Aionda Mail Security Whitepaper

Zero-Knowledge · Post-Quantum · Made in Germany

Version 1.4 — June 2026

Aionda GmbH
Stuttgart, Germany

contact-46epp9ba@contact.aionda.com
https://mail.aionda.com

PUBLIC DOCUMENT

Contents

1. Resumen ejecutivo

Aionda Mail es un servicio de correo electrónico Zero-Knowledge con cifrado post-cuántico, operado por Aionda GmbH en Stuttgart, Alemania. El servicio combina direcciones de correo desechables (DEAs) con un buzón de correo completamente cifrado — una combinación que ningún otro proveedor ofrece.

Propiedades de seguridad principales:

- **Arquitectura Zero-Knowledge:** Todo el cifrado y descifrado ocurre exclusivamente en el navegador del usuario. El servidor nunca tiene acceso al contenido de los correos en el buzón seguro, contraseñas ni claves de cifrado.
- **Seguridad post-cuántica:** El mecanismo de encapsulación de claves híbrido (X25519 + ML-KEM-1024) protege todos los datos contra ataques tanto de computadoras clásicas como cuánticas.
- **Autenticación Zero-Knowledge:** El protocolo OPAQUE (RFC 9807) asegura que las contraseñas nunca se transmiten ni se almacenan en el servidor — ni siquiera como hashes.
- **Shamir Secret Sharing (2-de-3):** La clave maestra del vault se divide en tres partes protegidas por contraseña, passkey y clave de recuperación. Dos cualesquiera de las partes reconstruyen la clave maestra.
- **Perfect Forward Secrecy:** Cada solicitud API utiliza un par de claves criptográficas único y de un solo uso. Comprometer una solicitud no afecta a ninguna otra.
- **Protección MITM (Guardian):** La extensión del navegador verifica de forma independiente todas las respuestas del servidor mediante firmas Ed25519 y detecta ataques de intermediario a través de la verificación de certificados TLS — incluso contra proxies corporativos y CDNs comprometidos.
- **Archivo de correo conforme a GoBD:** Las cuentas empresariales se benefician de una cadena de hashes a prueba de manipulación (blockchain SHA3-256) con contenido cifrado de extremo a extremo (Hybrid KEM), registro de auditoría completo, retención legal y retención configurable — conforme con la normativa alemana GoBD.
- **Sin recuperación de contraseña:** En caso de pérdida de la contraseña y todos los métodos de recuperación, los datos son irrecuperables. Esto es por diseño — demuestra que el servidor no puede acceder a los datos del usuario.

Jurisdicción: Legislación alemana (DSGVO/RGPD), sin intercambio de datos con agencias de inteligencia extranjeras.

---

2. Modelo de amenazas

2.1 Contra qué protege Aionda Mail

Amenaza	Protección
Compromiso del servidor (filtración de base de datos, acceso interno)	Todo el contenido del correo cifrado con claves que el servidor nunca posee
Espionaje de red (ISP, Wi-Fi, CDN)	Transporte API cifrado de extremo a extremo mediante Hybrid KEM

Amenaza	Protección
Inspección de CloudFlare	Las solicitudes API se cifran antes de salir del navegador; CloudFlare solo ve texto cifrado. La extensión Guardian detecta manipulación de respuestas mediante firmas Ed25519
Proxies MITM corporativos (ZScaler, Fortinet, etc.)	La extensión Guardian detecta certificados de proxy mediante lista de bloqueo de emisores (Firefox)
Ataques de computadoras cuánticas ("cosechar ahora, descifrar después")	ML-KEM-1024 (NIST FIPS 203) proporciona resistencia post-cuántica
Robo de base de datos de contraseñas	OPAQUE almacena solo registros criptográficos, no hashes de contraseñas
Ataques de fuerza bruta offline contra contraseñas	OPAQUE previene ataques offline; limitación de velocidad del lado del servidor previene ataques online
Análisis de tamaño de correos	El Bucket Padding oculta los tamaños reales de los correos
Canales laterales de compresión (CRIME/BREACH)	Bucket Padding aplicado después de la compresión
Enumeración de usuarios	Respuestas falsas determinísticas para cuentas inexistentes

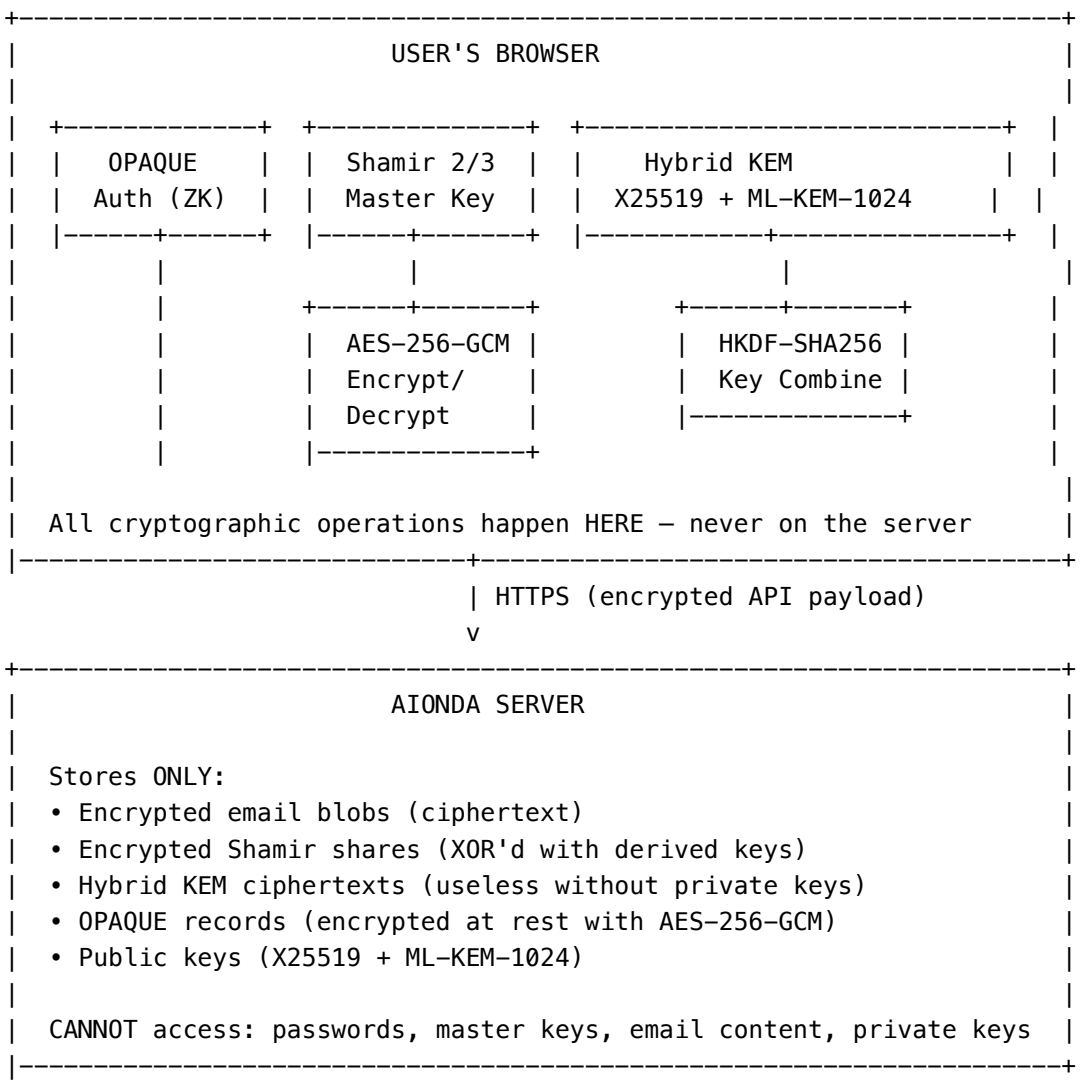
2.2 Contra qué NO protege Aionda Mail

Limitación	Explicación
Dispositivo comprometido	Si un malware controla el navegador, puede leer el contenido descifrado
Metadatos hacia destinatarios externos	Los correos hacia Gmail/Outlook viajan sin cifrar tras abandonar nuestros servidores (a menos que se use PGP)
Metadatos de correo en nuestros servidores	Las marcas de tiempo, direcciones IP y tamaños de correos cifrados son visibles para el servidor
Confianza en la entrega web	El navegador descarga JavaScript de nuestros servidores en cada visita (véase la Sección 19 para mitigaciones)
Criptoanálisis mediante coacción física	Ningún sistema criptográfico protege contra la coerción física

2.3 Principio de diseño

Aionda Mail sigue el modelo de **"servidor honesto"**: el sistema está diseñado de manera que incluso un servidor completamente comprometido — o un operador malicioso — no pueda descifrar los datos del usuario. La seguridad no depende de confiar en nosotros. Depende de las matemáticas.

3. Visión general de la arquitectura



3.1 Resumen de componentes

Componente	Propósito	Ubicación
OPAQUE (RFC 9807)	Autenticación de contraseña Zero-Knowledge	Cliente + Servidor
Shamir Secret Sharing	Protección de clave maestra del vault (umbral 2-de-3)	Solo cliente
Hybrid KEM (X25519 + ML-KEM-1024)	Encapsulación de claves post-cuántica para correos	Cliente + Servidor
AES-256-GCM	Cifrado simétrico autenticado	Cliente + Servidor
HKDF-SHA256	Derivación de claves a partir de secretos compartidos híbridos	Cliente + Servidor
BIP39	Codificación de clave de recuperación (mnemónico de 24 palabras)	Solo cliente

Componente	Propósito	Ubicación
WebAuthn PRF	Desbloqueo del vault basado en passkey	Solo cliente
Bucket Padding	Protección contra canales laterales	Cliente + Servidor

---

4. Autenticación Zero-Knowledge (OPAQUE)

4.1 ¿Por qué no el hashing de contraseñas?

Los servicios tradicionales almacenan hashes de contraseñas (bcrypt, Argon2). Aunque es mejor que el texto plano, este enfoque tiene debilidades fundamentales:

- El servidor ve la contraseña durante el inicio de sesión (aunque solo brevemente en RAM)
- Los hashes de contraseñas pueden ser atacados por fuerza bruta offline si la base de datos es robada
- El servidor podría ser modificado para registrar contraseñas

OPAQUE elimina los tres problemas. La contraseña nunca sale del navegador — ni como texto plano, ni como hash, en ninguna forma.

4.2 Cómo funciona OPAQUE

OPAQUE (RFC 9807) es un protocolo de intercambio de claves autenticado por contraseña asimétrico (aPAKE). Utiliza un mecanismo criptográfico de desafío-respuesta donde el servidor puede verificar que el usuario conoce la contraseña correcta sin jamás conocer cuál es esa contraseña.

Registro (una sola vez):

```
Browser                                    Server
--------                                   ------
1. Generate OPAQUE registration     ->
   request from password
                                           2. Create OPAQUE registration
                                              response (blind evaluation)
                                    <-
3. Finalize registration            ->      4. Store encrypted OPAQUE
   (export_key available)                      record (AES-256-GCM at rest)
```

Inicio de sesión (cada sesión):

```
Browser                                    Server
--------                                   ------
1. Generate login request (KE1)     ->      2. Look up encrypted OPAQUE record
   from password                              Decrypt with server key
                                              Generate response (KE2)
                                    <-
3. Verify server response
   Compute session key + KE3        ->      4. Verify KE3 (cryptographic proof)
   Password verified locally!                 If valid: session authenticated
```

```
                                    If invalid: reject (max 3 attempts)
```

Propiedades clave:

- La contraseña se verifica **del lado del cliente** en el paso 3 — el servidor nunca la ve
- El servidor almacena un **registro OPAQUE**, que no es un hash de contraseña y no puede ser atacado por fuerza bruta offline
- Los registros OPAQUE están adicionalmente **cifrados en reposo** con AES-256-GCM usando una clave del servidor
- **Protección contra enumeración de usuarios**: Las cuentas inexistentes reciben respuestas falsas determinísticas con temporización idéntica
- **Limitación de velocidad**: Máximo 3 intentos de autenticación por sesión, tiempo de espera de sesión de 120 segundos

4.3 Implementación

- **Biblioteca:** `@serenity-kit/opaque` (basada en WASM, de grado producción)
- **Componente del servidor:** Microservicio dedicado para operaciones criptográficas OPAQUE
- **Formato Base64:** base64url (seguro para URLs, sin relleno) para compatibilidad del protocolo
- **Registro de auditoría:** Todos los eventos de autenticación registrados con marcas de tiempo y direcciones IP

4.4 Migración de SRP

Las cuentas heredadas que utilizan SRP-6a se migran automáticamente a OPAQUE en el siguiente inicio de sesión. Después de la migración, el verificador SRP se elimina permanentemente. La migración es unidireccional — las cuentas no pueden revertir a SRP.

---

5. Clave maestra del Vault y Shamir Secret Sharing

5.1 Generación de la clave maestra

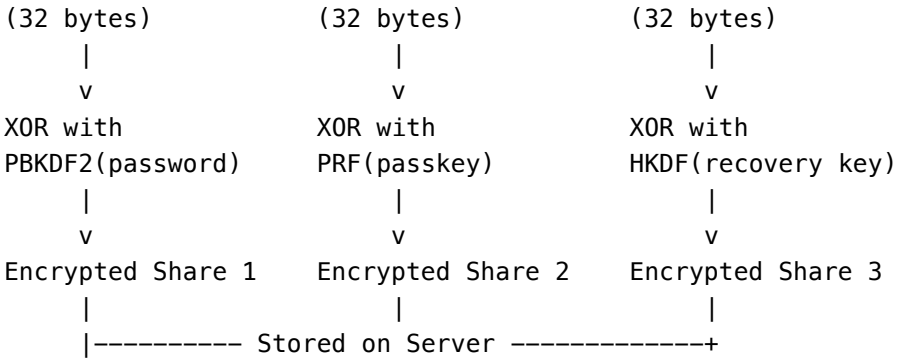
Cuando un usuario activa el buzón cifrado, se genera una **clave maestra de 256 bits (32 bytes)** utilizando el generador de números aleatorios criptográficamente seguro del navegador (`crypto.getRandomValues`).

