# Aionda Mail Security Whitepaper

Zero-Knowledge · Post-Quantum · Made in Germany

**Version 1.8** — 2026-10-13

**Aionda GmbH**
Stuttgart, Germany

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

PUBLIC DOCUMENT

## Contents

1. Resumo executivo . . . 3
2. Modelo de ameaças . . . 3
3. Visão geral da arquitetura . . . 4
4. Autenticação Zero-Knowledge (OPAQUE) . . . 6
5. Chave mestra do cofre & Shamir Secret Sharing . . . 8
6. KEM híbrida pós-quântica . . . 9
7. Pipeline de encriptação de e-mail . . . 10
8. Camada de transporte API encriptada . . . 12
9. Criptografia da partilha de pastas . . . 14
10. Vault Drive — Armazenamento de documentos encriptado . . . 15
11. Aionda Chat — Mensagens E2EE pós-quânticas . . . 21
12. Chamadas de vídeo Aionda — Videoconferência E2EE pós-quântica . . . 26
13. Tarefas Aionda - Gestão de tarefas encriptada de ponta a ponta . . . 31
14. Proteções contra canais auxiliares . . . 34
15. Gestão e ciclo de vida das chaves . . . 34
16. Mecanismo de recuperação . . . 35
17. Autenticação sem palavra-passe (Passkeys) . . . 37
18. Guardian: Proteção MITM & assinatura de respostas . . . 37
19. Arquivo empresarial de e-mail (Blockchain) . . . 41
20. O que o servidor vê — e o que não vê . . . 44
21. Referência de algoritmos . . . 45
22. Comparação com outros fornecedores . . . 47
23. Limitações & fronteiras honestas . . . 48
24. Plano futuro . . . 49
25. Iniciar sessão num dispositivo por código QR . . . 49
Histórico do documento . . . 50
Contacto . . . 52

## 1. Resumo executivo

A Aionda Mail é um serviço de e-mail encriptado zero-knowledge e pós-quântico, operado pela Aionda GmbH em Estugarda, Alemanha. O serviço combina endereços de e-mail descartáveis (DEAs) com uma caixa de correio totalmente encriptada — uma combinação que nenhum outro fornecedor oferece.

**Propriedades centrais de segurança:**

- **Arquitetura Zero-Knowledge:** A decifragem do conteúdo armazenado na caixa de correio é efetuada no dispositivo do utilizador, no navegador ou na aplicação nativa.
- **Segurança pós-quântica:** O Hybrid Key Encapsulation Mechanism (X25519 + ML-KEM-1024) protege todos os dados contra ataques clássicos e contra ataques de computadores quânticos.
- **Autenticação Zero-Knowledge:** O protocolo OPAQUE (RFC 9807) garante que as palavras-passe nunca são transmitidas ao servidor nem aí armazenadas — nem sequer como hashes.
- **Shamir Secret Sharing (2 em 3):** A chave mestra do cofre é dividida em três partes protegidas por palavra-passe, passkey e chave de recuperação. Quaisquer duas partes reconstroem a chave mestra.
- **Perfect Forward Secrecy:** Cada pedido à API utiliza um par de chaves criptográficas único e descartável. Comprometer um pedido não afeta nenhum outro.
- **Proteção MITM (Guardian):** A extensão de navegador verifica de forma independente todas as respostas do servidor via assinaturas Ed25519 e deteta ataques man-in-the-middle através da verificação do certificado TLS — mesmo contra proxies empresariais e CDNs comprometidas.
- **Arquivo de e-mail conforme GoBD:** Contas Enterprise beneficiam de uma cadeia de hash anti-falsificação (blockchain SHA3-256) com conteúdo encriptado de ponta a ponta (Hybrid KEM), trilho de auditoria completo, retenção legal e prazos de retenção configuráveis — conforme à regulamentação alemã GoBD.
- **Sem recuperação de palavra-passe:** Se a palavra-passe e todos os métodos de recuperação forem perdidos, os dados ficam irrecuperáveis. Isto é intencional — prova que o servidor não consegue aceder aos dados do utilizador.

**Jurisdição:** Direito alemão (DSGVO/GDPR), sem partilha de dados com serviços de informação estrangeiros.

---

## 2. Modelo de ameaças

### 2.1 Aquilo contra que a Aionda Mail protege

| Ameaça | Proteção |
|---|---|
| **Comprometimento do servidor** (fuga de base de dados, acesso interno) | Todo o conteúdo de e-mail encriptado com chaves que o servidor nunca possui |
| **Espionagem de rede** (ISP, Wi-Fi, CDN) | Transporte API encriptado de ponta a ponta via Hybrid KEM |

| Ameaça | Proteção |
|---|---|
| **Inspeção CloudFlare** | Os pedidos à API são encriptados antes de sair do navegador; a CloudFlare vê apenas ciphertext. A extensão Guardian deteta adulteração de respostas via assinaturas Ed25519 |
| **Proxies MITM corporativos** (ZScaler, Fortinet, etc.) | A extensão Guardian deteta certificados de proxy via lista de bloqueio de emissores (Firefox) |
| **Ataques de computadores quânticos** ("recolher agora, decifrar depois") | ML-KEM-1024 (NIST FIPS 203) oferece resistência pós-quântica |
| **Roubo da base de dados de palavras-passe** | O OPAQUE armazena apenas registos criptográficos, não hashes de palavras-passe |
| **Ataques de força bruta offline a palavras-passe** | O OPAQUE impede ataques offline; o rate limiting do lado do servidor impede ataques online |
| **Análise de tamanho de e-mail** | O bucket padding oculta os tamanhos reais dos e-mails |
| **Canais auxiliares de compressão** (CRIME/BREACH) | Bucket padding aplicado após a compressão |
| **Enumeração de utilizadores** | Respostas falsas determinísticas para contas inexistentes |

### 2.2 Aquilo contra que a Aionda Mail NÃO protege

| Limitação | Explicação |
|---|---|
| **Dispositivo comprometido** | Se o malware controlar o navegador, pode ler o conteúdo já decifrado |
| **Metadados para destinatários externos** | E-mails para Gmail/Outlook circulam em claro depois de saírem dos nossos servidores (a menos que se use PGP) |
| **Metadados de e-mail nos nossos servidores** | Timestamps, endereços IP e tamanhos de e-mails encriptados são visíveis para o servidor |
| **Confiança na entrega web** | O navegador transfere JavaScript dos nossos servidores em cada visita (ver Secção 20 para mitigações) |
| **Criptoanálise sob coação física** | Nenhum sistema criptográfico protege contra coerção física |

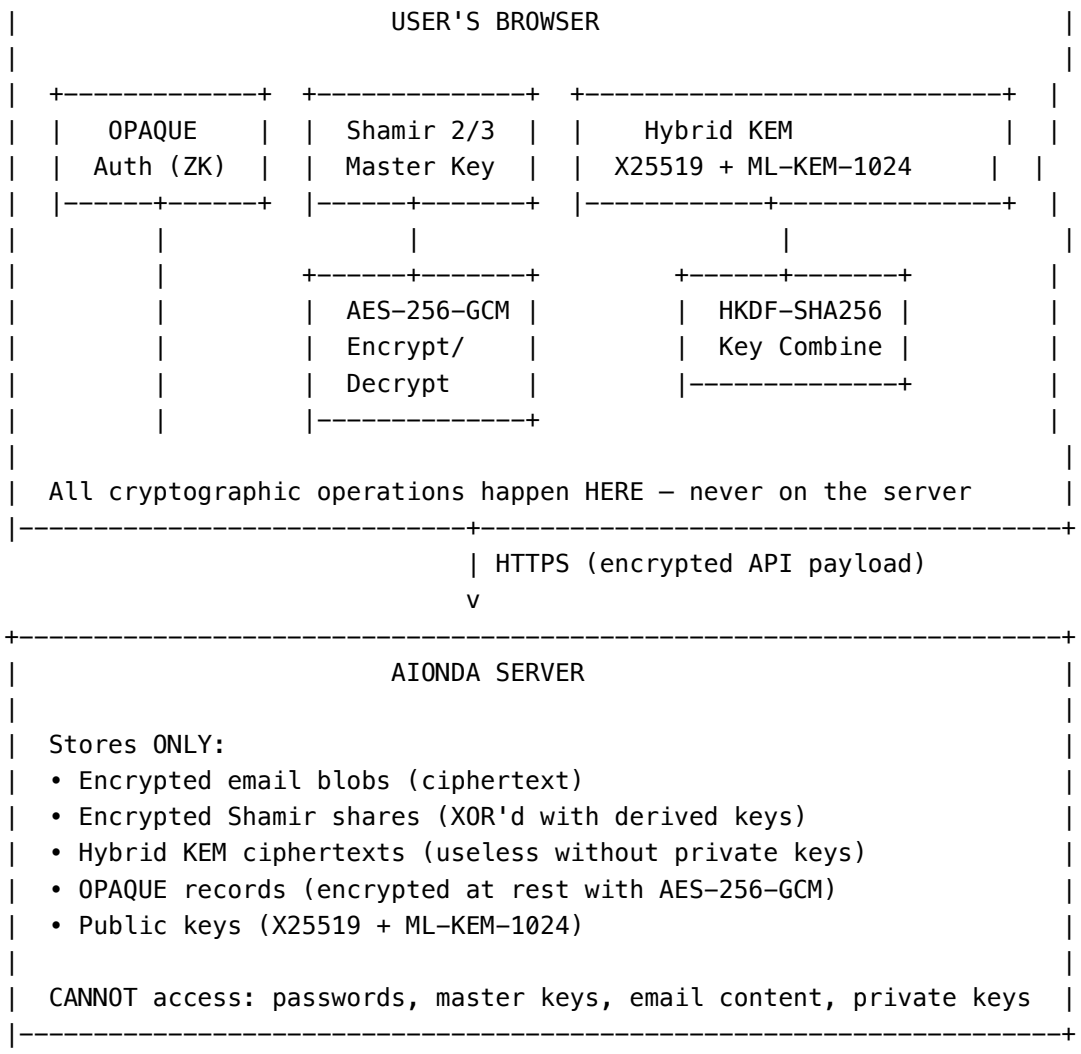
### 2.3 Princípio de design

A Aionda Mail segue o **modelo do "servidor honesto"**: o sistema é concebido para que mesmo um servidor totalmente comprometido — ou um operador malicioso — não consiga decifrar os dados do utilizador. A segurança não depende de confiar em nós. Depende de matemática.

---

## 3. Visão geral da arquitetura

+----------------------------------------------------------------------+




### 3.1 Visão geral dos componentes

| Componente | Função | Localização |
|---|---|---|
| OPAQUE (RFC 9807) | Autenticação de palavra-passe zero-knowledge | Cliente + Servidor |
| Shamir Secret Sharing | Proteção da chave mestra do cofre (limiar 2 em 3) | Apenas cliente |
| Hybrid KEM (X25519 + ML-KEM-1024) | Encapsulação de chave pós-quântica para e-mails | Cliente + Servidor |
| AES-256-GCM | Encriptação simétrica autenticada | Cliente + Servidor |
| HKDF-SHA256 | Derivação de chave a partir de segredos partilhados híbridos | Cliente + Servidor |
| BIP39 | Codificação da chave de recuperação (mnemónica de 24 palavras) | Apenas cliente |

| Componente | Função | Localização |
|---|---|---|
| WebAuthn PRF | Desbloqueio do cofre baseado em passkey | Apenas cliente |
| Bucket Padding | Proteção contra canais auxiliares | Cliente + Servidor |

### 3.2 Clientes nativos para iOS e macOS

A Aionda Mail disponibiliza aplicações nativas para iOS (iPhone e iPad) e macOS. Ambas utilizam uma base de código Swift/SwiftUI partilhada, com pontos de entrada e adaptações da interface específicos de cada plataforma. O pacote partilhado `AiondaMailCore` implementa as funções centrais, enquanto `AiondaMailUI` fornece a interface SwiftUI.

A interface nativa é distribuída como parte da aplicação instalada. Distingue-se assim do cliente do navegador, que carrega o código da aplicação através da web. As referências às API do navegador, a JavaScript e a sessionStorage neste documento descrevem a implementação web; não definem o armazenamento local nem o ambiente de execução dos clientes nativos.

Nativo não significa que todos os componentes dispensem tecnologias web: vistas WebKit integradas apresentam e-mails HTML, permitem redigi-los e fornecem a interface multimédia das videochamadas. A vista de leitura HTML desativa o JavaScript do conteúdo e utiliza um armazenamento de dados de sites não persistente. Estas definições de leitura não devem ser generalizadas ao editor nem às vistas de videochamada, que necessitam de JavaScript.

A arquitetura nativa, por si só, não garante a segurança. Continua a ser necessário confiar na aplicação instalada, nas suas atualizações e no sistema operativo; um dispositivo comprometido pode aceder ao conteúdo decifrado. A distribuição web continua a fazer parte da fronteira de confiança dos componentes integrados que carregam conteúdo web.

---

## 4. Autenticação Zero-Knowledge (OPAQUE)

### 4.1 Porque não hash de palavras-passe?

Os serviços tradicionais armazenam hashes de palavras-passe (bcrypt, Argon2). Embora melhor que texto em claro, esta abordagem tem fragilidades fundamentais:

- O servidor vê a palavra-passe durante o login (mesmo que apenas brevemente em RAM)
- Os hashes de palavras-passe podem ser atacados por força bruta offline se a base de dados for roubada
- O servidor pode ser modificado para registar palavras-passe

**O OPAQUE elimina os três problemas.** A palavra-passe nunca sai do navegador — nem em claro, nem como hash, nem em qualquer forma.

