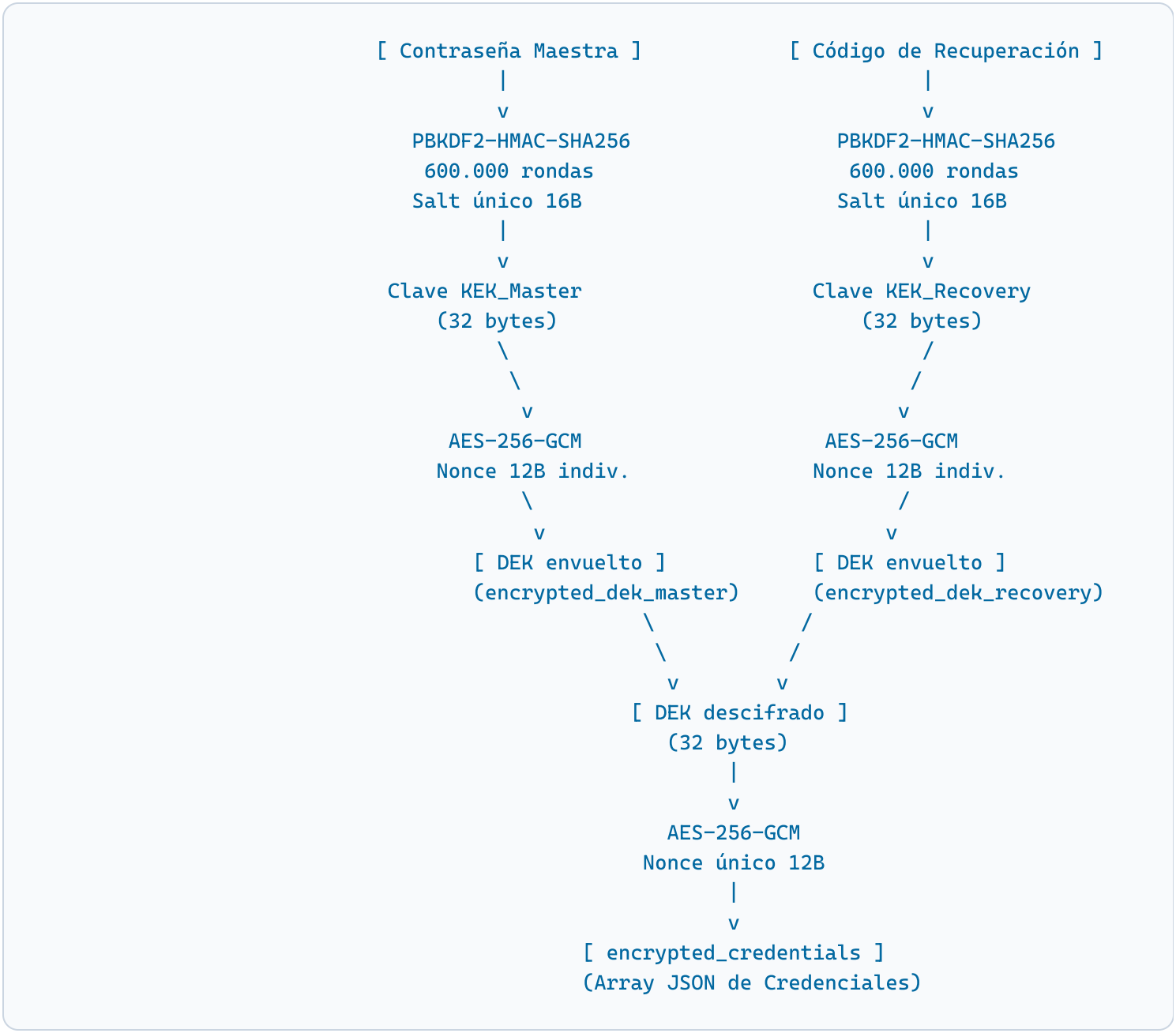
- Si el equipo tiene software espía que captura teclado (APIs de teclado en el SO) o pantalla, la contraseña maestra puede ser sustraída en el momento exacto en que se ingresa.
- En Windows, cualquier proceso que se ejecute bajo la misma sesión de usuario interactiva puede, en general, abrir el proceso de MPLock mediante `OpenProcess(PROCESS_VM_READ)` e inspeccionar su espacio de direcciones sin requerir privilegios de Administrador local ni elevación UAC. Mientras la bóveda permanece desbloqueada, los datos descifrados residen en la memoria RAM del proceso.
- Contraseñas cortas o predecibles anulan la protección de PBKDF2 frente a ataques de diccionario dirigidos.
- Al operar sin servidores ni almacenamiento en la nube, . Si se olvidan la clave maestra y el código de recuperación, los datos se pierden irreversiblemente.
- Al copiar una credencial o secreto, MPLock utiliza las extensiones nativas de Windows en `arboard` (`SetExtWindows::exclude_from_history()` y `exclude_from_cloud()` , `src-tauri/src/lib.rs`), instruyendo al sistema operativo para que no registre el secreto en el Historial del Portapapeles (`Win + V`) ni lo sincronice en la nube de Microsoft. Estas directivas son respetadas por Windows y por gestores de portapapeles bien portados, pero no necesariamente por herramientas de terceros no conformes. Asimismo, un temporizador en segundo plano vacía el portapapeles tras 30 segundos si el contenido aún coincide con el secreto copiado (función pura `should_clear_clipboard`). En un cierre normal de la aplicación, el manejador de eventos del ciclo de vida (`tauri::WindowEvent::CloseRequested / Destroyed`) intercepta el cierre y limpia inmediatamente el portapapeles si todavía contiene el secreto; no obstante, un cierre forzado del proceso (ej. `taskkill` o Administrador de tareas), un apagado repentino del equipo o un fallo imprevisto pueden impedir dicha ejecución y dejar el secreto en el portapapeles del sistema. La limpieza automática a los 30 segundos y la exclusión del historial de Windows (Win+V) se aplican a las copias hechas desde la aplicación de escritorio. El botón de copiar el código TOTP del popup de la extensión del navegador usa el portapapeles del navegador: ese valor no se limpia automáticamente ni se excluye del historial de Windows.
- Cuando las credenciales son enviadas a la extensión web a través de mensajería nativa para autorrellenar un formulario, residen temporalmente en el espacio de memoria del proceso del navegador (motor JavaScript/V8). MPLock no puede forzar un borrado determinista con ceros (`zeroize`) en la memoria RAM del navegador, dado que el recolector de basura de JavaScript maneja cadenas inmutables y el ciclo de vida de la memoria de la extensión escapa al control del binario de escritorio.
- AES-256-GCM comprueba la autenticidad del archivo que se está descifrando, pero . Si un atacante sustituye el archivo de