Esta clave maestra es la raíz de todo el cifrado. Nunca sale del navegador en texto plano. Nunca se almacena en ningún lugar — ni en el navegador, ni en el servidor, en ninguna forma.

5.2 Shamir Secret Sharing (2-de-3)

La clave maestra se divide en tres partes utilizando el esquema de **Shamir's Secret Sharing** sobre el campo de Galois GF(2^8) con el polinomio irreducible de AES (x^8 + x^4 + x^3 + x + 1).

```
Master Key (32 bytes, generated once)
     |
     |--- Shamir Split (k=2, n=3) ---+
     |                               |
     v                               v
Share 1             Share 2             Share 3
```



Propiedad de umbral: Cualesquiera 2 de las 3 partes son suficientes para reconstruir la clave maestra mediante interpolación de Lagrange. El servidor almacena solo las partes cifradas — y no puede descifrar ninguna de ellas.

5.3 Protección de las partes

Cada parte se combina mediante XOR con una clave derivada de un factor de autenticación diferente:

Parte	Protegida por	Derivación de clave
Parte 1	Contraseña	PBKDF2-SHA256, 600.000 iteraciones, salt aleatorio de 32 bytes
Parte 2	Passkey (FIDO2)	Extensión WebAuthn PRF, vinculada al hardware
Parte 3	Clave de recuperación	HKDF-SHA3-256 con salt vinculado a la cuenta

Escenarios de reconstrucción:

- **Inicio de sesión normal:** Contraseña (Parte 1) + Passkey (Parte 2) -> Clave maestra
- **Passkey perdido:** Contraseña (Parte 1) + Clave de recuperación (Parte 3) -> Clave maestra
- **Cambio de contraseña:** Passkey (Parte 2) + Clave de recuperación (Parte 3) -> Clave maestra

5.4 Protección del almacenamiento de sesión

Incluso dentro de una sesión del navegador, la clave maestra nunca se almacena en texto plano:

1. Se genera una clave AES-256 efímera
2. La clave maestra se cifra con esta clave efímera
3. Solo el blob cifrado se coloca en `sessionStorage`
4. La clave efímera existe solo en la memoria de JavaScript (recolectada por el garbage collector al cerrar la pestaña)

---

6. KEM híbrido post-cuántico

6.1 ¿Por qué post-cuántico?

Las computadoras cuánticas ejecutando el algoritmo de Shor podrían romper la criptografía de clave pública clásica (RSA, ECDH, X25519) en tiempo polinómico. Aunque las computadoras cuánticas a gran escala aún no existen, la amenaza de los ataques **"cosechar ahora, descifrar después"** es real: los adversarios podrían almacenar datos cifrados hoy y descifrarlos cuando las computadoras cuánticas estén disponibles.

6.2 Enfoque híbrido

Aionda Mail utiliza un **mecanismo de encapsulación de claves híbrido** que combina:

- **X25519** (Curve25519 ECDH) — seguridad clásica probada, nivel de seguridad de 128 bits
- **ML-KEM-1024** (NIST FIPS 203, anteriormente Kyber-1024) — seguridad post-cuántica, NIST Security Level 5

El enfoque híbrido proporciona **defensa en profundidad**: la clave combinada es segura mientras **al menos uno** de los dos algoritmos permanezca sin ser roto.

6.3 Proceso de encapsulación

```
Sender (encrypting an email):

1. Generate ephemeral X25519 keypair
2. X25519 key agreement with recipient's public key
   -> x25519SharedSecret (32 bytes)

3. ML-KEM-1024 encapsulation with recipient's public key
   -> mlKemSharedSecret (32 bytes) + mlKemCiphertext (1568 bytes)

4. Combine secrets:
   combinedSecret = x25519SharedSecret || mlKemSharedSecret  (64 bytes)

5. Derive final key:
   sharedSecret = HKDF-SHA256(
       ikm = combinedSecret,
       salt = nil,
       info = "trashmail-hybrid-kem-v1",
       length = 32
   )

6. Use sharedSecret to wrap the email's ephemeral AES-256 key
```

6.4 Proceso de desencapsulación

```
Recipient (decrypting an email):

1. X25519 key agreement:
   x25519Shared = X25519(recipientPrivateKey, ephemeralPublicKey)

2. ML-KEM-1024 decapsulation:
   mlKemShared = ML-KEM-1024.Decapsulate(mlKemCiphertext, recipientPrivateKey)
```

```
3. Combine and derive (identical to sender):
   sharedSecret = HKDF-SHA256(x25519Shared || mlKemShared, "trashmail-hybrid-kem-v1")

4. Unwrap email's ephemeral AES-256 key using sharedSecret
5. Decrypt email content with ephemeral key
```

6.5 Tamaños de claves

Parámetro	Tamaño	Estándar
X25519 public key	32 bytes	RFC 7748
X25519 private key	32 bytes	RFC 7748
ML-KEM-1024 public key	1.568 bytes	NIST FIPS 203
ML-KEM-1024 private key	3.168 bytes	NIST FIPS 203
ML-KEM-1024 ciphertext	1.568 bytes	NIST FIPS 203
Combined shared secret	32 bytes	HKDF-SHA256 output

6.6 Biblioteca

- **ML-KEM-1024:** `@noble/post-quantum` (auditada, implementación JavaScript pura)
- **X25519:** WebCrypto API (`crypto.subtle.deriveBits`)
- **HKDF:** `@noble/hashes` (conforme con RFC 5869)

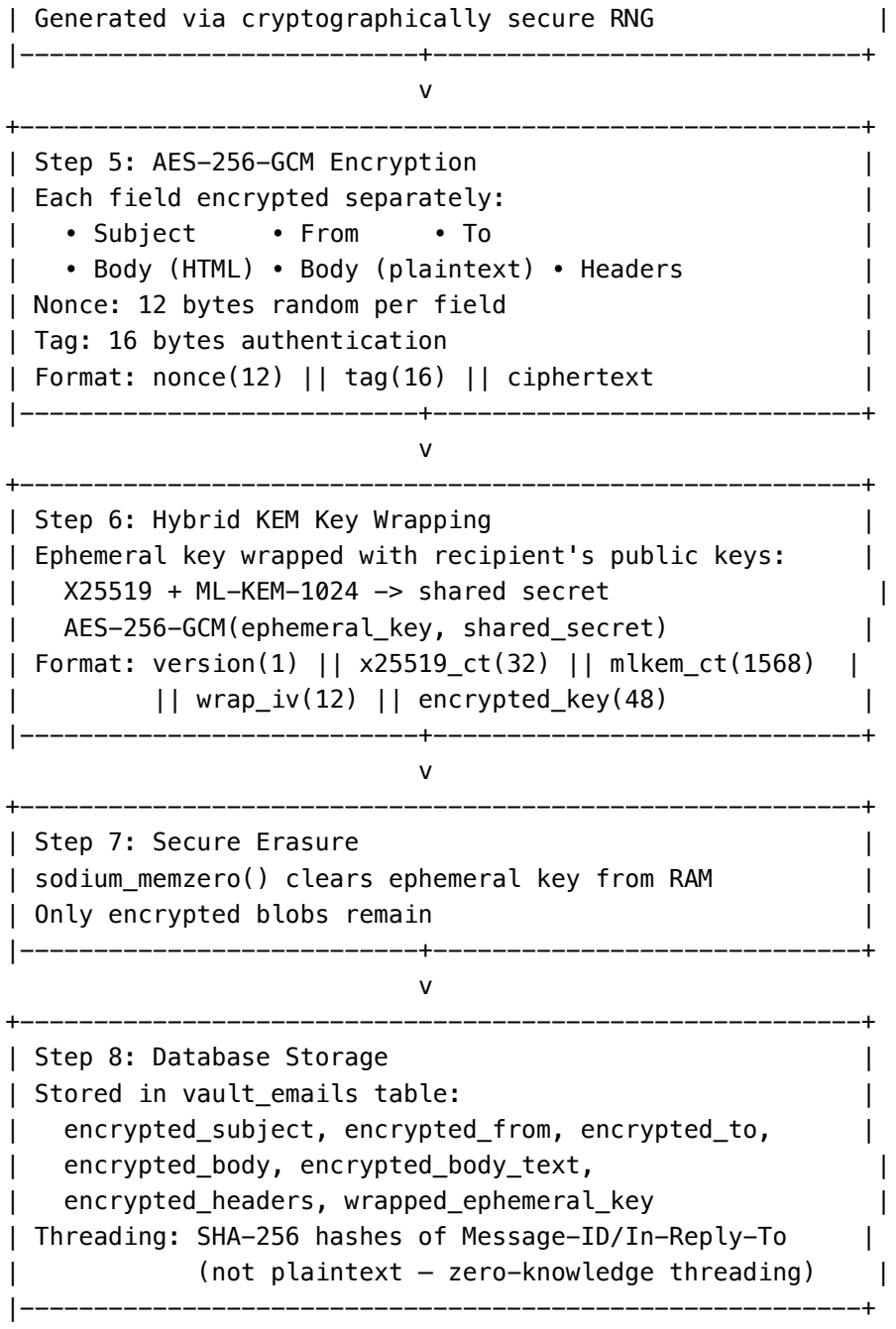
---

7. Pipeline de cifrado de correos electrónicos

7.1 Correo entrante (SMTP -> Almacenamiento cifrado)

Cuando un correo llega al servidor SMTP de Aionda Mail:

```
+-----------------------------------------------------------+
| Step 1: SMTP Reception                                    |
| Mail server receives email -> passes to SMTP handler      |
|-------------------------+---------------------------------+
                          v
+-----------------------------------------------------------+
| Step 2: Compression (optional)                            |
| gzip level 6 for text content                             |
| Skipped for: JPEG, PNG, MP4, ZIP, PDF, DOCX, etc.         |
|-------------------------+---------------------------------+
                          v
+-----------------------------------------------------------+
| Step 3: Bucket Padding                                    |
| Pad to fixed-size bucket (256B, 512B, 1KB, ..., 16MB)     |
| Format: 0xDEAD || length(4B) || data || random padding    |
|-------------------------+---------------------------------+
                          v
+-----------------------------------------------------------+
| Step 4: Generate Ephemeral Key                            |
| 32-byte random AES-256 key (unique per email)             |
```



7.2 Lectura de correo (descifrado en el navegador)

El proceso inverso ocurre completamente en el navegador:

1. Obtener el correo cifrado del servidor a través de la API cifrada
2. Analizar la clave efímera envuelta (extraer texto cifrado X25519 + texto cifrado ML-KEM)
3. **Desencapsulación Hybrid KEM** usando las claves privadas del vault -> secreto compartido
4. Desenvolver la clave AES-256 efímera
5. **Descifrado AES-256-GCM** de cada campo (asunto, remitente, destinatario, cuerpo, cabeceras)
6. Reordenar bytes: formato del servidor (nonce || tag || ct) -> formato WebCrypto (nonce ||

ct || tag)
7. Eliminar Bucket Padding (detectar bytes mágicos 0xDEAD)
8. Descomprimir si es gzip (detectar bytes mágicos 0x1F8B)
9. Decodificar UTF-8 a texto plano

7.3 Envío de correo

Al redactar y enviar un correo:

1. El cliente cifra el contenido del correo con un protocolo de desafío-respuesta
2. El servidor recibe la carga cifrada, descifra efímeramente (solo en memoria), envía vía SMTP
3. El servidor devuelve las cabeceras MIME generadas al cliente (cifradas)
4. El cliente cifra una copia con la clave maestra del vault y la almacena en la carpeta Enviados
5. El texto plano efímero del lado del servidor se descarta inmediatamente — nunca se escribe en disco

7.4 Adjuntos

Cada adjunto se cifra de forma independiente:

- Clave AES-256 efímera separada por adjunto
- Envoltura de clave Hybrid KEM separada por adjunto
- Nombre de archivo y tipo MIME cifrados por separado
- Sin compresión para formatos ya comprimidos (JPEG, ZIP, PDF, etc.)

7.5 Agrupación de hilos de correo (Zero-Knowledge)

La agrupación de hilos de correo (agrupar correos relacionados) utiliza únicamente **hashes SHA-256** de las cabeceras Message-ID e In-Reply-To. El servidor nunca ve las cadenas reales de Message-ID — puede agrupar correos por igualdad de hash sin conocer el contenido.

---

8. Capa de transporte API cifrada

8.1 Problema

Incluso con HTTPS, ciertos intermediarios pueden inspeccionar el tráfico:

- **CloudFlare** (CDN/protección DDoS) termina TLS y puede ver solicitudes en texto plano
- **Proxies corporativos** pueden realizar inspección TLS
- **Parámetros de API** (como `?cmd=read_email&id=123`) filtran metadatos

8.2 Solución: API cifrada de extremo a extremo

Toda la comunicación API está adicionalmente cifrada de extremo a extremo entre el navegador y el servidor de aplicación, dentro del túnel HTTPS:

```
Browser                                    Server
--------                                   ------

Phase 1: Key Exchange (once per session)
```

```
------------------------------------
GET /get_encryption_keys          ->        Return 20 pre-generated
                                           Hybrid KEM keypairs
                                  <-       {uuid, x25519_pub, mlkem_pub}

Phase 2: Encrypted Request (every API call)
------------------------------------
1. Pick random keypair from cache
2. Hybrid KEM encapsulate -> shared secret
3. gzip compress request payload
4. Bucket-pad compressed data
5. AES-256-GCM encrypt with shared secret
6. Generate ephemeral response keypair
7. POST /e {                      ->        8.  Validate key ownership
     encrypted_payload,                     9.  Hybrid KEM decapsulate
     key_uuid,                              10. AES-256-GCM decrypt
     x25519_ciphertext,                     11. Decompress
     mlkem_ciphertext,                      12. Route to API controller
     response_x25519_pub,                   13. Execute business logic
     response_mlkem_pub                     14. Encrypt response with
   }                                            client's response keys
                                  <-        15. Return encrypted response
16. Hybrid KEM decapsulate response
17. AES-256-GCM decrypt
18. Decompress -> plaintext response
```

8.3 Propiedades clave

- **Uso único:** Cada par de claves API se utiliza exactamente una vez, luego se invalida permanentemente
- **Perfect Forward Secrecy:** Comprometer la clave de una solicitud no afecta a ninguna otra solicitud
- **Vinculada a la sesión:** Las claves son reclamadas por una sesión específica y no pueden ser reutilizadas por otra
- **Pool de claves:** El servidor mantiene aproximadamente 100.000 pares de claves pre-generados
- **Re-obtención automática:** El cliente solicita automáticamente nuevas claves cuando la caché baja de 10
- **TTL de claves:** Las claves reclamadas expiran después de 24 horas
- **Bidireccional:** Tanto la solicitud COMO la respuesta están cifradas — el servidor nunca devuelve texto plano

8.4 Lo que CloudFlare ve

Con esta arquitectura, CloudFlare (o cualquier proxy que termine TLS) solo ve:

- `POST /e` — un único endpoint opaco
- Un blob binario de datos cifrados
- Sin nombres de comandos API, sin parámetros, sin IDs de correo, sin datos de usuario

9. Criptografía de carpetas compartidas

9.1 Modelo de compartición

Los usuarios pueden compartir carpetas cifradas con otros usuarios de Aionda Mail. El mecanismo de compartición utiliza el Hybrid KEM para cifrar una clave específica de la carpeta para cada destinatario.