### 4.2 Como funciona o OPAQUE

O OPAQUE (RFC 9807) é um protocolo aPAKE (Asymmetric Password-Authenticated Key Exchange). Utiliza um mecanismo criptográfico de challenge-response onde o servidor consegue verificar que o utilizador conhece a palavra-passe correta sem nunca saber qual é essa palavra-passe.

**Registo (uma única 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)
```

**Login (em cada sessão):**

```
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)
```

**Propriedades centrais:**

- A palavra-passe é verificada **no lado do cliente** no passo 3 — o servidor nunca a vê
- O servidor armazena um **registo OPAQUE**, que não é um hash de palavra-passe e não pode ser atacado offline
- Os registos OPAQUE são ainda **encriptados em repouso** com AES-256-GCM usando uma chave do lado do servidor
- **Proteção contra enumeração de utilizadores**: Contas inexistentes recebem respostas falsas determinísticas com timing idêntico
- **Rate limiting**: Máximo 3 tentativas de autenticação por sessão, timeout de sessão de 120 segundos

### 4.3 Implementação

- **Biblioteca:** `@serenity-kit/opaque` (baseada em WASM, qualidade de produção)
- **Componente do servidor:** Microsserviço dedicado para operações criptográficas OPAQUE
- **Formato Base64:** base64url (URL-safe, sem padding) para compatibilidade do protocolo
- **Registo de auditoria:** Todos os eventos de autenticação são registados com timestamps e endereços IP

### 4.4 Migração SRP

Legacy accounts using SRP-6a are automatically migrated to OPAQUE upon next login. After migration, the SRP verifier is permanently deleted. Migration is one-way — accounts cannot revert to SRP.

---

## 5. Chave mestra do cofre & Shamir Secret Sharing

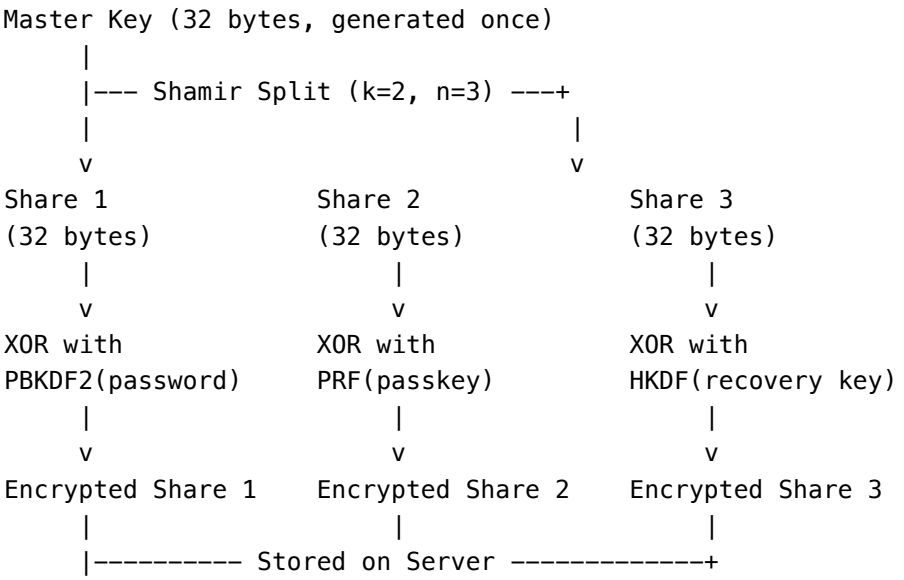
### 5.1 Geração da chave mestra

When a user activates the encrypted mailbox, a **256-bit (32-byte) master key** is generated using the browser's cryptographically secure random number generator (`crypto.getRandomValues`).

Esta chave mestra é a raiz de toda a encriptação. Nunca sai do navegador em claro. Nunca é armazenada em lado nenhum — nem no navegador, nem no servidor, nem em qualquer forma.

### 5.2 Shamir Secret Sharing (2 em 3)

A chave mestra é dividida em três partes usando o esquema **Shamir's Secret Sharing** sobre o corpo de Galois GF(2^8) com o polinómio irredutível do AES (x^8 + x^4 + x^3 + x + 1).




**Threshold property:** Any 2 of the 3 shares are sufficient to reconstruct the master key via Lagrange interpolation. The server stores only the encrypted shares — and cannot decrypt any of them.

### 5.3 Proteção das partes

Cada parte é combinada (XOR) com uma chave derivada de um factor de autenticação diferente:

| Share | Protected By | Key Derivation |
|---|---|---|
| Share 1 | Password | PBKDF2-SHA256, 600,000 iterations, 32-byte random salt |
| Share 2 | Passkey (FIDO2) | WebAuthn PRF extension, hardware-bound |
| Share 3 | Recovery Key | HKDF-SHA3-256 with account-bound salt |

**Reconstruction scenarios:**

- **Normal login:** Password (Share 1) + Passkey (Share 2) -> Master Key
- **Lost passkey:** Password (Share 1) + Recovery Key (Share 3) -> Master Key

- **Password change:** Passkey (Share 2) + Recovery Key (Share 3) -> Master Key

### 5.4 Proteção do session storage

Even within a browser session, the master key is never stored in plaintext:

1. An ephemeral AES-256 key is generated
2. The master key is encrypted with this ephemeral key
3. Only the encrypted blob is placed in `sessionStorage`
4. The ephemeral key exists only in JavaScript memory (garbage-collected on tab close)

---

## 6. KEM híbrida pós-quântica

### 6.1 Porquê pós-quântico?

Quantum computers running Shor's algorithm could break classical public-key cryptography (RSA, ECDH, X25519) in polynomial time. While large-scale quantum computers do not yet exist, the threat of **"harvest now, decrypt later"** attacks is real: adversaries could store encrypted data today and decrypt it when quantum computers become available.

### 6.2 Abordagem híbrida

A Aionda Mail utiliza um **Hybrid Key Encapsulation Mechanism** que combina:

- **X25519** (Curve25519 ECDH) — proven classical security, 128-bit security level
- **ML-KEM-1024** (NIST FIPS 203, formerly Kyber-1024) — post-quantum security, NIST Security Level 5

A abordagem híbrida fornece **defesa em profundidade**: a chave combinada é segura desde que **pelo menos um** dos dois algoritmos permaneça intacto.

### 6.3 Processo de encapsulação

```
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 Processo de decapsulação

```
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 Tamanhos das chaves

| Parameter | Size | Standard |
|---|---|---|
| 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` (audited, pure JavaScript implementation)
- **X25519:** WebCrypto API (`crypto.subtle.deriveBits`)
- **HKDF:** `@noble/hashes` (RFC 5869 compliant)

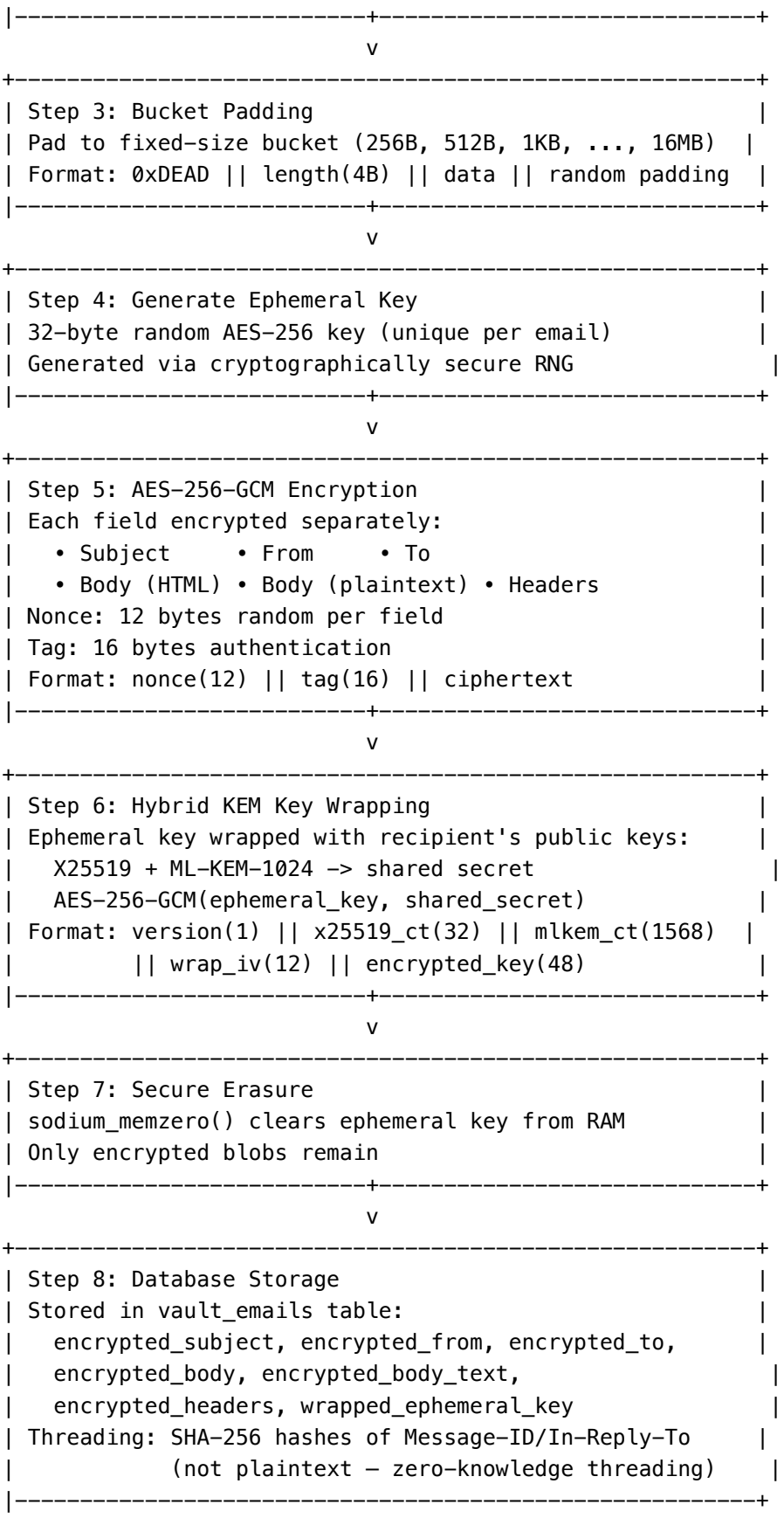
---

## 7. Pipeline de encriptação de e-mail

### 7.1 Incoming Email (SMTP -> Encrypted Storage)

Quando um e-mail chega ao servidor SMTP da 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.         |
```


Step 3: Bucket Padding
Pad to fixed-size bucket (256B, 512B, 1KB, ..., 16MB)
Format: 0xDEAD || length(4B) || data || random padding
Step 4: Generate Ephemeral Key
32-byte random AES-256 key (unique per email)
Generated via cryptographically secure RNG
Step 5: AES-256-GCM Encryption
Each field encrypted separately:
• Subject • From • To
• Body (HTML) • Body (plaintext) • Headers
Nonce: 12 bytes random per field
Tag: 16 bytes authentication
Format: nonce(12) || tag(16) || ciphertext
Step 6: Hybrid KEM Key Wrapping
Ephemeral key wrapped with recipient's public keys:
X25519 + ML-KEM-1024 -> shared secret
AES-256-GCM(ephemeral_key, shared_secret)
Format: version(1) || x25519_ct(32) || mlkem_ct(1568)
|| wrap_iv(12) || encrypted_key(48)
Step 7: Secure Erasure
sodium_memzero() clears ephemeral key from RAM
Only encrypted blobs remain
Step 8: Database Storage
Stored in vault_emails table:
encrypted_subject, encrypted_from, encrypted_to,
encrypted_body, encrypted_body_text,
encrypted_headers, wrapped_ephemeral_key
Threading: SHA-256 hashes of Message-ID/In-Reply-To
(not plaintext — zero-knowledge threading)

### 7.2 Leitura de e-mail (decifragem no navegador)

O processo inverso acontece inteiramente no navegador:

1. Obter o e-mail encriptado do servidor via API encriptada
2. Analisar a chave efémera encapsulada (extrair ciphertext X25519 + ciphertext ML-KEM)
3. **Decapsulação Hybrid KEM** usando as chaves privadas do cofre -> segredo partilhado
4. Desencapsular a chave efémera AES-256
5. **Decifragem AES-256-GCM** de cada campo (assunto, de, para, corpo, cabeçalhos)
6. Reordenar bytes: formato do servidor (nonce || tag || ct) -> formato WebCrypto (nonce || ct || tag)
7. Remover bucket padding (detetar magic bytes 0xDEAD)
8. Descomprimir se for gzip (detetar magic bytes 0x1F8B)
9. Descodificar UTF-8 para texto em claro

### 7.3 Envio de e-mail

Ao redigir e enviar um e-mail:

1. O cliente encripta o conteúdo do e-mail com um protocolo de challenge-response
2. O servidor recebe o payload encriptado, decifra-o efemeramente (apenas em memória), envia via SMTP
3. O servidor devolve os cabeçalhos MIME gerados ao cliente (encriptados)
4. O cliente encripta uma cópia com a chave mestra do cofre e armazena na pasta Enviados
5. O texto em claro efémero do lado do servidor é imediatamente descartado — nunca escrito no disco

### 7.4 Anexos

Cada anexo é encriptado independentemente:

- Chave efémera AES-256 separada por anexo
- Encapsulação Hybrid KEM separada por anexo
- Nome do ficheiro e tipo MIME encriptados separadamente
- Sem compressão para formatos já comprimidos (JPEG, ZIP, PDF, etc.)