la bóveda actual por una copia de respaldo legítima anterior, MPLock la descifrará con éxito sin alertar de que los datos han sido revertidos a un estado anterior, al no contar con un contador monótono en hardware (como TPM).

- Almacenar contraseñas y semillas secretas TOTP en la misma base de datos consolida ambos factores en un único vector de riesgo: si la bóveda se ve comprometida, el segundo factor se pierde simultáneamente.
- Binarios locales alterados, DLLs inyectadas o kernels manipulados invalidan el entorno de ejecución.

03. Arquitectura de la Bóveda (DEK Dual)

La persistencia y el empaquetado seguro se gestionan en los módulos `mplock-core/src/crypto.rs` y `mplock-core/src/storage.rs`.



3.1 Esquema de Clave de Sobre (DEK)

Al inicializar la bóveda (`storage::initialize_db` , `storage.rs:89-97`), se genera una clave aleatoria de 32 bytes (256 bits) denominada DEK. Esta clave se codifica en Base64 y se envuelve dos veces de forma independiente:

1. `encrypted_dek_master` : Cifrada con la clave derivada de la contraseña maestra (`storage.rs:102`).
2. `encrypted_dek_recovery` : Cifrada con la clave derivada del código de recuperación (`storage.rs:106`).

Las credenciales de usuario se cifran exclusivamente con la DEK. Este modelo permite que el cambio de contraseña maestra o la regeneración del código de recuperación solo requiera re-cifrar la clave DEK de 32 bytes, sin necesidad de reescribir toda la base de datos de credenciales (`src-tauri/src/lib.rs:408-456, 459-508` — componente cerrado).