9.2 Flujo de compartición

```
Folder Owner                                          Recipient
------------                                          ---------

1. Derive folder key from master key:
   folderKey = HKDF-SHA256(masterKey, folderUuid)

2. Fetch recipient's public keys:
   recipient.x25519_pub (32 bytes)
   recipient.mlkem_pub (1568 bytes)

3. Hybrid KEM encapsulate:
   hybridEncapsulate(recipient.x25519_pub, recipient.mlkem_pub)
   -> {x25519Ciphertext, mlKemCiphertext, sharedSecret}

4. Encrypt folder key:
   wrappedKey = AES-256-GCM(folderKey, sharedSecret, nonce)

5. Store on server:
   {x25519_ct, mlkem_ct, nonce, wrappedKey, permissions}

                                                      6. Fetch sharing record from server

                                                      7. Hybrid KEM decapsulate:
                                                         hybridDecapsulate(x25519_ct, mlkem_ct,
                                                           own_x25519_priv, own_mlkem_priv)
                                                         -> sharedSecret

                                                      8. Decrypt folder key:
                                                         folderKey = AES-GCM-decrypt(
                                                           wrappedKey, sharedSecret, nonce)

                                                      9. Decrypt emails in folder using folderKe
```

9.3 Modelo de permisos

Permiso	Capacidad
`readonly`	Leer correos de la carpeta (solo descifrar)
`einliefern`	Entregar nuevos correos en la carpeta
`bearbeiten`	Editar el contenido de la carpeta
`antworten`	Responder correos dentro de la carpeta

Permiso	Capacidad
`vollzugriff`	Acceso completo incluyendo transferencia de propiedad

9.4 Copia entre vaults (Re-Wrapping)

Cuando un destinatario copia un correo de una carpeta compartida a su propio vault, la clave del correo debe ser **re-envuelta** para su propio par de claves Hybrid KEM. Esto se realiza completamente del lado del cliente:

1. Descifrar la clave de la carpeta compartida usando las claves privadas del destinatario
2. Descifrar la clave efímera del correo usando la clave de la carpeta
3. Re-encapsular la clave efímera con las claves públicas propias del destinatario
4. Almacenar la copia re-envuelta en el vault del destinatario

El servidor facilita la transferencia pero nunca ve ningún material de clave en texto plano.

---

10. Vault Drive — Almacenamiento de documentos cifrado

Vault Drive es el almacenamiento de documentos Zero-Knowledge de Aionda Mail. A diferencia de los correos electrónicos, que utilizan claves efímeras por mensaje, Drive usa una **arquitectura de clave por archivo**: cada documento recibe su propia clave AES-256-GCM de 32 bytes, generada en el cliente y envuelta con la clave maestra.

10.1 ¿Por qué claves por archivo?

La arquitectura de clave por archivo ofrece tres ventajas clave:

1. **Compartición granular:** Documentos individuales pueden ser compartidos sin exponer la clave maestra. El servidor simplemente re-envuelve la clave del archivo para el destinatario — el contenido permanece sin cambios.
2. **Aislamiento ante compromisos:** Si una sola clave de archivo es comprometida (p.ej. a través de un enlace compartido), todos los demás documentos permanecen protegidos.
3. **Compartición eficiente sin re-cifrado:** Al compartir, el contenido no necesita ser re-cifrado — todos los destinatarios leen el mismo texto cifrado con una clave re-envuelta.

10.2 Flujo de subida

```
Cliente                                                 Servidor
-------                                                 --------

1. Generar clave de archivo aleatoria:
   fileKey = randomBytes(32)

2. Cifrar contenido, nombre, tipo MIME:
   encContent = AES-256-GCM(content, fileKey, nonce1)
   encName    = AES-256-GCM(name,    fileKey, nonce2)
   encMime    = AES-256-GCM(mime,    fileKey, nonce3)

3. Envolver clave de archivo con clave maestra:
```

```
   wrappedKey = AES-256-GCM(fileKey, masterKey, nonce4)

4. Enviar paquete cifrado:
   POST /e {
     encContent, encName, encMime,
     wrappedKey, wrappedKeyNonce
   }
                                                    5. Almacenar en vault_files + vault_file_c
                                                       (textos cifrados puros, sin material de
```

10.3 Flujo de descarga

```
1. Obtener chunk + wrapped_key del servidor
2. Desenvolver clave de archivo:
   fileKey = AES-256-GCM-decrypt(wrappedKey, masterKey, nonce)
3. Descifrar contenido, nombre, MIME:
   content = AES-256-GCM-decrypt(encContent, fileKey, nonce1)
```

El descifrado ocurre en un **Web Worker** que mantiene la clave maestra solo temporalmente en memoria. Después de la operación, la memoria se sobrescribe.

10.4 Compartición — solo re-envoltura de claves

La ventaja central de las claves por archivo se muestra al compartir: el contenido **no** se re-cifra. Solo la clave de archivo de 32 bytes se re-envuelve usando el KEM híbrido del destinatario.

```
Propietario                                         Destinatario
-----------                                         ------------

1. Desenvolver clave de archivo:
   fileKey = AES-GCM-decrypt(wrappedKey, masterKey)

2. Cargar claves públicas del destinatario:
   recipient.x25519_pub, recipient.mlkem_pub

3. Encapsulación KEM híbrida:
   hybridEncapsulate(recipient.x25519_pub,
                     recipient.mlkem_pub)
   -> {x25519_ct, mlkem_ct, sharedSecret}

4. Envolver clave de archivo con sharedSecret:
   shareWrappedKey = AES-256-GCM(fileKey,
                                 sharedSecret, nonce)

5. Almacenar registro de compartición:
   INSERT INTO vault_file_shares {
     file_uuid, recipient_account_id,
     x25519_ephemeral, mlkem_ciphertext,
     share_wrapped_key, permissions
   }
                                                    6. Obtener registro + chunk
                                                    7. Decapsulación KEM híbrida:
```

```
   hybridDecapsulate(x25519_ct, mlkem_ct,
     own_x25519_priv, own_mlkem_priv)
   -> sharedSecret
8. Desenvolver clave de archivo:
   fileKey = AES-GCM-decrypt(
     shareWrappedKey, sharedSecret)
9. Descifrar el MISMO texto cifrado:
   content = AES-GCM-decrypt(
     encContent, fileKey)
```

El propietario no necesita la clave maestra del destinatario — solo las claves públicas KEM híbridas, que el servidor almacena en texto claro.

10.5 Compartición de carpetas (recursiva)

Cuando una carpeta se comparte, el cliente procesa recursivamente todos los documentos y subcarpetas contenidos. Cada archivo recibe su propio registro de compartición conteniendo la clave de archivo del documento respectivo, re-envuelta para el destinatario. La carpeta en sí no tiene clave de carpeta — la estructura se enlaza mediante UUIDs padre.

Importante: El paso de encapsulación KEM se realiza **por operación**, no por archivo. Todos los archivos en una compartición por lotes usan el mismo `sharedSecret` — haciendo la operación eficiente sin reducir la seguridad (cada archivo aún tiene su propia clave).

10.6 Modelo de permisos

Permiso	Puede realizar
Lectura	Descargar, previsualizar, leer metadatos
Acceso total	+ renombrar, subir nueva versión, mover dentro de la carpeta compartida, subir nuevos documentos
No posible	Eliminar (solo propietario), sacar de carpeta compartida (solo propietario)

Los permisos se almacenan en el registro `vault_file_shares` y se aplican del lado del servidor. La criptografía protege el contenido — el control de acceso protege las operaciones.

10.7 Renombrar, sobrescribir y actualizaciones de metadatos

Para documentos compartidos, las operaciones de renombrar y sobrescribir se realizan directamente en el registro `vault_files` — no en los registros de compartición individuales. Todos los destinatarios ven inmediatamente el nuevo nombre o contenido, ya que referencian el mismo texto cifrado.

Al **sobrescribir** (nueva versión), se genera una **nueva clave de archivo**, el texto cifrado antiguo se reemplaza, y todos los registros de compartición existentes se re-envuelven del lado del cliente con la nueva clave. El propietario debe cargar las claves públicas de todos los destinatarios para esta operación.

10.8 Previsualización y descifrado en memoria

Imágenes, PDFs y otros documentos se descifran **exclusivamente en memoria** para la previsualización. No hay caché de disco con datos en texto claro. El cliente usa el **VaultDataLayer**,

una caché IndexedDB cifrada que almacena textos cifrados de manera persistente y solo los descifra bajo demanda.

Las miniaturas **nunca** se generan del lado del servidor — el servidor nunca ve el contenido. Las miniaturas se generan del lado del cliente a partir del original descifrado y también se almacenan en caché cifradas.

10.9 Lo que el servidor ve

Visible para el servidor	No visible
Contenido cifrado (bytes aleatorios)	Contenido en texto claro
Nombre de archivo cifrado (bytes aleatorios)	Nombre de archivo en texto claro
Tipo MIME cifrado (bytes aleatorios)	Tipo de archivo (imagen, PDF, texto…)
Tamaño del archivo (con padding a límites de bucket)	Tamaño original real
Marca de tiempo de subida	Clave de archivo, clave maestra
ID de cuenta del propietario	Contenidos de subcarpetas
UUID de carpeta padre (para estructura)	Nombres de carpetas (también cifrados)
Registros de compartición con textos cifrados KEM	Clave maestra del destinatario, clave de archivo desenvuelta

10.10 Aplicación de cuotas

La aplicación de cuotas se realiza del lado del servidor basándose en el **tamaño cifrado del archivo** (incluyendo el padding de bucket). El servidor no conoce el tamaño real en texto claro — el sobrecoste de padding lo paga intencionalmente el usuario para prevenir la fuga de tamaño.

Límites: - **Gratuito:** 100 MB de almacenamiento total, máx. 10 MB por documento - **Plus:** 1 GB de almacenamiento total, máx. 100 MB por documento

10.11 Compartición externa — enlaces públicos de compartición

Los documentos de Vault Drive pueden compartirse con destinatarios que **no disponen de una cuenta de Aionda Mail** mediante enlaces públicos servidos desde el dominio aislado `mail.aionda.com`. La función preserva el principio Zero-Knowledge tratando cada compartición como un **sobre criptográfico independiente**, desacoplado de la clave maestra del Vault del propietario.

Modos de protección:

Modo	Factor de confianza	Caso de uso
`link_only`	Fragmento de URL (nunca se envía al servidor)	Comodidad — cualquiera con el enlace puede acceder
`password`	Clave derivada mediante Argon2id	Transmisión de contraseña fuera de banda (SMS, llamada)

Modo	Factor de confianza	Caso de uso
`recipient_pubkey`	KEM híbrido (X25519 + ML-KEM-1024)	Seguridad post-cuántica cuando el destinatario ha registrado previamente claves públicas

10.11.1 Creación del enlace (cliente del propietario)

```
1. fileKey := AES-GCM-decrypt(vault_files.wrapped_key, masterKey)
2. shareKey := randomBytes(32)                          // efímera, por enlace
3. wrappedFileKey := AES-GCM(fileKey, shareKey)         // nuevo sobre
4. unlockVerifier := HMAC-SHA256(shareKey,
                     "unlock:" || shareUuid)            // prueba de conocimiento
5. Si hay protección por contraseña:
     salt := randomBytes(16)
     pwKey := Argon2id(password, salt, t=3, m=64MB, p=1)
     wrappedShareKey := AES-GCM(shareKey, pwKey)
     -> Enlace: https://mail.aionda.com/s/<shareUuid>
   En caso contrario (link_only):
     -> Enlace: https://mail.aionda.com/s/<shareUuid>#<base64(shareKey)>
       (fragmento de URL — los navegadores nunca transmiten fragmentos al servidor)

6. Metadatos cifrados (por enlace, con shareKey):
     encFilename, encMime, encMessage  // todos AES-GCM
7. POST /e  (transporte API cifrado, véase §8)
   -> el servidor guarda el registro del enlace — nunca ve texto en claro
```

La distribución del enlace la elige el propietario — no Aionda Mail. Tras crearlo, el propietario decide el canal: copiar el enlace, código QR, borrador `mailto:` en su propio cliente de correo, Signal, SMS, AirDrop o cualquier otro canal fuera de banda. Aionda Mail nunca ve, envía ni registra la entrega saliente — el servidor solo sabe qué share-UUIDs existen. Esto importa por dos razones: (a) el propietario elige el canal que considere más fiable (política de cumplimiento, mensajería preferida, escaneo QR cara a cara), y (b) para enlaces protegidos por contraseña permite la **separación de canales** — enlace por canal A, contraseña por canal B — de modo que un único canal comprometido nunca conceda acceso por sí solo.