### 7.5 Threading de e-mails (Zero-Knowledge)

O threading de e-mails (agrupar e-mails relacionados) usa apenas **hashes SHA-256** dos cabeçalhos Message-ID e In-Reply-To. O servidor nunca vê as strings reais dos Message-ID — pode agrupar e-mails por igualdade de hash sem conhecer o conteúdo.

---

## 8. Camada de transporte API encriptada

### 8.1 Problema

Mesmo com HTTPS, certos intermediários podem inspecionar tráfego:

- A **CloudFlare** (CDN/proteção DDoS) termina o TLS e pode ver pedidos em claro
- **Proxies empresariais** podem realizar inspeção TLS
- **Parâmetros de API** (como `?cmd=read_email&id=123`) revelam metadados

### 8.2 Solução: API encriptada de ponta a ponta

Toda a comunicação de API é adicionalmente encriptada de ponta a ponta entre o navegador e o servidor de aplicação, dentro do 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 Propriedades centrais

- **One-time use:** Each API keypair is used exactly once, then permanently invalidated
- **Perfect Forward Secrecy:** Compromising one request key does not affect any other request
- **Session-bound:** Keys are claimed by a specific session and cannot be reused by another
- **Key pool:** Server maintains approximately 100,000 pre-generated keypairs
- **Auto-refetch:** Client automatically requests new keys when cache drops below 10
- **Key TTL:** Claimed keys expire after 24 hours
- **Bidirectional:** Both request AND response are encrypted — the server never returns plaintext

### 8.4 O que a CloudFlare vê

With this architecture, CloudFlare (or any TLS-terminating proxy) sees only:

- `POST /e` — a single, opaque endpoint

- A binary blob of encrypted data
- No API command names, no parameters, no email IDs, no user data

---

## 9. Criptografia da partilha de pastas

### 9.1 Modelo de partilha

Users can share encrypted folders with other Aionda Mail users. The sharing mechanism uses the Hybrid KEM to encrypt a folder-specific key for each recipient.

### 9.2 Fluxo de partilha

```
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 permissões

| Permission | Capability |
|---|---|
| `readonly` | Read folder emails (decrypt only) |

| Permission | Capability |
|---|---|
| `einliefern` | Submit new emails into folder |
| `bearbeiten` | Edit folder contents |
| `antworten` | Reply to emails within folder |
| `vollzugriff` | Full access including ownership transfer |

### 9.4 Cópia entre cofres (re-encapsulação)

When a recipient copies an email from a shared folder to their own vault, the email key must be **re-wrapped** for their own Hybrid KEM keypair. This is performed entirely client-side:

1. Decrypt shared folder key using recipient's private keys
2. Decrypt email's ephemeral key using folder key
3. Re-encapsulate ephemeral key with recipient's own public keys
4. Store re-wrapped copy in recipient's vault

O servidor facilita a transferência mas nunca vê material de chave em claro.

---

## 10. Vault Drive — Armazenamento de documentos encriptado

Vault Drive is the zero-knowledge document storage of Aionda Mail. Unlike emails, which use per-message ephemeral keys, Drive uses a **per-file-key architecture**: every document receives its own 32-byte AES-256-GCM key, generated client-side and wrapped with the master key.

### 10.1 Porquê chaves por ficheiro?

A arquitetura de chave por ficheiro oferece três vantagens fundamentais:

1. **Granular sharing:** Individual documents can be shared without exposing the master key. The server simply re-wraps the file key for the recipient — the content remains unchanged.
2. **Compromise isolation:** If a single file key is compromised (e.g. via a shared link), all other documents remain protected.
3. **Efficient sharing without re-encryption:** When sharing, the content does not need to be re-encrypted — all recipients read the same ciphertext with a re-wrapped key.

### 10.2 Fluxo de carregamento

```
Client                                                  Server
------                                                  ------

1. Generate random file key:
   fileKey = randomBytes(32)

2. Encrypt content, name, MIME type:
   encContent = AES-256-GCM(content, fileKey, nonce1)
   encName    = AES-256-GCM(name,    fileKey, nonce2)
   encMime    = AES-256-GCM(mime,    fileKey, nonce3)
```

```
3. Wrap file key with master key:
   wrappedKey = AES-256-GCM(fileKey, masterKey, nonce4)

4. Send encrypted package:
   POST /e {
     encContent, encName, encMime,
     wrappedKey, wrappedKeyNonce
   }
                                                    5. Store in vault_files + vault_file_chunk
                                                       (pure ciphertexts, no key material)
```

### 10.3 Fluxo de transferência

```
1. Fetch chunk + wrapped_key from server
2. Unwrap file key:
   fileKey = AES-256-GCM-decrypt(wrappedKey, masterKey, nonce)
3. Decrypt content, name, MIME:
   content = AES-256-GCM-decrypt(encContent, fileKey, nonce1)
```

Decryption happens inside a **Web Worker** that holds the master key only temporarily in memory. After the operation, the memory is overwritten.

### 10.4 Partilha — apenas re-encapsulação de chave

A vantagem central das chaves por ficheiro mostra-se na partilha: o conteúdo **não** é re-encriptado. Apenas a chave de ficheiro de 32 bytes é re-encapsulada usando a Hybrid KEM do destinatário.

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

1. Unwrap file key:
   fileKey = AES-GCM-decrypt(wrappedKey, masterKey)

2. Fetch recipient's public keys:
   recipient.x25519_pub, recipient.mlkem_pub

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

4. Wrap file key with sharedSecret:
   shareWrappedKey = AES-256-GCM(fileKey,
                                 sharedSecret, nonce)

5. Store share record:
   INSERT INTO vault_file_shares {
     file_uuid, recipient_account_id,
     x25519_ephemeral, mlkem_ciphertext,
     share_wrapped_key, permissions
   }
```

```
6. Fetch share record + chunk
7. Hybrid KEM decapsulation:
   hybridDecapsulate(x25519_ct, mlkem_ct,
     own_x25519_priv, own_mlkem_priv)
   -> sharedSecret
8. Unwrap file key:
   fileKey = AES-GCM-decrypt(
     shareWrappedKey, sharedSecret)
9. Decrypt SAME ciphertext:
   content = AES-GCM-decrypt(
     encContent, fileKey)
```

O proprietário não precisa da chave mestra do destinatário — apenas das chaves públicas Hybrid KEM, que o servidor armazena em claro.

### 10.5 Partilha de pastas (recursiva)

When a folder is shared, the client processes all contained documents and subfolders recursively. Each file receives its own share record containing the respective document's file key, re-wrapped for the recipient. The folder itself has no folder key — the structure is linked via parent UUIDs.

**Important:** The KEM encapsulation step is performed **per operation**, not per file. All files in a batched share use the same `sharedSecret` — making the operation efficient without reducing security (each file still has its own file key).

### 10.6 Modelo de permissões

| Permission | Can perform |
|---|---|
| **Read** | Download, preview, read metadata |
| **Full access** | + rename, upload new version, move inside the shared folder, upload new documents |
| **Not possible** | Delete (owner only), move out of shared folder (owner only) |

Permissions are stored in the `vault_file_shares` record and enforced server-side. Cryptography protects the content — access control protects the operations.

### 10.7 Renomeação, sobrescrita e atualização de metadados

For shared documents, rename and overwrite operations are performed directly on the `vault_files` record — not on individual share records. All recipients immediately see the new name or content, as they reference the same ciphertext.

On **overwrite** (new version), a **new file key** is generated, the old ciphertext is replaced, and all existing share records are re-wrapped client-side with the new key. The owner must load the public keys of all recipients for this operation.

### 10.8 Pré-visualização e decifragem em memória

Images, PDFs and other documents are decrypted **exclusively in memory** for preview. There is no disk cache with plaintext data. The client uses the **VaultDataLayer**, an encrypted IndexedDB cache that persistently stores ciphertexts and only decrypts them on demand.

Thumbnails are **never** generated server-side — the server never sees content. Thumbnails are generated client-side from the decrypted original and cached encrypted as well.

### 10.9 O que o servidor vê

| Visible to server | Not visible |
|---|---|
| Encrypted content (random bytes) | Plaintext content |
| Encrypted filename (random bytes) | Plaintext filename |
| Encrypted MIME type (random bytes) | File type (image, PDF, text…) |
| File size (padded to bucket boundaries) | Actual original size |
| Upload timestamp | File key, master key |
| Owner account ID | Contents of subfolders |
| Parent folder UUID (for structure) | Folder names (also encrypted) |
| Share records with KEM ciphertexts | Recipient master key, unwrapped file key |

### 10.10 Aplicação de quota

Quota enforcement happens server-side based on the **encrypted file size** (including bucket padding). The server does not know the actual plaintext size — the padding overhead is intentionally paid by the user to prevent size leakage.

Limits: - **Free:** 100 MB total storage, max. 10 MB per document - **Plus:** 5 GB total storage, max. 100 MB per document

### 10.11 Partilha externa — ligações públicas

Vault Drive documents can be shared with recipients who **do not have an Aionda Mail account** via public share links served from the isolated domain `mail.aionda.com`. The feature preserves zero-knowledge by treating each share as an **independent crypto envelope**, decoupled from the owner's vault master key.

**Protection modes:**

| Mode | Trust factor | Use case |
|---|---|---|
| `link_only` | URL fragment (never sent to server) | Convenience — anyone with the link can access |
| `password` | Argon2id-derived key | Out-of-band password transfer (SMS, phone call) |
| `recipient_pubkey` | Hybrid KEM (X25519 + ML-KEM-1024) | Post-quantum secure when recipient has pre-registered public keys |

#### 10.11.1 Share creation (owner client)

```
1. fileKey := AES-GCM-decrypt(vault_files.wrapped_key, masterKey)
2. shareKey := randomBytes(32)                          // ephemeral, per share
3. wrappedFileKey := AES-GCM(fileKey, shareKey)         // new envelope
4. unlockVerifier := HMAC-SHA256(shareKey,
                     "unlock:" || shareUuid)            // proof-of-knowledge
```

```
5. If password protection:
     salt := randomBytes(16)
     pwKey := Argon2id(password, salt, t=3, m=64MB, p=1)
     wrappedShareKey := AES-GCM(shareKey, pwKey)
     -> Link: https://mail.aionda.com/s/<shareUuid>
   Else (link_only):
     -> Link: https://mail.aionda.com/s/<shareUuid>#<base64(shareKey)>
       (URL fragment — browsers never transmit fragments to the server)

6. Encrypted metadata (per-share, with shareKey):
     encFilename, encMime, encMessage  // all AES-GCM
7. POST /e  (encrypted API transport, see §8)
   -> server stores share record — never sees plaintext
```

**Link delivery is user-chosen, not routed through Aionda Mail.** After creation, the owner decides which channel transports the link: copy-to-clipboard, QR code, a pre-filled `mailto:` draft opened in the user's own mail client, Signal, SMS, AirDrop, or any other out-of-band channel. Aionda Mail never sees, sends, or logs the outgoing delivery — the server only learns which share-UUIDs exist. This matters for two reasons: (a) the owner can select the most trusted channel available to them (a corporate compliance policy, a preferred messenger, a verified face-to-face QR scan), and (b) for `password`-protected shares it enables **channel separation** — link via channel A, password via channel B — so a single compromised channel never grants access on its own.

**10.11.2 Share access (recipient, no account required)** The recipient visits `https://mail.aionda.com/s/<s`
The Share Page is a **separate bundle** from the Manager, with its own reproducible build, Subresource Integrity manifest, and a `/.well-known/integrity.json` endpoint for offline verification.

```
1. Share Page bootstrap (client generates ephemeral hybrid KEM keypair)
2. share_fetch_meta -> { wrapped_file_key, salt?, encrypted_message, ... }
3. Unlock phase — produces a one-time unlock_token:
     password mode:    server verifies Argon2id derivation -> unlock_token
     link_only mode:   client computes HMAC-SHA256(shareKey, "unlock:"||uuid)
                       server verifies against stored unlockVerifier
                       -> unlock_token
4. fileKey := AES-GCM-decrypt(wrappedFileKey, shareKey)
5. share_download_chunk (per chunk, unlock_token required)
6. Client hashes plaintext, calls share_confirm_download
```

The `unlock_token` unifies the flow across all three protection modes and enables centralized rate-limiting and audit-logging — link-only access requires proof of knowledge of the fragment, so bots and crawlers without the fragment cannot issue downloads.

**10.11.3 Enforced policies** Every share must declare the following at creation time (all enforced server-side):

- **`expires_at`** — mandatory, default 7 days, maximum 90 days
- **`max_downloads`** — mandatory counter, default 10

- **Rate limiting** — Argon2 attempts on password shares are throttled per IP hash and per share
- **`revoked_at`** — the owner can revoke any share with a single click; future unlock attempts and downloads return a unified "share unavailable" response (same error as expired or exhausted — no enumeration oracle)

**10.11.4 Tamper-evident audit chain** All relevant access events are appended to the existing `enterprise_audit_log` hash chain (SHA3-256, see §18.2). The following action types are reserved for external sharing:

`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`.

IP and user-agent hashes use a **daily rotating salt** (`vault_drive_external_share_daily_salts`, pruned after 30 days) — historical hashes become non-reversible after salt rotation for GDPR compliance, while short-term correlation remains available for the owner.

**10.11.5 Key rotation (share rewrap)** The owner can rotate a share at any time without re-uploading the content:

```
1. new_shareKey := randomBytes(32)
2. new_wrappedFileKey := AES-GCM(fileKey, new_shareKey)
3. INSERT new share row; old.linked_to_uuid := new.share_uuid
4. old.revoked_at := NOW()
```

Recipients using the old link receive "share unavailable". The owner forwards the new link. All access events from both shares remain linked via `linked_to_uuid` in the audit chain.