3.2 Cifrado Autenticado AES-256-GCM

- **Primitiva:** `aes_gcm::Aes256Gcm` (RustCrypto, `crypto.rs:1-3`).
- **Tamaño de Nonce / IV:** 12 bytes (96 bits) (`crypto.rs:15`, `IV_LEN = 12`).
- **Generación del Nonce:** En cada cifrado se genera un nonce aleatorio de 96 bits mediante `OsRng.fill_bytes(&mut iv)` (`crypto.rs:41-42`). Para el volumen esperado de operaciones de un usuario personal, la probabilidad de colisión aleatoria es estadísticamente despreciable.
- **Tag de Autenticación:** 16 bytes (128 bits) concatenados al texto cifrado, verificados obligatoriamente al descifrar (`crypto.rs:98-100`).

3.3 Derivación de Claves (PBKDF2) y Análisis Comparativo

- **Algoritmo:** PBKDF2 con HMAC-SHA256 (`crypto.rs:6, 10, 21-24`).
- **Iteraciones reales:** `600_000` rondas (`crypto.rs:13`, constante `PBKDF2_ITERATIONS`).
- **Salt criptográfico:** 16 bytes (128 bits) generados con `OsRng` por cada operación (`crypto.rs:14, 33-34`).
- **Limitación Arquitectónica (PBKDF2 frente a Argon2id):** PBKDF2 eleva sustancialmente el coste temporal frente a ataques de fuerza bruta en CPU, pero **no es un algoritmo duro en memoria (*memory-hard*)**. Algoritmos modernos como **Argon2id** o **scrypt** ofrecen una resistencia sensiblemente superior contra adversarios equipados con granjas masivas de GPUs, FPGAs o ASICs dedicados debido a que exigen un uso intensivo de ancho de banda y memoria RAM. La ausencia de Argon2id es una limitación conocida de la versión 1.x.

3.4 Formato del Archivo `credentials.enc`

El archivo en disco contiene un objeto JSON con tres campos (`storage.rs:40-44`):

```
{
  "encrypted_credentials": "<Base64_Payload>",
  "encrypted_dek_master": "<Base64_Payload>",
  "encrypted_dek_recovery": "<Base64_Payload>"
}
```

Cada payload Base64 almacena en binario: `[Salt (16B)] + [Nonce/IV (12B)] + [Ciphertext + Tag (variable)]` (`crypto.rs:27-30, 58-64`).

3.5 Formato del Respaldo Cifrado (`.mplockbackup`)

La exportación de respaldo cifrado (`src-tauri/src/lib.rs`) produce un archivo binario autónomo estructurado en dos regiones: un encabezado canónico de 30 bytes y un payload cifrado con el motor criptográfico de `mplock-core`:

- **Encabezado Binario (30 bytes en Little-Endian):**
  1. `Magic Bytes` (14 bytes): Cadena ASCII fija `MPLOCK_BACKUP\0` (identificador estricto de formato).
  2. `Version` (4 bytes, `u32` LE): Versión del formato (fijada en `1`; versiones superiores son rechazadas con error explicativo).
  3. `Iterations` (4 bytes, `u32` LE): Rondas de PBKDF2 (fijadas en `600_000`; valores < 600.000 o > 2.000.000 son rechazados por seguridad).
  4. `Salt Length` (4 bytes, `u32` LE): Longitud del salt aleatorio (`16` bytes).
  5. `Nonce Length` (4 bytes, `u32` LE): Longitud del vector de inicialización / nonce (`12` bytes).
- **Payload Cifrado:** Corresponde a la salida directa de `mplock_core::crypto::encrypt` aplicada sobre la estructura JSON serializada con las credenciales, notas y semillas TOTP, utilizando AES-256-GCM y una clave derivada mediante PBKDF2-HMAC-SHA256 con una contraseña de exportación independiente de la clave maestra.
- **Escritura Atómica:** El archivo se escribe primero en un fichero temporal en el mismo directorio (`.{nombre}.tmp.{random}`) y se renombra atómicamente al destino final tras sincronizar buffers en disco (`sync_all`), protegiendo la integridad del archivo ante cortes o fallos.

04. Aleatoriedad y Muestreo sin Sesgo

4.1 Entropía del Sistema Operativo (`OsRng`)

MPLock se apoya en `rand::rngs::OsRng` (`crypto.rs:8`, `storage.rs:2`, `password.rs:1`) para todas las decisiones criptográficas (sales, IVs, DEK y generación de contraseñas), el cual utiliza el generador criptográfico del sistema operativo a través de la biblioteca getrandom.