10.11.2 Acceso al enlace (destinatario, sin cuenta) El destinatario visita `https://mail.aionda.com/s/<sha`
La Share Page es un **bundle independiente** del Manager, con su propio build reproducible, manifiesto Subresource Integrity y un endpoint `/.well-known/integrity.json` para verificación offline.

```
1. Bootstrap de la Share Page (el cliente genera un keypair KEM híbrido efímero)
2. share_fetch_meta -> { wrapped_file_key, salt?, encrypted_message, ... }
3. Fase de unlock — produce un unlock_token de un solo uso:
     modo password:    el servidor verifica la derivación Argon2id -> unlock_token
     modo link_only:   el cliente calcula HMAC-SHA256(shareKey, "unlock:"||uuid)
                       el servidor lo verifica contra el unlockVerifier almacenado
                       -> unlock_token
4. fileKey := AES-GCM-decrypt(wrappedFileKey, shareKey)
```

5. share_download_chunk (por chunk, unlock_token obligatorio)
6. El cliente hashea el texto en claro, llama a share_confirm_download

El unlock_token unifica el flujo para los tres modos de protección y permite limitación de tasa y registro de auditoría centralizados — el acceso link_only exige prueba de conocimiento del fragmento, por lo que bots y crawlers sin fragmento no pueden iniciar descargas.

10.11.3 Políticas aplicadas Cada enlace debe declarar lo siguiente al crearse (todo se aplica del lado del servidor):

- **expires_at** — obligatorio, 7 días por defecto, máximo 90 días
- **max_downloads** — contador obligatorio, 10 por defecto
- **Limitación de tasa** — los intentos Argon2 en enlaces con contraseña se limitan por hash de IP y por enlace
- **revoked_at** — el propietario puede revocar cualquier enlace con un clic; los intentos de unlock y las descargas posteriores devuelven una respuesta unificada «enlace no disponible» (mismo error que caducado/agotado — sin oráculo de enumeración)

10.11.4 Cadena de auditoría inalterable Todos los eventos de acceso relevantes se añaden a la hash chain existente enterprise_audit_log (SHA3-256, véase §18.2). Los siguientes tipos de acción están reservados para la compartición externa:

EXT_SHARE_CREATED, EXT_SHARE_EMAIL_SENT, EXT_SHARE_VIEWED, EXT_SHARE_UNLOCKED, EXT_SHARE_PREVIEWED, EXT_SHARE_DOWNLOAD_STARTED, EXT_SHARE_DOWNLOAD_CONFIRMED, EXT_SHARE_INTEGRITY_BROKEN, EXT_SHARE_PASSWORD_FAIL, EXT_SHARE_REVOKED, EXT_SHARE_ROTATED, EXT_SHARE_EXPIRED.

Los hashes de IP y user-agent usan un **salt rotativo diario** (vault_drive_external_share_daily_salts, purgado tras 30 días) — los hashes históricos dejan de ser reversibles tras la rotación del salt (conformidad con el RGPD), manteniendo la correlación a corto plazo para el propietario.

10.11.5 Rotación de clave (rewrap del enlace) El propietario puede rotar un enlace en cualquier momento sin volver a subir el contenido:

1. new_shareKey := randomBytes(32)
2. new_wrappedFileKey := AES-GCM(fileKey, new_shareKey)
3. INSERT nueva fila de enlace; old.linked_to_uuid := new.share_uuid
4. old.revoked_at := NOW()

Los destinatarios con el enlace antiguo reciben «enlace no disponible». El propietario distribuye el nuevo enlace. Los eventos de acceso de ambos enlaces permanecen vinculados mediante linked_to_uuid en la cadena de auditoría.

10.11.6 Qué hace la revocación — y qué no La revocación **detiene la entrega futura por parte del servidor** del enlace. **No revoca** retroactivamente share keys, file keys ni chunks ya entregados. Un destinatario que ya haya descargado el texto en claro — o que haya capturado el fragmento del enlace — puede seguir utilizando ese material localmente. Para casos con mayores requisitos de seguridad, combina: expires_at corto, max_downloads bajo, protección con contraseña y la extensión de navegador Guardian en el destinatario.

10.11.7 Qué ve el servidor

El servidor ve	El servidor NO ve
`share_uuid`, `file_uuid`, `account_id` (propietario)	Nombre del archivo, tipo MIME, contenido, mensaje en texto claro
`wrapped_file_key`, `wrapped_share_key` (cifrados)	`share_key` (link_only: vive exclusivamente en el fragmento de URL, en el cliente)
`unlock_verifier` (salida HMAC, no reversible)	Contraseña, clave derivada por Argon2
`protection_mode`, `expires_at`, `max_downloads`, `allow_preview`	Identidad del destinatario (solo hash salado de IP/UA, purgado tras 30 días)
`encrypted_filename`, `encrypted_mime`, `encrypted_message` (cifrados)	Sus textos en claro
`sender_display_name` en claro (el destinatario lo ve antes del unlock)	
Entradas de la cadena de auditoría para cada evento de acceso	

---

11. Aionda Chat — Mensajería E2EE post-cuántica

Aionda Chat es la superficie de mensajería en tiempo real integrada en Aionda Mail. A diferencia de los correos electrónicos —que utilizan claves efímeras de un solo uso por mensaje— las conversaciones de chat son de larga duración y requieren **forward secrecy** y **post-compromise security** para cada mensaje individual. Para esto diseñamos un protocolo dedicado: **AAR (Aionda Async Ratchet)** — una variante post-cuántica del X3DH + Double Ratchet de Signal, en la cual cada paso Diffie-Hellman se reemplaza por un KEM híbrido (X25519 + ML-KEM-1024).

11.1 ¿Por qué un protocolo dedicado?

Correo electrónico y chat tienen perfiles de seguridad fundamentalmente diferentes:

Propiedad	Correo	Chat
Cadencia	Esporádica (minutos/horas)	Tiempo real (segundos)
Duración de sesión	Mensaje único	Continua, días a años
Unidad de forward secrecy	Por mensaje	Por mensaje **dentro** de una sesión larga
Recuperación post-compromiso	No requerida (clave one-shot)	Requerida — cada mensaje nuevo se recupera de un compromiso pasado
Asincronía	Alta (destinatario suele estar fuera de línea)	Alta (destinatario suele estar fuera de línea)

Propiedad	Correo	Chat
Dinámica de grupo	Lista de destinatarios estática	Adición/eliminación dinámica

Una conversación de chat que simplemente reutilice la pipeline de correo electrónico tendría que (a) quemar un nuevo par de claves efímeras en cada pulsación de tecla (inaceptablemente lento), o (b) compartir una única clave de conversación estática (sin forward secrecy). Ninguna es aceptable. AAR resuelve esto manteniendo un estado de clave con ratchet continuo por par —cada mensaje hace avanzar el estado de manera irreversible.

11.2 Bloques de construcción

AAR usa exactamente cuatro primitivas:

KEM híbrido	X25519 + ML-KEM-1024	(encapsulación de claves)
AEAD	AES-256-GCM	(cifrado de mensajes)
KDF	HKDF-SHA256	(derivación de claves)
Firmas	Ed25519	(vinculación de identidad, firma SPK)

No se introducen nuevas suposiciones criptográficas —cada primitiva también se usa en otros lugares del sistema y ha sido auditada de forma independiente.

11.3 KeyBundle — el material de handshake asíncrono

Cada usuario publica un **KeyBundle** en el servidor cuando activa el chat por primera vez. El bundle permite a los pares comenzar a enviar mensajes incluso cuando el usuario está fuera de línea.

KeyBundle (por cuenta, híbrido de extremo a extremo):

```
Clave de identidad (IK)    — larga duración
   |-- IK.x25519_pub
   |-- IK.mlkem_pub
   |-- IK.ed25519_pub      — usada para firmar la SPK

Signed Pre-Key (SPK)       — rotación semanal
   |-- SPK.x25519_pub
   |-- SPK.mlkem_pub
   |-- Firma Ed25519       — firma la concatenación, vincula SPK a IK

One-Time Pre-Keys (OPK)    — pool de ~100, consumidas atómicamente
   |-- OPK.x25519_pub
   |-- OPK.mlkem_pub
```

Almacenamiento del servidor: Solo se almacenan las mitades públicas de cada clave, más firmas y metadatos. Las mitades privadas nunca abandonan el navegador de origen. El servidor aplica un `UPDATE … LIMIT 1 SET consumed_ts = NOW()` atómico cuando se obtiene una OPK, garantizando la semántica de uso único bajo solicitudes concurrentes.

Cuando el pool de OPK cae por debajo de 20, el cliente lo rellena con un nuevo lote — esto

evita que el handshake asíncrono se degrade a un modo degenerado sin entropía one-time.

11.4 Handshake estilo PQXDH (X3DH-PQ)

Para iniciar una nueva conversación con Bob, Alice obtiene el bundle de Bob (`IK_B`, `SPK_B`, una `OPK_B`) y la firma Ed25519 de Bob sobre `SPK_B`. Alice verifica la firma y luego realiza cuatro encapsulaciones KEM híbridas:

```
1. Verificar firma Ed25519 sobre SPK_B   — vincula SPK a la identidad larga

2. Generar clave efímera de Alice (EK_A):
   EK_A.x25519, EK_A.mlkem  (one-time, descartada inmediatamente tras el handshake)

3. Cuatro encapsulaciones KEM híbridas:
   ss1 = HKEM(IK_A_priv  ⊗ SPK_B_pub)    — Identidad Alice -> SPK Bob
   ss2 = HKEM(EK_A_priv  ⊗ IK_B_pub)     — Efímera Alice -> Identidad Bob
   ss3 = HKEM(EK_A_priv  ⊗ SPK_B_pub)    — Efímera Alice -> SPK Bob
   ss4 = HKEM(EK_A_priv  ⊗ OPK_B_pub)    — Efímera Alice -> OPK Bob

   (cada ss_i es a su vez la combinación HKDF de un secreto X25519
    Y un secreto ML-KEM — véase la Sección 6.4)

4. Derivación de la clave raíz:
   SK = HKDF-SHA256(
          IKM  = ss1 || ss2 || ss3 || ss4,
          info = "AAR-X3DH-v1" || consumed_OPK_id
        )

5. Olvidar todo material efímero privado; SK siembra el ratchet.
```

La construcción garantiza:

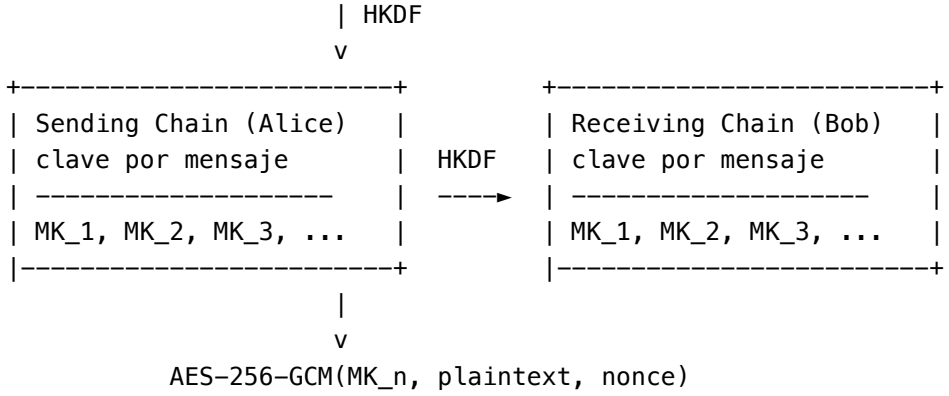
- **Autenticación mutua** mediante la `SPK_B` firmada con Ed25519 y la vinculación `IK_A`
- **Forward secrecy** incluso si `IK_B` se compromete posteriormente — `ss3` y `ss4` usan claves efímeras en ambos lados
- **Seguridad post-cuántica** porque cada `ss_i` incluye un término ML-KEM — un atacante que recolecte el tráfico actual para una futura computadora cuántica no puede reconstruir ninguno de los cuatro secretos
- **Resistencia a repeticiones** mediante el consumo atómico de OPK — el trigger del lado servidor de Bob se niega a liberar la misma OPK dos veces

El ID de la OPK consumida por Alice se vincula al parámetro `info` de HKDF — Bob solo puede reproducir la misma clave raíz con la misma OPK, y solo una vez.

11.5 Double Ratchet — claves por mensaje

Tras el handshake, ambos lados hacen avanzar un estado **Double Ratchet** por cada mensaje:

```
+-----------------------------------------+
|       Root Chain (ratchet DH)           |
|   avanza cada vez que se envía un nuevo  |
|   KEM híbrido                           |
|--------+--------------------------------+
```



Cada evento de chat lleva la **clave pública de ratchet** actual del remitente (par híbrido, ~1,6 KB). Cuando el destinatario recibe un mensaje cuya clave de ratchet difiere de la última vista, ambas partes realizan un nuevo KEM híbrido, derivan una nueva clave raíz y reinician sus chains. Este es el **ratchet DH** en terminología de Signal — excepto que cada paso es un KEM híbrido completo, no un único Diffie-Hellman.

Garantías por mensaje:

- **Forward secrecy:** la eliminación de `MK_n` hace indescifrable el mensaje `n`, incluso con el estado largo plazo completo en el momento de la captura
- **Post-compromise security:** una vez que ha ocurrido un nuevo paso DH-ratchet tras un compromiso, todos los mensajes posteriores están protegidos del atacante
- **Entrega fuera de orden:** las claves de mensaje se cachean para mensajes que llegan tarde (limitadas a 1000 claves por sesión para acotar la memoria)

11.6 Contenedor de mensaje

Cada evento de chat subido al servidor es un texto cifrado autocontenido:

```
MessageContainer (codificado en base64, almacenado en chat_events.payload):

+-----------------------------------------------------------------+
| ratchet_pub      :  Clave pública KEM híbrida (paso actual)  |
| prev_chain_len   :  Núm. de mensajes en la cadena anterior   |
| msg_number       :  Contador en la cadena actual             |
| ciphertext       :  AES-256-GCM(MK_n, plaintext || header)   |
| tag              :  Tag de autenticación AES-256-GCM         |
|-----------------------------------------------------------------+

El header dentro del plaintext AEAD contiene: login del remitente,
content_type (text/quote/file-reference), timestamp,
reply_to_event_uuid opcional.
```

El contenedor es opaco para el servidor — incluyendo el tipo de mensaje, el formato del nombre de visualización del remitente y cualquier contexto de respuesta citada. La base de datos de Aionda almacena el contenedor tal cual en `chat_events.payload`.