**10.11.6 What revocation does — and does not** Revocation **stops future server-side delivery** of the share. It does **not** retroactively revoke share keys, file keys, or chunks already delivered. A recipient who has already downloaded the plaintext — or who captured the link fragment — can continue to use that material locally. For use cases with higher assurance requirements, combine: short `expires_at`, low `max_downloads`, password protection, and the Guardian browser extension for the recipient.

**10.11.7 What the server sees**

| Visible to server | Not visible |
|---|---|
| `share_uuid`, `file_uuid`, `account_id` (owner) | Plaintext filename, MIME type, content, message |
| `wrapped_file_key`, `wrapped_share_key` (ciphertexts) | `share_key` (link-only: lives only in the URL fragment, client-side) |
| `unlock_verifier` (HMAC output, not reversible) | Password, Argon2-derived key |
| `protection_mode`, `expires_at`, `max_downloads`, `allow_preview` | Recipient identity (only salted IP/UA hash, pruned after 30 days) |
| `encrypted_filename`, `encrypted_mime`, `encrypted_message` (ciphertexts) | Their plaintexts |

| Visible to server | Not visible |
|---|---|
| Plaintext `sender_display_name` (recipient sees it before unlock)<br>Audit-chain entries for every access event | |

---

## 11. Aionda Chat — Mensagens E2EE pós-quânticas

O Aionda Chat é a superfície integrada de mensagens em tempo real da Aionda Mail. Ao contrário do e-mail — que utiliza chaves efémeras de uso único por mensagem — as conversas de chat são de longa duração e requerem **forward secrecy** e **post-compromise security** para cada mensagem individual. Para isso projetámos um protocolo dedicado: **AAR (Aionda Async Ratchet)** — uma variante pós-quântica do X3DH + Double Ratchet do Signal, onde cada passo Diffie–Hellman é substituído por uma KEM híbrida (X25519 + ML-KEM-1024).

### 11.1 Porquê um protocolo dedicado?

O e-mail e o chat têm perfis de segurança fundamentalmente diferentes:

| Property | Email | Chat |
|---|---|---|
| **Cadence** | Bursty (minutes/hours apart) | Real-time (seconds apart) |
| **Session length** | Single message | Continuous, days to years |
| **Forward secrecy unit** | Per message | Per message **within** a long session |
| **Post-compromise recovery** | Not required (one-shot key) | Required — each new message heals from past compromise |
| **Asynchrony** | High (recipient often offline) | High (recipient often offline) |
| **Group dynamics** | Static recipient list | Dynamic add/remove members |

A chat conversation that simply re-used the email pipeline would either (a) burn a fresh ephemeral keypair every keystroke (unacceptably slow), or (b) share a single static conversation key (no forward secrecy). Neither is acceptable. AAR resolves this by maintaining a continuously ratcheting key state per peer — every message advances the state irrevocably.

### 11.2 Blocos de construção

```
AAR uses exactly four primitives:

   Hybrid KEM         X25519 + ML-KEM-1024  (key encapsulation)
   AEAD               AES-256-GCM           (message encryption)
   KDF                HKDF-SHA256           (key derivation)
   Signatures         Ed25519               (identity-key binding, SPK signing)
```

```
No new cryptographic assumptions are introduced — every primitive is also
used elsewhere in the system and has been independently audited.
```

### 11.3 KeyBundle — o material de handshake assíncrono

Cada utilizador publica um **KeyBundle** no servidor quando o chat é activado pela primeira vez. O bundle permite aos pares começar a enviar mensagens mesmo enquanto o utilizador está offline.

```
KeyBundle (per account, hybrid throughout):

   Identity Key (IK)          — long-lived
      |-- IK.x25519_pub
      |-- IK.mlkem_pub
      |-- IK.ed25519_pub      — used to sign the SPK

   Signed Pre-Key (SPK)       — rotated weekly
      |-- SPK.x25519_pub
      |-- SPK.mlkem_pub
      |-- Ed25519 signature   — signs the concatenation, binds SPK to IK

   One-Time Pre-Keys (OPK)    — pool of ~100, consumed atomically
      |-- OPK.x25519_pub
      |-- OPK.mlkem_pub
```

**Server storage:** Only public halves of each key are stored, plus signatures and metadata. Private halves never leave the originating browser. The server enforces an atomic `UPDATE … LIMIT 1 SET consumed_ts = NOW()` when an OPK is fetched, guaranteeing single-use semantics under concurrent requests.

When the OPK pool falls below 20, the client tops up the pool with a fresh batch — this prevents the asynchronous handshake from degrading to a degenerate mode lacking one-time entropy.

### 11.4 Handshake estilo PQXDH (X3DH-PQ)

To start a new conversation with Bob, Alice fetches Bob's bundle (`IK_B`, `SPK_B`, one `OPK_B`) and Bob's Ed25519 signature on `SPK_B`. Alice verifies the signature, then performs four hybrid KEM encapsulations:

```
1. Verify Ed25519 signature on SPK_B    — binds SPK to long-term identity

2. Generate ephemeral Alice key (EK_A):
   EK_A.x25519, EK_A.mlkem  (one-time, discarded immediately after handshake)

3. Four hybrid KEM encapsulations:
   ss1 = HKEM(IK_A_priv  ⊗ SPK_B_pub)    — Alice identity -> Bob SPK
   ss2 = HKEM(EK_A_priv  ⊗ IK_B_pub)     — Alice ephemeral -> Bob identity
   ss3 = HKEM(EK_A_priv  ⊗ SPK_B_pub)    — Alice ephemeral -> Bob SPK
   ss4 = HKEM(EK_A_priv  ⊗ OPK_B_pub)    — Alice ephemeral -> Bob OPK

   (each ss_i is itself the HKDF combination of an X25519 secret
    AND an ML-KEM secret — see Section 6.4)
```

```
4. Root key derivation:
   SK = HKDF-SHA256(
          IKM  = ss1 || ss2 || ss3 || ss4,
          info = "AAR-X3DH-v1" || consumed_OPK_id
        )

5. Forget every private piece of ephemeral material; SK seeds the ratchet.
```

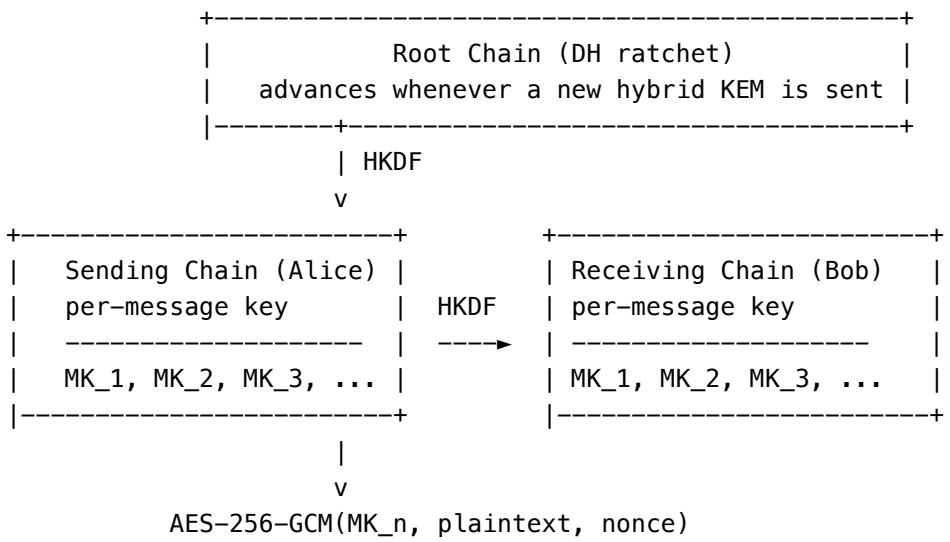
A construção garante:

- **Mutual authentication** via the Ed25519-signed `SPK_B` and `IK_A` binding
- **Forward secrecy** even if `IK_B` is later compromised — `ss3` and `ss4` use ephemeral keys on both sides
- **Post-quantum security** because every `ss_i` includes an ML-KEM term — an attacker harvesting today's traffic for a future quantum computer still cannot reconstruct any of the four secrets
- **Replay resistance** through atomic OPK consumption — Bob's server-side trigger refuses to release the same OPK twice

O OPK ID consumido pela Alice é ligado ao parâmetro `info` do HKDF — o lado do Bob só pode portanto reproduzir a mesma chave raiz com o mesmo OPK, e apenas uma vez.

### 11.5 Double Ratchet — chaves por mensagem

After the handshake, both sides advance a **Double Ratchet** state for every message:




Cada evento de chat transporta a **chave pública de ratchet** atual do remetente (par híbrido, ~1,6 KB). Quando o destinatário recebe uma mensagem cuja chave de ratchet difere da última vista, ambas as partes executam uma nova Hybrid KEM, derivam uma nova chave raiz e reiniciam as suas chains. Este é o **DH ratchet** na terminologia do Signal — exceto que cada passo é uma Hybrid KEM completa, e não um único Diffie–Hellman.

**Per-message guarantees:**

- **Forward secrecy:** Deletion of `MK_n` makes message `n` undecryptable, even given the full long-term state at the time of capture

- **Post-compromise security:** Once a fresh DH-ratchet step has occurred after a compromise, all subsequent messages are protected from the attacker
- **Out-of-order delivery:** Message keys are cached for late-arriving messages (capped at 1000 keys per session to bound memory)

### 11.6 Contentor de mensagem

Cada evento de chat carregado para o servidor é um ciphertext autocontido:

```
MessageContainer (base64-encoded, stored in chat_events.payload):

   +-------------------------------------------------------------+
   | ratchet_pub      :  Hybrid KEM public key (current step)    |
   | prev_chain_len   :  Number of messages in previous chain    |
   | msg_number       :  Counter within current chain            |
   | ciphertext       :  AES-256-GCM(MK_n, plaintext || header)  |
   | tag              :  AES-256-GCM authentication tag          |
   |-------------------------------------------------------------+

The header inside the AEAD plaintext contains: sender login string,
content_type (text/quote/file-reference), timestamp, optional
reply_to_event_uuid.
```

O contentor é opaco para o servidor — incluindo o tipo de mensagem, o formato do nome de visualização do remetente, e qualquer contexto de resposta citada. A base de dados da Aionda armazena o contentor tal qual em `chat_events.payload`.

### 11.7 Conversas em grupo — Sender Keys por destinatário

For room conversations (3+ participants), AAR uses a **per-recipient fan-out** model. The sender derives one sender-key chain per peer pair and encrypts a message once per recipient. Each recipient therefore receives a personally-addressed ciphertext, decryptable only with the keys derived from their own AAR session with the sender.

**Trade-off:** Fan-out cost is `O(N)` in recipients — acceptable for typical team sizes (≤25 participants). For larger groups a future MLS-based mode is planned (see Roadmap). The current model is preferable to a single shared group key because: (a) it preserves forward secrecy on a per-pair basis, (b) it does not require key-rotation on member removal beyond the bilateral level, and (c) it inherits the post-quantum guarantees of pairwise AAR without modification.

### 11.8 Contentor de conversa inicial

When Alice starts a new room with Bob, Carol, and Dan, the first message to each participant is wrapped as an **initial-handshake container** that carries (a) the X3DH handshake material consumed against that participant's bundle and (b) the first message itself. The receiver dispatches on the discriminator field:

```
{
  "type": "initial",
  "initialMessage": { /* handshake metadata, OPK id consumed */ },
  "encryptedMessage": "<base64 MessageContainer>"
}
```

Subsequent messages in the same room reuse the established AAR session and only contain

the `encryptedMessage` portion.

### 11.9 Transporte — API encriptada + Mercure SSE + WebSocket

O plano do chat utiliza três canais ortogonais:

| Channel | Direction | Purpose | Plaintext seen by server |
|---|---|---|---|
| **`/e` Encrypted API** | Client -> Server | Send message, fetch history, key-bundle ops (11 endpoints) | None — all 11 chat endpoints use the same hybrid-KEM transport documented in Section 8 |
| **Mercure SSE** | Server -> Client | Push delivery of new events (`chat.event_received`, `chat.read_receipt`) | None — the server pushes the same opaque MessageContainer it stored |
| **WebSocket `/chat/ws`** | Bi-directional | Reconnect backfill (`sync_request`), heartbeat, presence | Presence boolean (online/offline) and login-string only — never message content |

Presence is held in an in-memory map on `aionda_chat_realtime`; a single intranet-only HTTP endpoint exposes a `login -> online` lookup for the team picker so users can see whether a peer is reachable. No history, no read state, no content ever traverses the presence channel.

### 11.10 Confirmações de leitura

As confirmações de leitura são uma definição opt-in por conta (`chat_participants.send_read_receipts`). Quando ativadas, marcar uma mensagem como lida publica um evento `chat.read_receipt` que contém:

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

Note that `last_read_event_uuid` is **not** the message content — it is a server-allocated identifier already known to the server. No additional information leaks beyond "Alice has now seen messages up to event X." Recipients who disable read receipts (`send_read_receipts = false`) never emit such events, and the server enforces the toggle.

### 11.11 Integração na cadeia de auditoria

Para contas Enterprise, cada operação de chat é adicionada ao log de auditoria infalsificável existente (`enterprise_audit_log`, cadeia de hash SHA3-256 — ver §18). Os tipos de ação reservados são:

`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`.

O registo de auditoria contém apenas o UUID da conversa, o login do actor, e um timestamp — **nunca** o payload encriptado, as chaves de ratchet, ou os conteúdos das mensagens.

