Aionda Mail Security Whitepaper

Zero-Knowledge · Post-Quantum · Made in Germany

Version 1.10 — 2026-11-08

Aionda GmbH
Stuttgart, Germany

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

PUBLIC DOCUMENT

Contents

1. Zusammenfassung . . . 3
2. Bedrohungsmodell . . . 3
3. Architekturübersicht . . . 5
4. Zero-Knowledge-Authentifizierung (OPAQUE) . . . 8
5. Vault Master Key & Shamir Secret Sharing . . . 9
6. Post-Quantum Hybrid KEM . . . 11
7. E-Mail-Verschlüsselungspipeline . . . 12
8. Verschlüsselte API-Transportschicht . . . 14
9. Ordnerfreigabe-Kryptografie . . . 16
10. Vault Drive — Verschlüsselter Dokumentenspeicher . . . 17
11. Aionda Chat — Post-Quantum-E2EE-Messaging . . . 23
12. Aionda Video-Anrufe — Post-Quantum-E2EE-Konferenzen . . . 29
13. Aionda Aufgaben - Ende-zu-Ende-verschlüsselte Aufgabenverwaltung . . . 33
14. Seitenkanalschutz . . . 37
15. Schlüsselverwaltung & Lebenszyklus . . . 37
16. Wiederherstellungsmechanismus . . . 39
17. Passwortlose Authentifizierung (Passkeys) . . . 40
18. Guardian: MITM-Schutz & Response-Signierung . . . 41
19. Enterprise-E-Mail-Archiv (Blockchain) . . . 44
20. Was der Server sieht — und was nicht . . . 48
21. Algorithmenreferenz . . . 49
22. Vergleich mit anderen Anbietern . . . 51
23. Einschränkungen & ehrliche Grenzen . . . 52
24. Roadmap . . . 54
25. Geräteanmeldung per QR-Code . . . 54
26. Quelloffener MCP-Client: das Protokoll zum Nachprüfen . . . 55
Dokumentenhistorie . . . 56
Kontakt . . . 59

1. Zusammenfassung

Aionda Mail ist ein Zero-Knowledge-, Post-Quantum-verschlüsselter E-Mail-Dienst, betrieben von der Aionda GmbH in Stuttgart, Deutschland. Der Dienst kombiniert Wegwerf-E-Mail-Adressen (DEAs) mit einem vollständig verschlüsselten Postfach — eine Kombination, die kein anderer Anbieter bietet.

Zentrale Sicherheitseigenschaften:

- **Zero-Knowledge-Architektur:** Die clientseitige Entschlüsselung gespeicherter Postfachinhalte erfolgt auf dem Gerät des Nutzers, im Browser oder in der nativen App.
- **Post-Quantum-Sicherheit:** Ein hybrider Key Encapsulation Mechanism (X25519 + ML-KEM-1024) schützt alle Daten sowohl gegen klassische als auch gegen Quantencomputer-Angriffe.
- **Quelloffener MCP-Client (Apache-2.0):** Das Protokoll und alle Datenformate sind durch den quelloffenen MCP-Client öffentlich dokumentiert und nachprüfbar - einschließlich der Ende-zu-Ende-Verschlüsselung mit Post-Quanten-Verfahren (Kapitel 26).
- **Zero-Knowledge-Authentifizierung:** Das OPAQUE-Protokoll (RFC 9807) stellt sicher, dass Passwörter niemals an den Server übertragen oder dort gespeichert werden — nicht einmal als Hashes.
- **Shamir Secret Sharing (2-of-3):** Der Vault Master Key wird in drei Anteile aufgeteilt, geschützt durch Passwort, Passkey und Wiederherstellungsschlüssel. Beliebige zwei Anteile rekonstruieren den Master Key.
- **Perfect Forward Secrecy:** Jede API-Anfrage verwendet ein einmaliges, einzigartiges kryptografisches Schlüsselpaar. Die Kompromittierung einer Anfrage betrifft keine andere.
- **MITM-Schutz (Guardian):** Die Browser-Erweiterung verifiziert alle Serverantworten unabhängig über Ed25519-Signaturen und erkennt Man-in-the-Middle-Angriffe durch TLS-Zertifikatsverifizierung — auch gegen Firmen-Proxys und kompromittierte CDNs.
- **GoBD-konformes E-Mail-Archiv:** Enterprise-Konten profitieren von einer manipulationssicheren Hash-Kette (SHA3-256 Blockchain) mit Ende-zu-Ende-verschlüsseltem Inhalt (Hybrid KEM), vollständigem Audit Trail, Legal Hold und konfigurierbarer Aufbewahrung — konform mit den deutschen GoBD-Vorschriften.
- **Kein Passwort-Recovery:** Bei Verlust des Passworts und aller Wiederherstellungsmethoden sind die Daten unwiederbringlich verloren. Das ist beabsichtigt — es beweist, dass der Server nicht auf die Nutzerdaten zugreifen kann.

Rechtsraum: Deutsches Recht (DSGVO/GDPR), keine Datenweitergabe an ausländische Geheimdienste.

---

2. Bedrohungsmodell

2.1 Wovor Aionda Mail schützt

Bedrohung	Schutz
Serverkompromittierung (Datenbankleck, Insider-Zugriff)	Alle E-Mail-Inhalte mit Schlüsseln verschlüsselt, die der Server nie besitzt

Bedrohung	Schutz
Netzwerküberwachung (ISP, WLAN, CDN)	Ende-zu-Ende-verschlüsselter API-Transport via Hybrid KEM
CloudFlare-Inspektion	API-Anfragen werden verschlüsselt, bevor sie den Browser verlassen; CloudFlare sieht nur Chiffretext. Die Guardian-Erweiterung erkennt Antwortmanipulation über Ed25519-Signaturen
Firmen-MITM-Proxys (ZScaler, Fortinet, etc.)	Guardian-Erweiterung erkennt Proxy-Zertifikate über Aussteller-Blocklist (Firefox)
Quantencomputer-Angriffe (“harvest now, decrypt later”)	ML-KEM-1024 (NIST FIPS 203) bietet Post-Quantum-Resistenz
Passwort-Datenbankdiebstahl	OPAQUE speichert nur kryptografische Records, keine Passwort-Hashes
Offline-Brute-Force-Angriffe auf Passwörter	OPAQUE verhindert Offline-Angriffe; serverseitige Ratenlimitierung verhindert Online-Angriffe
Analyse der E-Mail-Größe	Bucket Padding verschleiert die tatsächlichen E-Mail-Größen
Kompressions-Seitenkanäle (CRIME/BREACH)	Bucket Padding wird nach der Kompression angewendet
Nutzerenumeration	Deterministische Fake-Antworten für nicht existierende Konten

2.2 Wovor Aionda Mail NICHT schützt

Einschränkung	Erklärung
Kompromittiertes Gerät	Wenn Malware den Browser kontrolliert, kann sie entschlüsselte Inhalte lesen
Metadaten an externe Empfänger	E-Mails an Gmail/Outlook reisen nach Verlassen unserer Server unverschlüsselt (sofern nicht PGP verwendet wird)
E-Mail-Metadaten auf unseren Servern	Zeitstempel, IP-Adressen und verschlüsselte E-Mail-Größen sind für den Server sichtbar
Vertrauen bei Web-Zustellung	Der Browser lädt bei jedem Besuch JavaScript von unseren Servern herunter (siehe Abschnitt 20 für Gegenmaßnahmen)
Rubber-Hose-Kryptoanalyse	Kein kryptografisches System schützt gegen physischen Zwang

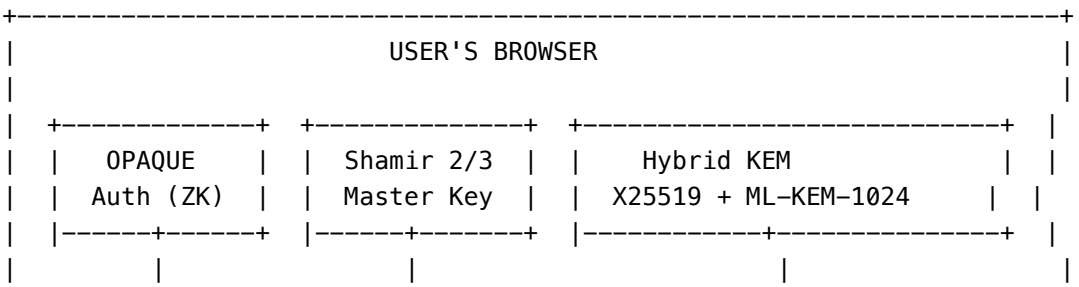
Einschränkung	Erklärung
Schlüsseltausch durch den Server	Der Server verteilt die öffentlichen Schlüssel. Ein böswilliger Server kann einem Absender für einen Kontakt, mit dem dieser noch nie geteilt hat, einen eigenen Schlüssel unterschieben (Trust on first use). Für spätere Wechsel erlaubt der Entwurf in Abschnitt 15.3 (signierte Schlüsselwechsel) einem pinnenden Client, einen legitimen von einem untergeschobenen Wechsel zu unterscheiden – sobald der Serverteil ausgeliefert ist und die Clients Ketten prüfen. Bis dahin hält der Apple-Client bei jedem Wechsel an (seit 4. Oktober 2026), und der Web-Client kapselt auf den Schlüssel, den der Server nennt. Ein Server kann einen Client ausserdem auf einen älteren, legitim signierten Zwischenstand lenken; das schadet nur, wenn dieser Zwischenschlüssel kompromittiert ist. Mit der Einführung des Fingerabdrucks, der den Signierschlüssel einschliesst, halten pinnende Clients einmal bei jedem Kontakt mit Signierschlüssel an; ein untergeschobener Schlüssel fällt dabei nur auf, wenn der Kunde den Abdruck wirklich vergleicht.

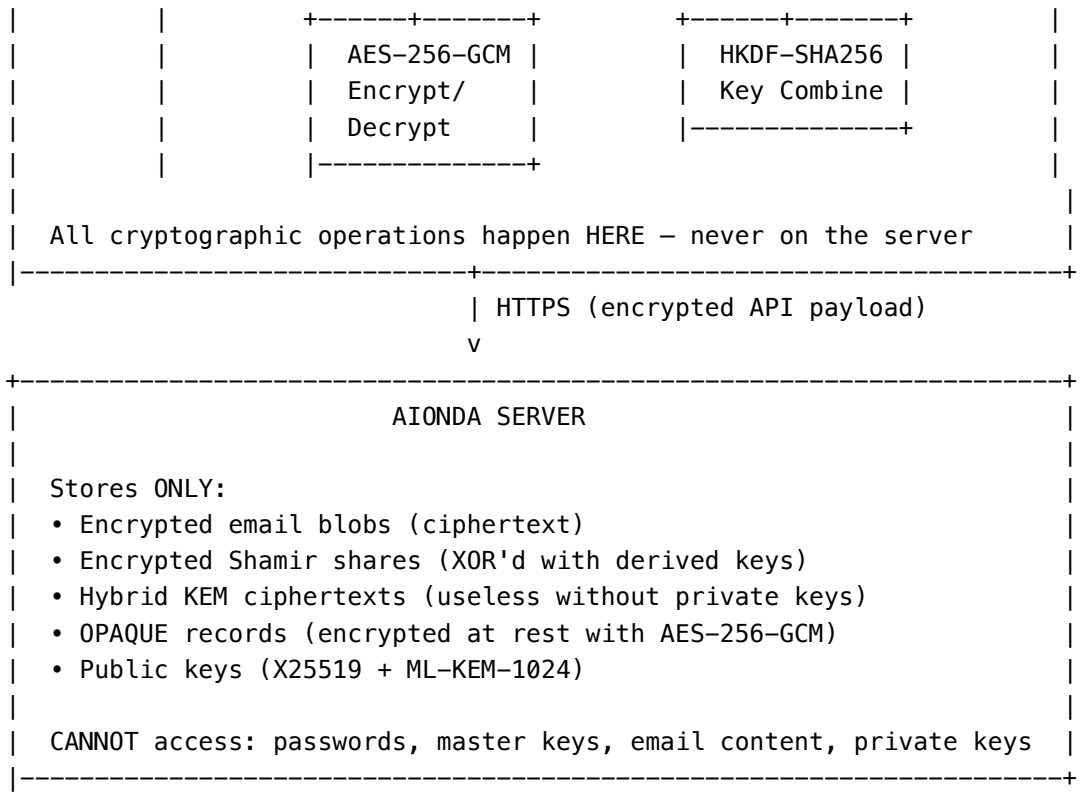
2.3 Designprinzip

Aionda Mail folgt dem **“Honest Server”-Modell**: Das System ist so konzipiert, dass selbst ein vollständig kompromittierter Server — oder ein böswilliger Betreiber — die Nutzerdaten nicht entschlüsseln kann. Sicherheit basiert nicht auf Vertrauen in uns. Sie basiert auf Mathematik.

---

3. Architekturübersicht





3.1 Komponentenübersicht

Komponente	Zweck	Ort
OPAQUE (RFC 9807)	Zero-Knowledge-Passwort-Authentifizierung	Client + Server
Shamir Secret Sharing	Vault Master Key Schutz (2-of-3 Threshold)	Nur Client
Hybrid KEM (X25519 + ML-KEM-1024)	Post-Quantum Key Encapsulation für E-Mails	Client + Server
AES-256-GCM	Authentifizierte symmetrische Verschlüsselung	Client + Server
HKDF-SHA256	Schlüsselableitung aus hybriden Shared Secrets	Client + Server
BIP39	Wiederherstellungsschlüssel-Kodierung (24-Wort-Mnemonik)	Nur Client
WebAuthn PRF	Passkey-basiertes Vault-Unlock	Nur Client
Bucket Padding	Seitenkanalschutz	Client + Server

3.2 Native Clients für iOS und macOS

Aionda Mail bietet native Anwendungen für iOS (iPhone und iPad) und macOS. Beide nutzen eine gemeinsame Swift-/SwiftUI-Codebasis mit plattformspezifischen Anwendungseinstiegen

und Anpassungen der Benutzeroberfläche. Das gemeinsame Paket `AiondaMailCore` implementiert die Kernfunktionen; `AiondaMailUI` stellt die SwiftUI-Oberfläche bereit.

Die native Anwendungsoberfläche wird als Teil der installierten App ausgeliefert. Das unterscheidet sie vom Browser-Client, der seinen Anwendungscode über die Web-Auslieferung lädt. Verweise auf Browser-APIs, JavaScript und sessionStorage in diesem Dokument beschreiben die Web-Implementierung; sie legen nicht die lokale Speicherung oder Laufzeit der nativen Clients fest.

Nativ bedeutet nicht, dass jede Komponente auf Webtechnologie verzichtet: Eingebettete WebKit-Ansichten stellen HTML-E-Mails dar, unterstützen das Verfassen von HTML-Nachrichten und bilden die Medienoberfläche für Videokonferenzen. Die HTML-Leseansicht deaktiviert Inhalts-JavaScript und verwendet einen nicht persistenten Website-Datenspeicher. Diese Einstellungen der Leseansicht gelten nicht pauschal für den Editor oder die Videokonferenz-Ansichten, die JavaScript benötigen.

Die native Architektur allein ist keine Sicherheitsgarantie. Vertrauen in die installierte App, ihre Updates und das Betriebssystem bleibt erforderlich; ein kompromittiertes Endgerät kann auf entschlüsselte Inhalte zugreifen. Bei eingebetteten Komponenten, die Webinhalte laden, bleibt die Web-Auslieferung Teil der Vertrauensgrenze.

3.3 Android-Client

Aionda Mail für Android ist eine native App (Kotlin, Jetpack Compose). Die Kryptografie liegt in einem gemeinsamen Rust-Kern, der über JNI eingebunden ist: ML-KEM-1024 und X25519 für den hybriden KEM, AES-256-GCM, ML-DSA für Signaturen und OPAQUE (`opaque-ke`) für die Anmeldung. Die Verfahren und Datenformate sind dieselben wie in Web-App, Apple-Apps und MCP-Client.