4.2 Eliminación del Sesgo de Módulo (*Modulo Bias*)

El generador (`password::generate_secure_password`, `password.rs:21-63`) utiliza un alfabeto de 78 caracteres legibles (26 minúsculas, 26 mayúsculas, 10 dígitos y 16 símbolos).

En implementaciones tradicionales que reducen bytes mediante la operación módulo (`byte % 78`), dado que 256 no es divisible de forma exacta por 78 ($256 = 3 \times 78 + 22$), los primeros 22 caracteres tienen 4 valores posibles de entrada frente a solo 3 valores de los 56 caracteres restantes. Esto generaba una probabilidad de $\approx 1,5625\%$ frente a $\approx 1,1719\%$, lo que implicaba un sesgo estadístico medido del 33% en favor de ciertos caracteres.

> En `password.rs:44` se utiliza `rng.gen_range(0..allowed_base.len())`, que implementa internamente muestreo uniforme por rechazo (*rejection sampling*), descartando los residuos numéricos y asegurando una probabilidad idéntica de $\frac{1}{78}$ para cada carácter.
>
> Asimismo, el generador aplica un muestreo de rechazo global sobre la contraseña completa (`password.rs:58-61`): si la cadena generada aleatoriamente carece de al menos un carácter de cada una de las 4 clases (minúscula, mayúscula, dígito y símbolo), la contraseña entera es descartada y regenerada de forma no determinista, evitando posiciones fijas predecibles.

05. Gestión de Secretos en Memoria (Zeroize)

Para reducir la persistencia de datos confidenciales en memoria RAM tras su uso, `mplock-core` utiliza el trait `Zeroize` (`crypto.rs:11`, `storage.rs:8`):

- **Claves intermedias derivadas:** `key_bytes.zeroize()` se ejecuta inmediatamente después de instanciar el cifrador AES (`crypto.rs:49, 94`).
- **Texto plano de la DEK:** `dek_str.zeroize()` se invoca tras decodificar sus bytes binarios (`storage.rs:171`).
- **Estructura Credential:** Implementa `#[derive(Zeroize)]` con el atributo `#[zeroize(drop)]` (`storage.rs:23-24`), sobrescribiendo campos de usuario, contraseña y notas al destruirse el objeto.
- **Contraseña Maestra en Proceso:** Tanto en la app de escritorio (`src-tauri/src/lib.rs:74, 77`) como en el host nativo (`src-tauri/src/native_host.rs:77, 92, 123`), el buffer de la clave maestra ejecuta `pass.zeroize()` al bloquearse la base de datos o cerrarse el subproceso.

> **Límites Técnicos: Por Qué Hablamos de "Mitigación" y no de "Eliminación Garantizada"**
>
> - A partir de `mplock-core v0.1.1` y `src-tauri`, se habilita explícitamente la feature `features = ["zeroize"]` en `aes-gcm = "0.10"`. Esto permite que la estructura interna `Aes256Gcm`, que contiene las subclaves expandidas de AES, sobrescriba su estado interno con ceros al ejecutarse su destructor (`drop`).
> - Si Windows vuelca páginas de RAM a disco antes de que se ejecute el destructor `drop`, fragmentos de texto plano pueden persistir en el almacenamiento no volátil.
> - Registros intermedios del procesador pueden retener valores transitorios antes de la ejecución de la rutina de limpieza.
> - Las credenciales mostradas en la interfaz gráfica cruzan hacia React y el motor V8. El recolector de basura gestiona cadenas inmutables que no pueden ser sobrescritas manualmente con ceros desde el código de la UI.

06. Motor 2FA / TOTP (RFC 6238)

Implementado en `mplock-core/src/totp.rs` como componente nativo en Rust:

- **Algoritmo:** HMAC-SHA1 según el estándar universal RFC 6238 (`totp.rs:85`).
- **Paso temporal:** 30 segundos (`totp.rs:88, 95`).
- **Formato de salida:** Código numérico de 6 dígitos con expiración regresiva calculada: `30 - (timestamp % 30)` (`totp.rs:96`).
- **Normalización Base32:** El analizador `parse_otpauth_uri` (`totp.rs:16-72`) admite URIs `otpauth://totp/...` o cadenas Base32 en bruto, descartando guiones y espacios y validando de forma estricta el alfabeto RFC 4648 (`A-Z`, `2-7`).
- **Limitación en Parámetros de URIs:** La función `parse_otpauth_uri` extrae únicamente el parámetro `secret=`, ignorando silenciosamente `algorithm`, `digits` y `period`. Si un servicio externo requiere HMAC-SHA256 o 8 dígitos, los códigos generados por MPLock serán inválidos para dicho servicio.
- **Custodia de Semillas:** La semilla Base32 reside cifrada en la bóveda cifrada (credentials.enc) y **jamás se envía a la extensión del navegador**. El host nativo calcula el código de 6 dígitos en memoria y transfiere únicamente el código temporal resultante (`src-tauri/src/native_host.rs:180-194` — componente cerrado).