### 11.12 O que o servidor vê

| Visible to server | Not visible |
|---|---|
| `conversation_uuid`, list of participants (by `mail_account.name`) | Plaintext message content |
| Encrypted `MessageContainer` payloads (opaque ciphertexts) | Ratchet keys, chain keys, message keys |
| Public halves of IK, SPK, OPK; Ed25519 signatures on SPK | Private halves of any keypair |
| OPK consumption counter and timestamp | Which OPK was consumed for which conversation (correlation prevented by HKDF-info binding only existing client-side) |
| Conversation cadence (timestamps of `chat_events` rows) | Quoted reply chains, file attachments inside the container |
| Presence boolean (online/offline) | Typing indicators (peer-to-peer only, never reach the server) |
| Audit-chain entries for Enterprise accounts | Plaintext content of audit entries |

### 11.13 Auditorias criptográficas

O protocolo AAR foi implementado de raiz em TypeScript e passou por três auditorias independentes assistidas por IA, documentadas em `typescript/manager/chat/crypto/AUDIT.md`, `AUDIT_SECOND_OPINION.md` e `AUDIT_THIRD_OPINION.md`. As conclusões de cada ronda foram integradas antes do deployment público. As principais superfícies auditadas foram: simetria do handshake, reserva de OPK sob concorrência, deep-cloning do estado imutável do ratchet, e o framing AEAD do `MessageContainer`.

### 11.14 Limites de implementação

The chat code base lives in `typescript/manager/chat/` (~9,000 lines of TypeScript) and `includes/classes/Ap` (PHP service layer). Key audit artifacts:

```
crypto/aar-types.ts            — type-only declarations, no logic
crypto/aar-keybundle.ts        — KeyBundle creation, rotation, serialization
crypto/aar-x3dh.ts             — Handshake (initiator + responder paths)
crypto/aar-double-ratchet.ts   — Root chain, sending chain, receiving chain
```

The clean separation between protocol logic (`crypto/`) and transport/UI (`chat-store.ts`, `chat-panel.ts`, …) ensures that the cryptographic core can be re-audited without UI scope creep.

---

## 12. Chamadas de vídeo Aionda — Videoconferência E2EE pós-quântica

A Aionda Mail inclui chamadas de vídeo de grupo encriptadas de ponta a ponta (suite criptográfica `AIONDA_PQ_CALL_V1`). O relé de média nunca vê áudio ou vídeo em texto simples — encaminha apenas fotogramas cifrados. As chaves de grupo são acordadas com um handshake **híbrido pós-quântico**.

### 12.1 Objetivos de segurança & modelo de confiança

**No âmbito — o protocolo garante:**

- **Confidencialidade** da média de áudio e vídeo, de ponta a ponta.
- **Autenticidade** do grupo — cada participante é criptograficamente um membro do mesmo grupo MLS, verificado fora de banda através da Short Authentication String (Secção 12.5).
- **Sigilo futuro (forward secrecy)** — a média passada permanece secreta mesmo que as chaves atuais sejam mais tarde comprometidas.
- **Segurança pós-compromisso (post-compromise security)** — o grupo recupera a confidencialidade depois de um membro comprometido ser removido e o grupo renovar as chaves.
- **Confidencialidade pós-quântica** — o estabelecimento do grupo utiliza a KEM híbrida X-Wing.
- **Independência do servidor** — uma SFU, servidor de sinalização ou camada de armazenamento maliciosos ou comprometidos não conseguem ler nem injetar média.

**Fora do âmbito (explicitamente *não* protegido):**

- Análise de tráfego — a temporização dos pacotes, os tamanhos dos pacotes, o número de participantes e a duração da chamada são observáveis (Secção 12.9).
- Compromisso do endpoint (ver a hipótese de confiança abaixo).
- Malware ao nível do sistema operativo ou apreensão física do dispositivo.

**Hipótese de confiança no endpoint.** O modelo pressupõe um navegador e um sistema operativo não comprometidos. Os fotogramas são selados dentro da pipeline de média do navegador (`RTCRtpScriptTransform`), mas um endpoint comprometido pode ler a média em texto simples *antes* da encriptação ou *depois* da desencriptação. O modelo de confiança na entrega web e as suas mitigações (assinatura de respostas Guardian) são tratados na Secção 18 e na Secção 23.

### 12.2 Porquê um protocolo dedicado?

A videoconferência em tempo real tem dois requisitos que as camadas de e-mail e de chat não têm:

1. **Encriptação por fotograma à velocidade da linha.** Os fotogramas de áudio/vídeo têm de ser encriptados individualmente para que uma Selective Forwarding Unit (SFU) os possa encaminhar, descartar e reordenar sem nunca os desencriptar. Este é o papel do **SFrame** (RFC 9605).
2. **Acordo dinâmico de chaves de grupo.** Os participantes entram e saem a meio da chamada. Cada alteração de composição tem de renovar as chaves do grupo com sigilo futuro (forward secrecy) e segurança pós-compromisso (post-compromise security). Este é o papel do **MLS** (RFC 9420).

A SFU do lado do servidor (`mediasoup`) é tratada como **infraestrutura não confiável**: é um encaminhador de pacotes de conhecimento zero, tal como a camada de armazenamento do e-mail.

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

O perfil MLS substitui as primitivas clássicas de uma suite MLS padrão por híbridos pós-quânticos.

| Role | Algorithm | Notes |
|---|---|---|
| HPKE KEM | **X-Wing** (ML-KEM-768 + X25519) | Hybrid; `draft-connolly-cfrg-xwing-kem` |
| Signatures | **ML-DSA-65** | NIST FIPS 204 (Dilithium), Level 3 |
| KDF | HKDF-SHA-256 | RFC 5869 |
| AEAD (HPKE + SFrame) | **AES-256-GCM** | O AES-256 oferece uma margem confortável contra ataques quânticos de pesquisa conhecidos (Grover) |

**Sem downgrade.** Não existe fallback exclusivamente clássico. Se o handshake pós-quântico não puder ser concluído, a chamada falha de forma fechada (fail closed) — um downgrade é um erro rígido, nunca um enfraquecimento silencioso.

**Maturidade do X-Wing.** O X-Wing é atualmente um draft do IETF/CFRG (`draft-connolly-cfrg-xwing-kem`). O perfil atual adota-o como KEM híbrida preferida; revisões futuras do protocolo poderão migrar para uma KEM híbrida finalizada e normalizada pelo CFRG/NIST, mantendo as camadas MLS e SFrame inalteradas.

**Agilidade criptográfica.** O identificador da suite criptográfica (`0xFF01`) determina por completo a KEM, o algoritmo de assinatura, o KDF e o AEAD. Uma futura `AIONDA_PQ_CALL_V2` pode substituir qualquer primitiva sem alterar as camadas superiores do protocolo; as suites são selecionadas por identificador e nunca negociadas silenciosamente para baixo.

### 12.4 Acordo de chaves de grupo (MLS, RFC 9420)

Cada participante é representado na árvore de ratchet MLS por um **KeyPackage** (chave pública KEM X-Wing + credencial ML-DSA-65). O grupo evolui através de *épocas*:

- **Welcome / Commit** — um membro que entra é adicionado com um segredo de caminho encapsulado com X-Wing; a árvore de ratchet avança para uma nova época.
- **Renovação de chaves a cada alteração de composição.** Qualquer alteração — uma entrada, uma saída, a substituição de um dispositivo ou a rotação de uma chave de identidade — faz o grupo avançar para uma nova época. É isto que proporciona o sigilo futuro (um novo participante não consegue ler média passada) e a segurança pós-compromisso (um membro removido não consegue ler média futura).
- **Remove-Commit.** Quando um participante sai (ou é detetado um par obsoleto/zombie), o proprietário emite um Remove-Commit que renova as chaves do grupo, de modo que um membro que saiu não consiga desencriptar fotogramas subsequentes, mesmo que tenha retido material de chave antigo.

A construção híbrida foi concebida de forma a que o compromisso de apenas **um** componente — ML-KEM-768 *ou* X25519 — não revele, por si só, o segredo de grupo.

### 12.5 Identidade & autenticação

- **Credenciais.** Cada membro transporta uma **Credential** MLS que vincula a identidade de um participante à sua chave de assinatura ML-DSA-65. Cada mensagem de handshake MLS (Welcome / Commit / Update) é assinada com essa chave, pelo que a SFU ou o

servidor de sinalização não conseguem forjar operações de grupo.

- **Short Authentication String (SAS).** A ligação à sala transporta um **segredo pré-partilhado** de alta entropia no fragmento de URL, que nunca é enviado ao servidor em texto simples. O exportador do grupo MLS e este segredo são combinados numa Short Authentication String, cuja impressão digital o cliente **fixa (pin)**. Uma divergência — por exemplo, um servidor de sinalização ou uma SFU a tentar inserir um participante malicioso — desencadeia um **aborto rígido sem qualquer fallback suave**. Isto vincula o grupo criptográfico ao segredo de sala fora de banda e derrota o man-in-the-middle do lado do servidor.
- **Fator de autenticação.** Para uma sala de convidados, *a posse da ligação à sala equivale à posse da credencial da sala* — o segredo da sala é o único fator de autenticação. Os participantes Aionda autenticados resgatam adicionalmente um token de sinalização assinado e limitado à conta (Secção 12.8).

### 12.6 Encriptação de média (SFrame, RFC 9605)

Cada faixa de média é encriptada fotograma a fotograma **antes** de chegar à SFU. O ciclo de vida da chave:

```
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
```

- A chave SFrame é derivada do **MLS Exporter**, pelo que apenas existe nos membros autenticados do grupo — nunca no servidor.
- A encriptação corre num Worker dedicado `RTCRtpScriptTransform` (a API normalizada Encoded-Transform, disponível no Chromium e no Safari/WebKit), de modo que os fotogramas são selados na pipeline de média do navegador.
- O codec é VP8; a SFU efetua o encaminhamento de keyframes apenas com base em metadados — nunca inspeciona o payload encriptado.

### 12.7 Proteção contra repetição (replay)

Cada fotograma encriptado transporta um **contador** SFrame monotónico dentro do seu cabeçalho autenticado e está vinculado à época MLS atual. Os recetores acompanham o contador mais elevado visto por remetente (um armazenamento de contadores do lado da receção, persistido). Um fotograma com um contador duplicado ou regredido, ou um vinculado a uma época obsoleta, é rejeitado — um ciphertext capturado não pode ser reproduzido para dentro da sessão.

### 12.8 Sinalização & modelo de acesso

- **Sinalização** corre sobre um WebSocket autenticado (`aionda_signaling`). Cada sessão é controlada por um **JWT** de curta duração, emitido pelo servidor de aplicação depois de o cliente resgatar o segredo da sala através da API encriptada `/e` (ver Secção 8). O

token é simétrico (HS256) e validado apenas entre o servidor de aplicação e o único serviço de sinalização; autoriza o estabelecimento da sessão e nunca transporta média nem segredos de grupo. O servidor de sinalização retransmite as mensagens MLS Welcome/Commit, mas não consegue lê-las.

- **Ligação à sala = credencial portadora (bearer).** Qualquer pessoa que detenha a ligação completa (UUID da sala + segredo do fragmento) pode entrar — o modelo é intencionalmente "quem tem a ligação". Os IDs de base de dados nunca são expostos; as salas são endereçadas apenas por UUIDs imprevisíveis.
- **Controlo do proprietário.** O proprietário da sala (verificado do lado do servidor pela titularidade da conta) pode marcar uma sala como *terminada* para bloquear futuras entradas, e pode remover um participante presente através de um MLS Remove-Commit (Secção 12.4).

### 12.9 O que a SFU / o servidor vê — e limitações de metadados

| A infraestrutura PODE ver | A infraestrutura NÃO PODE ver |
|---|---|
| Tamanhos dos fotogramas encriptados, temporização, metadados de encaminhamento | Conteúdo de áudio ou vídeo (os fotogramas são selados com AES-256-GCM) |
| Ciphertext do handshake MLS (Welcome/Commit) | Segredos de grupo ou chaves SFrame (derivados do lado do cliente a partir do exportador MLS) |
| UUID da sala, número de participantes, subject do JWT | O segredo da ligação à sala (reside no fragmento de URL) |
| Candidatos ICE / endereços de transporte | Qualquer texto simples que lhe permitisse entrar ou desencriptar a sessão |

**Metadados que o sistema intencionalmente não oculta:** endereços IP expostos à infraestrutura ICE / STUN / TURN, temporização dos pacotes, tamanhos dos pacotes, número de participantes e duração da chamada. A resistência à análise de tráfego não é um objetivo de conceção (Secção 12.1).

### 12.10 Propriedades de segurança (resumo)

| Propriedade | Garantia |
|---|---|
| Confidencialidade | Apenas os membros atuais do grupo MLS possuem as chaves SFrame. |
| Autenticação | Todas as mensagens MLS são assinadas com ML-DSA-65; o grupo é vinculado ao segredo da sala através da SAS fixada. |
| Sigilo futuro (forward secrecy) | O compromisso dos segredos atuais não revela média passada. |
| Segurança pós-compromisso | A renovação de chaves restaura a confidencialidade depois de um membro ser removido. |
| Independência do servidor | O compromisso do servidor de sinalização, da SFU ou do armazenamento não expõe a média. |

| Propriedade | Garantia |
|---|---|
| Pós-quântica | O estabelecimento do grupo assenta na KEM híbrida X-Wing (ML-KEM-768 + X25519). |

Estas propriedades descrevem a *conceção* do protocolo. Está planeada uma auditoria criptográfica independente, por terceiros, do subsistema de chamadas; até à sua publicação, as garantias acima são uma autoavaliação.