Lokale Geheimnisse (zum Beispiel Sitzungs- und Entsperrmaterial) legt die App nur verschlüsselt ab, mit einem AES-256-GCM-Schlüssel im Android Keystore, auf Geräten mit StrongBox im Sicherheitschip. HTML-E-Mails zeigt eine WebView ohne JavaScript, ohne DOM-Speicher, ohne Datei- und Inhaltszugriff und ohne Netzzugriff an; Bilder aus dem Netz lädt die App erst auf ausdrücklichen Wunsch, ohne Cookie und ohne Referer.

Bezug: Google Play und ein eigenes F-Droid-Repository (`https://aionda.com/fdroid/repo`) mit einer Variante ohne Google-Dienste, signiert mit einem eigenen Release-Schlüssel.

3.4 Alle Clients im Überblick

Client	Bezug	Technik	Kryptografie läuft	Quelltext
Web-App	Browser, `mail.aionda.com`	TypeScript, bei jedem Aufruf vom Server geladen (Abschnitt 23.1)	im Browser	nicht offen
iOS und iPadOS	App Store	Swift/SwiftUI, `AiondaMailCore` (Abschnitt 3.2)	auf dem Gerät	nicht offen

Client	Bezug	Technik	Kryptografie läuft	Quelltext
macOS	Mac App Store	gemeinsame Codebasis mit iOS (Abschnitt 3.2)	auf dem Gerät	nicht offen
Android	Google Play, eigenes F-Droid-Repository	Kotlin/Jetpack Compose, Rust-Kryptokern (Abschnitt 3.3)	auf dem Gerät	nicht offen
Browser-Erweiterung	Chrome, Firefox; Safari als Teil der macOS-App	Guardian-Prüfung in Chrome und Firefox (Kapitel 18), Wegwerfadressen	in der Erweiterung	nicht offen
MCP-Client	npm, Node.js	TypeScript (Kapitel 26)	auf dem Rechner des Nutzers	**offen, Apache-2.0**

Alle Clients verwenden dieselben Verfahren und Datenformate (Kapitel 6 und 7). Eine Windows-App (WinUI 3 mit demselben Rust-Kern wie Android) ist in Entwicklung und noch nicht veröffentlicht.

---

4. Zero-Knowledge-Authentifizierung (OPAQUE)

4.1 Warum kein Passwort-Hashing?

Traditionelle Dienste speichern Passwort-Hashes (bcrypt, Argon2). Obwohl das besser als Klartext ist, hat dieser Ansatz grundlegende Schwächen:

- Der Server sieht das Passwort während des Logins (auch wenn nur kurz im RAM)
- Passwort-Hashes können offline per Brute-Force geknackt werden, wenn die Datenbank gestohlen wird
- Der Server könnte so modifiziert werden, dass er Passwörter mitloggt

OPAQUE eliminiert alle drei Probleme. Das Passwort verlässt niemals den Browser — nicht als Klartext, nicht als Hash, in keiner Form.

4.2 Wie OPAQUE funktioniert

OPAQUE (RFC 9807) ist ein asymmetrisches Password-Authenticated Key Exchange (aPAKE) Protokoll. Es verwendet einen kryptografischen Challenge-Response-Mechanismus, bei dem der Server verifizieren kann, dass der Nutzer das richtige Passwort kennt, ohne jemals zu erfahren, wie dieses Passwort lautet.

Registrierung (einmalig):

```
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 (bei jeder Sitzung):

```
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)
```

Zentrale Eigenschaften:

- Das Passwort wird **clientseitig** in Schritt 3 verifiziert — der Server sieht es nie
- Der Server speichert einen **OPAQUE Record**, der kein Passwort-Hash ist und nicht offline per Brute-Force geknackt werden kann
- OPAQUE Records werden zusätzlich **at rest mit AES-256-GCM** unter Verwendung eines serverseitigen Schlüssels verschlüsselt
- **Schutz vor Nutzerenumeration**: Nicht existierende Konten erhalten deterministische Fake-Antworten mit identischem Timing
- **Ratenlimitierung**: Maximal 3 Authentifizierungsversuche pro Sitzung, 120 Sekunden Sitzungs-Timeout

4.3 Implementierung

- **Bibliothek:** `@serenity-kit/opaque` (WASM-basiert, produktionsreif)
- **Serverkomponente:** Dedizierter Microservice für kryptografische OPAQUE-Operationen
- **Base64-Format:** base64url (URL-sicher, ohne Padding) für Protokollkompatibilität
- **Audit-Logging:** Alle Authentifizierungsereignisse mit Zeitstempeln und IP-Adressen protokolliert

4.4 SRP-Migration

Legacy-Konten, die SRP-6a verwenden, werden beim nächsten Login automatisch zu OPAQUE migriert. Nach der Migration wird der SRP-Verifier dauerhaft gelöscht. Die Migration ist einmalig — Konten können nicht zu SRP zurückkehren.

---

5. Vault Master Key & Shamir Secret Sharing

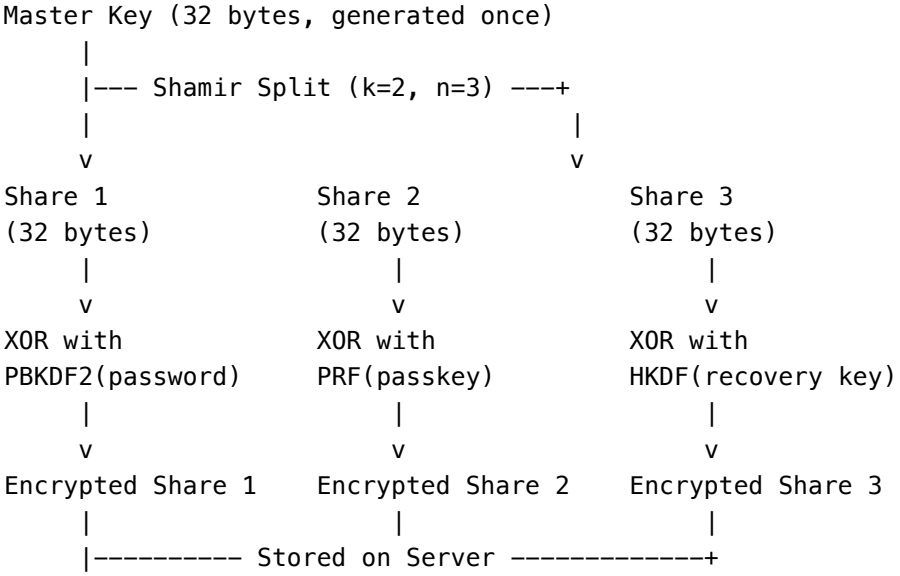
5.1 Master Key Generierung

Wenn ein Nutzer das verschlüsselte Postfach aktiviert, wird ein **256-Bit (32-Byte) Master Key** mit dem kryptografisch sicheren Zufallszahlengenerator des Browsers (`crypto.getRandomValues`) generiert.

Dieser Master Key ist die Wurzel aller Verschlüsselung. Er verlässt niemals den Browser im Klartext. Er wird nirgendwo gespeichert — nicht im Browser, nicht auf dem Server, in keiner Form.

5.2 Shamir Secret Sharing (2-of-3)

Der Master Key wird mittels **Shamirs Secret Sharing**-Verfahren über dem Galois-Feld GF(2^8) mit dem AES-irreduziblen Polynom (x^8 + x^4 + x^3 + x + 1) in drei Anteile aufgeteilt.



Threshold-Eigenschaft: Beliebige 2 der 3 Anteile reichen aus, um den Master Key via Lagrange-Interpolation zu rekonstruieren. Der Server speichert nur die verschlüsselten Anteile — und kann keinen davon entschlüsseln.

5.3 Anteilsschutz

Jeder Anteil wird mit einem Schlüssel per XOR verknüpft, der aus einem anderen Authentifizierungsfaktor abgeleitet wird:

Anteil	Geschützt durch	Schlüsselableitung
Anteil 1	Passwort	PBKDF2-SHA256, 600.000 Iterationen, 32-Byte-Zufallssalt
Anteil 2	Passkey (FIDO2)	WebAuthn PRF Extension, hardwaregebunden
Anteil 3	Wiederherstellungsschlüssel	HKDF-SHA3-256 mit kontogebundenem Salt

Rekonstruktionsszenarien:

- **Normaler Login:** Passwort (Anteil 1) + Passkey (Anteil 2) -> Master Key

- **Passkey verloren:** Passwort (Anteil 1) + Wiederherstellungsschlüssel (Anteil 3) -> Master Key
- **Passwortwechsel:** Passkey (Anteil 2) + Wiederherstellungsschlüssel (Anteil 3) -> Master Key

5.4 Session-Storage-Schutz

Selbst innerhalb einer Browser-Sitzung wird der Master Key niemals im Klartext gespeichert:

1. Ein ephemerer AES-256-Schlüssel wird generiert
2. Der Master Key wird mit diesem ephemeren Schlüssel verschlüsselt
3. Nur der verschlüsselte Blob wird in `sessionStorage` abgelegt
4. Der ephemere Schlüssel existiert nur im JavaScript-Speicher (wird beim Schließen des Tabs garbage-collected)

---

6. Post-Quantum Hybrid KEM

6.1 Warum Post-Quantum?

Quantencomputer, die Shors Algorithmus ausführen, könnten klassische Public-Key-Kryptografie (RSA, ECDH, X25519) in polynomialer Zeit brechen. Obwohl großskalige Quantencomputer noch nicht existieren, ist die Bedrohung durch **“harvest now, decrypt later”**-Angriffe real: Angreifer könnten verschlüsselte Daten heute speichern und entschlüsseln, sobald Quantencomputer verfügbar werden.

6.2 Hybrider Ansatz

Aionda Mail verwendet einen **hybriden Key Encapsulation Mechanism**, der kombiniert:

- **X25519** (Curve25519 ECDH) — bewährte klassische Sicherheit, 128-Bit-Sicherheitsniveau
- **ML-KEM-1024** (NIST FIPS 203, ehemals Kyber-1024) — Post-Quantum-Sicherheit, NIST Security Level 5

Der hybride Ansatz bietet **Defense-in-Depth**: Der kombinierte Schlüssel ist sicher, solange **mindestens einer** der beiden Algorithmen ungebrochen bleibt.

6.3 Encapsulation-Prozess

```
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 Decapsulation-Prozess

```
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 Schlüsselgrößen

Parameter	Größe	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 Bibliothek

- **ML-KEM-1024:** `@noble/post-quantum` (auditiert, reine JavaScript-Implementierung)
- **X25519:** WebCrypto API (`crypto.subtle.deriveBits`)
- **HKDF:** `@noble/hashes` (RFC 5869 konform)

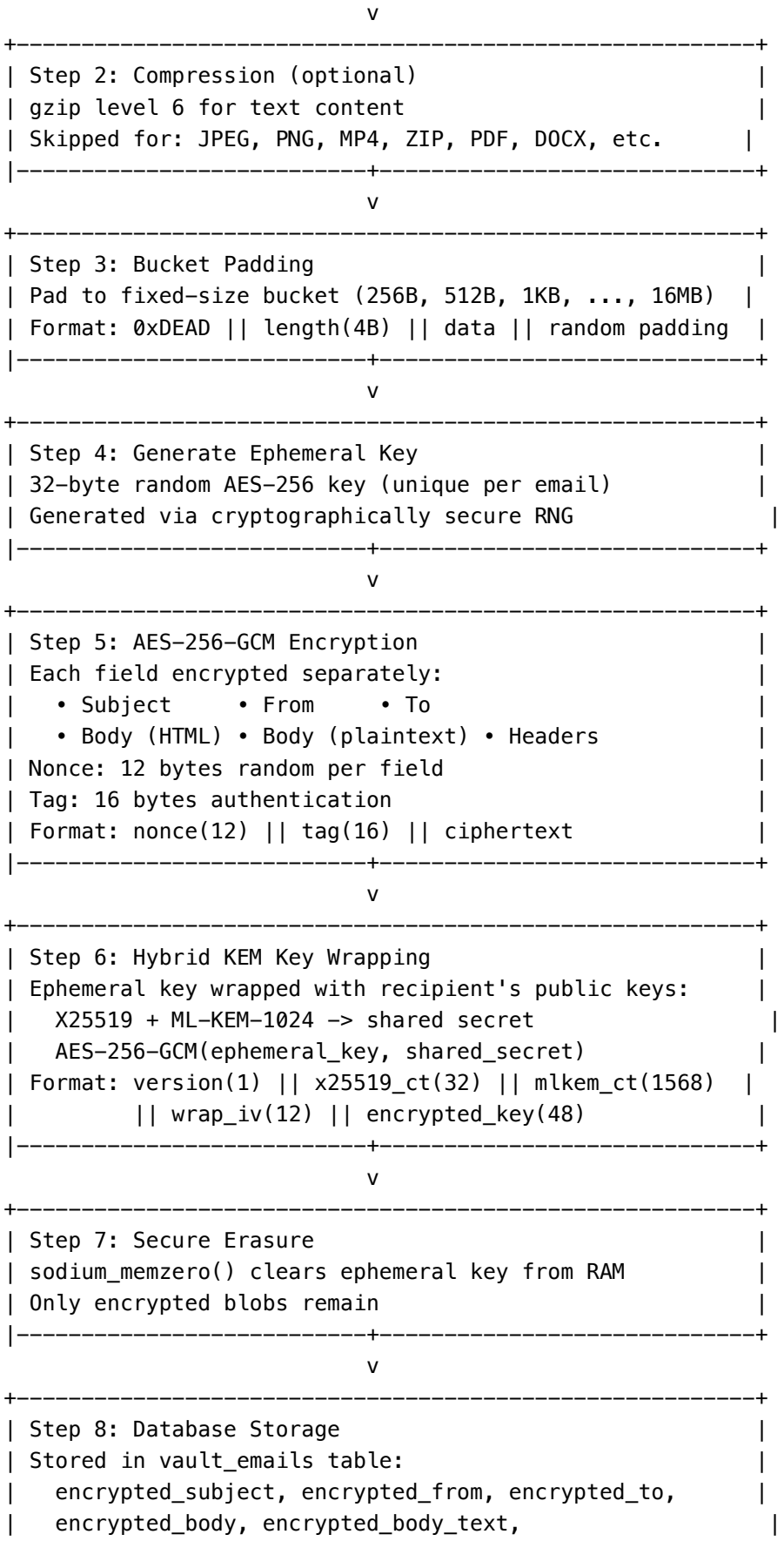
---

7. E-Mail-Verschlüsselungspipeline

7.1 Eingehende E-Mail (SMTP -> Verschlüsselte Speicherung)

Wenn eine E-Mail bei Aionda Mails SMTP-Server eintrifft:

```
+-----------------------------------------------------------+
| Step 1: SMTP Reception                                    |
| Mail server receives email -> passes to SMTP handler      |
|-------------------------+---------------------------------+
```


v
Step 2: Compression (optional)
gzip level 6 for text content
Skipped for: JPEG, PNG, MP4, ZIP, PDF, DOCX, etc.
v
Step 3: Bucket Padding
Pad to fixed-size bucket (256B, 512B, 1KB, ..., 16MB)
Format: 0xDEAD || length(4B) || data || random padding
v
Step 4: Generate Ephemeral Key
32-byte random AES-256 key (unique per email)
Generated via cryptographically secure RNG
v
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
v
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)
v
Step 7: Secure Erasure
sodium_memzero() clears ephemeral key from RAM
Only encrypted blobs remain
v
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 E-Mail lesen (Browser-Entschlüsselung)

Der umgekehrte Prozess findet vollständig im Browser statt:

1. Verschlüsselte E-Mail vom Server über die verschlüsselte API abrufen
2. Wrapped Ephemeral Key parsen (X25519-Chiffretext + ML-KEM-Chiffretext extrahieren)
3. **Hybrid KEM Decapsulation** mit den Vault Private Keys -> Shared Secret
4. Ephemeren AES-256-Schlüssel entpacken
5. **AES-256-GCM-Entschlüsselung** jedes Feldes (Betreff, Von, An, Body, Headers)
6. Byte-Reihenfolge anpassen: Server-Format (nonce || tag || ct) -> WebCrypto-Format (nonce || ct || tag)
7. Bucket Padding entfernen (0xDEAD Magic Bytes erkennen)
8. Dekomprimieren falls gzip (0x1F8B Magic Bytes erkennen)
9. UTF-8 zu Klartext dekodieren

7.3 E-Mail senden

Beim Verfassen und Senden einer E-Mail:

1. Der Client verschlüsselt den E-Mail-Inhalt mit einem Challenge-Response-Protokoll
2. Der Server empfängt den verschlüsselten Payload, entschlüsselt ephemer (nur im Speicher) und sendet via SMTP
3. Der Server gibt die generierten MIME-Header an den Client zurück (verschlüsselt)
4. Der Client verschlüsselt eine Kopie mit dem Vault Master Key und speichert sie im Gesendet-Ordner
5. Der ephemere serverseitige Klartext wird sofort verworfen — nie auf die Festplatte geschrieben

7.4 Anhänge

Jeder Anhang wird unabhängig verschlüsselt:

- Separater ephemerer AES-256-Schlüssel pro Anhang
- Separates Hybrid KEM Key Wrapping pro Anhang
- Dateiname und MIME-Typ separat verschlüsselt
- Keine Kompression für bereits komprimierte Formate (JPEG, ZIP, PDF, etc.)

7.5 E-Mail-Threading (Zero-Knowledge)

E-Mail-Threading (Gruppierung zusammengehöriger E-Mails) verwendet ausschließlich **SHA-256-Hashes** der Message-ID- und In-Reply-To-Header. Der Server sieht nie die tatsächlichen Message-ID-Strings — er kann E-Mails anhand der Hash-Gleichheit gruppieren, ohne den Inhalt zu kennen.

---

8. Verschlüsselte API-Transportschicht

8.1 Problem

Selbst mit HTTPS können bestimmte Zwischeninstanzen den Datenverkehr inspizieren:

- **CloudFlare** (CDN/DDoS-Schutz) terminiert TLS und kann Klartext-Anfragen sehen
- **Firmen-Proxys** können TLS-Inspektion durchführen
- **API-Parameter** (wie `?cmd=read_email&id=123`) verraten Metadaten

8.2 Lösung: Ende-zu-Ende-verschlüsselte API

Die gesamte API-Kommunikation wird zusätzlich Ende-zu-Ende zwischen Browser und Anwendungsserver verschlüsselt, innerhalb des HTTPS-Tunnels:

```
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 Zentrale Eigenschaften

- **Einmalverwendung:** Jedes API-Schlüsselpaar wird genau einmal verwendet und dann dauerhaft invalidiert
- **Perfect Forward Secrecy:** Die Kompromittierung eines Request-Schlüssels betrifft keinen anderen Request
- **Sitzungsgebunden:** Schlüssel werden von einer bestimmten Sitzung beansprucht und können nicht von einer anderen wiederverwendet werden
- **Schlüsselpool:** Der Server hält ca. 100.000 vorgegenerierte Schlüsselpaare bereit

- **Auto-Refetch:** Der Client fordert automatisch neue Schlüssel an, wenn der Cache unter 10 fällt
- **Schlüssel-TTL:** Beanspruchte Schlüssel verfallen nach 24 Stunden
- **Bidirektional:** Sowohl Request ALS AUCH Response sind verschlüsselt — der Server gibt niemals Klartext zurück

8.4 Was CloudFlare sieht

Mit dieser Architektur sieht CloudFlare (oder jeder TLS-terminierende Proxy) nur:

- `POST /e` — einen einzelnen, opaken Endpunkt
- Einen binären Blob verschlüsselter Daten
- Keine API-Befehlsnamen, keine Parameter, keine E-Mail-IDs, keine Nutzerdaten

---

9. Ordnerfreigabe-Kryptografie

9.1 Freigabemodell

Nutzer können verschlüsselte Ordner mit anderen Aionda-Mail-Nutzern teilen. Der Freigabemechanismus verwendet den Hybrid KEM, um einen ordnerspezifischen Schlüssel für jeden Empfänger zu verschlüsseln.

9.2 Freigabe-Ablauf

```
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 Berechtigungsmodell

Berechtigung	Fähigkeit
`readonly`	Ordner-E-Mails lesen (nur Entschlüsselung)
`einliefern`	Neue E-Mails in den Ordner einfügen
`bearbeiten`	Ordnerinhalte bearbeiten
`antworten`	Auf E-Mails innerhalb des Ordners antworten
`vollzugriff`	Voller Zugriff einschließlich Eigentumsübertragung

9.4 Cross-Vault Copy (Re-Wrapping)

Wenn ein Empfänger eine E-Mail aus einem geteilten Ordner in seinen eigenen Vault kopiert, muss der E-Mail-Schlüssel für sein eigenes Hybrid-KEM-Schlüsselpaar **re-wrapped** werden. Dies geschieht vollständig clientseitig:

1. Shared Folder Key mit den privaten Schlüsseln des Empfängers entschlüsseln
2. Ephemeren Schlüssel der E-Mail mit dem Folder Key entschlüsseln
3. Ephemeren Schlüssel mit den eigenen öffentlichen Schlüsseln des Empfängers erneut encapsulieren
4. Re-wrapped Kopie im Vault des Empfängers speichern

Der Server erleichtert den Transfer, sieht aber nie irgendein Schlüsselmaterial im Klartext.

---

10. Vault Drive — Verschlüsselter Dokumentenspeicher

Vault Drive ist der Zero-Knowledge-Dokumentenspeicher von Aionda Mail. Im Gegensatz zu E-Mails, die ephemere Schlüssel pro Nachricht verwenden, nutzt Drive eine **Per-File-Key-Architektur**: Jedes Dokument bekommt einen eigenen 32-Byte AES-256-GCM-Schlüssel, der clientseitig generiert und mit dem Master Key gewrapped wird.

10.1 Warum Per-File-Key?

Die Per-File-Key-Architektur bietet drei zentrale Vorteile:

1. **Granulares Teilen:** Einzelne Dokumente können geteilt werden, ohne den Master Key preiszugeben. Der Server re-wrappt lediglich den File Key für den Empfänger — der Inhalt bleibt unverändert.
2. **Compromise Isolation:** Wird ein einzelner File Key kompromittiert (z.B. über einen geteilten Link), sind alle anderen Dokumente weiterhin geschützt.
3. **Effizientes Sharing ohne Re-Encryption:** Beim Teilen muss der Inhalt nicht neu verschlüsselt werden — alle Empfänger lesen denselben Ciphertext mit einem re-wrapped Key.

10.2 Upload-Ablauf

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

1. Generiere zufälligen File Key:
   fileKey = randomBytes(32)

2. Verschlüssele Content, Name, MIME-Typ:
   encContent = AES-256-GCM(content, fileKey, nonce1)
   encName    = AES-256-GCM(name,    fileKey, nonce2)
   encMime    = AES-256-GCM(mime,    fileKey, nonce3)

3. Wrappe File Key mit Master Key:
   wrappedKey = AES-256-GCM(fileKey, masterKey, nonce4)

4. Sende verschlüsseltes Paket:
   POST /e {
     encContent, encName, encMime,
     wrappedKey, wrappedKeyNonce
   }
                                                        5. Speichere in vault_files + vault_file_c
                                                           (reine Ciphertexte, kein Schlüsselmater
```

10.3 Download-Ablauf

```
1. Lade Chunk + wrapped_key vom Server
2. Unwrap File Key:
   fileKey = AES-256-GCM-decrypt(wrappedKey, masterKey, nonce)
3. Entschlüssele Content, Name, MIME:
   content = AES-256-GCM-decrypt(encContent, fileKey, nonce1)
```

Die Entschlüsselung findet in einem **Web Worker** statt, der den Master Key nur temporär im Speicher hält. Nach der Operation wird der Speicher überschrieben.

10.4 Sharing — Nur Schlüssel-Rewrap

Der zentrale Vorteil von Per-File-Keys zeigt sich beim Teilen: Der Content wird **nicht** neu verschlüsselt. Stattdessen wird nur der 32-Byte File Key mit dem Hybrid KEM des Empfängers neu verpackt.

```
Eigentümer                                              Empfänger
----------                                              ---------

1. Unwrap File Key:
   fileKey = AES-GCM-decrypt(wrappedKey, masterKey)

2. Lade Public Keys des Empfängers:
   recipient.x25519_pub, recipient.mlkem_pub

3. Hybrid KEM Encapsulation:
   hybridEncapsulate(recipient.x25519_pub,
                     recipient.mlkem_pub)
```

```
   -> {x25519_ct, mlkem_ct, sharedSecret}

4. Wrappe File Key mit sharedSecret:
   shareWrappedKey = AES-256-GCM(fileKey,
                                 sharedSecret, nonce)

5. Speichere Share-Record:
   INSERT INTO vault_file_shares {
     file_uuid, recipient_account_id,
     x25519_ephemeral, mlkem_ciphertext,
     share_wrapped_key, permissions
   }
```

```
6. Lade 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. Entschlüssele GLEICHEN Ciphertext:
   content = AES-GCM-decrypt(
     encContent, fileKey)
```

Der Eigentümer benötigt den Master Key des Empfängers nicht — nur die öffentlichen Hybrid-KEM-Schlüssel, die der Server im Klartext speichert.

10.5 Ordner-Sharing (rekursiv)

Wird ein Ordner geteilt, verarbeitet der Client rekursiv alle enthaltenen Dokumente und Unterordner. Jede Datei erhält einen eigenen Share-Record mit dem File Key des jeweiligen Dokuments, re-wrapped für den Empfänger. Der Ordner selbst hat keinen Ordner-Key — die Struktur wird über Eltern-UUIDs verknüpft.

Wichtig: Der KEM-Encapsulation-Schritt wird **pro Operation** durchgeführt, nicht pro Datei. Alle Dateien eines gebatchten Sharings verwenden dasselbe `sharedSecret` — was den Vorgang effizient macht, ohne die Sicherheit zu reduzieren (jede Datei hat dennoch ihren eigenen File Key).

10.6 Berechtigungsmodell

Berechtigung	Kann ausführen
Lesen	Herunterladen, Vorschau, Metadaten lesen
Vollzugriff	+ Umbenennen, neue Version hochladen, innerhalb des geteilten Ordners verschieben, neue Dokumente hochladen
Nicht möglich	Löschen (nur Eigentümer), aus geteiltem Ordner rausverschieben (nur Eigentümer)

Die Berechtigungen werden im `vault_file_shares`-Record gespeichert und serverseitig durchge-

setzt. Die Kryptografie schützt den Inhalt — die Zugriffskontrolle schützt die Operationen.

10.7 Rename, Overwrite und Metadaten-Updates

Bei geteilten Dokumenten werden Rename- und Overwrite-Operationen direkt auf dem `vault_files`-Record ausgeführt — nicht auf den individuellen Share-Records. Alle Empfänger sehen sofort den neuen Namen bzw. den neuen Inhalt, da sie denselben Ciphertext referenzieren.

Beim **Overwrite** (neue Version) wird ein **neuer File Key** generiert, der alte Ciphertext wird ersetzt, und alle bestehenden Share-Records werden clientseitig mit dem neuen Key re-wrapped. Der Eigentümer muss für diese Operation die Public Keys aller Empfänger laden.

10.8 Vorschau und In-Memory-Entschlüsselung

Bilder, PDFs und andere Dokumente werden für die Vorschau **ausschließlich im Arbeitsspeicher** entschlüsselt. Es gibt keinen Disk-Cache mit Plaintext-Daten. Der Client nutzt den **VaultDataLayer**, einen verschlüsselten IndexedDB-Cache, der Ciphertexte persistent zwischenspeichert und nur bei Bedarf entschlüsselt.

Thumbnails werden serverseitig **niemals** generiert — der Server sieht nie den Inhalt. Thumbnails werden clientseitig aus dem entschlüsselten Original erzeugt und ebenfalls verschlüsselt zwischengespeichert.

10.9 Was der Server sieht

Sichtbar für den Server	Nicht sichtbar
Verschlüsselter Content (Zufallsbytes)	Plaintext-Inhalt
Verschlüsselter Dateiname (Zufallsbytes)	Plaintext-Dateiname
Verschlüsselter MIME-Typ (Zufallsbytes)	Dateityp (Bild, PDF, Textdatei…)
Dateigröße (gepaddet auf Bucket-Grenzen)	Tatsächliche Originalgröße
Upload-Zeitpunkt	File Key, Master Key
Eigentümer-Account-ID	Inhalte von Unterordnern
Elternordner-UUID (für Struktur)	Ordnernamen (ebenfalls verschlüsselt)
Share-Records mit KEM-Ciphertexten	Empfänger-Master-Key, entwrappter File Key

10.10 Quota-Durchsetzung

Die Quota-Prüfung erfolgt serverseitig anhand der **verschlüsselten Dateigröße** (inklusive Bucket-Padding). Der Server kennt die tatsächliche Plaintext-Größe nicht — der Padding-Overhead wird bewusst vom Nutzer bezahlt, um Größen-Leakage zu verhindern.

Limits: - **Kostenlos:** 100 MB Gesamtspeicher, max. 10 MB pro Dokument - **Plus:** 5 GB Gesamtspeicher, max. 100 MB pro Dokument

10.11 Externes Teilen — öffentliche Share-Links

Vault-Drive-Dokumente können mit Empfängern geteilt werden, die **kein Aionda-Mail-Konto haben**, über öffentliche Share-Links auf der isolierten Domain `mail.aionda.com`. Die Funktion bleibt Zero-Knowledge, weil jeder Share als **eigenständige Krypto-Hülle** behandelt wird, entkoppelt vom Vault-Master-Key des Eigentümers.

Schutzmodi:

Modus	Vertrauensfaktor	Einsatz
`link_only`	URL-Fragment (wird nie an den Server übertragen)	Komfort — jeder mit dem Link kann zugreifen
`password`	Argon2id-abgeleiteter Schlüssel	Passwort-Übergabe außerhalb des Kanals (SMS, Telefon)
`recipient_pubkey`	Hybrid KEM (X25519 + ML-KEM-1024)	Post-Quantum-sicher, wenn der Empfänger vorab Public Keys hinterlegt hat

10.11.1 Share-Erstellung (Eigentümer-Client)

```
1. fileKey := AES-GCM-decrypt(vault_files.wrapped_key, masterKey)
2. shareKey := randomBytes(32)                          // ephemer, pro Share
3. wrappedFileKey := AES-GCM(fileKey, shareKey)         // neue Hülle
4. unlockVerifier := HMAC-SHA256(shareKey,
                     "unlock:" || shareUuid)            // Proof-of-Knowledge
5. Bei Passwort-Schutz:
     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>
   Sonst (link_only):
     -> Link: https://mail.aionda.com/s/<shareUuid>#<base64(shareKey)>
       (URL-Fragment — Browser übertragen Fragmente nie an den Server)

6. Verschlüsselte Metadaten (pro Share, mit shareKey):
     encFilename, encMime, encMessage  // alle AES-GCM
7. POST /e  (verschlüsselter API-Transport, siehe §8)
   -> Server speichert Share-Record — sieht nie Klartext
```

Die Übermittlung wählt der Eigentümer selbst — nicht Aionda Mail. Nach dem Erstellen entscheidet der Eigentümer, über welchen Kanal der Link geht: Link kopieren, QR-Code, `mailto:`-Entwurf im lokalen Mail-Programm, Signal, SMS, AirDrop oder jeder andere Out-of-Band-Kanal. Aionda Mail sieht, sendet oder loggt die ausgehende Zustellung nicht — der Server weiß nur, welche Share-UUIDs existieren. Das ist aus zwei Gründen wichtig: (a) Der Eigentümer wählt den für ihn vertrauenswürdigsten Kanal (Corporate-Compliance, bevorzugter Messenger, QR-Code face-to-face), und (b) bei passwortgeschützten Shares ermöglicht es **Kanal-Trennung** — Link über Kanal A, Passwort über Kanal B — sodass ein kompromittierter Kanal allein niemals Zugriff gewährt.