11.7 Conversaciones de grupo — Sender Keys por destinatario

Para conversaciones de sala (3+ participantes), AAR utiliza un modelo de **fan-out por destinatario**. El remitente deriva una sender-key chain por par de pares y cifra un mensaje una vez

por destinatario. Cada destinatario recibe entonces un texto cifrado dirigido personalmente, descifrable solo con las claves derivadas de su propia sesión AAR con el remitente.

Trade-off: El coste de fan-out es `O(N)` en destinatarios — aceptable para tamaños de equipo típicos (≤25 participantes). Para grupos más grandes se planea un futuro modo basado en MLS (véase Roadmap). El modelo actual es preferible a una sola clave de grupo compartida porque: (a) preserva la forward secrecy a nivel de par, (b) no requiere rotación de claves más allá del nivel bilateral cuando se elimina un miembro, y (c) hereda las garantías post-cuánticas del AAR por pares sin modificación.

11.8 Contenedor de conversación inicial

Cuando Alice inicia una nueva sala con Bob, Carol y Dan, el primer mensaje a cada participante se envuelve como un **contenedor de handshake inicial** que transporta (a) el material de handshake X3DH consumido contra el bundle de ese participante y (b) el primer mensaje en sí. El receptor despacha por el campo discriminador:

```
{
  "type": "initial",
  "initialMessage": { /* metadatos del handshake, OPK id consumida */ },
  "encryptedMessage": "<base64 MessageContainer>"
}
```

Los mensajes posteriores en la misma sala reutilizan la sesión AAR establecida y solo contienen la parte `encryptedMessage`.

11.9 Transporte — API cifrada + Mercure SSE + WebSocket

El plano de chat utiliza tres canales ortogonales:

Canal	Dirección	Propósito	Plaintext visto por el servidor
API cifrada `/e`	Cliente -> Servidor	Enviar mensaje, obtener historial, operaciones key-bundle (11 endpoints)	Ninguno — los 11 endpoints de chat utilizan el mismo transporte hybrid-KEM documentado en la Sección 8
Mercure SSE	Servidor -> Cliente	Push de entrega de nuevos eventos (`chat.event_received`, `chat.read_receipt`)	Ninguno — el servidor empuja el mismo MessageContainer opaco que almacenó
WebSocket `/chat/ws`	Bidireccional	Backfill de reconexión (`sync_request`), heartbeat, presencia	Solo booleano de presencia (en línea/fuera de línea) y string de login — nunca contenido de mensajes

La presencia se mantiene en un mapa en memoria en `aionda_chat_realtime`; un único endpoint HTTP intranet expone una búsqueda `login -> online` para el selector de equipo, de

modo que los usuarios puedan ver si un par está disponible. Ningún historial, ningún estado de lectura, ningún contenido atraviesa nunca el canal de presencia.

11.10 Confirmaciones de lectura

Las confirmaciones de lectura son una opción opt-in por cuenta (`chat_participants.send_read_receipts`). Si están activadas, marcar un mensaje como leído publica un evento `chat.read_receipt` que contiene:

```
{ conversation_uuid, last_read_event_uuid, account_login }
```

Nótese que `last_read_event_uuid` **no** es el contenido del mensaje — es un identificador asignado por el servidor que ya conoce. No se filtra información adicional más allá de "Alice ha visto los mensajes hasta el evento X". Los destinatarios que desactivan las confirmaciones de lectura (`send_read_receipts = false`) nunca emiten tales eventos, y el servidor aplica el toggle.

11.11 Integración de la cadena de auditoría

Para las cuentas Enterprise, cada operación de chat se añade al registro de auditoría infalsificable existente (`enterprise_audit_log`, cadena de hash SHA3-256 — véase §18). Los tipos de acción reservados son:

`CHAT_CONVERSATION_CREATED`, `CHAT_PARTICIPANT_ADDED`, `CHAT_PARTICIPANT_REMOVED`, `CHAT_MESSAGE_SENT`, `CHAT_MESSAGE_READ`, `CHAT_KEYBUNDLE_PUBLISHED`, `CHAT_KEYBUNDLE_ROTATED`, `CHAT_OPK_TOPPED_UP`, `CHAT_CONVERSATION_LEFT`.

El registro de auditoría contiene únicamente el UUID de la conversación, el login del actor y un timestamp — **nunca** el payload cifrado, las claves de ratchet o los contenidos de los mensajes.

11.12 Lo que el servidor ve

Visible para el servidor	No visible
`conversation_uuid`, lista de participantes (por `mail_account.name`)	Contenido de mensajes en claro
Payloads `MessageContainer` cifrados (textos cifrados opacos)	Claves de ratchet, claves de chain, claves de mensaje
Mitades públicas de IK, SPK, OPK; firmas Ed25519 sobre SPK	Mitades privadas de cualquier par de claves
Contador y timestamp de consumo de OPK	Qué OPK se consumió para qué conversación (correlación impedida por la vinculación HKDF-info que solo existe del lado del cliente)
Cadencia de conversación (timestamps de las filas `chat_events`)	Cadenas de respuestas citadas, archivos adjuntos en el contenedor
Booleano de presencia (en línea/fuera de línea)	Indicadores de escritura (peer-to-peer, nunca llegan al servidor)
Entradas de la cadena de auditoría para cuentas Enterprise	Contenidos en claro de las entradas de auditoría

11.13 Auditorías criptográficas

El protocolo AAR se implementó desde cero en TypeScript y pasó por tres auditorías independientes asistidas por IA, documentadas en `typescript/manager/chat/crypto/AUDIT.md`, `AUDIT_SECOND_OPINION.md` y `AUDIT_THIRD_OPINION.md`. Los hallazgos de cada ronda se incorporaron antes del despliegue público. Las superficies principales auditadas fueron: simetría del handshake, reserva OPK bajo concurrencia, deep-cloning del estado inmutable del ratchet, y el framing AEAD del `MessageContainer`.

11.14 Límites de implementación

La base de código del chat vive en `typescript/manager/chat/` (~9.000 líneas de TypeScript) e `includes/classes/Api/Services/Chat/` (capa de servicios PHP). Artefactos clave de auditoría:

```
crypto/aar-types.ts            — solo declaraciones de tipos, sin lógica
crypto/aar-keybundle.ts        — creación, rotación, serialización del KeyBundle
crypto/aar-x3dh.ts             — Handshake (caminos iniciador y respondedor)
crypto/aar-double-ratchet.ts   — Root chain, sending chain, receiving chain
```

La separación limpia entre la lógica del protocolo (`crypto/`) y el transporte/UI (`chat-store.ts`, `chat-panel.ts`, …) garantiza que el núcleo criptográfico pueda re-auditarse sin desviación de alcance UI.

---

12. Aionda Video Calls — Videoconferencia E2EE post-cuántica

Aionda Mail incluye videollamadas grupales cifradas de extremo a extremo (suite criptográfica `AIONDA_PQ_CALL_V1`). El relé de medios nunca ve el audio ni el vídeo en claro — solo reenvía fotogramas cifrados. Las claves de grupo se acuerdan con un handshake **híbrido post-cuántico**.

12.1 Objetivos de seguridad y modelo de confianza

Dentro del alcance — el protocolo proporciona:

- **Confidencialidad** de los medios de audio y vídeo, de extremo a extremo.
- **Autenticidad** del grupo — cada participante es criptográficamente miembro del mismo grupo MLS, verificado fuera de banda mediante el Short Authentication String (Sección 12.5).
- **Forward secrecy** — los medios pasados permanecen secretos aunque las claves actuales se vean comprometidas más tarde.
- **Post-compromise security** — el grupo recupera la confidencialidad después de que un miembro comprometido es eliminado y el grupo se re-clava.
- **Confidencialidad post-cuántica** — el establecimiento del grupo utiliza el KEM híbrido X-Wing.
- **Independencia del servidor** — una SFU, un servidor de señalización o una capa de almacenamiento maliciosos o comprometidos no pueden leer ni inyectar medios.

Fuera del alcance (explícitamente *no* defendido):

- Análisis de tráfico — el timing de los paquetes, los tamaños de los paquetes, el número de participantes y la duración de la llamada son observables (Sección 12.9).

- Compromiso del endpoint (véase la suposición de confianza más abajo).
- Malware a nivel de sistema operativo o incautación física del dispositivo.

Suposición de confianza del endpoint. El modelo asume un navegador y un sistema operativo no comprometidos. Los fotogramas se sellan dentro de la pipeline de medios del navegador (`RTCRtpScriptTransform`), pero un endpoint comprometido puede leer los medios en claro *antes* del cifrado o *después* del descifrado. El modelo de confianza de la entrega web y sus mitigaciones (firma de respuestas de Guardian) se tratan en la Sección 17 y la Sección 22.

12.2 ¿Por qué un protocolo dedicado?

La videoconferencia en tiempo real tiene dos requisitos que las capas de correo y de chat no tienen:

1. **Cifrado por fotograma a velocidad de línea.** Los fotogramas de audio/vídeo deben cifrarse individualmente para que una Selective Forwarding Unit (SFU) pueda enrutarlos, descartarlos y reordenarlos sin descifrarlos jamás. Este es el papel de **SFrame** (RFC 9605).
2. **Acuerdo dinámico de claves de grupo.** Los participantes entran y salen durante la llamada. Cada cambio de membresía debe re-clavar el grupo con forward secrecy y post-compromise security. Este es el papel de **MLS** (RFC 9420).

La SFU del lado del servidor (`mediasoup`) se trata como **infraestructura no confiable**: es un enrutador de paquetes de conocimiento cero, exactamente como la capa de almacenamiento para el correo.

12.3 Suite criptográfica — `AIONDA_PQ_CALL_V1` (`0xFF01`)

El perfil MLS sustituye los primitivos clásicos de una suite MLS estándar por híbridos post-cuánticos.

Rol	Algoritmo	Notas
HPKE KEM	**X-Wing** (ML-KEM-768 + X25519)	Híbrido; `draft-connolly-cfrg-xwing-kem`
Firmas	**ML-DSA-65**	NIST FIPS 204 (Dilithium), Level 3
KDF	HKDF-SHA-256	RFC 5869
AEAD (HPKE + SFrame)	**AES-256-GCM**	AES-256 ofrece un margen cómodo frente a los ataques cuánticos de búsqueda conocidos (Grover)

Sin downgrade. No existe un fallback clásico exclusivo. Si el handshake post-cuántico no puede completarse, la llamada falla de forma cerrada (fail closed) — un downgrade es un error duro, nunca un debilitamiento silencioso.

Madurez de X-Wing. X-Wing es actualmente un draft del IETF/CFRG (`draft-connolly-cfrg-xwing-kem`). El perfil actual lo adopta como el KEM híbrido preferido; futuras revisiones del protocolo podrían migrar a un KEM híbrido finalizado y estandarizado por CFRG/NIST preservando las capas MLS y SFrame sin cambios.

Agilidad criptográfica. El identificador de la suite criptográfica (`0xFF01`) determina por completo el KEM, el algoritmo de firma, el KDF y el AEAD. Una futura `AIONDA_PQ_CALL_V2` puede reemplazar cualquier primitivo sin cambiar las capas superiores del protocolo; las suites se seleccionan por identificador y nunca se negocian silenciosamente a la baja.

12.4 Acuerdo de claves de grupo (MLS, RFC 9420)

Cada participante está representado en el ratchet tree MLS por un **KeyPackage** (clave KEM pública X-Wing + credencial ML-DSA-65). El grupo evoluciona a través de *épocas*:

- **Welcome / Commit** — un miembro que se une se añade con un path secret encapsulado con X-Wing; el ratchet tree avanza a una nueva época.
- **Re-clavado en cada cambio de membresía.** Cualquier cambio — una incorporación, una salida, un reemplazo de dispositivo o una rotación de clave de identidad — hace avanzar el grupo a una nueva época. Esto es lo que aporta la forward secrecy (un nuevo miembro no puede leer medios pasados) y la post-compromise security (un miembro eliminado no puede leer medios futuros).
- **Remove-Commit.** Cuando un participante se va (o se detecta un peer obsoleto/zombie), el propietario emite un Remove-Commit que re-clava el grupo, de modo que un miembro que se haya marchado no pueda descifrar fotogramas posteriores aunque haya conservado material de clave antiguo.

La construcción híbrida está diseñada de modo que comprometer solo **un** componente — ML-KEM-768 *o* X25519 — no revela por sí mismo el secreto de grupo.

12.5 Identidad y autenticación

- **Credenciales.** Cada miembro porta una **Credential** MLS que vincula la identidad de un participante a su clave de firma ML-DSA-65. Cada mensaje del handshake MLS (Welcome / Commit / Update) se firma con esa clave, por lo que la SFU o el servidor de señalización no pueden falsificar operaciones de grupo.
- **Short Authentication String (SAS).** El enlace de la sala transporta un **secreto precompartido** de alta entropía en el fragmento URL, que nunca se envía al servidor en claro. El exportador del grupo MLS y este secreto se combinan en un Short Authentication String, cuya huella el cliente **fija (pin)**. Una discrepancia — por ejemplo, un servidor de señalización o una SFU que intente insertar un participante malicioso — desencadena un **aborto duro sin fallback suave**. Esto vincula el grupo criptográfico al secreto de la sala fuera de banda y frustra el man-in-the-middle del lado del servidor.
- **Factor de autenticación.** Para una sala de invitados, *la posesión del enlace de la sala equivale a la posesión de la credencial de la sala* — el secreto de la sala es el único factor de autenticación. Los participantes Aionda autenticados canjean además un token de señalización firmado y limitado al alcance de la cuenta (Sección 12.8).

12.6 Cifrado de medios (SFrame, RFC 9605)

Cada pista de medios se cifra fotograma a fotograma **antes** de llegar a la SFU. El ciclo de vida de la clave:

```
MLS group secret
      |  MLS Exporter (RFC 9420 §8)
      v
SFrame base key
      |  HKDF(base_key, KID = sender || epoch)
```

```
      v
per-sender key
      |  AES-256-GCM(nonce = frame counter, aad = sframe_header)
      v
per-frame ciphertext || tag      ->  wire = sframe_header || ciphertext || tag
```

- La clave SFrame se deriva del **MLS Exporter**, por lo que solo existe en los miembros del grupo autenticados — nunca en el servidor.
- El cifrado se ejecuta en un Worker dedicado `RTCRtpScriptTransform` (la API estandarizada Encoded-Transform, disponible en Chromium y Safari/WebKit), de modo que los fotogramas se sellan en la pipeline de medios del navegador.
- El códec es VP8; la SFU realiza el enrutamiento de keyframes solo sobre metadatos — nunca inspecciona el payload cifrado.