---

## 13. Tarefas Aionda - Gestão de tarefas encriptada de ponta a ponta

O Aionda Mail inclui um gestor de tarefas encriptado: listas de tarefas, subtarefas, notas, datas de vencimento, lembretes e tarefas recorrentes. Segue as mesmas regras zero-knowledge da caixa de correio. Tudo o que o utilizador escreve é encriptado no navegador; o servidor guarda apenas texto cifrado, mais um pequeno conjunto de campos de planeamento deliberadamente em claro.

### 13.1 Uma chave por tarefa

Cada tarefa e cada lista de tarefas recebe a sua própria chave de 32 bytes acabada de gerar (AES-256-GCM), criada no navegador com o gerador de aleatoriedade da API WebCrypto. Os campos de conteúdo são encriptados com essa chave; a própria chave é depois envolvida com a chave-mestra do cofre:

```
chave de tarefa aleatória de 32 bytes
      |  AES-256-GCM, IV de 96 bits novo para cada campo
      v
encrypted_title / encrypted_notes / encrypted_steps / encrypted_metadata

chave-mestra do cofre
      |  HKDF-SHA-256(salt = 32 bytes aleatórios, info = "trashmail-tasks-v1")
      v
chave de envolvimento
      |  AES-256-GCM
      v
wrapped_key = version(1) || salt(32) || IV(12) || texto cifrado || tag
```

O servidor guarda apenas `wrapped_key` e as colunas cifradas. Não recebe a chave da tarefa, nem a chave-mestra, nem qualquer texto em claro. Sem a chave-mestra, que nunca sai do navegador, a chave envolvida não pode ser aberta.

**Separação de domínios.** A etiqueta HKDF `trashmail-tasks-v1` é específica do módulo de tarefas. Sem uma etiqueta destas, uma chave envolvida produzida por outro módulo (calendário, contactos) seria intermutável com uma chave de tarefa. A etiqueta liga cada chave envolvida ao módulo que a criou.

**Desencriptação independente.** As tarefas e as listas são desencriptadas uma a uma. Um registo ilegível é anotado no log e ignorado, em vez de fazer falhar toda a vista, pelo que um texto cifrado danificado não deixa o utilizador sem acesso às restantes tarefas.

### 13.2 O que é encriptado

| Campo | Conteúdo | Proteção |
|---|---|---|
| `encrypted_title` | Título da tarefa | AES-256-GCM com a chave da tarefa |
| `encrypted_notes` | Nota de texto livre | AES-256-GCM com a chave da tarefa |
| `encrypted_steps` | Array JSON das subtarefas (texto e estado de conclusão) | AES-256-GCM com a chave da tarefa |
| `encrypted_metadata` | JSON, incluindo a regra de recorrência (RRULE) | AES-256-GCM com a chave da tarefa |
| `encrypted_name` | Nome de uma lista de tarefas | AES-256-GCM com a chave da lista |

Cada campo é selado com um IV de 96 bits próprio e novo, pelo que o mesmo texto em claro nunca produz o mesmo texto cifrado.

**Os nomes das listas também são encriptados.** Uma lista chamada “Aquisição Contoso” é conteúdo, não metadado, e é tratada como tal. O servidor vê que uma conta tem cinco listas, mas não como se chamam.

### 13.3 Metadados deliberadamente em claro

Um pequeno conjunto de campos de planeamento é guardado em claro, para que o servidor possa filtrar, ordenar, contar e disparar lembretes sem desencriptar nada:

| Campo em claro | Finalidade |
|---|---|
| Data de vencimento | Lista inteligente “Planeado”, deteção de atrasos, ordenação |
| Hora do lembrete | Disparar o lembrete no momento certo |
| Marca de importante | Lista inteligente “Importante” |
| Marca de concluído e hora de conclusão | Separar tarefas abertas de tarefas concluídas |
| Data de “O meu dia” | Lista inteligente “O meu dia”, válida apenas para o dia atual |
| Posição | Ordenação manual dentro de uma lista |
| Atribuição de lista (UUID) | Agrupamento das tarefas sob a respetiva lista |
| E-mail associado (UUID) | Referência ao e-mail a partir do qual a tarefa foi criada |

**O servidor fica a saber quando algo vence, nunca o que é.** Pode determinar que uma conta tem catorze tarefas abertas, três delas com vencimento a 3 de agosto e uma marcada como importante. Não consegue ler um único título, nota, subtarefa, regra de recorrência ou nome de lista.

É o mesmo compromisso já assumido para as horas dos compromissos do calendário, e é divulgado em vez de escondido. A alternativa - encriptar também a data de vencimento - obrigaria o cliente a descarregar e desencriptar o conjunto completo de tarefas para cada filtro, cada contador e cada lembrete. Os campos de planeamento não transportam conteúdo, e os objetos a que pertencem são endereçados exclusivamente por UUID.

### 13.4 A lista “E-mail sinalizado”

A lista “E-mail sinalizado” não é um segundo repositório de dados. É uma vista sobre a estrela que já existe nos e-mails da caixa de correio, pelo que não são criados dados em claro adicionais para ela.

Os assuntos apresentados nessa lista são desencriptados exclusivamente do lado do cliente, através do caminho habitual de desencriptação de e-mail (Section 7.2): a chave do e-mail em causa é desembrulhada no navegador e o assunto é aí desencriptado. O módulo de tarefas nunca manipula uma chave de e-mail nem pede um assunto ao servidor.

### 13.5 Transporte, endereçamento e modo de supervisão

- **Transporte encriptado.** Todos os endpoints de tarefas passam pela camada de transporte encriptada `/e` (Section 8). Pedido e resposta são selados com o KEM híbrido, pelo que uma CDN ou qualquer outro intermediário vê apenas um `POST /e` com uma carga opaca, e não qual a tarefa criada, renomeada ou concluída.
- **Apenas UUID.** As tarefas, as listas e os e-mails associados são endereçados exclusivamente por UUID. Os identificadores internos da base de dados nunca são expostos, o que elimina a enumeração como via de ataque.
- **Modo de supervisão.** Quando um proprietário administra a caixa de correio de um colaborador ou de um agente de IA, o módulo de tarefas aplica a mesma separação de chaves do resto do cofre: as chaves de tarefa são envolvidas com a chave-mestra do contexto supervisionado, não com a do proprietário. Se essa chave de contexto não estiver disponível, a operação falha de forma dura. Não existe, deliberadamente, qualquer recurso à chave do proprietário - um recurso silencioso escreveria as tarefas de um colaborador com a chave errada. Isso seria um enfraquecimento criptográfico, e um enfraquecimento tem de falhar de forma ruidosa em vez de ser bem-sucedido em silêncio.

### 13.6 Validação de entrada contra truncagem silenciosa

O comprimento de cada bloco cifrado é verificado no servidor antes da escrita:

| Campo | Limite |
|---|---|
| Título da tarefa, nome da lista | 2048 bytes |
| Nota | 1 MiB |
| Subtarefas | 256 KiB |
| Metadados | 256 KiB |
| Chave envolvida | de 60 a 1700 bytes |

Os valores demasiado grandes são rejeitados com um erro explícito. Trata-se de um controlo de integridade com consequências diretas para a confidencialidade: consoante o modo SQL, a base de dados truncaria de outra forma um valor demasiado grande em silêncio. Um texto cifrado AES-GCM truncado falha a autenticação e o seu conteúdo fica permanentemente ilegível, também para o utilizador. Recusar a escrita é o único comportamento seguro.

### 13.7 O que o servidor vê - e o que não vê

| O servidor PODE ver | O servidor NÃO PODE ver |
|---|---|
| Datas de vencimento, horas de lembrete, datas de “O meu dia” | Títulos das tarefas, notas, subtarefas |
| Marcas de importante e concluído, hora de conclusão | O nome de qualquer lista de tarefas |
| Número de tarefas e listas, a sua ordem e agrupamento | Regras de recorrência e todos os restantes metadados |
| O UUID de um e-mail associado | O conteúdo ou o assunto desse e-mail |
| Tamanhos do texto cifrado, datas de criação e modificação | Qualquer material de chave: chaves de tarefa, de lista e chave-mestra |

---

## 14. Proteções contra canais auxiliares

### 14.1 Bucket Padding

**Problema:** Os tamanhos de e-mails encriptados podem revelar informação. Um atacante que observe os comprimentos dos ciphertexts pode inferir o conteúdo (por exemplo, um e-mail de 50 bytes é provavelmente “OK, obrigado” enquanto um e-mail de 500 KB contém anexos).

**Solução:** Todos os dados são alinhados para “buckets” de tamanho fixo antes da encriptação:

```
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]
```

**Exemplo:** Um e-mail de 523 bytes é alinhado para 1.024 bytes. Um observador vê apenas “e-mail de 1KB” — não o tamanho real de 523 bytes.

### 14.2 Compressão antes da encriptação

Os dados são comprimidos com gzip (nível 6) **antes** da encriptação. Esta é a única ordem correta:

- A compressão após encriptação falharia (dados encriptados têm entropia máxima)
- O bucket padding após a compressão previne ataques tipo CRIME/BREACH que exploram rácios de compressão

### 14.3 Privacidade do threading

Os threads de e-mail usam hashes SHA-256 dos cabeçalhos Message-ID em vez de identificadores em claro. O servidor pode agrupar e-mails relacionados por igualdade de hash sem saber os identificadores reais das mensagens.

---

## 15. Gestão e ciclo de vida das chaves

### 15.1 Hierarquia de chaves

```
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
```

### 15.2 Armazenamento de chaves

| Key | Storage Location | Protection |
|---|---|---|
| Master Key | Nowhere (reconstructed on-demand from Shamir shares) | Shamir 2-of-3 |
| Vault private keys | Server (encrypted) | AES-256-GCM with master key |
| Vault public keys | Server (plaintext) | Not sensitive — public by definition |
| Email ephemeral keys | Server (wrapped) | Hybrid KEM encapsulation |
| OPAQUE records | Server (encrypted at rest) | AES-256-GCM with server key |
| Encrypted Shamir shares | Server | XOR with password/passkey/recovery derived keys |
| API transport keys | Server (pre-generated pool) | One-time use, 24h TTL |

### 15.3 Fingerprints de chaves

Each vault keypair has a SHA-256 fingerprint stored on the server. This allows:

- Audit trail of key rotations
- Detection of unauthorized key changes
- Client-side verification of key integrity

---

## 16. Mecanismo de recuperação

### 16.1 Chave de recuperação (mnemónica BIP39)

Durante a configuração do cofre, o utilizador recebe uma **frase de recuperação de 24 palavras** gerada a partir de 256 bits de entropia, codificada usando a norma 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
```

### 16.2 Derivação da chave de recuperação

```
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)
```

### 16.3 O que o servidor armazena

O servidor armazena **apenas o hash SHA3-256** de uma chave de verificação derivada. Não armazena:

- As palavras de recuperação
- A entropia
- A própria chave de verificação

#### 16.3.4 Recovery Flow

1. User enters 24-word recovery phrase
2. Client derives entropy -> HKDF-SHA3-256 -> SHA3-256 -> verification hash
3. Client sends verification hash to server (never the plaintext key)
4. Server compares with stored hash
5. If match: All 2FA methods are disabled, user sets up fresh authentication
6. Recovery key is revoked after single use

#### 16.3.5 Rate Limiting

- Maximum 3 verification attempts per hour
- 60-minute lockout after exceeding limit
- One-time use: recovery key is permanently revoked after successful use

#### 16.3.6 No Password Recovery

**Não existe reposição de palavra-passe via e-mail.** Se um utilizador perde a sua palavra-passe E todos os outros factores de autenticação (passkey + chave de recuperação), os seus dados ficam permanentemente inacessíveis. Esta é a prova fundamental de que o zero-knowledge funciona — se o serviço pudesse recuperar dados do utilizador, também os poderia ler.

---

## 17. Autenticação sem palavra-passe (Passkeys)

### 17.1 Extensão WebAuthn PRF

A Aionda Mail suporta passkeys FIDO2 (chaves de segurança em hardware, autenticadores biométricos) para login sem palavra-passe e desbloqueio do cofre.

A **extensão WebAuthn PRF (Pseudo-Random Function)** fornece uma saída determinística de 32 bytes ligada à passkey e credencial específicas. Esta saída é usada para proteger a Parte 2 do Shamir.

### 17.2 Como 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)
```

### 17.3 Múltiplas passkeys

Os utilizadores podem registar várias passkeys (por exemplo, MacBook Touch ID, iPhone Face ID, YubiKey). Cada passkey protege independentemente a sua própria cópia da Parte 2. Qualquer passkey combinada com a palavra-passe é suficiente para desbloquear o cofre.

---

## 18. Guardian: Proteção MITM & assinatura de respostas

### 18.1 O problema com aplicações web

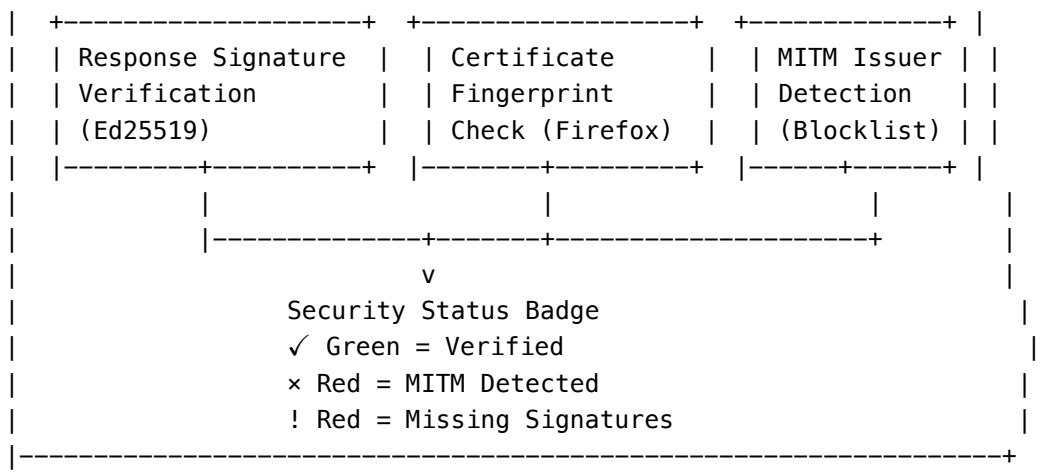
Every web application has an inherent trust problem: the browser downloads JavaScript from the server on every visit. A man-in-the-middle (MITM) attacker — whether a compromised CDN, corporate proxy, or rogue ISP — could theoretically inject modified code that exfiltrates encryption keys.