10.11.2 Share-Zugriff (Empfänger, kein Konto nötig)

Der Empfänger öffnet `https://mail.aionda.com/s/<`
Die Share-Page ist ein **eigenständiges Bundle** getrennt vom Manager, mit reproduzierbarem Build, Subresource-Integrity-Manifest und einem `/.well-known/integrity.json`-Endpunkt zur Offline-Verifikation.

```
1. Share-Page-Bootstrap (Client erzeugt ephemeres Hybrid-KEM-Keypair)
2. share_fetch_meta -> { wrapped_file_key, salt?, encrypted_message, ... }
3. Unlock-Phase — liefert ein einmaliges unlock_token:
```

```
     password-Modus:   Server verifiziert Argon2id-Ableitung -> unlock_token
     link_only-Modus:  Client berechnet HMAC-SHA256(shareKey, "unlock:"||uuid)
                       Server prüft gegen gespeicherten unlockVerifier
                       -> unlock_token
4. fileKey := AES-GCM-decrypt(wrappedFileKey, shareKey)
5. share_download_chunk (pro Chunk, unlock_token erforderlich)
6. Client hasht Klartext, ruft share_confirm_download
```

Das `unlock_token` vereinheitlicht den Ablauf für alle drei Schutzmodi und ermöglicht zentrales Rate-Limiting und Audit-Logging — link_only-Zugriffe erfordern Kenntnis des Fragments, sodass Bots und Crawler ohne Fragment keine Downloads auslösen können.

10.11.3 Erzwungene Richtlinien Jeder Share muss bei Erstellung folgendes deklarieren (alles serverseitig durchgesetzt):

- **`expires_at`** — Pflicht, Default 7 Tage, Maximum 90 Tage
- **`max_downloads`** — Pflicht-Zähler, Default 10
- **Rate-Limiting** — Argon2-Versuche bei Passwort-Shares werden pro IP-Hash und pro Share gedrosselt
- **`revoked_at`** — Der Eigentümer kann jeden Share per Klick widerrufen; künftige Unlock-Versuche und Downloads liefern eine einheitliche „Share nicht verfügbar"-Antwort (gleicher Fehler wie abgelaufen/ausgeschöpft — kein Enumerations-Orakel)

10.11.4 Manipulationssichere Audit-Kette Alle relevanten Zugriffsevents werden an die bestehende `enterprise_audit_log`-Hash-Chain angehängt (SHA3-256, siehe §18.2). Folgende Action-Types sind für externes Teilen reserviert:

`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- und User-Agent-Hashes verwenden einen **täglich rotierenden Salt** (`vault_drive_external_share_daily_` nach 30 Tagen gelöscht) — historische Hashes sind nach Salt-Rotation nicht mehr rückführbar (DSGVO-Konformität), die kurzfristige Korrelation für den Eigentümer bleibt möglich.

10.11.5 Schlüssel-Rotation (Share-Rewrap) Der Eigentümer kann einen Share jederzeit rotieren, ohne den Inhalt neu hochzuladen:

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

Empfänger mit dem alten Link erhalten „Share nicht verfügbar". Der Eigentümer verteilt den neuen Link. Zugriffsevents beider Shares bleiben über `linked_to_uuid` in der Audit-Kette verknüpft.

10.11.6 Was Widerruf leistet — und was nicht Widerruf **stoppt die künftige serverseitige Auslieferung** des Shares. Er widerruft **nicht** rückwirkend bereits ausgelieferte Share-Keys, File-Keys oder Chunks. Wer den Klartext bereits heruntergeladen oder das Link-Fragment abgegriffen hat, kann das Material lokal weiterverwenden. Für höhere Schutzbedürfnisse empfehlen

wir die Kombination aus: kurzem `expires_at`, niedrigem `max_downloads`, Passwort-Schutz und Guardian-Browser-Erweiterung für den Empfänger.

10.11.7 Was der Server sieht

Server sieht	Server sieht NICHT
`share_uuid`, `file_uuid`, `account_id` (Eigentümer)	Klartext-Dateiname, MIME-Typ, Inhalt, Nachricht
`wrapped_file_key`, `wrapped_share_key` (Ciphertexte)	`share_key` (bei link_only: lebt nur im URL-Fragment, clientseitig)
`unlock_verifier` (HMAC-Ausgabe, nicht umkehrbar)	Passwort, Argon2-abgeleiteter Schlüssel
`protection_mode`, `expires_at`, `max_downloads`, `allow_preview`	Empfänger-Identität (nur gesalzener IP/UA-Hash, nach 30 Tagen gelöscht)
`encrypted_filename`, `encrypted_mime`, `encrypted_message` (Ciphertexte)	Deren Klartexte
Klartext `sender_display_name` (Empfänger sieht ihn vor dem Unlock)	
Audit-Chain-Einträge für jedes Zugriffsevent	

---

11. Aionda Chat — Post-Quantum-E2EE-Messaging

Aionda Chat ist die integrierte Echtzeit-Messaging-Oberfläche von Aionda Mail. Im Gegensatz zu E-Mails — die einmalige ephemere Schlüssel pro Nachricht verwenden — sind Chat-Konversationen langlebig und benötigen für jede einzelne Nachricht **Forward Secrecy** und **Post-Compromise Security**. Dafür haben wir ein eigenes Protokoll entwickelt: **AAR (Aionda Async Ratchet)** — eine Post-Quantum-Variante von Signals X3DH + Double Ratchet, in der jeder Diffie-Hellman-Schritt durch eine hybride KEM (X25519 + ML-KEM-1024) ersetzt wird.

11.1 Warum ein eigenes Protokoll?

E-Mail und Chat haben grundlegend unterschiedliche Sicherheitsprofile:

Eigenschaft	E-Mail	Chat
Frequenz	Sporadisch (Minuten/Stunden)	Echtzeit (Sekunden)
Sitzungsdauer	Einzelne Nachricht	Kontinuierlich, Tage bis Jahre
Forward-Secrecy-Einheit	Pro Nachricht	Pro Nachricht **innerhalb** einer langen Sitzung
Post-Compromise Recovery	Nicht erforderlich (One-Shot-Key)	Erforderlich — jede neue Nachricht heilt einen früheren Compromise
Asynchronität	Hoch (Empfänger oft offline)	Hoch (Empfänger oft offline)

Eigenschaft	E-Mail	Chat
Gruppendynamik	Statische Empfängerliste	Dynamisches Hinzufügen/Entfernen

Eine Chat-Konversation, die einfach die E-Mail-Pipeline wiederverwendet, würde entweder (a) bei jedem Tastendruck ein frisches ephemeres Schlüsselpaar verbrennen (inakzeptabel langsam) oder (b) einen einzigen statischen Konversationsschlüssel teilen (keine Forward Secrecy). Beides ist nicht akzeptabel. AAR löst das, indem ein kontinuierlich rachetnder Schlüsselzustand pro Peer gepflegt wird — jede Nachricht treibt den Zustand unwiderruflich vorwärts.

11.2 Bausteine

```
AAR verwendet exakt vier Primitive:

   Hybride KEM        X25519 + ML-KEM-1024  (Schlüssel-Kapselung)
   AEAD               AES-256-GCM           (Nachrichten-Verschlüsselung)
   KDF                HKDF-SHA256           (Schlüsselableitung)
   Signaturen         Ed25519               (Identitätsbindung, SPK-Signatur)

Es werden keine neuen kryptografischen Annahmen eingeführt – jedes Primitiv
wird auch anderswo im System eingesetzt und wurde unabhängig auditiert.
```

11.3 KeyBundle — das asynchrone Handshake-Material

Jeder Nutzer veröffentlicht beim ersten Aktivieren des Chats ein **KeyBundle** auf dem Server. Das Bundle ermöglicht es Peers, Nachrichten zu beginnen, auch wenn der Nutzer offline ist.

```
KeyBundle (pro Account, durchgehend hybrid):

   Identitätsschlüssel (IK)   – langlebig
      |-- IK.x25519_pub
      |-- IK.mlkem_pub
      |-- IK.ed25519_pub      – wird zum Signieren des SPK genutzt

   Signed Pre-Key (SPK)       – wöchentliche Rotation
      |-- SPK.x25519_pub
      |-- SPK.mlkem_pub
      |-- Ed25519-Signatur    – signiert die Konkatenation, bindet SPK an IK

   One-Time Pre-Keys (OPK)    – Pool von ~100, atomar konsumiert
      |-- OPK.x25519_pub
      |-- OPK.mlkem_pub
```

Server-Speicherung: Es werden nur die öffentlichen Hälften aller Schlüssel gespeichert, plus Signaturen und Metadaten. Die privaten Hälften verlassen den ursprünglichen Browser nie. Der Server erzwingt ein atomares `UPDATE … LIMIT 1 SET consumed_ts = NOW()`, wenn ein OPK abgerufen wird, und garantiert damit Single-Use-Semantik unter konkurrierenden Anfragen.

Wenn der OPK-Pool unter 20 fällt, füllt der Client den Pool mit einem frischen Batch wieder auf — das verhindert, dass der asynchrone Handshake in einen degenerierten Modus ohne

One-Time-Entropie zurückfällt.

11.4 PQXDH-artiger Handshake (X3DH-PQ)

Um eine neue Konversation mit Bob zu starten, ruft Alice Bobs Bundle (`IK_B`, `SPK_B`, einen `OPK_B`) und Bobs Ed25519-Signatur auf `SPK_B` ab. Alice verifiziert die Signatur und führt dann vier hybride KEM-Encapsulations durch:

```
1. Verifiziere Ed25519-Signatur auf SPK_B  — bindet SPK an Langzeitidentität

2. Erzeuge ephemeren Alice-Schlüssel (EK_A):
   EK_A.x25519, EK_A.mlkem  (einmalig, sofort nach dem Handshake verworfen)

3. Vier hybride KEM-Encapsulations:
   ss1 = HKEM(IK_A_priv  ⊗ SPK_B_pub)    — Alice-Identität -> Bob-SPK
   ss2 = HKEM(EK_A_priv  ⊗ IK_B_pub)     — Alice-Ephemer -> Bob-Identität
   ss3 = HKEM(EK_A_priv  ⊗ SPK_B_pub)    — Alice-Ephemer -> Bob-SPK
   ss4 = HKEM(EK_A_priv  ⊗ OPK_B_pub)    — Alice-Ephemer -> Bob-OPK

   (jedes ss_i ist selbst die HKDF-Kombination aus einem X25519-Geheimnis
    UND einem ML-KEM-Geheimnis — siehe Abschnitt 6.4)

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

5. Vergiss jedes private ephemere Material; SK seedet das Ratchet.
```

Die Konstruktion garantiert:

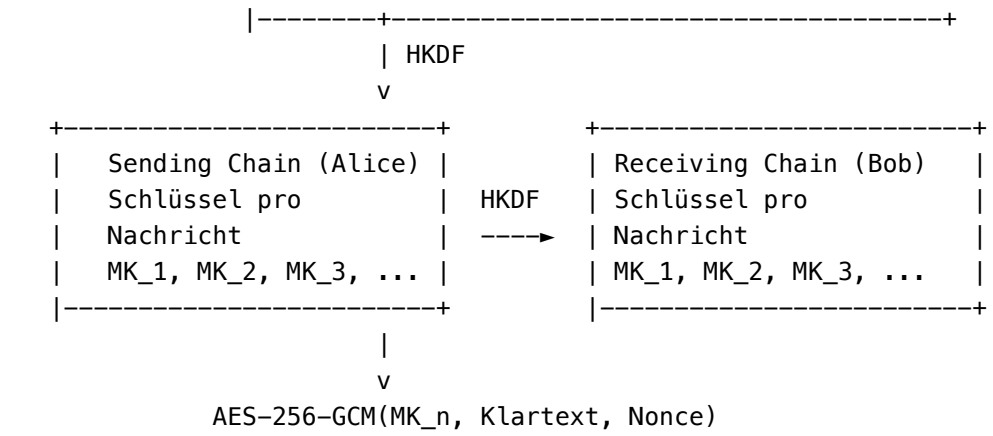
- **Wechselseitige Authentifizierung** über das Ed25519-signierte `SPK_B` und die `IK_A`-Bindung
- **Forward Secrecy** auch wenn `IK_B` später kompromittiert wird — `ss3` und `ss4` nutzen ephemere Schlüssel auf beiden Seiten
- **Post-Quantum-Sicherheit**, weil jedes `ss_i` einen ML-KEM-Term enthält — ein Angreifer, der heutigen Traffic für einen zukünftigen Quantencomputer sammelt, kann keines der vier Geheimnisse rekonstruieren
- **Replay-Resistenz** durch atomare OPK-Konsumption — Bobs serverseitiger Trigger weigert sich, denselben OPK zweimal herauszugeben

Die von Alice konsumierte OPK-ID wird in den `info`-Parameter von HKDF eingebunden — Bobs Seite kann denselben Root-Key daher nur mit derselben OPK reproduzieren, und nur einmal.

11.5 Double Ratchet — Schlüssel pro Nachricht

Nach dem Handshake schreitet auf beiden Seiten für jede Nachricht ein **Double Ratchet**-Zustand voran:

```
+-----------------------------------------------+
|       Root Chain (DH-Ratchet)                 |
|   schreitet voran, sobald eine neue           |
|   hybride KEM gesendet wird                   |
```



Jedes Chat-Event trägt den aktuellen **Ratchet-Public-Key** des Senders (hybrides Paar, ~1,6 KB). Wenn der Empfänger eine Nachricht erhält, deren Ratchet-Schlüssel sich vom zuletzt gesehenen unterscheidet, führen beide Parteien eine frische hybride KEM aus, leiten einen neuen Root-Key ab und resetten ihre Chains. Das ist der **DH-Ratchet** in Signal-Terminologie — nur dass jeder Schritt eine vollständige hybride KEM ist, nicht ein einzelner Diffie-Hellman.

Garantien pro Nachricht:

- **Forward Secrecy:** Das Löschen von `MK_n` macht Nachricht `n` undechiffrierbar, selbst bei Übergabe des vollständigen Langzeitzustands zum Zeitpunkt der Aufzeichnung
- **Post-Compromise Security:** Sobald nach einem Compromise ein frischer DH-Ratchet-Schritt stattgefunden hat, sind alle nachfolgenden Nachrichten vor dem Angreifer geschützt
- **Out-of-Order-Zustellung:** Nachrichten-Schlüssel werden für spät eintreffende Nachrichten gecached (gedeckelt auf 1000 Schlüssel pro Sitzung, um Speicher zu begrenzen)

11.6 Nachrichten-Container

Jedes auf den Server hochgeladene Chat-Event ist ein in sich geschlossenes Ciphertext-Paket:

```
MessageContainer (base64-kodiert, gespeichert in chat_events.payload):

+-------------------------------------------------------------+
| ratchet_pub      :  Hybride KEM-Public-Key (aktueller Step)  |
| prev_chain_len   :  Anzahl Nachrichten im vorherigen Chain   |
| msg_number       :  Zähler innerhalb des aktuellen Chains    |
| ciphertext       :  AES-256-GCM(MK_n, Klartext || Header)    |
| tag              :  AES-256-GCM-Authentifizierungs-Tag       |
|-------------------------------------------------------------+

Der Header innerhalb des AEAD-Klartexts enthält: Sender-Login,
content_type (text/quote/file-reference), Timestamp, optionales
reply_to_event_uuid.
```

Der Container ist für den Server opak — einschließlich des Nachrichtentyps, des Sender-Anzeige-Namens-Formats und jeglichen zitierten Antwort-Kontextes. Aiondas Datenbank speichert den Container as-is in `chat_events.payload`.

11.7 Gruppen-Konversationen — Sender-Keys pro Empfänger

Für Raum-Konversationen (3+ Teilnehmer) verwendet AAR ein **Per-Recipient-Fan-out**-Modell. Der Sender leitet eine Sender-Key-Chain pro Peer-Paar ab und verschlüsselt eine Nachricht einmal pro Empfänger. Jeder Empfänger erhält daher einen persönlich adressierten Ciphertext, entschlüsselbar nur mit den Schlüsseln aus seiner eigenen AAR-Sitzung mit dem Sender.

Trade-off: Die Fan-out-Kosten sind `O(N)` in Empfängern — akzeptabel für typische Teamgrößen (≤25 Teilnehmer). Für größere Gruppen ist ein zukünftiger MLS-basierter Modus geplant (siehe Roadmap). Das aktuelle Modell ist einem einzelnen geteilten Gruppenschlüssel vorzuziehen, weil: (a) es Forward Secrecy auf Per-Pair-Basis bewahrt, (b) es bei Mitglieder-Entfernung keine Schlüsselrotation jenseits der bilateralen Ebene erfordert, und (c) es die Post-Quantum-Garantien des paarweisen AAR ohne Modifikation erbt.