07. Licenciamiento Offline (Ed25519)

La validación de licencias de MPLock opera de forma local mediante firmas digitales asimétricas sobre la curva elíptica Ed25519 (`mplock-core/src/license.rs`).

7.1 Estructura Binaria de la Licencia (78 Bytes)

La clave de licencia es una cadena Base32 RFC 4648 sin relleno, distribuida en 25 bloques de 5 caracteres (125 caracteres legibles). Al decodificarse produce 78 bytes binarios (`license.rs:6-7, 97-108`):

- **Bytes 0..14 (Payload binario):** Versión de formato (1 byte), código de plan (1 byte: Pro/Trial), timestamp de emisión (4 bytes BE), timestamp de expiración (4 bytes BE, 0 para perpetua), y hash truncado de identidad (4 bytes).
- **Bytes 14..78 (Firma Ed25519):** Firma criptográfica de 64 bytes generada con la clave privada del emisor sobre el payload de 14 bytes.

7.2 Verificación Asimétrica y Privacidad

La aplicación contiene embebida únicamente la clave pública de verificación (`license.rs:112-119`). No se almacena la dirección de correo del comprador en la clave; solo viaja un hash truncado de 4 bytes (`license.rs:156-159`). Asimismo, se comprueba una lista negra basada en el hash SHA-256 de las firmas revocadas (`license.rs:121-130`).

> Una validación offline basada en firmas asimétricas no puede verificar si una misma clave ha sido introducida en múltiples equipos simultáneos. Aunque la aplicación vincula la clave cifrada en disco al hardware local (`MachineGuid` en el registro de Windows, `src-tauri/src/license.rs`), el modelo acepta el posible uso compartido no autorizado como una decisión de diseño priorizada en favor de no realizar llamadas de telemetría a servidores remotos.

08. Extensión del Navegador y Host Nativo

La comunicación entre el navegador (Chrome, Edge, Firefox) y la aplicación de escritorio utiliza el protocolo estándar de **Mensajería Nativa (Native Messaging)** a través de tuberías estándar `stdin/stdout` (`src-tauri/src/native_host.rs` — componente cerrado).

- **Formato:** Prefijo de 4 bytes Little-Endian con la longitud del mensaje JSON (`native_host.rs:18-29, 128-137`). Límite de seguridad de 1 MB contra agotamiento de memoria (`native_host.rs:30`).
- **Claves de Registro Verificadas:** El host nativo se registra exclusivamente en el Registro de Windows (`HKEY_CURRENT_USER` , verificado en `src-tauri/src/lib.rs:595-608`):
  - Google Chrome: `HKCU\Software\Google\Chrome\NativeMessagingHosts\com.passwords.app`
  - Microsoft Edge: `HKCU\Software\Microsoft\Edge\NativeMessagingHosts\com.passwords.app`
  - Mozilla Firefox: `HKCU\Software\Mozilla\NativeMessagingHosts\com.passwords.app`
- **Orígenes Autorizados en Manifiestos:**

  - Chrome y Edge (`allowed_origins` , `src-tauri/src/lib.rs:614-616`): `chrome-extension://gianaainehpppkelimbojoafaomdpipa/` (identificador en Chrome Web Store: `gianaainehpppkelimbojoafaomdpipa`).
  - Firefox (`allowed_extensions` , `src-tauri/src/lib.rs:597-599`): `password-manager@example.com` . *(Nota técnica: Este identificador quedó fijado permanentemente por Mozilla AMO porque fue el ID con el que se subió la extensión por primera vez al catálogo de Mozilla Add-ons; no es un valor temporal ni un requisito exigido técnicamente para enlazar el host nativo, sino la identidad inmutable asignada a la extensión en dicho catálogo).*