12.7 Protección contra replay

Cada fotograma cifrado lleva un **contador** SFrame monotónico dentro de su cabecera autenticada, y está vinculado a la época MLS actual. Los receptores rastrean el contador más alto visto por emisor (un almacén de contadores persistido del lado del receptor). Un fotograma con un contador duplicado o retrocedido, o uno vinculado a una época obsoleta, se rechaza — un texto cifrado capturado no puede reinyectarse (replay) en la sesión.

12.8 Señalización y modelo de acceso

- **La señalización** se ejecuta sobre un WebSocket autenticado (`aionda_signaling`). Cada sesión está protegida por un **JWT** de corta duración emitido por el servidor de aplicaciones después de que el cliente canjea el secreto de la sala a través de la API cifrada `/e` (véase la Sección 8). El token es simétrico (HS256) y se valida únicamente entre el servidor de aplicaciones y el único servicio de señalización; autoriza el establecimiento de la sesión y nunca transporta medios ni secretos de grupo. El servidor de señalización transmite los mensajes MLS Welcome/Commit pero no puede leerlos.
- **Enlace de la sala = credencial portadora.** Cualquiera que posea el enlace completo (UUID de la sala + secreto del fragmento) puede unirse — el modelo es intencionadamente "quien tenga el enlace". Los IDs de base de datos nunca se exponen; las salas se direccionan únicamente mediante UUIDs no adivinables.
- **Control del propietario.** El propietario de la sala (verificado del lado del servidor por la propiedad de la cuenta) puede marcar una sala como *finalizada* para bloquear futuras incorporaciones, y puede eliminar a un participante presente mediante un MLS Remove-Commit (Sección 12.4).

12.9 Lo que la SFU / el servidor ve — y limitaciones de los metadatos

La infraestructura SÍ puede ver	La infraestructura NO puede ver
Tamaños de los fotogramas cifrados, timing, metadatos de enrutamiento	Contenido de audio o vídeo (los fotogramas están sellados con AES-256-GCM)
Texto cifrado del handshake MLS (Welcome/Commit)	Secretos de grupo o claves SFrame (derivados del lado del cliente desde el MLS exporter)
UUID de la sala, número de participantes, subject del JWT	El secreto del enlace de la sala (vive en el fragmento URL)

Candidatos ICE / direcciones de transporte	Cualquier texto en claro que le permitiría unirse o descifrar la sesión

Metadatos que el sistema intencionadamente no oculta: las direcciones IP expuestas a la infraestructura ICE / STUN / TURN, el timing de los paquetes, los tamaños de los paquetes, el número de participantes y la duración de la llamada. La resistencia al análisis de tráfico no es un objetivo de diseño (Sección 12.1).

12.10 Propiedades de seguridad (resumen)

Propiedad	Garantía
Confidencialidad	Solo los miembros actuales del grupo MLS poseen las claves SFrame.
Autenticación	Todos los mensajes MLS se firman con ML-DSA-65; el grupo se vincula al secreto de la sala mediante el SAS fijado.
Forward secrecy	El compromiso de los secretos actuales no revela los medios pasados.
Post-compromise security	El re-clavado restaura la confidencialidad después de que se elimina un miembro.
Independencia del servidor	El compromiso del servidor de señalización, la SFU o el almacenamiento no expone los medios.
Post-cuántico	El establecimiento del grupo se basa en el KEM híbrido X-Wing (ML-KEM-768 + X25519).

Estas propiedades describen el *diseño* del protocolo. Está planificada una auditoría criptográfica independiente de terceros del subsistema de llamadas; hasta que se publique, las garantías anteriores son una autoevaluación.

---

13. Protecciones contra canales laterales

13.1 Bucket Padding

Problema: Los tamaños de los correos cifrados pueden filtrar información. Un atacante observando las longitudes del texto cifrado podría inferir contenido (p. ej., un correo de 50 bytes probablemente es “OK, gracias” mientras que un correo de 500 KB contiene adjuntos).

Solución: Todos los datos se rellenan a “buckets” de tamaño fijo antes del cifrado:

```
Bucket sizes: 256B, 512B, 1KB, 2KB, 4KB, 8KB, 16KB, 32KB,
              64KB, 128KB, 256KB, 512KB, 1MB, 2MB, 4MB, 8MB, 16MB

Format: [0xDEAD magic][4-byte length][actual data][random padding to bucket boundary]
```

Ejemplo: Un correo de 523 bytes se rellena a 1.024 bytes. Un observador solo ve “correo de 1 KB” — no el tamaño real de 523 bytes.

13.2 Compresión antes del cifrado

Los datos se comprimen con gzip (nivel 6) **antes** del cifrado. Este es el único orden correcto:

- La compresión después del cifrado fallaría (los datos cifrados tienen entropía máxima)
- El Bucket Padding después de la compresión previene ataques tipo CRIME/BREACH que explotan las ratios de compresión

13.3 Privacidad en la agrupación de hilos

Los hilos de correo utilizan hashes SHA-256 de las cabeceras Message-ID en lugar de identificadores en texto plano. El servidor puede agrupar correos relacionados por igualdad de hash sin conocer los identificadores reales de los mensajes.

---

14. Gestión y ciclo de vida de claves

14.1 Jerarquía de claves

```
Vault Master Key (32 bytes, generated once per account)
   |
   |--- Vault Keypair (Hybrid KEM)
   |    |-- X25519 public key (32 bytes) — stored plaintext on server
   |    |-- X25519 private key (32 bytes) — encrypted with master key
   |    |-- ML-KEM-1024 public key (1568 bytes) — stored plaintext on server
   |    |-- ML-KEM-1024 private key (3168 bytes) — encrypted with master key
   |
   |--- Per-Email Ephemeral Keys (32 bytes each)
   |    |-- Wrapped with recipient's Hybrid KEM public keys
   |
   |--- Per-Attachment Ephemeral Keys (32 bytes each)
   |    |-- Wrapped independently per attachment
   |
   |--- Folder Keys (derived via HKDF per folder)
   |    |-- Shared via Hybrid KEM encapsulation per recipient
   |
   |--- Signature Encryption Key (derived from master key)
        |-- Encrypts email signature templates
```

14.2 Almacenamiento de claves

Clave	Ubicación de almacenamiento	Protección
Clave maestra	En ningún lugar (reconstruida bajo demanda a partir de las partes de Shamir)	Shamir 2-de-3
Claves privadas del vault	Servidor (cifradas)	AES-256-GCM con clave maestra
Claves públicas del vault	Servidor (texto plano)	No sensibles — públicas por definición

Clave	Ubicación de almacenamiento	Protección
Claves efímeras de correo	Servidor (envueltas)	Encapsulación Hybrid KEM
Registros OPAQUE	Servidor (cifrados en reposo)	AES-256-GCM con clave del servidor
Partes de Shamir cifradas	Servidor	XOR con claves derivadas de contraseña/passkey/recuperación
Claves de transporte API	Servidor (pool pre-generado)	Uso único, TTL de 24h

14.3 Huellas digitales de claves

Cada par de claves del vault tiene una huella digital SHA-256 almacenada en el servidor. Esto permite:

- Registro de auditoría de rotaciones de claves
- Detección de cambios de claves no autorizados
- Verificación de integridad de claves del lado del cliente

---

15. Mecanismo de recuperación

15.1 Clave de recuperación (mnemónico BIP39)

Durante la configuración del vault, se presenta al usuario una **frase de recuperación de 24 palabras** generada a partir de 256 bits de entropía, codificada utilizando el estándar BIP39:

```
Example: apple river mountain sunset golden bridge falcon ocean
         crystal thunder meadow silver dolphin forest marble castle
         velvet compass harbor window ancient pepper rocket shield
```

15.2 Derivación de la clave de recuperación

```
1. Generate: 256 bits random entropy
2. Encode:   BIP39 mnemonic (24 words, 11 bits per word)
3. Derive:   verificationKey = HKDF-SHA3-256(
                 entropy,
                 salt = accountId,
                 info = "trashmail-recovery-verify"
             )
4. Hash:     verificationHash = SHA3-256(verificationKey)
5. Store:    Server stores ONLY verificationHash (32 bytes)
```

15.3 Lo que el servidor almacena

El servidor almacena **solo el hash SHA3-256** de una clave de verificación derivada. No almacena:

- Las palabras de recuperación

- La entropía
- La clave de verificación en sí

15.4 Flujo de recuperación

1. El usuario introduce la frase de recuperación de 24 palabras
2. El cliente deriva entropía -> HKDF-SHA3-256 -> SHA3-256 -> hash de verificación
3. El cliente envía el hash de verificación al servidor (nunca la clave en texto plano)
4. El servidor compara con el hash almacenado
5. Si coincide: Todos los métodos 2FA se desactivan, el usuario configura autenticación nueva
6. La clave de recuperación se revoca después de un solo uso

15.5 Limitación de velocidad

- Máximo 3 intentos de verificación por hora
- Bloqueo de 60 minutos después de exceder el límite
- Uso único: la clave de recuperación se revoca permanentemente después del uso exitoso

15.6 Sin recuperación de contraseña

No hay restablecimiento de contraseña por correo electrónico. Si un usuario pierde su contraseña Y todos los demás factores de autenticación (passkey + clave de recuperación), sus datos son permanentemente inaccesibles. Esta es la prueba fundamental de que Zero-Knowledge funciona — si pudiéramos recuperar sus datos, también podríamos leerlos.

---

16. Autenticación sin contraseña (Passkeys)

16.1 Extensión WebAuthn PRF

Aionda Mail soporta passkeys FIDO2 (llaves de seguridad hardware, autenticadores biométricos) para inicio de sesión sin contraseña y desbloqueo del vault.

La **extensión WebAuthn PRF (Pseudo-Random Function)** proporciona una salida determinística de 32 bytes vinculada al passkey y credencial específicos. Esta salida se utiliza para proteger la Parte 2 de Shamir.

16.2 Cómo funciona

```
1. Registration:
   - User creates passkey via navigator.credentials.create()
   - PRF extension generates hardware-bound output
   - Output XOR'd with Shamir Share 2 -> encrypted share stored on server

2. Authentication:
   - User authenticates with passkey (biometric/PIN)
   - PRF extension reproduces same 32-byte output
   - Output XOR'd with encrypted share -> Shamir Share 2 recovered
   - Combined with Share 1 (password) -> Master Key reconstructed
```

```
3. Vault Unlock (passwordless):
   - If both passkey (Share 2) and password (Share 1) available -> immediate unlock
   - Password verified via OPAQUE (separate from passkey auth)
```

16.3 Múltiples Passkeys

Los usuarios pueden registrar múltiples passkeys (p. ej., MacBook Touch ID, iPhone Face ID, YubiKey). Cada passkey protege de forma independiente su propia copia de la Parte 2. Cualquier passkey individual combinado con la contraseña es suficiente para desbloquear el vault.

---

17. Guardian: Protección MITM y firma de respuestas

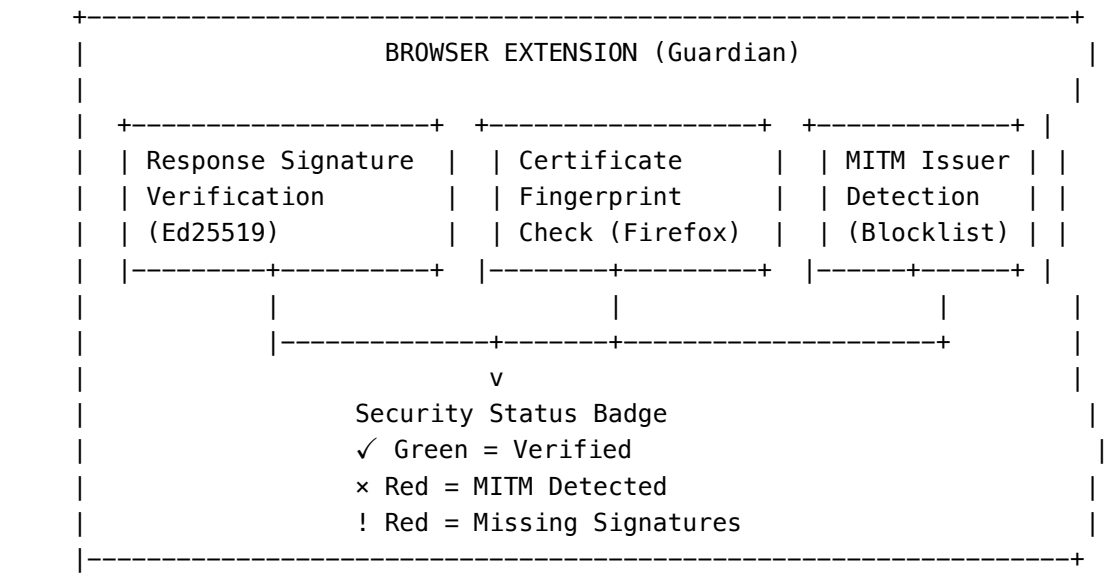
17.1 El problema con las aplicaciones web

Toda aplicación web tiene un problema inherente de confianza: el navegador descarga JavaScript del servidor en cada visita. Un atacante de intermediario (MITM) — ya sea un CDN comprometido, un proxy corporativo o un ISP malintencionado — podría teóricamente inyectar código modificado que exfiltre claves de cifrado.

Aionda Mail aborda esto con el **módulo Guardian**, un componente de extensión del navegador (disponible para Chrome y Firefox) que verifica de forma independiente la integridad del servidor.

17.2 Visión general de la arquitectura

El módulo Guardian opera dentro del Service Worker de la extensión del navegador — completamente independiente del JavaScript de la aplicación web. Realiza tres tipos de verificación:



17.3 Verificación de firma de respuesta (Ed25519)

Cada respuesta API del servidor de Aionda Mail está firmada criptográficamente usando **Ed25519** (Edwards-curve Digital Signature Algorithm).