11.8 Initialer Konversations-Container

Wenn Alice einen neuen Raum mit Bob, Carol und Dan startet, wird die erste Nachricht an jeden Teilnehmer als **Initial-Handshake-Container** gewrappt, der (a) das X3DH-Handshake-Material gegen das Bundle dieses Teilnehmers und (b) die erste Nachricht selbst trägt. Der Empfänger dispatcht auf das Discriminator-Feld:

```
{
  "type": "initial",
  "initialMessage": { /* Handshake-Metadaten, konsumierte OPK-ID */ },
  "encryptedMessage": "<base64 MessageContainer>"
}
```

Folgenachrichten im selben Raum nutzen die etablierte AAR-Sitzung und enthalten nur noch den `encryptedMessage`-Teil.

11.9 Transport — Verschlüsselte API + Mercure SSE + WebSocket

Die Chat-Ebene nutzt drei orthogonale Kanäle:

Kanal	Richtung	Zweck	Vom Server gesehener Klartext
`/e` Verschlüsselte API	Client -> Server	Nachricht senden, History abrufen, KeyBundle-Operationen (11 Endpoints)	Keiner — alle 11 Chat-Endpoints nutzen denselben Hybrid-KEM-Transport aus Abschnitt 8
Mercure SSE	Server -> Client	Push-Zustellung neuer Events (`chat.event_received`, `chat.read_receipt`)	Keiner — der Server pusht denselben opaken MessageContainer, den er gespeichert hat

Kanal	Richtung	Zweck	Vom Server gesehener Klartext
WebSocket `/chat/ws`	Bidirektional	Reconnect-Backfill (`sync_request`), Heartbeat, Presence	Nur Presence-Boolean (online/offline) und Login-String — niemals Nachrichten-Inhalte

Presence wird in einer In-Memory-Map auf `aionda_chat_realtime` gehalten; ein einziger intranet-only HTTP-Endpoint stellt eine `login -> online`-Lookup für den Team-Picker bereit, damit Nutzer sehen können, ob ein Peer erreichbar ist. Keine History, kein Read-State, kein Inhalt durchquert je den Presence-Kanal.

11.10 Lesebestätigungen

Lesebestätigungen sind eine Opt-in-Einstellung pro Account (`chat_participants.send_read_receipts`). Wenn aktiviert, publiziert das Markieren einer Nachricht als gelesen ein `chat.read_receipt`-Event mit:

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

Beachte: `last_read_event_uuid` ist **nicht** der Nachrichteninhalt — es ist ein servergenerierter Identifier, der dem Server bereits bekannt ist. Es leakt nichts über "Alice hat jetzt Nachrichten bis Event X gesehen" hinaus. Empfänger, die Lesebestätigungen deaktiviert haben (`send_read_receipts = false`), emittieren niemals solche Events, und der Server erzwingt das Toggle.

11.11 Audit-Chain-Integration

Für Enterprise-Accounts wird jede Chat-Operation an die existierende manipulationssichere Audit-Log (`enterprise_audit_log`, SHA3-256-Hash-Chain — siehe §18) angehängt. Reservierte Aktionstypen sind:

`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`.

Der Audit-Eintrag enthält nur die Konversations-UUID, das Login des Akteurs und einen Timestamp — **niemals** den verschlüsselten Payload, Ratchet-Schlüssel oder Nachrichteninhalte.

11.12 Was der Server sieht

Vom Server sichtbar	Nicht sichtbar
`conversation_uuid`, Liste der Teilnehmer (per `mail_account.name`)	Nachrichteninhalte im Klartext
Verschlüsselte `MessageContainer`-Payloads (opake Ciphertexte)	Ratchet-Schlüssel, Chain-Keys, Message-Keys
Öffentliche Hälften von IK, SPK, OPK; Ed25519-Signaturen auf SPK	Private Hälften jeglicher Schlüsselpaare
OPK-Konsumptions-Counter und Timestamp	Welcher OPK für welche Konversation konsumiert wurde (Korrelation nur clientseitig via HKDF-info-Bindung verhindert)

Vom Server sichtbar	Nicht sichtbar
Konversations-Kadenz (Timestamps der `chat_events`-Zeilen)	Zitierte Antwort-Ketten, Datei-Anhänge im Container
Presence-Boolean (online/offline)	Tipp-Indikatoren (peer-to-peer, erreichen den Server nie)
Audit-Chain-Einträge für Enterprise-Accounts	Klartextinhalte der Audit-Einträge

11.13 Kryptografische Audits

Das AAR-Protokoll wurde von Grund auf in TypeScript implementiert und durchlief drei unabhängige KI-gestützte Audits, dokumentiert in `typescript/manager/chat/crypto/AUDIT.md`, `AUDIT_SECOND_OPINION.md` und `AUDIT_THIRD_OPINION.md`. Erkenntnisse jeder Runde wurden vor dem öffentlichen Deployment integriert. Hauptauditflächen waren: Handshake-Symmetrie, OPK-Reservierung unter Concurrency, Deep-Cloning unveränderlicher Ratchet-State und das AEAD-Framing des `MessageContainer`.

11.14 Implementierungsgrenzen

Die Chat-Codebasis lebt in `typescript/manager/chat/` (~9.000 Zeilen TypeScript) und `includes/classes/Api` (PHP-Service-Schicht). Wesentliche Audit-Artefakte:

```
crypto/aar-types.ts            — nur Typen, keine Logik
crypto/aar-keybundle.ts        — KeyBundle-Erstellung, Rotation, Serialisierung
crypto/aar-x3dh.ts             — Handshake (Initiator- und Responder-Pfad)
crypto/aar-double-ratchet.ts   — Root-Chain, Sending-Chain, Receiving-Chain
```

Die saubere Trennung zwischen Protokoll-Logik (`crypto/`) und Transport/UI (`chat-store.ts`, `chat-panel.ts`, …) stellt sicher, dass der kryptografische Kern ohne UI-Scope-Creep re-auditiert werden kann.

---

12. Aionda Video-Anrufe — Post-Quantum-E2EE-Konferenzen

Aionda Mail enthält Ende-zu-Ende-verschlüsselte Gruppen-Videoanrufe (Cipher-Suite `AIONDA_PQ_CALL_V1`). Das Medien-Relay sieht niemals Audio oder Video im Klartext — es leitet ausschließlich Chiffretext-Frames weiter. Gruppenschlüssel werden über einen **Post-Quantum-Hybrid**-Handshake vereinbart.

12.1 Sicherheitsziele & Vertrauensmodell

Im Geltungsbereich — das Protokoll bietet:

- **Vertraulichkeit** von Audio- und Video-Medien, Ende zu Ende.
- **Authentizität** der Gruppe — jeder Teilnehmer ist kryptografisch Mitglied derselben MLS-Gruppe, out-of-band verifiziert über den Short Authentication String (Abschnitt 12.5).
- **Forward Secrecy** — vergangene Medien bleiben geheim, selbst wenn aktuelle Schlüssel später kompromittiert werden.
- **Post-Compromise Security** — die Gruppe stellt die Vertraulichkeit wieder her, nachdem ein kompromittiertes Mitglied entfernt wurde und die Gruppe neu verschlüsselt.
- **Post-Quantum-Vertraulichkeit** — die Gruppenetablierung nutzt den X-Wing-Hybrid-KEM.

- **Server-Unabhängigkeit** — eine bösartige oder kompromittierte SFU, ein Signaling-Server oder eine Speicherschicht kann Medien weder lesen noch einschleusen.

Außerhalb des Geltungsbereichs (explizit *nicht* abgewehrt):

- Traffic-Analyse — Paket-Timing, Paketgrößen, Teilnehmerzahl und Anrufdauer sind beobachtbar (Abschnitt 12.9).
- Endpunkt-Kompromittierung (siehe Vertrauensannahme unten).
- Malware auf Betriebssystem-Ebene oder physische Beschlagnahmung des Geräts.

Endpunkt-Vertrauensannahme. Das Modell setzt einen nicht kompromittierten Browser und ein nicht kompromittiertes Betriebssystem voraus. Frames werden innerhalb der Medien-Pipeline des Browsers versiegelt (`RTCRtpScriptTransform`), aber ein kompromittierter Endpunkt kann Klartext-Medien *vor* der Verschlüsselung oder *nach* der Entschlüsselung lesen. Das Vertrauensmodell der Web-Zustellung und seine Gegenmaßnahmen (Guardian-Response-Signierung) werden in Abschnitt 18 und Abschnitt 23 behandelt.

12.2 Warum ein eigenes Protokoll?

Echtzeit-Konferenzen haben zwei Anforderungen, die die E-Mail- und Chat-Schicht nicht haben:

1. **Per-Frame-Verschlüsselung bei voller Übertragungsrate.** Audio-/Video-Frames müssen einzeln verschlüsselt werden, damit eine Selective Forwarding Unit (SFU) sie routen, verwerfen und umsortieren kann, ohne sie jemals zu entschlüsseln. Das ist die Rolle von **SFrame** (RFC 9605).
2. **Dynamische Gruppen-Schlüsseleinigung.** Teilnehmer treten dem Anruf mitten in der Sitzung bei und verlassen ihn. Jede Mitgliedschaftsänderung muss die Gruppe mit Forward Secrecy und Post-Compromise Security neu verschlüsseln. Das ist die Rolle von **MLS** (RFC 9420).

Die serverseitige SFU (`mediasoup`) wird als **nicht vertrauenswürdige Infrastruktur** behandelt: Sie ist ein Zero-Knowledge-Paket-Router, genau wie die Speicherschicht für E-Mails.

12.3 Cipher-Suite — `AIONDA_PQ_CALL_V1` (`0xFF01`)

Das MLS-Profil ersetzt die klassischen Primitive einer Standard-MLS-Suite durch Post-Quantum-Hybride.

Rolle	Algorithm/Standard	Anmerkungen
HPKE KEM	**X-Wing** (ML-KEM-768 + X25519)	Hybrid; `draft-connolly-cfrg-xwing-kem`
Signaturen	**ML-DSA-65**	NIST FIPS 204 (Dilithium), Level 3
KDF	HKDF-SHA-256	RFC 5869
AEAD (HPKE + SFrame)	**AES-256-GCM**	AES-256 bietet eine komfortable Marge gegen bekannte Quanten-Suchangriffe (Grover)

Kein Downgrade. Es gibt keinen rein klassischen Fallback. Kann der Post-Quantum-Handshake nicht abgeschlossen werden, scheitert der Anruf fail-closed — ein Downgrade ist ein harter

Fehler, niemals eine stille Abschwächung.

X-Wing-Reifegrad. X-Wing ist derzeit ein IETF/CFRG-Draft (`draft-connolly-cfrg-xwing-kem`). Das aktuelle Profil übernimmt es als bevorzugten Hybrid-KEM; zukünftige Protokollrevisionen können auf einen finalisierten, CFRG/NIST-standardisierten Hybrid-KEM migrieren, während die MLS- und SFrame-Schichten unverändert erhalten bleiben.

Kryptografische Agilität. Die Cipher-Suite-Kennung (`0xFF01`) bestimmt vollständig den KEM, den Signaturalgorithmus, die KDF und die AEAD. Eine zukünftige `AIONDA_PQ_CALL_V2` kann jedes Primitiv ersetzen, ohne die höheren Protokollschichten zu ändern; Suiten werden per Kennung ausgewählt und niemals stillschweigend nach unten ausgehandelt.

12.4 Gruppen-Schlüsseleinigung (MLS, RFC 9420)

Jeder Teilnehmer wird im MLS-Ratchet-Baum durch ein **KeyPackage** repräsentiert (öffentlicher X-Wing-KEM-Schlüssel + ML-DSA-65-Credential). Die Gruppe entwickelt sich über *Epochen*:

- **Welcome / Commit** — ein beitretendes Mitglied wird mit einem X-Wing-verkapselten Path-Secret hinzugefügt; der Ratchet-Baum schreitet in eine neue Epoche fort.
- **Neuverschlüsselung bei jeder Mitgliedschaftsänderung.** Jede Änderung — ein Beitritt, ein Verlassen, ein Geräteaustausch oder eine Rotation des Identitätsschlüssels — führt die Gruppe in eine frische Epoche. Das liefert Forward Secrecy (ein neuer Teilnehmer kann vergangene Medien nicht lesen) und Post-Compromise Security (ein entferntes Mitglied kann zukünftige Medien nicht lesen).
- **Remove-Commit.** Wenn ein Teilnehmer geht (oder ein veralteter/Zombie-Peer erkannt wird), gibt der Eigentümer einen Remove-Commit aus, der die Gruppe neu verschlüsselt, sodass ein ausgeschiedenes Mitglied nachfolgende Frames nicht entschlüsseln kann — selbst wenn es altes Schlüsselmaterial behalten hat.

Die Hybrid-Konstruktion ist so ausgelegt, dass die Kompromittierung nur **einer** Komponente — ML-KEM-768 *oder* X25519 — für sich allein das Gruppengeheimnis nicht preisgibt.

12.5 Identität & Authentifizierung

- **Credentials.** Jedes Mitglied trägt ein MLS-**Credential**, das eine Teilnehmeridentität an seinen ML-DSA-65-Signaturschlüssel bindet. Jede MLS-Handshake-Nachricht (Welcome / Commit / Update) wird mit diesem Schlüssel signiert, sodass die SFU oder der Signaling-Server Gruppenoperationen nicht fälschen kann.
- **Short Authentication String (SAS).** Der Raum-Link trägt ein **vorab geteiltes Geheimnis** mit hoher Entropie im URL-Fragment, das niemals im Klartext an den Server gesendet wird. Der MLS-Gruppen-Exporter und dieses Geheimnis werden zu einem Short Authentication String zusammengeführt, dessen Fingerprint der Client **pinnt**. Eine Nichtübereinstimmung — etwa ein Signaling-Server oder eine SFU, die versucht, einen unautorisierten Teilnehmer einzuschleusen — löst einen **harten Abbruch ohne Soft-Fallback** aus. Das bindet die kryptografische Gruppe an das out-of-band Raumgeheimnis und vereitelt serverseitigen Man-in-the-Middle.
- **Authentifizierungsfaktor.** Bei einem Gast-Raum ist *der Besitz des Raum-Links gleichbedeutend mit dem Besitz des Raum-Credentials* — das Raumgeheimnis ist der einzige Authentifizierungsfaktor. Authentifizierte Aionda-Teilnehmer lösen zusätzlich ein signiertes, kontogebundenes Signaling-Token ein (Abschnitt 12.8).

12.6 Medienverschlüsselung (SFrame, RFC 9605)

Jeder Medien-Track wird Frame für Frame verschlüsselt, **bevor** er die SFU erreicht. Der Schlüssel-Lebenszyklus:

```
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
```

- Der SFrame-Schlüssel wird aus dem **MLS-Exporter** abgeleitet, existiert also nur auf authentifizierten Gruppenmitgliedern — niemals auf dem Server.
- Die Verschlüsselung läuft in einem dedizierten `RTCRtpScriptTransform`-Worker (die standardisierte Encoded-Transform-API, verfügbar in Chromium und Safari/WebKit), sodass Frames in der Medien-Pipeline des Browsers versiegelt werden.
- Der Codec ist VP8; die SFU führt das Keyframe-Routing nur anhand von Metadaten durch — sie inspiziert niemals die verschlüsselte Payload.

12.7 Replay-Schutz

Jeder verschlüsselte Frame trägt einen monotonen SFrame-**Counter** in seinem authentifizierten Header und ist an die aktuelle MLS-Epoche gebunden. Empfänger verfolgen den höchsten gesehenen Counter pro Sender (ein persistierter empfängerseitiger Counter-Store). Ein Frame mit einem doppelten oder zurückgesetzten Counter oder einer, der an eine veraltete Epoche gebunden ist, wird abgelehnt — ein aufgezeichneter Chiffretext kann nicht in die Sitzung wiedereingespielt werden.