Aionda Mail addresses this with the **Guardian module**, a browser extension component (available for Chrome and Firefox) that independently verifies server integrity.

### 18.2 Visão geral da arquitetura

The Guardian module operates inside the browser extension's Service Worker — completely independent from the web application's JavaScript. It performs three types of verification:

```
+-------------------------------------------------------------+
|                    BROWSER EXTENSION (Guardian)              |
|                                                             |
```




### 18.3 Verificação da assinatura de resposta (Ed25519)

Every API response from Aionda Mail's server is cryptographically signed using **Ed25519** (Edwards-curve Digital Signature Algorithm).

**Signing process (server-side):**

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>

**Verification process (browser extension):**

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

**Propriedades centrais:**

- **Replay protection:** 5-minute timestamp window prevents replaying old responses
- **Tamper detection:** Any modification to the response body invalidates the signature
- **Key isolation:** Public keys are bundled inside the extension (not downloaded from the server)
- **Environment separation:** Dev keys (`dev-2026-01`) cannot be used on production URLs and vice versa

### 18.4 Gestão da chave pública Ed25519

Public keys are shipped with the browser extension in `public_key.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"
        }
    }
}
```

- **Key format:** SPKI DER (Subject Public Key Info, Distinguished Encoding Rules)
- **Tamanho da chave:** 32 bytes (256-bit Ed25519 public key)
- **Signature size:** 64 bytes (fixed)
- **Rotation:** New keys are added before old keys expire; extension updates deliver new keys
- **No server trust:** Keys are embedded in the extension binary, not fetched from the server

### 18.5 Verificação do certificado TLS (Firefox)

On Firefox, the Guardian module performs additional TLS certificate verification using the `browser.webRequest.getSecurityInfo()` API (not available in Chrome due to Manifest V3 limitations).

**Verification flow:**

```
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
```

**Why issuer-based validation instead of pinning?** CloudFlare (used as CDN) rotates leaf certificates across edge servers. Traditional certificate pinning (matching exact fingerprints) would cause false positives. Issuer-based validation is more robust: the issuing CA is stable even when leaf certificates change.

### 18.6 Self-Certificate Fetching (anti-spoofing)

The server's certificate endpoint uses a clever anti-spoofing technique:

```
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
```

The server essentially asks "what certificate does the outside world see for my domain?" — and signs the answer so the extension can trust it.

### 18.7 Indicadores de estado de segurança

The extension displays a badge on the browser toolbar:

| Badge | Color | Meaning |
|---|---|---|
| ✓ | Green | All responses verified — signatures valid |
| **!** | Orange | Using deprecated signing key (rotation pending) |
| × | Red | MITM detected — signature verification failed |
| ! | Red | Missing signatures — responses not signed |
| [Shield] | Blue | Protected mode — no verification performed yet |

### 18.8 Cobertura de ameaças

| Attack | Detection Method | Browser |
|---|---|---|
| Corporate MITM proxy (ZScaler, Fortinet) | Certificate issuer blocklist | Firefox |
| Modified API responses | Ed25519 signature verification | Chrome + Firefox |
| Replay attacks | 5-minute timestamp window | Chrome + Firefox |
| CDN compromise (CloudFlare) | Response signature mismatch | Chrome + Firefox |
| Certificate substitution | Issuer comparison + server self-check | Firefox |
| Dev/prod key confusion | Environment-bound key IDs | Chrome + Firefox |

### 18.9 Limitações

- **Chrome Manifest V3:** Cannot inspect TLS certificates — only response signature verification is available
- **Extension required:** Users without the extension do not benefit from Guardian protections
- **Ed25519 is not post-quantum:** Signature verification uses classical cryptography. A sufficiently powerful quantum computer could theoretically forge Ed25519 signatures.

---

## 19. Arquivo empresarial de e-mail (Blockchain)

### 19.1 Visão geral

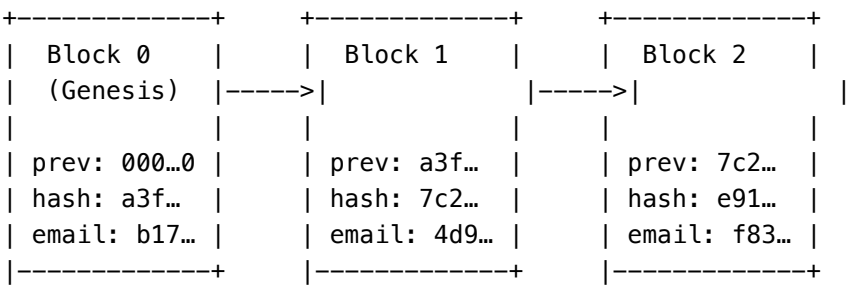
Aionda Mail's Enterprise plan includes a **GoBD-compliant email archive** secured by a cryptographic hash chain (blockchain). Every archived email becomes an immutable block in a per-company chain. Any tampering — modification, deletion, or insertion of blocks — is cryptographically detectable.

The archive combines two independent security layers:

1. **Hash chain (SHA3-256):** Guarantees integrity and immutability — proves no email was altered or removed after archival
2. **Hybrid KEM encryption (CAK):** Guarantees confidentiality — the server cannot read archived email content

### 19.2 Arquitetura da cadeia de hash

Each archived email becomes a block in a sequential, tamper-evident chain:




**Block hash calculation:**

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

**Properties:**

- **Hash algorithm:** SHA3-256 (NIST FIPS 202)
- **Genesis block:** `prev_block_hash` = 64 zeros, `block_number` = 0
- **Sequential numbering:** Enforced by database `UNIQUE KEY (company_uuid, block_number)`
- **One chain per company:** Complete isolation between enterprises
- **Email hash:** `SHA3-256(sender || recipient || timestamp || size)` — integrity proof of the original email data

### 19.3 Deteção de adulteração

The chain verification algorithm detects any form of tampering:

```
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
```

| Tampering Attempt | Detection |
|---|---|
| Modify email content | `email_hash` changes -> `block_hash` recalculation fails |
| Modify metadata (sender, domain, timestamp) | Included in hash input -> `block_hash` mismatch |
| Delete a block | Next block's `prev_block_hash` becomes orphaned |
| Insert a block | Breaks sequential `block_number` + `prev_block_hash` chain |
| Reorder blocks | `UNIQUE KEY` constraint + sequential verification prevents this |
| Replace entire chain | Genesis block hash would differ from any external backup |

**Verification result** reports the exact block number where tampering was detected, with expected vs. actual hash values for forensic analysis.

### 19.4 Chave de arquivo da empresa (CAK) — encriptação Zero-Knowledge

Archive contents are encrypted end-to-end using a **Company Archive Key** — a Hybrid KEM keypair (X25519 + ML-KEM-1024) generated client-side by the company owner.

```
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
```

**Key distribution to other employees (Admin, Compliance Officer):**

1. Owner decrypts CAK private keys with their password
2. Owner re-wraps private keys with target employee's password-derived key
3. Server stores the re-wrapped copy on the employee's record
4. Each authorized employee has their own independently wrapped copy

The server never sees the CAK private keys in plaintext.

### 19.5 O que é encriptado

When an email is archived, two layers of encryption are applied:

**Encrypted metadata** (AES-256-GCM with 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"
}
```

**Encrypted email content** (separate Hybrid KEM key wrapping):

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

**Zero-knowledge enforcement:** After encryption, the plaintext metadata fields in the database (sender_address, recipient_address, domains) are replaced with their SHA3-256 hashes. The server stores only hashes — the original values exist only inside the encrypted blobs.

### 19.6 Trilho de auditoria

Every action on the archive is logged in an **independent audit chain** (also hash-chained with SHA3-256):

| Action | When Logged |
|---|---|
| `EMAIL_RECEIVED` / `EMAIL_SENT` / `DRAFT_ARCHIVED` | Email archived |
| `VIEW_EMAIL` / `VIEW_ATTACHMENT` | Employee reads archived email |
| `SEARCH_ARCHIVE` | Search performed |

| Action | When Logged |
|---|---|
| `EXPORT_EMAIL` / `EXPORT_REPORT` | Data exported |
| `VERIFY_CHAIN` / `CHAIN_VERIFIED_OK` / `CHAIN_VERIFIED_BROKEN` | Integrity check |
| `LEGAL_HOLD_SET` / `LEGAL_HOLD_RELEASED` | Legal hold toggled |
| `ARCHIVE_DECRYPT` | CAK used to decrypt content |
| `ADMIN_ACCESS` | Administrative action |

Each audit entry records: actor (UUID + role), IP address, session ID, target email hash, and whether the chain was valid at the time of access.

### 19.7 Retenção legal e prazos de retenção

- **Retention period:** Configurable per company (default: 10 years), calculated per email as `archived_at + retention_years`
- **Legal hold:** Individual emails can be placed under legal hold, preventing deletion until released. Includes reason, actor, and timestamp.
- **GoBD compliance:** The combination of immutable hash chain, complete audit trail, configurable retention, and legal hold satisfies the requirements of the German GoBD (Grundsätze zur ordnungsmäßigen Führung und Aufbewahrung von Büchern, Aufzeichnungen und Unterlagen in elektronischer Form sowie zum Datenzugriff).

### 19.8 Exportação forense

Authorized users (Owner, Admin) can export the complete chain state for independent verification:

- Full chain data with all block hashes
- Verification result (valid/broken, broken block number if applicable)
- Expected vs. actual hash values for forensic analysis
- Last 50 audit log entries
- JSON format for external re-verification with any SHA3-256 implementation

---

## 20. O que o servidor vê — e o que não vê

This section explicitly documents the zero-knowledge boundary.

### 20.1 O que o servidor PODE ver

| Data | Why Visible | Mitigation |
|---|---|---|
| IP address | TCP/IP requirement | Use VPN/Tor if desired |
| Timestamps | Email reception time | Inherent to email protocol |
| Encrypted email blobs | Stored for retrieval | AES-256-GCM encrypted, key unknown to server |
| Padded ciphertext sizes | Storage requirement | Bucket padding hides actual sizes |

| Data | Why Visible | Mitigation |
|---|---|---|
| Recipient DEA address | Routing requirement | DEA is disposable, not the real address |
| Account existence | Authentication flow | User enumeration protection deployed |
| Public keys | Required for encryption by server | Public by definition, not sensitive |
| Encrypted Shamir shares | Storage for user | XOR'd with keys server doesn't know |
| OPAQUE records | Authentication protocol | Not password hashes, encrypted at rest |

### 20.2 O que o servidor NÃO PODE ver

| Data | Why Invisible |
|---|---|
| Email content (subject, body, headers) | Encrypted with ephemeral keys wrapped via Hybrid KEM |
| User password | OPAQUE — password never transmitted |
| Master key | Reconstructed only in browser from Shamir shares |
| Vault private keys | Encrypted with master key before storage |
| Email ephemeral keys | Wrapped with Hybrid KEM, server lacks private keys |
| Recovery key / mnemonic | Only SHA3-256 hash of derived key stored |
| Passkey PRF outputs | Hardware-bound, never leave authenticator |
| Folder names | Encrypted with folder-specific keys |
| Email signatures | Encrypted with master key |
| API request content | Encrypted via /e transport layer |
| API response content | Encrypted before transmission |

### 20.3 Garantia criptográfica

Even with full access to:

- The complete database
- All network traffic
- The server's source code and configuration
- All OPAQUE records and server keys

…an attacker **cannot** decrypt a single email without the user's password (or passkey + recovery key). This is not a policy — it is a mathematical impossibility enforced by the cryptographic design.