- **Flujo Real de la Contraseña Maestra en Native Messaging:** La contraseña maestra nunca viaja por redes externas ni servidores remotos. No obstante, en el canal local de mensajería nativa, sí viaja en texto plano JSON por `stdin` cuando el usuario desbloquea la base de datos desde el popup de la extensión (acción `"unlock"` con payload `request["masterPassword"]` , `src-tauri/src/native_host.rs:67-89`). El subproceso del host nativo retiene la clave maestra en la variable `master_password: Option<Vec<u8>>` mientras la sesión permanezca desbloqueada, aplicando `zeroize()` al bloquear o terminar (`native_host.rs:77, 92, 123`).
- **Segregación de Datos y Memoria en el Navegador:** El host devuelve únicamente la credencial individual que coincide con el dominio consultado (`system` , `username` , `password` , `notes`) y el código temporal TOTP ya calculado. La base de datos completa, la clave DEK, las semillas secretas Base32 de TOTP y el código de recuperación no cruzan el canal ni se transmiten al navegador. La contraseña recibida por la extensión reside temporalmente en la memoria JavaScript del navegador gestionada por su recolector de basura, la cual MPLock no puede sobreescribir con ceros desde el proceso de escritorio.
- **Permisos Declarados en la Extensión (`manifest.json`):**
  - `nativeMessaging` : Imprescindible para lanzar y comunicarse con el binario local `mplock.exe` .
  - `activeTab` : Acceso a la URL y título de la pestaña activa únicamente cuando el usuario pulsa el icono de la extensión.
  - `scripting` : Permite insertar las credenciales devueltas exclusivamente en los campos de usuario y contraseña de la pestaña activa tras pulsar "Rellenar". No se solicita el permiso global `<all_urls>` .
- **Exportación a CSV en Texto Plano (Riesgo Operativo del Cliente):** La función de exportación a CSV genera un archivo en texto plano sin cifrar con todos los secretos del almacén (usuarios, contraseñas y semillas TOTP) para permitir la migración hacia otros gestores. Aunque la aplicación mitiga la fuga accidental exigiendo la reautenticación mediante contraseña maestra y la escritura explícita de la palabra de confirmación `EXPORTAR` , una vez volcado al sistema de archivos el fichero resultante queda expuesto sin protección criptográfica y debe ser eliminado de forma segura por el usuario tras completar la migración.

09. Delimitación de Código Abierto vs. Lógica Comercial

Componente	Ubicación	Licencia	Estado	Contenido y Verificabilidad
`mplock-core`	github.com/eagoudet/mplock-core	MIT / Apache-2.0	**Abierto**	Criptografía (AES-GCM, PBKDF2), almacenamiento de bóveda (DEK dual), generador de contraseñas (rejection sampling), motor TOTP y verificación asimétrica Ed25519.
	`src-tauri/src/license.rs`	Propietaria	Cerrado	Límites de cuota (15 credenciales en Free/Trial), cálculo de días restantes, comprobación de hardware local (`MachineGuid`) y clave pública de producción. *Componente no abierto, no verificable públicamente.*
	`src-tauri/src/native_host.rs`	Propietaria	Cerrado	Comunicación Native Messaging por stdio, caché de sesión desbloqueada y cálculo efímero de TOTP. *Componente no abierto, no verificable públicamente.*
	`src/` , `src-tauri/`	Propietaria	Cerrado	Interfaz gráfica React 19, hooks de instalación NSIS para Windows y empaquetado Tauri v2. *Componente no abierto, no verificable públicamente.*

Declaración de transparencia: No afirmamos que toda la aplicación de escritorio sea abierta; afirmamos que el núcleo criptográfico que custodia y procesa los secretos del usuario (`mplock-core`) es independiente, abierto y públicamente auditable.

10. Pruebas Automatizadas y Reproducibilidad

El espacio de trabajo cuenta con un total de **50 pruebas unitarias automatizadas** verificadas mediante `cargo test`. Sin embargo, **únicamente las 14 pruebas de `mplock-core` son reproducibles públicamente** por cualquier usuario o auditor externo:

- **14 pruebas abiertas en `mplock-core`** Validan vectores oficiales del RFC 6238, ciclo de cifrado/descifrado, resistencia ante payloads corruptos, muestreo de rechazo en contraseñas y ciclo completo de inicialización con DEK dual.
- **36 pruebas en `tauri_app_lib`** Validan políticas de cuota en planes comerciales, control de acceso en host nativo, migraciones y retrocompatibilidad de almacenes, sanitización del portapapeles y exclusión en historial, y la suite completa de 9 pruebas de respaldo cifrado (`.mplockbackup` : ciclo roundtrip de cifrado/descifrado, rechazo de contraseña errónea, detección de alteración de bytes, detección de archivos truncados, validación de versiones soportadas y límites de iteraciones, respeto del límite de 15 cuentas en importación, escritura atómica en disco y rechazo de contraseña de exportación idéntica a la maestra).