12.8 Signaling- & Zugriffsmodell

- **Signaling** läuft über einen authentifizierten WebSocket (`aionda_signaling`). Jede Sitzung wird durch ein kurzlebiges **JWT** abgesichert, das vom Anwendungsserver ausgestellt wird, nachdem der Client das Raumgeheimnis über die verschlüsselte `/e`-API einlöst (siehe Abschnitt 8). Das Token ist symmetrisch (HS256) und wird ausschließlich zwischen dem Anwendungsserver und dem einzigen Signaling-Dienst validiert; es autorisiert das Session-Setup und trägt niemals Medien- oder Gruppengeheimnisse. Der Signaling-Server leitet MLS-Welcome-/Commit-Nachrichten weiter, kann sie aber nicht lesen.
- **Raum-Link = Bearer-Credential.** Jeder, der den vollständigen Link besitzt (Raum-UUID + Fragment-Geheimnis), darf beitreten — das Modell ist bewusst „wer den Link hat". Datenbank-IDs werden niemals offengelegt; Räume werden ausschließlich über nicht erratbare UUIDs adressiert.
- **Eigentümer-Kontrolle.** Der Raum-Eigentümer (serverseitig über die Konto-Eigentümerschaft verifiziert) kann einen Raum als *beendet* markieren, um zukünftige Beitritte zu blockieren, und einen anwesenden Teilnehmer über einen MLS-Remove-Commit entfernen (Abschnitt 12.4).

12.9 Was die SFU / der Server sieht — und Metadaten-Grenzen

Die Infrastruktur KANN sehen	Die Infrastruktur KANN NICHT sehen
Verschlüsselte Frame-Größen, Timing, Routing-Metadaten	Audio- oder Video-Inhalte (Frames sind AES-256-GCM-versiegelt)
MLS-Handshake-Chiffretext (Welcome/Commit)	Gruppengeheimnisse oder SFrame-Schlüssel (clientseitig aus dem MLS-Exporter abgeleitet)
Raum-UUID, Teilnehmerzahl, JWT-Subject	Das Raum-Link-Geheimnis (lebt im URL-Fragment)
ICE-Kandidaten / Transport-Adressen	Jeglichen Klartext, der das Beitreten oder Entschlüsseln der Sitzung erlauben würde

Metadaten, die das System bewusst nicht verbirgt: IP-Adressen, die der ICE-/STUN-/TURN-Infrastruktur offengelegt werden, Paket-Timing, Paketgrößen, Teilnehmerzahl und Anrufdauer. Resistenz gegen Traffic-Analyse ist kein Designziel (Abschnitt 12.1).

12.10 Sicherheitseigenschaften (Zusammenfassung)

Eigenschaft	Garantie
Vertraulichkeit	Nur aktuelle MLS-Gruppenmitglieder besitzen die SFrame-Schlüssel.
Authentifizierung	Alle MLS-Nachrichten sind mit ML-DSA-65 signiert; die Gruppe ist über den gepinnten SAS an das Raumgeheimnis gebunden.
Forward Secrecy	Die Kompromittierung aktueller Geheimnisse gibt vergangene Medien nicht preis.
Post-Compromise Security	Die Neuverschlüsselung stellt die Vertraulichkeit wieder her, nachdem ein Mitglied entfernt wurde.
Server-Unabhängigkeit	Die Kompromittierung des Signaling-Servers, der SFU oder des Speichers gibt keine Medien preis.
Post-Quantum	Die Gruppenetablierung beruht auf dem X-Wing-Hybrid-KEM (ML-KEM-768 + X25519).

Diese Eigenschaften beschreiben das Protokoll-*Design*. Ein unabhängiges kryptografisches Drittaudit des Anruf-Subsystems ist geplant; bis zu dessen Veröffentlichung sind die obigen Garantien eine Selbsteinschätzung.

---

13. Aionda Aufgaben - Ende-zu-Ende-verschlüsselte Aufgabenverwaltung

Aionda Mail enthält eine verschlüsselte Aufgabenverwaltung: Aufgabenlisten, Teilschritte, Notizen, Fälligkeiten, Erinnerungen und wiederkehrende Aufgaben. Sie folgt denselben Zero-Knowledge-Regeln wie das Postfach. Alles, was ein Nutzer schreibt, wird im Browser ver-

schlüsselt; der Server speichert Chiffrat sowie eine kleine, bewusst gewählte Menge an Klartext-Feldern für die Terminplanung.

13.1 Schlüssel pro Aufgabe

Jede Aufgabe und jede Aufgabenliste erhält einen eigenen, frisch erzeugten 32-Byte-Schlüssel (AES-256-GCM), der im Browser mit dem Zufallszahlengenerator der WebCrypto-API erzeugt wird. Die Inhaltsfelder werden mit diesem Schlüssel verschlüsselt; der Schlüssel selbst wird anschließend mit dem Master Key des Vaults gewrappt:

```
zufälliger 32-Byte-Aufgabenschlüssel
      |  AES-256-GCM, frischer 96-Bit-IV pro Feld
      v
encrypted_title / encrypted_notes / encrypted_steps / encrypted_metadata

Vault Master Key
      |  HKDF-SHA-256(salt = 32 Zufallsbytes, info = "trashmail-tasks-v1")
      v
Wrapping-Schlüssel
      |  AES-256-GCM
      v
wrapped_key = version(1) || salt(32) || IV(12) || Chiffrat || Tag
```

Der Server speichert ausschließlich `wrapped_key` und die Chiffrat-Spalten. Er erhält weder den Aufgabenschlüssel noch den Master Key noch irgendeinen Klartext. Ohne den Master Key, der den Browser nie verlässt, lässt sich der gewrappte Schlüssel nicht öffnen.

Domänentrennung. Das HKDF-Label `trashmail-tasks-v1` gilt ausschließlich für das Aufgabenmodul. Ohne ein solches Label wäre ein gewrappter Schlüssel aus einem anderen Modul (Kalender, Kontakte) mit einem Aufgabenschlüssel austauschbar. Das Label bindet jeden gewrappten Schlüssel an das Modul, das ihn erzeugt hat.

Unabhängige Entschlüsselung. Aufgaben und Listen werden einzeln entschlüsselt. Ein nicht lesbarer Datensatz wird protokolliert und übersprungen, statt die gesamte Ansicht scheitern zu lassen. Ein beschädigtes Chiffrat kann den Nutzer damit nicht von seinen übrigen Aufgaben aussperren.

13.2 Was verschlüsselt wird

Feld	Inhalt	Schutz
`encrypted_title`	Titel der Aufgabe	AES-256-GCM mit dem Schlüssel der Aufgabe
`encrypted_notes`	Freitext-Notiz	AES-256-GCM mit dem Schlüssel der Aufgabe
`encrypted_steps`	JSON-Array der Teilschritte (Text und Erledigt-Status)	AES-256-GCM mit dem Schlüssel der Aufgabe
`encrypted_metadata`	JSON, einschließlich der Wiederholungsregel (RRULE)	AES-256-GCM mit dem Schlüssel der Aufgabe
`encrypted_name`	Name einer Aufgabenliste	AES-256-GCM mit dem Schlüssel der Liste

Jedes Feld wird mit einem eigenen, frischen 96-Bit-IV versiegelt. Gleicher Klartext ergibt damit nie gleiches Chiffrat.

Auch Listennamen sind verschlüsselt. Eine Liste namens “Akquise Contoso” ist Inhalt, nicht Metadatum, und wird entsprechend behandelt. Der Server sieht, dass ein Konto fünf Listen besitzt, nicht aber, wie sie heißen.

13.3 Bewusste Klartext-Metadaten

Eine kleine Menge an Feldern zur Terminplanung wird im Klartext gespeichert, damit der Server filtern, sortieren, zählen und Erinnerungen auslösen kann, ohne etwas zu entschlüsseln:

Klartext-Feld	Zweck
Fälligkeitsdatum	Intelligente Liste “Geplant”, Erkennung überfälliger Aufgaben, Sortierung
Erinnerungszeit	Auslösen der Erinnerung zum richtigen Zeitpunkt
Wichtig-Kennzeichen	Intelligente Liste “Wichtig”
Erledigt-Kennzeichen und Erledigungszeit	Trennung offener von erledigten Aufgaben
“Mein Tag”-Datum	Intelligente Liste “Mein Tag”, die nur für den aktuellen Tag gilt
Position	Manuelle Sortierung innerhalb einer Liste
Listenzuordnung (UUID)	Gruppierung der Aufgaben unter ihrer Liste
Verknüpfte E-Mail (UUID)	Verweis auf die E-Mail, aus der eine Aufgabe entstanden ist

Der Server erfährt, wann etwas fällig ist, nie was es ist. Er kann feststellen, dass ein Konto vierzehn offene Aufgaben hat, drei davon fällig am 3. August, eine davon als wichtig markiert. Er kann keinen einzigen Titel, keine Notiz, keinen Teilschritt, keine Wiederholungsregel und keinen Listennamen lesen.

Das ist derselbe Kompromiss, der bereits für die Terminzeiten des Kalenders eingegangen wurde, und er wird offengelegt statt verschwiegen. Die Alternative - auch das Fälligkeitsdatum zu verschlüsseln - würde den Client zwingen, für jeden Filter, jeden Zähler und jede Erinnerung den kompletten Aufgabenbestand herunterzuladen und zu entschlüsseln. Die Planungsfelder tragen keinen Inhalt, und die zugehörigen Objekte werden ausschließlich über UUIDs adressiert.

13.4 Die Liste “Gekennzeichnete E-Mail”

Die Liste “Gekennzeichnete E-Mail” ist kein zweiter Datenspeicher. Sie ist eine Sicht auf das Stern-Kennzeichen, das für E-Mails im Postfach ohnehin existiert. Es entstehen dafür also keine zusätzlichen Klartextdaten.

Die dort angezeigten Betreffzeilen werden ausschließlich clientseitig entschlüsselt, über den regulären Entschlüsselungspfad für E-Mails (Abschnitt 7.2): Der Schlüssel der jeweiligen E-Mail

wird im Browser entpackt und der Betreff dort entschlüsselt. Das Aufgabenmodul verarbeitet zu keinem Zeitpunkt einen E-Mail-Schlüssel und fordert nie einen Betreff vom Server an.

13.5 Transport, Adressierung und Supervision-Modus

- **Verschlüsselter Transport.** Sämtliche Aufgaben-Endpunkte laufen über die verschlüsselte Transportschicht `/e` (Abschnitt 8). Anfrage und Antwort werden mit dem Hybrid-KEM versiegelt. Ein CDN oder ein anderer Vermittler sieht nur `POST /e` mit einer undurchsichtigen Nutzlast, nicht aber, welche Aufgabe angelegt, umbenannt oder abgeschlossen wurde.
- **Ausschließlich UUIDs.** Aufgaben, Listen und verknüpfte E-Mails werden ausschließlich über UUIDs adressiert. Interne Datenbank-IDs werden nie nach außen gegeben, womit Enumeration als Angriffsweg entfällt.
- **Supervision-Modus.** Verwaltet ein Eigentümer das Postfach eines Mitarbeiters oder eines KI-Agenten, gelten dieselben Regeln zur Schlüsseltrennung wie im übrigen Vault: Aufgabenschlüssel werden mit dem Master Key des überwachten Kontexts gewrappt, nicht mit dem des Eigentümers. Ist dieser Kontextschlüssel nicht verfügbar, bricht die Operation hart ab. Ein Rückfall auf den Schlüssel des Eigentümers ist bewusst nicht vorgesehen - ein stiller Rückfall würde die Aufgaben eines Mitarbeiters mit dem falschen Schlüssel schreiben. Das ist ein kryptografisches Downgrade, und ein Downgrade muss laut scheitern statt still zu gelingen.

13.6 Eingabevalidierung gegen stille Trunkierung

Jedes Chiffrat wird serverseitig auf seine Länge geprüft, bevor es geschrieben wird:

Feld	Grenze
Aufgabentitel, Listenname	2.048 Byte
Notiz	1 MiB
Teilschritte	256 KiB
Metadaten	256 KiB
Gewrappter Schlüssel	60 bis 1.700 Byte

Zu große Werte werden mit einem expliziten Fehler abgewiesen. Das ist eine Integritätsprüfung mit unmittelbarer Folge für die Vertraulichkeit: Je nach SQL-Modus würde die Datenbank einen zu großen Wert sonst still abschneiden. Ein abgeschnittenes AES-GCM-Chiffrat scheitert an der Authentifizierung, und der Inhalt ist danach dauerhaft unlesbar - auch für den Nutzer. Den Schreibvorgang zu verweigern ist das einzig sichere Verhalten.

13.7 Was der Server sieht - und was nicht

Der Server KANN sehen	Der Server KANN NICHT sehen
Fälligkeitsdaten, Erinnerungszeiten, “Mein Tag”-Daten	Aufgabentitel, Notizen, Teilschritte
Wichtig- und Erledigt-Kennzeichen, Erledigungszeit	Den Namen einer Aufgabenliste
Anzahl der Aufgaben und Listen, ihre Reihenfolge und Gruppierung	Wiederholungsregeln und alle weiteren Metadaten
Die UUID einer verknüpften E-Mail	Inhalt oder Betreff dieser E-Mail

Der Server KANN sehen	Der Server KANN NICHT sehen
Chiffratgrößen, Erstellungs- und Änderungszeitpunkte	Jegliches Schlüsselmaterial - Aufgaben-, Listen- und Master Key

---

14. Seitenkanalschutz

14.1 Bucket Padding

Problem: Verschlüsselte E-Mail-Größen können Informationen preisgeben. Ein Angreifer, der Chiffretext-Längen beobachtet, könnte auf den Inhalt schließen (z.B. eine 50-Byte-E-Mail ist wahrscheinlich "OK, danke", während eine 500-KB-E-Mail Anhänge enthält).

Lösung: Alle Daten werden vor der Verschlüsselung auf Festgrößen-"Buckets" aufgefüllt:

```
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]
```

Beispiel: Eine 523-Byte-E-Mail wird auf 1.024 Bytes aufgefüllt. Ein Beobachter sieht nur "1-KB-E-Mail" — nicht die tatsächliche 523-Byte-Größe.

14.2 Kompression vor Verschlüsselung

Daten werden mit gzip (Level 6) **vor** der Verschlüsselung komprimiert. Das ist die einzig korrekte Reihenfolge:

- Kompression nach der Verschlüsselung würde fehlschlagen (verschlüsselte Daten haben maximale Entropie)
- Das Bucket Padding nach der Kompression verhindert CRIME/BREACH-artige Angriffe, die Kompressionsraten ausnutzen

14.3 Threading-Privatsphäre

E-Mail-Threads verwenden SHA-256-Hashes der Message-ID-Header anstelle von Klartext-Identifikatoren. Der Server kann zusammengehörige E-Mails anhand der Hash-Gleichheit gruppieren, ohne die tatsächlichen Nachrichten-Identifikatoren zu kennen.