Proceso de firma (lado del servidor):

```
1. Server generates API response body (JSON)
2. Construct signing input: responseBody + "|" + unixTimestamp
3. Sign with Ed25519 private key -> 64-byte signature
4. Attach HTTP headers:
   X-Aionda-Signature: <base64(signature)>
   X-Aionda-Timestamp: <unix_timestamp>
   X-Aionda-Key-Id: <key_identifier>
```

Proceso de verificación (extensión del navegador):

```
1. Extract signature, timestamp, and key ID from HTTP headers
2. Look up Ed25519 public key by key ID (bundled in extension)
3. Verify key has not expired (valid_from / valid_until)
4. Check timestamp freshness: |now - timestamp| ≤ 300 seconds
5. Reconstruct signing input: responseBody + "|" + timestamp
6. crypto.subtle.verify("Ed25519", publicKey, signature, data)
7. If invalid -> MITM alert, red badge
```

Propiedades clave:

- **Protección contra repetición:** Ventana de tiempo de 5 minutos previene la reproducción de respuestas antiguas
- **Detección de manipulación:** Cualquier modificación al cuerpo de la respuesta invalida la firma
- **Aislamiento de claves:** Las claves públicas están empaquetadas dentro de la extensión (no se descargan del servidor)
- **Separación de entornos:** Las claves de desarrollo (`dev-2026-01`) no pueden usarse en URLs de producción y viceversa

17.4 Gestión de claves públicas Ed25519

Las claves públicas se distribuyen con la extensión del navegador en `public_key.json`:

```json
{
    "keys": {
        "prod-2026-01": {
            "algorithm": "Ed25519",
            "public_key": "<base64 SPKI DER>",
            "valid_from": "2026-01-13T00:00:00Z",
            "valid_until": "2027-01-13T00:00:00Z"
        }
    }
}
```

- **Formato de clave:** SPKI DER (Subject Public Key Info, Distinguished Encoding Rules)
- **Tamaño de clave:** 32 bytes (clave pública Ed25519 de 256 bits)
- **Tamaño de firma:** 64 bytes (fijo)
- **Rotación:** Se agregan nuevas claves antes de que expiren las anteriores; las actualizaciones de la extensión entregan las nuevas claves
- **Sin confianza en el servidor:** Las claves están incrustadas en el binario de la extensión, no se obtienen del servidor

17.5 Verificación de certificados TLS (Firefox)

En Firefox, el módulo Guardian realiza una verificación adicional de certificados TLS utilizando la API `browser.webRequest.getSecurityInfo()` (no disponible en Chrome debido a las limitaciones de Manifest V3).

Flujo de verificación:

```
1. Browser extension intercepts HTTPS response
2. Extract TLS certificate chain from browser's security info:
   - Leaf certificate fingerprint (SHA-256)
   - Issuer Distinguished Name (O=, CN=)
   - Subject (CN=)

3. Check against known MITM issuers (hardcoded blocklist):
   ZScaler, Netskope, Fortinet, Palo Alto, Blue Coat,
   Check Point, Barracuda, Sophos, WatchGuard, Cisco Umbrella
   -> If match: MITM detected, show warning

4. Check against trusted issuers:
   Google Trust Services, Cloudflare, Let's Encrypt,
   DigiCert, Sectigo
   -> If match AND subject matches expected domain: OK

5. If unknown issuer: Fetch server's own certificate fingerprint
   - Server connects to itself via external routing (prevents spoofing)
   - Response is Ed25519 signed (prevents MITM from lying about cert)
   - Compare issuer organization with browser's certificate issuer
   -> If mismatch: MITM suspected, show warning
```

¿Por qué validación basada en emisor en lugar de pinning? CloudFlare (usado como CDN) rota los certificados hoja entre los servidores de borde. El pinning de certificados tradicional (que coincide con huellas digitales exactas) causaría falsos positivos. La validación basada en emisor es más robusta: la CA emisora es estable incluso cuando los certificados hoja cambian.

17.6 Obtención de certificado propio (Anti-Spoofing)

El endpoint de certificado del servidor utiliza una técnica ingeniosa anti-spoofing:

```
Server connects to cert.trashmail.com (or cert-subdomain.domain)
with SNI = mail.aionda.com

-> Forces external routing through CloudFlare
-> Receives the actual certificate that users see
-> Prevents localhost spoofing
-> Response signed with Ed25519 to prevent tampering
```

El servidor esencialmente pregunta “¿qué certificado ve el mundo exterior para mi dominio?” — y firma la respuesta para que la extensión pueda confiar en ella.

17.7 Indicadores de estado de seguridad

La extensión muestra una insignia en la barra de herramientas del navegador:

Insignia	Color	Significado
✓	Verde	Todas las respuestas verificadas — firmas válidas
!	Naranja	Usando clave de firma obsoleta (rotación pendiente)
×	Rojo	MITM detectado — verificación de firma falló
!	Rojo	Firmas faltantes — respuestas no firmadas
[Shield]	Azul	Modo protegido — aún no se ha realizado verificación

17.8 Cobertura de amenazas

Ataque	Método de detección	Navegador
Proxy MITM corporativo (ZScaler, Fortinet)	Lista de bloqueo de emisores de certificados	Firefox
Respuestas API modificadas	Verificación de firma Ed25519	Chrome + Firefox
Ataques de repetición	Ventana de tiempo de 5 minutos	Chrome + Firefox
Compromiso de CDN (CloudFlare)	Discrepancia de firma de respuesta	Chrome + Firefox
Sustitución de certificado	Comparación de emisor + auto-verificación del servidor	Firefox
Confusión de clave dev/prod	IDs de clave vinculados al entorno	Chrome + Firefox

17.9 Limitaciones

- **Chrome Manifest V3:** No puede inspeccionar certificados TLS — solo está disponible la verificación de firma de respuesta
- **Extensión requerida:** Los usuarios sin la extensión no se benefician de las protecciones de Guardian
- **Ed25519 no es post-cuántico:** La verificación de firmas utiliza criptografía clásica. Una computadora cuántica suficientemente potente podría teóricamente falsificar firmas Ed25519.

---

18. Archivo empresarial de correo electrónico (Blockchain)

18.1 Visión general

El plan Enterprise de Aionda Mail incluye un **archivo de correo electrónico conforme a GoBD** asegurado por una cadena de hashes criptográfica (blockchain). Cada correo archivado se convierte en un bloque inmutable en una cadena por empresa. Cualquier manipulación — modificación, eliminación o inserción de bloques — es criptográficamente detectable.

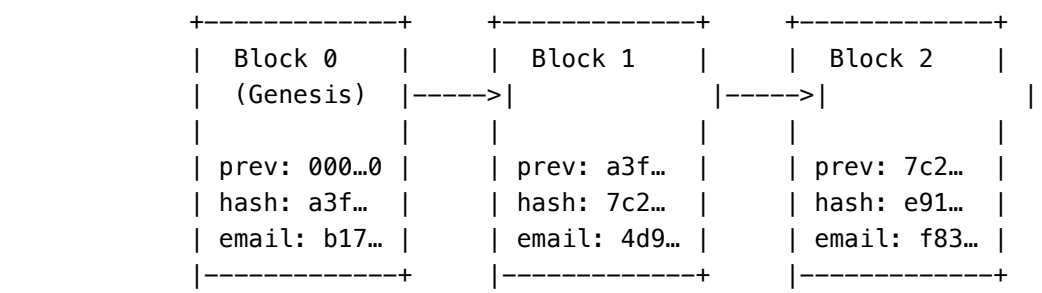
El archivo combina dos capas de seguridad independientes:

1. **Cadena de hashes (SHA3-256):** Garantiza integridad e inmutabilidad — demuestra que ningún correo fue alterado o eliminado después del archivado

2. **Cifrado Hybrid KEM (CAK):** Garantiza confidencialidad — el servidor no puede leer el contenido de los correos archivados

18.2 Arquitectura de la cadena de hashes

Cada correo archivado se convierte en un bloque en una cadena secuencial a prueba de manipulación:



Cálculo del hash de bloque:

```
block_hash = SHA3-256(
    prev_block_hash  || "|" ||
    timestamp        || "|" ||
    email_hash       || "|" ||
    direction        || "|" ||
    sender_domain    || "|" ||
    recipient_domain
)
```

Propiedades:

- **Algoritmo de hash:** SHA3-256 (NIST FIPS 202)
- **Bloque génesis:** `prev_block_hash` = 64 ceros, `block_number` = 0
- **Numeración secuencial:** Aplicada por la base de datos `UNIQUE KEY (company_uuid, block_number)`
- **Una cadena por empresa:** Aislamiento completo entre empresas
- **Hash de correo:** `SHA3-256(sender || recipient || timestamp || size)` — prueba de integridad de los datos del correo original

18.3 Detección de manipulación

El algoritmo de verificación de la cadena detecta cualquier forma de manipulación:

```
For each block (ordered by block_number ASC):

  1. Verify link:    block.prev_block_hash == expected_prev_hash
  2. Recalculate:    expected = SHA3-256(prev_hash | timestamp | email_hash | ...)
  3. Verify content: block.block_hash == expected
  4. Advance:        expected_prev_hash = block.block_hash

If ANY check fails -> chain is broken at block N
```

Intento de manipulación	Detección
Modificar contenido del correo	`email_hash` cambia -> falla el recálculo de `block_hash`
Modificar metadatos (remitente, dominio, marca de tiempo)	Incluido en la entrada del hash -> discrepancia de `block_hash`
Eliminar un bloque	El `prev_block_hash` del siguiente bloque queda huérfano
Insertar un bloque	Rompe el `block_number` secuencial + cadena `prev_block_hash`
Reordenar bloques	La restricción `UNIQUE KEY` + verificación secuencial lo previene
Reemplazar toda la cadena	El hash del bloque génesis diferiría de cualquier copia de seguridad externa

El **resultado de verificación** reporta el número de bloque exacto donde se detectó la manipulación, con los valores de hash esperados vs. reales para análisis forense.

18.4 Company Archive Key (CAK) — Cifrado Zero-Knowledge

Los contenidos del archivo se cifran de extremo a extremo usando una **Company Archive Key** — un par de claves Hybrid KEM (X25519 + ML-KEM-1024) generado del lado del cliente por el propietario de la empresa.

```
Company Owner's Browser                          Server
-----------------------                          ------

1. Generate Hybrid KEM keypair (client-side):
   X25519 keypair (32 + 32 bytes)
   ML-KEM-1024 keypair (1568 + 3168 bytes)

2. Derive wrapping key from password:
   wrappingKey = HKDF-SHA256(
     password,
     salt = "trashmail-archive-{account_id}",
     info = "trashmail-archive-key-wrap",
     length = 32
   )

3. Wrap private keys:
   AES-256-GCM(x25519_priv || mlkem_priv, wrappingKey)

4. Send to server:                          ->   Store:
   • Public keys (plaintext)                     archive_x25519_pub
   • Wrapped private keys (encrypted)            archive_mlkem_pub
```

`wrapped_archive_key`

Distribución de claves a otros empleados (Administrador, Oficial de Cumplimiento):

1. El propietario descifra las claves privadas del CAK con su contraseña
2. El propietario re-envuelve las claves privadas con la clave derivada de la contraseña del empleado objetivo
3. El servidor almacena la copia re-envuelta en el registro del empleado
4. Cada empleado autorizado tiene su propia copia envuelta de forma independiente

El servidor nunca ve las claves privadas del CAK en texto plano.

18.5 Qué se cifra

Cuando un correo se archiva, se aplican dos capas de cifrado:

Metadatos cifrados (AES-256-GCM con Hybrid KEM):

```
{
  "d": "INBOUND",
  "s": "user@example.com",
  "r": "admin@company.de",
  "sd": "example.com",
  "rd": "company.de",
  "sz": 45000,
  "ts": "2026-02-27T10:30:00Z",
  "ha": true,
  "ac": 3,
  "en": "John Doe"
}
```

Contenido cifrado del correo (envoltura de clave Hybrid KEM separada):

```
{
  "subject": "Meeting notes",
  "body": "<html>...</html>",
  "from": "sender@domain.com",
  "to": "recipient@company.de"
}
```

Aplicación Zero-Knowledge: Después del cifrado, los campos de metadatos en texto plano en la base de datos (sender_address, recipient_address, domains) se reemplazan con sus hashes SHA3-256. El servidor almacena solo hashes — los valores originales existen solo dentro de los blobs cifrados.

18.6 Registro de auditoría

Cada acción sobre el archivo se registra en una **cadena de auditoría independiente** (también encadenada por hashes con SHA3-256):

Acción	Cuándo se registra
`EMAIL_RECEIVED` / `EMAIL_SENT` / `DRAFT_ARCHIVED`	Correo archivado
`VIEW_EMAIL` / `VIEW_ATTACHMENT`	Empleado lee correo archivado

Acción	Cuándo se registra
`SEARCH_ARCHIVE`	Búsqueda realizada
`EXPORT_EMAIL` / `EXPORT_REPORT`	Datos exportados
`VERIFY_CHAIN` / `CHAIN_VERIFIED_OK` / `CHAIN_VERIFIED_BROKEN`	Verificación de integridad
`LEGAL_HOLD_SET` / `LEGAL_HOLD_RELEASED`	Retención legal activada/desactivada
`ARCHIVE_DECRYPT`	CAK utilizado para descifrar contenido
`ADMIN_ACCESS`	Acción administrativa

Cada entrada de auditoría registra: actor (UUID + rol), dirección IP, ID de sesión, hash del correo objetivo, y si la cadena era válida en el momento del acceso.

18.7 Retención legal y período de retención

- **Período de retención:** Configurable por empresa (predeterminado: 10 años), calculado por correo como `archived_at + retention_years`
- **Retención legal:** Los correos individuales pueden ser puestos bajo retención legal, impidiendo su eliminación hasta que se libere. Incluye razón, actor y marca de tiempo.
- **Cumplimiento GoBD:** La combinación de cadena de hashes inmutable, registro de auditoría completo, retención configurable y retención legal satisface los requisitos de la normativa alemana GoBD (Grundsätze zur ordnungsmäßigen Führung und Aufbewahrung von Büchern, Aufzeichnungen und Unterlagen in elektronischer Form sowie zum Datenzugriff).