Para reproducir las pruebas del núcleo abierto localmente:

```
# Clonar y verificar el núcleo criptográfico
git clone https://github.com/eagoudet/mplock-core.git
cd mplock-core
cargo test
```

11. Limitaciones Conocidas y Estado de Auditoría

> **Estado de Auditoría Externa**
>
> El diseño y código fuente de MPLock (como Cure53, Trail of Bits o NCC Group). Se desaconseja asumir garantías implícitas hasta que una evaluación profesional colegiada sea contratada y sus resultados publicados íntegramente.

Binarios, Compilaciones y Firma Digital

- **Código cerrado del instalador:** La aplicación de escritorio es de código cerrado y no dispone de compilaciones reproducibles (*reproducible builds*). Un auditor externo no puede verificar de forma automatizada que el instalador distribuido (`mplock_setup.exe`) coincida de manera exacta con el código fuente del núcleo abierto.

- **Firma Digital del Instalador:** El instalador actualmente no cuenta con firma Authenticode EV/OV comercial, lo que provoca la advertencia preventiva de Microsoft Defender SmartScreen durante la descarga e instalación en Windows.
- **Compromiso:** Publicar el hash criptográfico SHA-256 del instalador en cada lanzamiento de GitHub Releases para verificación de integridad de descarga.

Canal de Reporte de Vulnerabilidades

Si detectas una debilidad criptográfica o un problema de seguridad en `mplock-core` o en la aplicación de escritorio, agradecemos tu reporte coordinado a través del canal de contacto:

- **Canal Oficial:** mplocksoporte@gmail.com
- **Política de Divulgación:** Consulta los plazos de respuesta y procedimientos en SECURITY.md.

12. Historial de Versiones y Matriz de Trazabilidad

Historial de Revisiones

Versión 1.5 (14 de Octubre de 2026): Ajustes de redacción: sin términos absolutos; descripción del alfabeto del código de recuperación.
Versión 1.4 (3 de Octubre de 2026): Especificación técnica formal del formato de respaldo cifrado `.mplockbackup` (encabezado canónico de 30 bytes, validación de versiones e iteraciones, PBKDF2 600.000 rondas, AES-256-GCM y escritura atómica); inclusión de la exportación a CSV en texto plano en la matriz de riesgos operativos del cliente (sec. 8); ampliación de la suite de pruebas a 50 pruebas unitarias automatizadas (14 en `mplock-core` y 36 en `src-tauri`, con 9 pruebas dedicadas al ciclo de vida de respaldos).
Versión 1.3 (2 de Octubre de 2026): Lanzamiento de release v1.4.0: integración de `Zeroize` en `LAST_COPIED_SECRET` y vaciado en el cierre de ventana y bloqueo; precisión técnica sobre límites de cierre forzado y herramientas de portapapeles de terceros; actualización del conteo total de pruebas a 40 (14 núcleo + 26 desktop); sustitución de números de línea por identificadores de función en la trazabilidad interna; actualización de extensiones a v1.10.0 y restricción de instalaciones locales no empaquetadas en Chrome.
Versión 1.2 (2 de Octubre de 2026): Actualización técnica integral: actualización de `mplock-core` a `v0.1.1`; habilitación de la feature `features = ["zeroize"]` en `aes-gcm = "0.10"` en core y desktop para borrado automático de subclaves en memoria RAM tras drop; integración de extensiones nativas de Windows en `arboard` (`SetExtWindows::exclude_from_history / exclude_from_cloud`) mitigando persistencia en `Win + V` y nube; temporizador nativo en segundo plano (Tauri/Rust) para vaciado a los 30 s condicionado a coincidencia y limpieza automática al cerrar la ventana; delimitación de memoria JavaScript en extensiones; aclaración sobre ID inmutable de Firefox fijado por Mozilla AMO (`password-manager@example.com`).
Versión 1.1 (2 de Octubre de 2026): Revisión técnica rigurosa: corrección de flujo de contraseña maestra en Native Messaging, documentación de IDs reales de extensión, aclaración sobre límites del portapapeles, especificación de límites de PBKDF2 frente a Argon2id, advertencia de lectura de memoria en Windows por procesos del mismo usuario, documentación de parámetros ignorados en TOTP y aclaración de reproducibilidad pública de pruebas.