---

15. Schlüsselverwaltung & Lebenszyklus

15.1 Schlüsselhierarchie

```
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 Schlüsselspeicherung

Schlüssel	Speicherort	Schutz
Master Key	Nirgendwo (wird on-demand aus Shamir-Anteilen rekonstruiert)	Shamir 2-of-3
Vault Private Keys	Server (verschlüsselt)	AES-256-GCM mit Master Key
Vault Public Keys	Server (Klartext)	Nicht sensitiv — per Definition öffentlich
E-Mail Ephemeral Keys	Server (wrapped)	Hybrid KEM Encapsulation
OPAQUE Records	Server (at rest verschlüsselt)	AES-256-GCM mit Server-Schlüssel
Verschlüsselte Shamir-Anteile	Server	XOR mit passwort-/passkey-/recovery-abgeleiteten Schlüsseln
API-Transportschlüssel	Server (vorgegenerierter Pool)	Einmalverwendung, 24h TTL

15.3 Schlüssel-Fingerprints und signierte Schlüsselwechsel

Berichtigt am 7. Oktober 2026. Frühere Fassungen dieses Abschnitts behaupteten einen serverseitig gespeicherten Fingerprint mit „Erkennung unautorisierter Schlüsseländerungen" und „clientseitiger Verifizierung der Schlüsselintegrität". Gemessen am 4. Oktober 2026: Die Spalte `vault_keys.key_fingerprint` stand im Schema, aber **kein Code schrieb oder las sie**, und kein Client verglich einen gelieferten Schlüssel mit etwas Gemerktem. Was der Entwurf jetzt festlegt (Vertrag `docs/VAULT_SCHLUESSELWECHSEL_SIGNATUR.md`, v2; Server-Umsetzung in Abnahme, zum Zeitpunkt dieser Änderung **nicht ausgeliefert**):

Fingerabdruck (clientseitig, v2). `SHA-256("aionda-postfachschluessel-fingerabdruck-v2" ‖ len ‖ X25519 ‖ len ‖ ML-KEM-1024 ‖ len ‖ ML-DSA-65-Signierschlüssel)`; ein Postfach ohne Signierschlüssel trägt die Länge 0 bei. Angezeigt als die ersten 128 Bit in Viergruppen (`1A2B-3C4D-…`). Zwei Menschen vergleichen ihn auf einem anderen Weg. Weil der Signierschlüssel Teil davon ist, fällt ein untergeschobener Signierschlüssel beim Vergleich auf; der Server berechnet ihn nicht für Vertrauensentscheidungen.

Signierte Schlüsselwechsel (Server). Jedes Postfach bekommt einen ML-DSA-65-Signierschlüssel

(FIPS 204); sein Seed liegt AES-256-GCM-verschlüsselt unter dem Master Key, mit zweck- und kontogebundenen Zusatzdaten. Einrichten oder Ersetzen dieses Schlüssels über das dafür vorgesehene Kommando verlangt einen Nachweis des Master Keys: Der Server kapselt eine Challenge an die eigenen Hybrid-KEM-Schlüssel des Postfachs, und nur ein Client, der sie öffnen kann, signiert die Einrichtungsakte. Der ältere Weg der Postfach-Neuanlage nimmt einen ganzen neuen Schlüsselsatz weiterhin ohne diesen Nachweis an; ein pinnender Kontakt sieht dann einen sichtbaren Halt, keinen stillen Wechsel (im Vertrag als Grenze benannt). Sobald ein Signierschlüssel eingerichtet ist und die Prüfung verfügbar ist, lehnt der Server unsignierte Rotationen ab. Bei einer Rotation des KEM-Schlüsselpaars signiert der Client eine kanonische Wechselakte (alter Fingerabdruck, neue X25519- und ML-KEM-Schlüssel, neuer Signierschlüssel, Anmeldename, Zeitpunkt) mit dem **alten** Signierschlüssel. Der Server prüft, bevor er etwas speichert (ungültig Ablehnung; Prüfung nicht möglich Ablehnung statt stiller Annahme), und liefert die Kette der Wechsel zusammen mit den öffentlichen Schlüsseln aus. Eine Rotation ohne Signatur wird weiter angenommen, der Ketteneintrag ist dann sichtbar unsigniert.

Was ein pinnender Client laut Entwurf tut. Beim ersten Kontakt KEM-Schlüssel und Signierschlüssel zusammen merken (Trust on first use); einen geänderten Schlüssel nur übernehmen, wenn eine Kette gültiger Signaturen vom gemerkten zum gelieferten Satz führt; sonst anhalten und den Kunden Fingerabdrücke vergleichen lassen – auch dann, wenn für einen ohne Signierschlüssel gemerkten Kontakt erstmals einer genannt wird. Der Apple-Client pinnt seit dem 4. Oktober 2026 die KEM-Schlüssel (jeder Wechsel hält an); Kettenprüfung und Fingerabdruck v2 sind dort ein Folgeticket (`4a91f154`), der Web-Client pinnt noch nicht (`b5243fcc`).

16. Wiederherstellungsmechanismus

16.1 Wiederherstellungsschlüssel (BIP39-Mnemonik)

Während der Vault-Einrichtung wird dem Nutzer eine **24-Wort-Wiederherstellungsphrase** präsentiert, die aus 256 Bit Entropie generiert und mit dem BIP39-Standard kodiert wird:

```
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 Ableitung des Wiederherstellungsschlüssels

```
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 Was der Server speichert

Der Server speichert **nur den SHA3-256-Hash** eines abgeleiteten Verifizierungsschlüssels. Er speichert nicht:

- Die Wiederherstellungswörter

- Die Entropie
- Den Verifizierungsschlüssel selbst

16.4 Wiederherstellungsablauf

1. Der Nutzer gibt die 24-Wort-Wiederherstellungsphrase ein
2. Der Client leitet ab: Entropie -> HKDF-SHA3-256 -> SHA3-256 -> Verifizierungs-Hash
3. Der Client sendet den Verifizierungs-Hash an den Server (niemals den Klartext-Schlüssel)
4. Der Server vergleicht mit dem gespeicherten Hash
5. Bei Übereinstimmung: Alle 2FA-Methoden werden deaktiviert, der Nutzer richtet eine neue Authentifizierung ein
6. Der Wiederherstellungsschlüssel wird nach einmaliger Verwendung widerrufen

16.5 Ratenlimitierung

- Maximal 3 Verifizierungsversuche pro Stunde
- 60-Minuten-Sperrung nach Überschreitung des Limits
- Einmalverwendung: Der Wiederherstellungsschlüssel wird nach erfolgreicher Nutzung dauerhaft widerrufen

16.6 Kein Passwort-Recovery

Es gibt kein Passwort-Reset per E-Mail. Wenn ein Nutzer sein Passwort UND alle anderen Authentifizierungsfaktoren (Passkey + Wiederherstellungsschlüssel) verliert, sind seine Daten dauerhaft unzugänglich. Das ist der fundamentale Beweis dafür, dass Zero-Knowledge funktioniert — wenn wir die Daten wiederherstellen könnten, könnten wir sie auch lesen.

---

17. Passwortlose Authentifizierung (Passkeys)

17.1 WebAuthn PRF Extension

Aionda Mail unterstützt FIDO2-Passkeys (Hardware-Sicherheitsschlüssel, biometrische Authentifikatoren) für passwortloses Login und Vault-Unlock.

Die **WebAuthn PRF (Pseudo-Random Function) Extension** liefert einen deterministischen 32-Byte-Output, der an den spezifischen Passkey und die Credential gebunden ist. Dieser Output wird zum Schutz von Shamir-Anteil 2 verwendet.

17.2 Funktionsweise

```
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 Mehrere Passkeys

Nutzer können mehrere Passkeys registrieren (z.B. MacBook Touch ID, iPhone Face ID, YubiKey). Jeder Passkey schützt unabhängig seine eigene Kopie von Anteil 2. Ein einzelner Passkey in Kombination mit dem Passwort reicht aus, um den Vault zu entsperren.

---

18. Guardian: MITM-Schutz & Response-Signierung

18.1 Das Problem mit Webanwendungen

Jede Webanwendung hat ein inhärentes Vertrauensproblem: Der Browser lädt bei jedem Besuch JavaScript vom Server herunter. Ein Man-in-the-Middle (MITM)-Angreifer — sei es ein kompromittiertes CDN, ein Firmen-Proxy oder ein bösartiger ISP — könnte theoretisch modifizierten Code einschleusen, der Verschlüsselungsschlüssel exfiltriert.

Aionda Mail begegnet diesem Problem mit dem **Guardian-Modul**, einer Browser-Erweiterungskomponente (verfügbar für Chrome und Firefox), die die Serverintegrität unabhängig verifiziert.

18.2 Architekturübersicht

Das Guardian-Modul arbeitet im Service Worker der Browser-Erweiterung — vollständig unabhängig vom JavaScript der Webanwendung. Es führt drei Arten von Verifizierung durch:

```
+-------------------------------------------------------------------+
|                    BROWSER EXTENSION (Guardian)                    |
|                                                                   |
|  +--------------------+  +------------------+  +-------------+ |
|  | Response Signature  |  | Certificate      |  | MITM Issuer | |
|  | Verification        |  | Fingerprint      |  | Detection   | |
|  | (Ed25519)           |  | Check (Firefox)  |  | (Blocklist) | |
|  |---------+----------+  |--------+---------+  |-----+------+ |
|            |                      |                    |        |
|            |--------------+-------+--------------------+        |
|                           v                                     |
|                 Security Status Badge                            |
|                 ✓ Green = Verified                               |
|                 × Red = MITM Detected                            |
|                 ! Red = Missing Signatures                       |
|-------------------------------------------------------------------+
```

18.3 Response-Signaturverifizierung (Ed25519)

Jede API-Antwort von Aionda Mails Server wird kryptografisch mit **Ed25519** (Edwards-Kurve Digital Signature Algorithm) signiert.

Signierungsprozess (serverseitig):

```
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>
```

Verifizierungsprozess (Browser-Erweiterung):

```
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
```

Zentrale Eigenschaften:

- **Replay-Schutz:** 5-Minuten-Zeitfenster verhindert das Wiedereinspielen alter Antworten
- **Manipulationserkennung:** Jede Änderung am Response Body invalidiert die Signatur
- **Schlüsselisolation:** Öffentliche Schlüssel sind in der Erweiterung gebündelt (nicht vom Server heruntergeladen)
- **Umgebungstrennung:** Dev-Schlüssel (`dev-2026-01`) können nicht auf Produktions-URLs verwendet werden und umgekehrt

18.4 Ed25519 Public Key Management

Öffentliche Schlüssel werden mit der Browser-Erweiterung in `public_key.json` ausgeliefert:

```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"
        }
    }
}
```

- **Schlüsselformat:** SPKI DER (Subject Public Key Info, Distinguished Encoding Rules)
- **Schlüsselgröße:** 32 Bytes (256-Bit Ed25519 Public Key)
- **Signaturgröße:** 64 Bytes (fest)
- **Rotation:** Neue Schlüssel werden hinzugefügt, bevor alte ablaufen; Erweiterungs-Updates liefern neue Schlüssel
- **Kein Serververtrauen:** Schlüssel sind in der Erweiterungs-Binary eingebettet, nicht vom Server abgerufen

18.5 TLS-Zertifikatsverifizierung (Firefox)

Unter Firefox führt das Guardian-Modul eine zusätzliche TLS-Zertifikatsverifizierung mittels der `browser.webRequest.getSecurityInfo()`-API durch (unter Chrome aufgrund von Manifest-V3-Einschränkungen nicht verfügbar).

Verifizierungsablauf:

```
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
```

Warum Aussteller-basierte Validierung statt Pinning? CloudFlare (als CDN genutzt) rotiert Leaf-Zertifikate über Edge-Server hinweg. Traditionelles Certificate Pinning (Abgleich exakter Fingerprints) würde Fehlalarme verursachen. Aussteller-basierte Validierung ist robuster: Die ausstellende CA bleibt stabil, auch wenn sich Leaf-Zertifikate ändern.

18.6 Self-Certificate-Fetching (Anti-Spoofing)

Der Zertifikats-Endpunkt des Servers nutzt eine clevere Anti-Spoofing-Technik:

```
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
```

Der Server fragt im Grunde: “Welches Zertifikat sieht die Außenwelt für meine Domain?” — und signiert die Antwort, damit die Erweiterung ihr vertrauen kann.

18.7 Sicherheitsstatus-Indikatoren

Die Erweiterung zeigt ein Badge in der Browser-Toolbar an:

Badge	Farbe	Bedeutung
✓	Grün	Alle Antworten verifiziert — Signaturen gültig

Badge	Farbe	Bedeutung
!	Orange	Veralteter Signaturschlüssel in Verwendung (Rotation steht an)
×	Rot	MITM erkannt — Signaturverifizierung fehlgeschlagen
!	Rot	Fehlende Signaturen — Antworten nicht signiert
[Shield]	Blau	Geschützter Modus — noch keine Verifizierung durchgeführt

18.8 Bedrohungsabdeckung

Angriff	Erkennungsmethode	Browser
Firmen-MITM-Proxy (ZScaler, Fortinet)	Zertifikats-Aussteller-Blocklist	Firefox
Modifizierte API-Antworten	Ed25519-Signaturverifizierung	Chrome + Firefox
Replay-Angriffe	5-Minuten-Zeitfenster	Chrome + Firefox
CDN-Kompromittierung (CloudFlare)	Response-Signatur-Abweichung	Chrome + Firefox
Zertifikatsersetzung	Aussteller-Vergleich + Server-Selbstcheck	Firefox
Dev/Prod-Schlüsselverwechslung	Umgebungsgebundene Key IDs	Chrome + Firefox

18.9 Einschränkungen

- **Chrome Manifest V3:** Kann TLS-Zertifikate nicht inspizieren — nur Response-Signaturverifizierung ist verfügbar
- **Erweiterung erforderlich:** Nutzer ohne die Erweiterung profitieren nicht vom Guardian-Schutz
- **Ed25519 ist nicht Post-Quantum:** Signaturverifizierung verwendet klassische Kryptografie. Ein ausreichend leistungsfähiger Quantencomputer könnte theoretisch Ed25519-Signaturen fälschen.

---

19. Enterprise-E-Mail-Archiv (Blockchain)

19.1 Überblick

Der Enterprise-Plan von Aionda Mail umfasst ein **GoBD-konformes E-Mail-Archiv**, das durch eine kryptografische Hash-Kette (Blockchain) gesichert ist. Jede archivierte E-Mail wird zu einem unveränderlichen Block in einer unternehmensspezifischen Kette. Jede Manipulation — Änderung, Löschung oder Einfügung von Blöcken — ist kryptografisch erkennbar.

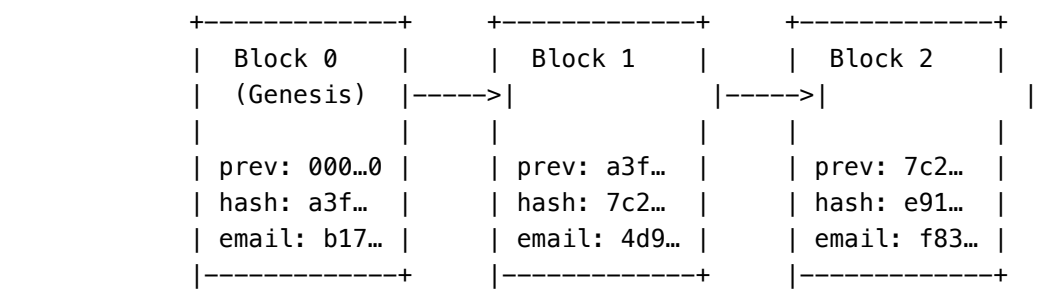
Das Archiv kombiniert zwei unabhängige Sicherheitsschichten:

1. **Hash-Kette (SHA3-256):** Garantiert Integrität und Unveränderlichkeit — beweist, dass keine E-Mail nach der Archivierung geändert oder entfernt wurde

2. **Hybrid-KEM-Verschlüsselung (CAK):** Garantiert Vertraulichkeit — der Server kann archivierte E-Mail-Inhalte nicht lesen

19.2 Hash-Ketten-Architektur

Jede archivierte E-Mail wird zu einem Block in einer sequenziellen, manipulationssicheren Kette:



Block-Hash-Berechnung:

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

Eigenschaften:

- **Hash-Algorithmus:** SHA3-256 (NIST FIPS 202)
- **Genesis-Block:** `prev_block_hash` = 64 Nullen, `block_number` = 0
- **Sequenzielle Nummerierung:** Erzwungen durch Datenbank `UNIQUE KEY (company_uuid, block_number)`
- **Eine Kette pro Unternehmen:** Vollständige Isolation zwischen Unternehmen
- **E-Mail-Hash:** `SHA3-256(sender || recipient || timestamp || size)` — Integritätsnachweis der ursprünglichen E-Mail-Daten

19.3 Manipulationserkennung

Der Ketten-Verifizierungsalgorithmus erkennt jede Form der Manipulation:

```
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
```

Manipulationsversuch	Erkennung
E-Mail-Inhalt ändern	`email_hash` ändert sich -> `block_hash`-Neuberechnung schlägt fehl
Metadaten ändern (Absender, Domain, Zeitstempel)	Im Hash-Input enthalten -> `block_hash`-Abweichung
Einen Block löschen	`prev_block_hash` des nächsten Blocks wird verwaist
Einen Block einfügen	Bricht sequenzielle `block_number` + `prev_block_hash`-Kette
Blöcke umordnen	`UNIQUE KEY`-Constraint + sequenzielle Verifizierung verhindert dies
Gesamte Kette ersetzen	Genesis-Block-Hash würde sich von jedem externen Backup unterscheiden

Verifizierungsergebnis meldet die exakte Blocknummer, an der die Manipulation erkannt wurde, mit erwartetem vs. tatsächlichem Hash für forensische Analyse.