18.8 Exportación forense

Los usuarios autorizados (Propietario, Administrador) pueden exportar el estado completo de la cadena para verificación independiente:

- Datos completos de la cadena con todos los hashes de bloques
- Resultado de verificación (válida/rota, número de bloque roto si aplica)
- Valores de hash esperados vs. reales para análisis forense
- Últimas 50 entradas del registro de auditoría
- Formato JSON para re-verificación externa con cualquier implementación de SHA3-256

---

19. Lo que el servidor ve — y lo que no

Esta sección documenta explícitamente la frontera Zero-Knowledge.

19.1 El servidor PUEDE ver

Dato	Por qué es visible	Mitigación
Dirección IP	Requisito TCP/IP	Se recomienda el uso de VPN/Tor si se desea mayor privacidad
Marcas de tiempo	Hora de recepción del correo	Inherente al protocolo de correo
Blobs de correo cifrados	Almacenados para recuperación	Cifrados con AES-256-GCM, clave desconocida para el servidor
Tamaños de texto cifrado con relleno	Requisito de almacenamiento	El Bucket Padding oculta los tamaños reales
Dirección DEA del destinatario	Requisito de enrutamiento	La DEA es desechable, no la dirección real
Existencia de la cuenta	Flujo de autenticación	Protección contra enumeración de usuarios implementada
Claves públicas	Requeridas para el cifrado por el servidor	Públicas por definición, no sensibles
Partes de Shamir cifradas	Almacenamiento para el usuario	XOR con claves que el servidor no conoce
Registros OPAQUE	Protocolo de autenticación	No son hashes de contraseñas, cifrados en reposo

19.2 El servidor NO PUEDE ver

Dato	Por qué es invisible
Contenido del correo (asunto, cuerpo, cabeceras)	Cifrado con claves efímeras envueltas vía Hybrid KEM
Contraseña del usuario	OPAQUE — la contraseña nunca se transmite
Clave maestra	Reconstruida solo en el navegador a partir de las partes de Shamir
Claves privadas del vault	Cifradas con la clave maestra antes del almacenamiento
Claves efímeras de correo	Envueltas con Hybrid KEM, el servidor carece de claves privadas
Clave de recuperación / mnemónico	Solo se almacena el hash SHA3-256 de la clave derivada
Salidas PRF del passkey	Vinculadas al hardware, nunca salen del autenticador

Dato	Por qué es invisible
Nombres de carpetas	Cifrados con claves específicas de carpeta
Firmas de correo	Cifradas con la clave maestra
Contenido de solicitudes API	Cifrado vía capa de transporte /e
Contenido de respuestas API	Cifrado antes de la transmisión

19.3 Garantía criptográfica

Incluso con acceso completo a:

- La base de datos completa
- Todo el tráfico de red
- El código fuente y configuración del servidor
- Todos los registros OPAQUE y claves del servidor

…un atacante **no puede** descifrar un solo correo sin la contraseña del usuario (o passkey + clave de recuperación). Esto no es una política — es una imposibilidad matemática impuesta por el diseño criptográfico.

---

20. Referencia de algoritmos

20.1 Tabla completa de algoritmos

Componente	Algoritmo	Parámetros	Estándar
Autenticación de contraseña	OPAQUE	RFC 9807, aPAKE	RFC 9807
Derivación de clave de contraseña	PBKDF2-SHA256	600.000 iteraciones, salt 32B, salida 32B	NIST SP 800-132
Cifrado del vault	AES-256-GCM	Clave 256-bit, nonce 96-bit, tag 128-bit	NIST SP 800-38D
Intercambio de claves clásico	X25519	Curve25519, 256-bit	RFC 7748
KEM post-cuántico	ML-KEM-1024	Kyber-1024, NIST Level 5	NIST FIPS 203
Derivación de claves híbrida	HKDF-SHA256	64B IKM, info=“trashmail-hybrid-kem-v1”	RFC 5869
Compartición de secretos	Shamir SSS	k=2, n=3, GF(2^8)	Shamir (1979)
Codificación de clave de recuperación	BIP39	Entropía 256-bit, 24 palabras	BIP-0039
Derivación de clave de recuperación	HKDF-SHA3-256	Salt vinculado a la cuenta	NIST FIPS 202

Componente	Algoritmo	Parámetros	Estándar
Verificación de clave de recuperación	SHA3-256	Salida 32 bytes	NIST FIPS 202
Cifrado de registros OPAQUE	AES-256-GCM	Cifrado en reposo del lado del servidor	NIST SP 800-38D
Desbloqueo de vault por passkey	WebAuthn PRF	Basado en HMAC, vinculado al hardware	WebAuthn Level 2
Compresión	gzip	Nivel 6	RFC 1952
Bucket Padding	Personalizado	17 tamaños (256B–16MB), magic 0xDEAD	—
Firma de respuestas	Ed25519	Clave 256-bit, firma 512-bit	RFC 8032
Cadena de hashes del archivo	SHA3-256	Hash por bloque, encadenamiento secuencial	NIST FIPS 202
Envoltura de clave del archivo (CAK)	HKDF-SHA256 + AES-256-GCM	Clave de envoltura derivada de contraseña	RFC 5869 / NIST SP 800-38D
Verificación de certificados	SHA-256	Comparación de huella digital de certificado TLS	—
Agrupación de hilos de correo	SHA-256	Hash de Message-ID	NIST FIPS 180-4
Claves de grupo de videollamadas	MLS + X-Wing	Suite `AIONDA_PQ_CALL_V1` (0xFF01), ML-KEM-768 + X25519	RFC 9420 / draft-connolly-cfrg-xwing-kem
Firmas de videollamadas	ML-DSA-65	Dilithium, NIST Level 3	NIST FIPS 204
Medios de videollamadas	SFrame (AES-256-GCM)	Por fotograma, clave del MLS exporter	RFC 9605

20.2 Niveles de seguridad

Algoritmo	Seguridad clásica	Seguridad post-cuántica
X25519	128-bit	Roto por el algoritmo de Shor
ML-KEM-1024	Equivalente a 256-bit	NIST Level 5 (≈AES-256)
AES-256-GCM	256-bit	128-bit (algoritmo de Grover)
SHA-256	256-bit	128-bit (algoritmo de Grover)
SHA3-256	256-bit	128-bit (algoritmo de Grover)
Hybrid KEM (combinado)	128-bit (limitado por X25519)	Level 5 (limitado por ML-KEM)

21. Comparación con otros proveedores

Característica	Aionda Mail	Tuta Mail	Proton Mail
País	Alemania (Stuttgart)	Alemania (Hannover)	Suiza
Zero-Knowledge	Sí (OPAQUE + criptografía del lado del cliente)	Sí	Sí
Post-cuántico	Sí (ML-KEM-1024 + X25519 Hybrid)	Sí (basado en Kyber)	En desarrollo
Protocolo de contraseña	OPAQUE (RFC 9807) — la contraseña nunca sale del navegador	bcrypt (contraseña enviada al servidor sobre TLS)	Basado en SRP
Asunto cifrado	Sí	Sí	No
Cabeceras cifradas	Sí	Parcial	No
Nombres de contactos cifrados	Sí (en el vault)	Sí	No
Direcciones de correo desechables	Sí (función principal, ilimitadas para Plus)	No	Sí (vía SimpleLogin)
Extensión del navegador	Sí (Chrome + Firefox)	No	Vía SimpleLogin
Compartición de carpetas	Sí (Hybrid KEM por destinatario)	Limitado	Sí
Código fuente abierto del cliente	No	Sí	Sí
Auditoría de seguridad	Planificada	Sí	Sí
Recuperación de contraseña	No (por diseño)	No (por diseño)	No (por diseño)
Soporte de passkey	Sí (FIDO2 + PRF)	Sí	Sí
Soporte PGP	Sí (entrante + saliente)	No (protocolo propio)	Sí (OpenPGP)
Archivo de correo conforme a GoBD	Sí (cadena de hashes SHA3-256 + Hybrid KEM)	No	No
Detección MITM (ext. navegador)	Sí (firmas Ed25519 + verificación TLS)	No	No
Perfect Forward Secrecy (API)	Sí (claves efímeras por solicitud)	Desconocido	Desconocido
Ofuscación de tamaño de correo	Sí (Bucket Padding)	Desconocido	No

---

22. Limitaciones y fronteras honestas

22.1 Modelo de confianza de aplicaciones web

Aionda Mail es una aplicación web. En cada carga de página, el navegador descarga JavaScript de nuestros servidores. Un atacante sofisticado que comprometa nuestros servidores podría teóricamente servir JavaScript modificado que exfiltre claves.

Mitigaciones actuales:

- Hashes de Subresource Integrity (SRI) en todas las etiquetas de script
- Cabeceras Content Security Policy (CSP) que restringen las fuentes de scripts
- Todo el código criptográfico crítico está incluido en el paquete principal de la aplicación
- **Extensión del navegador Guardian** (Sección 17): Verificación de firma Ed25519 en todas las respuestas API detecta manipulación del lado del servidor; verificación de certificados TLS (Firefox) detecta proxies MITM

Mitigaciones planificadas:

- Caché de Service Worker para operación offline (reduce la frecuencia de confianza en cada carga)

22.2 Visibilidad de metadatos

Aunque el contenido del correo está completamente cifrado, ciertos metadatos son visibles para el servidor:

- Cuándo se recibieron los correos (marcas de tiempo)
- Qué dirección DEA recibió el correo
- Tamaño aproximado del correo (dentro de los límites del bucket)
- Patrones de actividad de la cuenta

22.3 Registro de procesamiento de correos

Con fines de diagnóstico, Aionda Mail incluye un **registro de procesamiento de correos** opcional que puede almacenar temporalmente el contenido bruto de los correos entrantes. Esta función es configurable por dirección de correo desechable (DEA) y puede activarse o desactivarse en la configuración de la DEA (“Registrar contenido del correo”).

Cuando está activado (configurable por DEA):

- El mensaje SMTP bruto completo (encabezados + cuerpo) se almacena en texto plano en el servidor
- Eliminación automática tras un breve período de retención (menos de 7 días)
- Accesible únicamente para el propietario de la cuenta a través de la API autenticada
- Propósito: solución de problemas de entrega, verificación del reenvío, revisión de decisiones de filtrado de spam

Cuando está desactivado:

- No se almacena ningún contenido de correo en el registro de procesamiento
- Solo se registran metadatos (dirección del remitente, marca de tiempo, estado de entrega)
- El cifrado del vault sigue siendo el único mecanismo de almacenamiento

Importante: Este registro de procesamiento es independiente del vault cifrado. Los correos almacenados en el vault siempre están cifrados con Hybrid KEM, independientemente de la

configuración del registro. El registro de procesamiento existe como una función heredada del sistema de reenvío de correos y proporciona transparencia operativa. Los usuarios que requieran almacenamiento estrictamente Zero-Knowledge para todos los correos deben desactivar esta opción.

22.4 Seguridad del correo externo

Los correos enviados a o recibidos de direcciones que no son de Aionda viajan a través de la infraestructura estándar de correo (SMTP). Aunque se almacenan cifrados en el vault, el contenido del correo fue visible durante el tránsito a menos que se haya utilizado cifrado PGP.

22.5 Sin custodia de claves

No existe clave maestra, puerta trasera ni mecanismo de recuperación disponible para Aionda GmbH. Si un usuario pierde su contraseña y todos los métodos de recuperación, sus datos se pierden permanentemente. Esta es una decisión de diseño intencional que demuestra la integridad del modelo Zero-Knowledge.

---

23. Hoja de ruta

Hito	Estado	Objetivo
Arquitectura Zero-Knowledge	Completado	—
Hybrid KEM post-cuántico (ML-KEM-1024)	Completado	—
Autenticación OPAQUE (RFC 9807)	Completado	—
Shamir Secret Sharing (2-de-3)	Completado	—
Capa de transporte API cifrada	Completado	—
Soporte Passkey/WebAuthn PRF	Completado	—
Calendario cifrado de extremo a extremo	Completado	—
Archivo de correo conforme a GoBD (Blockchain)	Completado	—
Protección MITM Guardian (Ed25519)	Completado	—
Verificación de certificados TLS (Firefox)	Completado	—

Historial del documento

Versión	Fecha	Cambios
1.0	Marzo 2026	Publicación inicial
1.1	Abril 2026	Nuevo capítulo 10: Vault Drive (arquitectura de clave por archivo, compartición mediante re-envoltura de claves)

Versión	Fecha	Cambios
1.2	Abril 2026	Sección 10.11: Compartición externa — enlaces públicos con sobres KEM híbridos, flujo unlock_token, modo de contraseña Argon2id, fragmento URL para link_only, cadena de auditoría inalterable (EXT_SHARE_*) y distribución del enlace elegida por el propietario
1.3	Mayo 2026	Nuevo capítulo 11: Aionda Chat — mensajería en tiempo real E2EE post-cuántica vía AAR (Aionda Async Ratchet), una variante híbrida X25519 + ML-KEM-1024 del Signal X3DH + Double Ratchet. Forward secrecy por mensaje, post-compromise security e integración con la cadena de auditoría Enterprise. Capítulos siguientes renumerados 12-22. Traducciones PT-BR y PT-PT añadidas. Bug de doble numeración en el renderizado del PDF corregido (numeración automática de LaTeX desactivada; los números de capítulo provienen ahora únicamente de los encabezados).
1.4	junio de 2026	Nuevo capítulo 12: Videollamadas Aionda — videoconferencia grupal E2EE poscuántica. Acuerdo de claves de grupo MLS (RFC 9420) con la suite criptográfica `AIONDA_PQ_CALL_V1` (X-Wing = ML-KEM-768 + X25519, firmas ML-DSA-65), cifrado de medios fotograma a fotograma AES-256-GCM SFrame (RFC 9605) con clave derivada del exportador MLS, vinculación SAS y un relé SFU de conocimiento cero. Capítulos siguientes renumerados 13-23.

Contacto

Aionda GmbH Stephan Ferraro Stuttgart, Alemania

Email: contact-46epp9ba@contact.aionda.com Web: https://mail.aionda.com

Este documento describe la arquitectura de seguridad de Aionda Mail a fecha de junio de 2026. Los sistemas criptográficos evolucionan — este documento se actualizará conforme la arquitectura cambie.