Matriz de Trazabilidad Técnica (Afirmación → Código Verificado)

Parámetro / Afirmación Técnica	Valor Verificado en Código	Archivo en Repos
PBKDF2 Iteraciones	`600_000` rondas	`mplock-core/src/crypto`
PBKDF2 Función Hash	`SimpleHmac<Sha256>`	`mplock-core/src/crypto`
Longitud Salt Criptográfico	16 bytes (128 bits)	`mplock-core/src/crypto`
Longitud Nonce / IV	12 bytes (96 bits)	`mplock-core/src/crypto`
Algoritmo Simétrico	AES-256-GCM	`mplock-core/src/crypto`
Payload Empaquetado Base64	`[Salt (16B)] + [IV (12B)] + [Ciphertext + Tag]`	`mplock-core/src/crypto`
Zeroize en Claves Derivadas	`key_bytes.zeroize()`	`mplock-core/src/crypto`
Zeroize en Cifrador AES (Feature)	`aes-gcm = { version = "0.10", features = ["zeroize"] }`	`mplock-core/Ca`
Limpieza Portapapeles (30 s)	`copy_to_clipboard_with_autoclear`	`src-tauri/src/T`
Generación DEK (32 bytes)	`OsRng.fill_bytes(&mut dek)`	`mplock-core/src/storage`
Envoltura Dual de la DEK	`encrypted_dek_master` y `recovery`	`mplock-core/src/storage`
Código de Recuperación	24 caracteres (32 símbolos sin caracteres ambiguos (O, 0, I, 1), 120 bits)	`mplock-core/src/storage`

Parámetro / Afirmación Técnica	Valor Verificado en Código	Archivo en Repos
Zeroize en Struct Credential	`#[zeroize(drop)]` en `struct Credential`	`mplock-core/src/storage`
Rejection Sampling en Contraseñas	`rng.gen_range(0..78)` + loop de 4 clases	`mplock-core/src/passwor`
Parámetros TOTP	RFC 6238, SHA1, 6 dígitos, paso 30s	`mplock-core/src`
Normalización Base32 TOTP	RFC 4648, descarte de espacios y guiones	`mplock-core/src`
Longitud Licencia Binaria	78 bytes (14 payload + 64 firma)	`mplock-core/src/license`
Verificación Ed25519	`verifying_key.verify(payload, &sig)`	`mplock-core/src/license`
Hash SHA-256 Revocación	Cálculo de hash de firma contra lista negra	`mplock-core/src/license`
Native Messaging Stdio	4-byte LE length + JSON UTF-8	`src-tauri/src/native`
Límite Mensaje Nativo	1 MB (1024 * 1024 bytes)	`src-tauri/src/native`
Allowed Extensions Firefox	`["password-manager@example.com"]`	`src-tauri/src/`
Allowed Origins Chrome/Edge	`["chrome-extension://gianaainehpppkelimbojoafaomdpipa/"]`	`src-tauri/src/`
Aislamiento de Semilla 2FA	Solo se envía código efímero de 6 dígitos	`src-tauri/src/native`

Parámetro / Afirmación Técnica	Valor Verificado en Código	Archivo en Repos
Tope 15 Cuentas Free/Trial	`check_credential_limit`	`src-tauri/src/`
Vinculación a Hardware	`MachineGuid` en Registro Windows	`src-tauri/src/licens`
Encabezado Respaldo Cifrado	`MPLOCK_BACKUP\0` (30B LE: v1, 600k iter, salt 16B, nonce 12B)	`src-tauri/src/`
Escritura Atómica en Disco	Archivo temporal + `sync_all` + `fs::rename`	`src-tauri/src/`
Exportación CSV Segura	Reautenticación con contraseña maestra + confirmación "EXPORTAR"	`src-tauri/src/` `src/ExportModal`

© 2026 MPLock. Documento técnico de acceso público.

Núcleo criptográfico dual-licensed bajo MIT y Apache-2.0 en GitHub.