---

## 21. Referência de algoritmos

### 21.1 Tabela completa de algoritmos

| Component | Algorithm | Parameters | Standard |
|---|---|---|---|
| Password authentication | OPAQUE | RFC 9807, aPAKE | RFC 9807 |
| Password key derivation | PBKDF2-SHA256 | 600,000 iterations, 32B salt, 32B output | NIST SP 800-132 |
| Vault encryption | AES-256-GCM | 256-bit key, 96-bit nonce, 128-bit tag | NIST SP 800-38D |
| Classical key exchange | X25519 | Curve25519, 256-bit | RFC 7748 |
| Post-quantum KEM | ML-KEM-1024 | Kyber-1024, NIST Level 5 | NIST FIPS 203 |
| Hybrid key derivation | HKDF-SHA256 | 64B IKM, info="trashmail-hybrid-kem-v1" | RFC 5869 |
| Secret sharing | Shamir SSS | k=2, n=3, GF(2^8) | Shamir (1979) |
| Recovery key encoding | BIP39 | 256-bit entropy, 24 words | BIP-0039 |
| Recovery key derivation | HKDF-SHA3-256 | Account-bound salt | NIST FIPS 202 |
| Recovery key verification | SHA3-256 | 32-byte output | NIST FIPS 202 |
| OPAQUE record encryption | AES-256-GCM | Server-side at-rest encryption | NIST SP 800-38D |
| Passkey vault unlock | WebAuthn PRF | HMAC-based, hardware-bound | WebAuthn Level 2 |
| Compression | gzip | Level 6 | RFC 1952 |
| Bucket padding | Custom | 17 sizes (256B–16MB), 0xDEAD magic | — |
| Response signing | Ed25519 | 256-bit key, 512-bit signature | RFC 8032 |
| Archive hash chain | SHA3-256 | Per-block hash, sequential linking | NIST FIPS 202 |
| Archive key wrapping (CAK) | HKDF-SHA256 + AES-256-GCM | Password-derived wrapping key | RFC 5869 / NIST SP 800-38D |
| Certificate verification | SHA-256 | TLS cert fingerprint comparison | — |
| Email threading | SHA-256 | Hash of Message-ID | NIST FIPS 180-4 |
| Chaves de grupo de chamada de vídeo | MLS + X-Wing | Suite `AIONDA_PQ_CALL_V1` (0xFF01), ML-KEM-768 + X25519 | RFC 9420 / draft-connolly-cfrg-xwing-kem |
| Assinaturas de chamada de vídeo | ML-DSA-65 | Dilithium, NIST Nível 3 | NIST FIPS 204 |
| Média de chamada de vídeo | SFrame (AES-256-GCM) | Por fotograma, chave do exportador MLS | RFC 9605 |

### 21.2 Níveis de segurança

| Algorithm | Classical Security | Post-Quantum Security |
|---|---|---|
| X25519 | 128-bit | Broken by Shor's algorithm |
| ML-KEM-1024 | 256-bit equivalent | NIST Level 5 (≈AES-256) |
| AES-256-GCM | 256-bit | 128-bit (Grover's algorithm) |
| SHA-256 | 256-bit | 128-bit (Grover's algorithm) |
| SHA3-256 | 256-bit | 128-bit (Grover's algorithm) |
| Hybrid KEM (combined) | 128-bit (X25519 bound) | Level 5 (ML-KEM bound) |

---

## 22. Comparação com outros fornecedores

| Feature | Aionda Mail | Tuta Mail | Proton Mail |
|---|---|---|---|
| **Country** | Germany (Stuttgart) | Germany (Hannover) | Switzerland |
| **Zero-Knowledge** | Yes (OPAQUE + client-side crypto) | Yes | Yes |
| **Post-Quantum** | Yes (ML-KEM-1024 + X25519 Hybrid) | Yes (Kyber-based) | In development |
| **Password protocol** | OPAQUE (RFC 9807) — password never leaves browser | bcrypt (password sent to server over TLS) | SRP-based |
| **Subject encrypted** | Yes | Yes | No |
| **Headers encrypted** | Yes | Partial | No |
| **Contact names encrypted** | Yes (in vault) | Yes | No |
| **Disposable email addresses** | Yes (core feature, unlimited for Plus) | No | Yes (via SimpleLogin) |
| **Browser addon** | Yes (Chrome + Firefox) | No | Via SimpleLogin |
| **Folder sharing** | Yes (Hybrid KEM per recipient) | Limited | Yes |
| **Open source client** | No | Yes | Yes |
| **Security audit** | Planned | Yes | Yes |
| **Password recovery** | No (by design) | No (by design) | No (by design) |
| **Passkey support** | Yes (FIDO2 + PRF) | Yes | Yes |
| **PGP support** | Yes (incoming + outgoing) | No (own protocol) | Yes (OpenPGP) |
| **GoBD-compliant email archive** | Yes (SHA3-256 hash chain + Hybrid KEM) | No | No |

| Feature | Aionda Mail | Tuta Mail | Proton Mail |
|---|---|---|---|
| **MITM detection (browser ext.)** | Yes (Ed25519 signatures + TLS check) | No | No |
| **Perfect Forward Secrecy (API)** | Yes (per-request ephemeral keys) | Unknown | Unknown |
| **Email size obfuscation** | Yes (bucket padding) | Unknown | No |

---

## 23. Limitações & fronteiras honestas

### 23.1 Modelo de confiança da aplicação web

Aionda Mail is a web application. On every page load, the browser downloads JavaScript from our servers. A sophisticated attacker who compromises our servers could theoretically serve modified JavaScript that exfiltrates keys.

**Current mitigations:**

- Subresource Integrity (SRI) hashes on all script tags
- Content Security Policy (CSP) headers restrict script sources
- All critical cryptographic code is included in the main application bundle
- **Guardian browser extension** (Section 18): Ed25519 signature verification on all API responses detects server-side tampering; TLS certificate verification (Firefox) detects MITM proxies

**Planned mitigations:**

- Service Worker caching for offline operation (reduces trust-on-load frequency)

### 23.2 Visibilidade de metadados

While email content is fully encrypted, certain metadata is visible to the server:

- When emails were received (timestamps)
- Which DEA address received the email
- Approximate email size (within bucket boundaries)
- Account activity patterns

### 23.3 Registo de processamento de e-mail

For diagnostic purposes, Aionda Mail includes an optional **email processing log** that can temporarily store the raw content of incoming emails. This feature is configurable per disposable email address (DEA) and can be enabled or disabled in the DEA settings (“Log email content”).

When enabled (opt-in per DEA):

- The full raw SMTP message (headers + body) is stored in plaintext on the server
- Automatic deletion after a short retention period (less than 7 days)
- Accessible only to the account owner via authenticated API

- Purpose: troubleshooting delivery issues, verifying forwarding, reviewing spam filtering decisions

When disabled:

- No email content is stored in the processing log
- Only metadata is logged (sender address, timestamp, delivery status)
- Vault encryption remains the sole storage mechanism

**Important:** This processing log is independent of the encrypted vault. Emails stored in the vault are always encrypted with Hybrid KEM regardless of the log setting. The processing log exists as a legacy feature from the email forwarding system and provides operational transparency. Users who require strict zero-knowledge storage for all emails should disable this option.

### 23.4 Segurança de e-mail externo

Emails sent to or received from non-Aionda addresses travel through the standard email infrastructure (SMTP). While stored encrypted in the vault, the email content was visible during transit unless PGP encryption was used.

### 23.5 Sem key escrow

Não existe chave mestra, backdoor ou mecanismo de recuperação disponível para a Aionda GmbH. Se um utilizador perde a sua palavra-passe e todos os métodos de recuperação, os seus dados ficam permanentemente perdidos. Esta é uma decisão de design intencional que prova a integridade do modelo zero-knowledge.

---

## 24. Plano futuro

| Marco | Estado | Meta |
|---|---|---|
| Arquitetura Zero-Knowledge | Concluído | — |
| Hybrid KEM pós-quântica (ML-KEM-1024) | Concluído | — |
| Autenticação OPAQUE (RFC 9807) | Concluído | — |
| Shamir Secret Sharing (2 em 3) | Concluído | — |
| Camada de transporte API encriptada | Concluído | — |
| Suporte para Passkey/WebAuthn PRF | Concluído | — |
| Calendário encriptado de ponta a ponta | Concluído | — |
| Tarefas encriptadas de ponta a ponta | Concluído | — |
| Arquivo de e-mail conforme GoBD (Blockchain) | Concluído | — |
| Proteção MITM Guardian (Ed25519) | Concluído | — |
| Verificação de certificado TLS (Firefox) | Concluído | — |

---

## 25. Iniciar sessão num dispositivo por código QR

### 25.1 Utilização e validade

Um cliente já autenticado e desbloqueado apresenta **Ligar um novo dispositivo (código QR)**. O novo dispositivo lê o convite. Ambos apresentam um código de comparação de oito algaris-

mos; o utilizador deve compará-los e confirmar expressamente no dispositivo existente antes da transferência. Web, iOS e macOS podem apresentar o convite; o destinatário deve suportar o protocolo e o respetivo perfil de transporte de confiança.

O convite expira após **15 minutos (900 segundos)**. Este prazo fixo é uma decisão do produto, não uma norma do setor. Consultas e novas tentativas não o prolongam. Um convite concluído não permite ligar outro dispositivo. O cancelamento ou a expiração invalida uma transferência incompleta, mas não termina a sessão de um dispositivo que já concluiu o processo.

### 25.2 Conteúdo do QR e transferência encriptada

O QR `AIONDA_DEVICE_PAIRING_V2` contém um UUID de emparelhamento, o nome de utilizador, a origem do serviço, os instantes de criação e expiração, uma impressão digital SHA-256 que associa as chaves públicas efémeras da origem aos metadados do convite, e o perfil de transporte e a origem de autenticação. **Não contém a palavra-passe da conta, palavras de recuperação, token de acesso ou chave-mestra do cofre**. O nome de utilizador é legível: não publiques nem partilhes a imagem do QR.

O destinatário compara a impressão digital do QR com as chaves públicas fornecidas pelo intermediário. X25519 + ML-KEM-1024 estabelece um segredo partilhado. HKDF-SHA-256 deriva chaves separadas por função e associadas à transcrição do protocolo; AES-256-GCM encripta e autentica o pedido, a prontidão, a aprovação e o recibo. A conta, o ambiente, o destinatário e a validade ficam associados à troca. Uma impressão digital, associação de transporte ou mensagem autenticada inválida causa um erro, sem alternativa em texto simples.

Após a confirmação, a origem envia um **novo token de acesso do dispositivo e a chave-mestra do cofre** no pacote de aprovação encriptado, não a palavra-passe ou as palavras de recuperação existentes. O token é registado com OPAQUE. O servidor retransmite dados encriptados e gere o ciclo de vida e os registos; o destinatário desencripta o pacote e obtém acesso efetivo à caixa de correio. A aprovação é, por isso, sensível. Revogar o token impede autenticações posteriores, mas não apaga dados ou chaves já obtidos pelo dispositivo.

### 25.3 Riscos do navegador e proteções

O emparelhamento a partir da Web apresenta um aviso de risco traduzido e exige uma autenticação recente: o servidor só aceita criar o convite nos 120 segundos seguintes a uma verificação OPAQUE ou passkey bem-sucedida e associada à conta. A caixa deve estar desbloqueada. A origem Web fica associada à sua sessão; o destinatário utiliza um perfil de autenticação fixo e de confiança, não um destino arbitrário indicado no QR.

**Prefere a aplicação nativa iOS ou macOS como origem.** A encriptação de ponta a ponta protege a troca apenas quando ambos os extremos executam código fiável. Um proxy de inspeção TLS como o ZScaler, uma extensão maliciosa ou uma distribuição Web comprometida pode alterar o código e ler segredos antes da encriptação. HTTPS, o QR e códigos coincidentes não tornam seguro um extremo comprometido. O Guardian pode acrescentar verificações se estiver instalado e ativo, mas não é obrigatório nem pressuposto nesta garantia. Nunca aproves pedidos inesperados; compara os números nos teus dois dispositivos.

---

## Histórico do documento

| Version | Date | Changes |
|---|---|---|
| 1.0 | March 2026 | Initial publication |
| 1.1 | April 2026 | New chapter 10: Vault Drive (per-file-key architecture, sharing via key rewrap) |
| 1.2 | April 2026 | Section 10.11: External Sharing — public share links with hybrid KEM envelopes, unlock_token flow, Argon2id password mode, URL fragment for link-only, tamper-evident audit chain (EXT_SHARE_*), and user-chosen link distribution |
| 1.3 | May 2026 | New chapter 11: Aionda Chat — post-quantum E2EE real-time messaging via AAR (Aionda Async Ratchet), a hybrid X25519 + ML-KEM-1024 variant of Signal X3DH + Double Ratchet. Per-message forward secrecy, post-compromise security, and integration with the Enterprise audit chain. Subsequent chapters renumbered 12-22. PT-BR and PT-PT translations added. Fixed double-numbering bug in PDF rendering (LaTeX auto-numbering disabled; chapter numbers now sourced from headings only). |
| 1.4 | junho de 2026 | Novo capítulo 12: Chamadas de vídeo Aionda — videoconferência de grupo E2EE pós-quântica. Acordo de chaves de grupo MLS (RFC 9420) com a suite criptográfica `AIONDA_PQ_CALL_V1` (X-Wing = ML-KEM-768 + X25519, assinaturas ML-DSA-65), cifragem de média fotograma a fotograma AES-256-GCM SFrame (RFC 9605) com chave derivada do exportador MLS, ligação SAS e um relé SFU de conhecimento zero. Capítulos seguintes renumerados 13-23. |

| Version | Date | Changes |
|---|---|---|
| 1.5 | julho de 2026 | Novo capítulo 13: Tarefas Aionda - gestão de tarefas encriptada de ponta a ponta. Chaves AES-256-GCM por tarefa e por lista, envolvidas com a chave-mestra do cofre (HKDF-SHA-256 com separação de domínios específica do módulo), títulos, notas, subtarefas, nomes de listas e regras de recorrência encriptados, exposição honesta dos metadados de planeamento deliberadamente em claro, lista de e-mails sinalizados assente na estrela já existente, separação de chaves no modo de supervisão e validação de comprimento no servidor contra a truncagem silenciosa do texto cifrado. Capítulos seguintes renumerados 14-24. |
| 1.7 | 13 outubro 2026 | 25: Iniciar sessão num dispositivo por código QR (Web, iOS, macOS). |
| 1.8 | 13 outubro 2026 | 3.2: Clientes nativos iOS/macOS, arquitetura Swift/SwiftUI partilhada e fronteiras de confiança dos componentes WebKit integrados. |

---

## Contacto

**Aionda GmbH** Stephan Ferraro Stuttgart, Germany

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

---

*Este documento descreve a arquitetura de segurança da Aionda Mail à data de 13 de outubro de 2026. Os sistemas criptográficos evoluem — este documento será atualizado à medida que a arquitetura mudar.*