19.4 Company Archive Key (CAK) — Zero-Knowledge-Verschlüsselung

Archivinhalte werden Ende-zu-Ende mit einem **Company Archive Key** verschlüsselt — einem Hybrid-KEM-Schlüsselpaar (X25519 + ML-KEM-1024), das clientseitig vom Unternehmenseigner generiert wird.

```
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
```

Schlüsselverteilung an andere Mitarbeiter (Admin, Compliance Officer):

1. Der Eigentümer entschlüsselt die CAK Private Keys mit seinem Passwort
2. Der Eigentümer re-wrapped die Private Keys mit dem passwortabgeleiteten Schlüssel des Zielmitarbeiters
3. Der Server speichert die re-wrapped Kopie im Record des Mitarbeiters
4. Jeder autorisierte Mitarbeiter hat seine eigene unabhängig gewrappte Kopie

Der Server sieht die CAK Private Keys niemals im Klartext.

19.5 Was verschlüsselt wird

Wenn eine E-Mail archiviert wird, werden zwei Verschlüsselungsschichten angewendet:

Verschlüsselte Metadaten (AES-256-GCM mit 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"
}
```

Verschlüsselter E-Mail-Inhalt (separates Hybrid KEM Key Wrapping):

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

Zero-Knowledge-Durchsetzung: Nach der Verschlüsselung werden die Klartext-Metadatenfelder in der Datenbank (sender_address, recipient_address, domains) durch ihre SHA3-256-Hashes ersetzt. Der Server speichert nur Hashes — die Originalwerte existieren nur innerhalb der verschlüsselten Blobs.

19.6 Audit Trail

Jede Aktion am Archiv wird in einer **unabhängigen Audit-Kette** protokolliert (ebenfalls hash-verkettet mit SHA3-256):

Aktion	Wann protokolliert
`EMAIL_RECEIVED` / `EMAIL_SENT` / `DRAFT_ARCHIVED`	E-Mail archiviert
`VIEW_EMAIL` / `VIEW_ATTACHMENT`	Mitarbeiter liest archivierte E-Mail
`SEARCH_ARCHIVE`	Suche durchgeführt

Aktion	Wann protokolliert
`EXPORT_EMAIL` / `EXPORT_REPORT`	Daten exportiert
`VERIFY_CHAIN` / `CHAIN_VERIFIED_OK` / `CHAIN_VERIFIED_BROKEN`	Integritätsprüfung
`LEGAL_HOLD_SET` / `LEGAL_HOLD_RELEASED`	Legal Hold ein-/ausgeschaltet
`ARCHIVE_DECRYPT`	CAK zum Entschlüsseln verwendet
`ADMIN_ACCESS`	Administrative Aktion

Jeder Audit-Eintrag enthält: Akteur (UUID + Rolle), IP-Adresse, Session-ID, Ziel-E-Mail-Hash und ob die Kette zum Zeitpunkt des Zugriffs gültig war.

19.7 Legal Hold & Aufbewahrung

- **Aufbewahrungsfrist:** Konfigurierbar pro Unternehmen (Standard: 10 Jahre), berechnet pro E-Mail als `archived_at + retention_years`
- **Legal Hold:** Einzelne E-Mails können unter Legal Hold gestellt werden, was die Löschung bis zur Aufhebung verhindert. Enthält Begründung, Akteur und Zeitstempel.
- **GoBD-Konformität:** Die Kombination aus unveränderlicher Hash-Kette, vollständigem Audit Trail, konfigurierbarer Aufbewahrung und Legal Hold erfüllt die Anforderungen der deutschen GoBD (Grundsätze zur ordnungsmäßigen Führung und Aufbewahrung von Büchern, Aufzeichnungen und Unterlagen in elektronischer Form sowie zum Datenzugriff).

19.8 Forensischer Export

Autorisierte Nutzer (Eigentümer, Admin) können den vollständigen Kettenstatus für unabhängige Verifizierung exportieren:

- Vollständige Kettendaten mit allen Block-Hashes
- Verifizierungsergebnis (gültig/gebrochen, Blocknummer des Bruchs falls zutreffend)
- Erwarteter vs. tatsächlicher Hash für forensische Analyse
- Letzte 50 Audit-Log-Einträge
- JSON-Format zur externen Nachverifizierung mit jeder SHA3-256-Implementierung

---

20. Was der Server sieht — und was nicht

Dieser Abschnitt dokumentiert explizit die Zero-Knowledge-Grenze.

20.1 Der Server KANN sehen

Daten	Warum sichtbar	Gegenmaßnahme
IP-Adresse	TCP/IP-Anforderung	Bei Bedarf VPN/Tor verwenden
Zeitstempel	Zeitpunkt des E-Mail-Empfangs	Inhärent zum E-Mail-Protokoll
Verschlüsselte E-Mail-Blobs	Gespeichert für den Abruf	AES-256-GCM verschlüsselt, Schlüssel dem Server unbekannt

Daten	Warum sichtbar	Gegenmaßnahme
Aufgefüllte Chiffretext-Größen	Speicheranforderung	Bucket Padding verbirgt tatsächliche Größen
Empfänger-DEA-Adresse	Routing-Anforderung	DEA ist einmalig, nicht die echte Adresse
Kontoexistenz	Authentifizierungsablauf	Schutz vor Nutzerenumeration implementiert
Öffentliche Schlüssel	Für serverseitige Verschlüsselung erforderlich	Per Definition öffentlich, nicht sensitiv
Verschlüsselte Shamir-Anteile	Speicherung für den Nutzer	XOR-verknüpft mit Schlüsseln, die der Server nicht kennt
OPAQUE Records	Authentifizierungsprotokoll	Keine Passwort-Hashes, at rest verschlüsselt

20.2 Der Server KANN NICHT sehen

Daten	Warum unsichtbar
E-Mail-Inhalt (Betreff, Body, Header)	Verschlüsselt mit ephemeren Schlüsseln, gewrappt via Hybrid KEM
Nutzerpasswort	OPAQUE — Passwort wird nie übertragen
Master Key	Wird nur im Browser aus Shamir-Anteilen rekonstruiert
Vault Private Keys	Mit Master Key vor der Speicherung verschlüsselt
E-Mail Ephemeral Keys	Mit Hybrid KEM gewrappt, Server hat keine Private Keys
Wiederherstellungsschlüssel / Mnemonik	Nur SHA3-256-Hash des abgeleiteten Schlüssels gespeichert
Passkey PRF Outputs	Hardwaregebunden, verlassen nie den Authentifikator
Ordnernamen	Verschlüsselt mit ordnerspezifischen Schlüsseln
E-Mail-Signaturen	Verschlüsselt mit Master Key
API-Anfrageinhalte	Verschlüsselt über /e Transportschicht
API-Antwortinhalte	Vor der Übertragung verschlüsselt

20.3 Kryptografische Garantie

Selbst mit vollem Zugriff auf:

- Die vollständige Datenbank
- Den gesamten Netzwerkverkehr
- Den Quellcode und die Konfiguration des Servers
- Alle OPAQUE Records und Server-Schlüssel

…kann ein Angreifer **keine einzige E-Mail** entschlüsseln, ohne das Passwort des Nutzers (oder Passkey + Wiederherstellungsschlüssel). Das ist keine Richtlinie — es ist eine mathematische Unmöglichkeit, die durch das kryptografische Design erzwungen wird.

---

21. Algorithmenreferenz

21.1 Vollständige Algorithmentabelle

Komponente	Algorithmus	Parameter	Standard
Passwort-Authentifizierung	OPAQUE	RFC 9807, aPAKE	RFC 9807
Passwort-Schlüsselableitung	PBKDF2-SHA256	600.000 Iterationen, 32B Salt, 32B Output	NIST SP 800-132
Vault-Verschlüsselung	AES-256-GCM	256-Bit Schlüssel, 96-Bit Nonce, 128-Bit Tag	NIST SP 800-38D
Klassischer Schlüsselaustausch	X25519	Curve25519, 256-Bit	RFC 7748
Post-Quantum KEM	ML-KEM-1024	Kyber-1024, NIST Level 5	NIST FIPS 203
Hybride Schlüsselableitung	HKDF-SHA256	64B IKM, info="trashmail-hybrid-kem-v1"	RFC 5869
Secret Sharing	Shamir SSS	k=2, n=3, GF(2^8)	Shamir (1979)
Wiederherstellungsschlüssel-Kodierung	BIP39	256-Bit Entropie, 24 Wörter	BIP-0039
Wiederherstellungsschlüssel-Ableitung	HKDF-SHA3-256	Kontogebundener Salt	NIST FIPS 202
Wiederherstellungsschlüssel-Verifizierung	SHA3-256	32-Byte Output	NIST FIPS 202
OPAQUE Record-Verschlüsselung	AES-256-GCM	Serverseitige At-Rest-Verschlüsselung	NIST SP 800-38D
Passkey Vault-Unlock	WebAuthn PRF	HMAC-basiert, hardwaregebunden	WebAuthn Level 2
Kompression	gzip	Level 6	RFC 1952
Bucket Padding	Custom	17 Größen (256B–16MB), 0xDEAD Magic	—
Response-Signierung	Ed25519	256-Bit Schlüssel, 512-Bit Signatur	RFC 8032
Archiv-Hash-Kette	SHA3-256	Per-Block-Hash, sequenzielle Verkettung	NIST FIPS 202
Archiv Key Wrapping (CAK)	HKDF-SHA256 + AES-256-GCM	Passwortabgeleiteter Wrapping Key	RFC 5869 / NIST SP 800-38D
Zertifikatsverifizierung	SHA-256	TLS-Zertifikats-Fingerprint-Vergleich	—
E-Mail-Threading	SHA-256	Hash der Message-ID	NIST FIPS 180-4

Komponente	Algorithmus	Parameter	Standard
Video-Anruf-Gruppenschlüssel	MLS + X-Wing	Suite `AIONDA_PQ_CALL_V1` (0xFF01), ML-KEM-768 + X25519	RFC 9420 / draft-connolly-cfrg-xwing-kem
Video-Anruf-Signaturen	ML-DSA-65	Dilithium, NIST Level 3	NIST FIPS 204
Video-Anruf-Medien	SFrame (AES-256-GCM)	Per-Frame, Schlüssel aus MLS-Exporter	RFC 9605

21.2 Sicherheitsniveaus

Algorithmus	Klassische Sicherheit	Post-Quantum-Sicherheit
X25519	128-Bit	Gebrochen durch Shors Algorithmus
ML-KEM-1024	256-Bit-Äquivalent	NIST Level 5 (≈AES-256)
AES-256-GCM	256-Bit	128-Bit (Grovers Algorithmus)
SHA-256	256-Bit	128-Bit (Grovers Algorithmus)
SHA3-256	256-Bit	128-Bit (Grovers Algorithmus)
Hybrid KEM (kombiniert)	128-Bit (X25519-gebunden)	Level 5 (ML-KEM-gebunden)

---

22. Vergleich mit anderen Anbietern

Merkmal	Aionda Mail	Tuta Mail	Proton Mail
Land	Deutschland (Stuttgart)	Deutschland (Hannover)	Schweiz
Zero-Knowledge	Ja (OPAQUE + clientseitige Krypto)	Ja	Ja
Post-Quantum	Ja (ML-KEM-1024 + X25519 Hybrid)	Ja (Kyber-basiert)	In Entwicklung
Passwort-Protokoll	OPAQUE (RFC 9807) — Passwort verlässt nie den Browser	bcrypt (Passwort wird per TLS an Server gesendet)	SRP-basiert
Betreff verschlüsselt	Ja	Ja	Nein
Header verschlüsselt	Ja	Teilweise	Nein
Kontaktnamen verschlüsselt	Ja (im Vault)	Ja	Nein

Merkmal	Aionda Mail	Tuta Mail	Proton Mail
Wegwerf-E-Mail-Adressen (DEAs)	Ja (Kernfunktion, unbegrenzt für Plus)	Nein	Ja (via SimpleLogin)
Browser-Erweiterung	Ja (Chrome + Firefox)	Nein	Via SimpleLogin
Ordnerfreigabe	Ja (Hybrid KEM pro Empfänger)	Eingeschränkt	Ja
Open-Source Client	Teilweise (MCP-Client, Apache-2.0)	Ja	Ja
Sicherheitsaudit	Geplant	Ja	Ja
Passwort-Recovery	Nein (beabsichtigt)	Nein (beabsichtigt)	Nein (beabsichtigt)
Passkey-Unterstützung	Ja (FIDO2 + PRF)	Ja	Ja
PGP-Unterstützung	Ja (eingehend + ausgehend)	Nein (eigenes Protokoll)	Ja (OpenPGP)
GoBD-konformes E-Mail-Archiv	Ja (SHA3-256 Hash-Kette + Hybrid KEM)	Nein	Nein
MITM-Erkennung (Browser-Erw.)	Ja (Ed25519-Signaturen + TLS-Check)	Nein	Nein
Perfect Forward Secrecy (API)	Ja (Einmal-Schlüssel pro Request)	Unbekannt	Unbekannt
E-Mail-Größenverschleierung	Ja (Bucket Padding)	Unbekannt	Nein

---

23. Einschränkungen & ehrliche Grenzen

23.1 Vertrauensmodell für Webanwendungen

Aionda Mail ist eine Webanwendung. Bei jedem Seitenaufruf lädt der Browser JavaScript von unseren Servern herunter. Ein raffinierter Angreifer, der unsere Server kompromittiert, könnte theoretisch modifiziertes JavaScript ausliefern, das Schlüssel exfiltriert.

Aktuelle Gegenmaßnahmen:

- Subresource Integrity (SRI) Hashes auf allen Script-Tags
- Content Security Policy (CSP) Header beschränken Script-Quellen
- Aller kritischer kryptografischer Code ist im Haupt-Application-Bundle enthalten
- **Guardian Browser-Erweiterung** (Abschnitt 18): Ed25519-Signaturverifizierung auf allen API-Antworten erkennt serverseitige Manipulation; TLS-Zertifikatsverifizierung (Firefox) erkennt MITM-Proxys

Geplante Gegenmaßnahmen:

- Service Worker Caching für Offline-Betrieb (reduziert Vertrauensabhängigkeit beim Laden)

23.2 Metadaten-Sichtbarkeit

Während E-Mail-Inhalte vollständig verschlüsselt sind, sind bestimmte Metadaten für den Server sichtbar:

- Wann E-Mails empfangen wurden (Zeitstempel)
- Welche DEA-Adresse die E-Mail empfangen hat
- Ungefähre E-Mail-Größe (innerhalb der Bucket-Grenzen)
- Kontoaktivitätsmuster

23.3 E-Mail-Verarbeitungsprotokoll

Zu Diagnosezwecken enthält Aionda Mail ein optionales **E-Mail-Verarbeitungsprotokoll**, das den Rohinhalt eingehender E-Mails vorübergehend speichern kann. Diese Funktion ist pro Wegwerf-E-Mail-Adresse (DEA) konfigurierbar und kann in den DEA-Einstellungen aktiviert oder deaktiviert werden ("E-Mail-Inhalt protokollieren").

Bei Aktivierung (pro DEA einstellbar):

- Die vollständige SMTP-Rohnachricht (Header + Body) wird im Klartext auf dem Server gespeichert
- Automatische Löschung nach einer kurzen Aufbewahrungsfrist (unter 7 Tagen)
- Nur für den Kontoinhaber über die authentifizierte API zugänglich
- Zweck: Fehlerbehebung bei Zustellproblemen, Überprüfung der Weiterleitung, Bewertung von Spam-Filterentscheidungen

Bei Deaktivierung:

- Es werden keine E-Mail-Inhalte im Verarbeitungsprotokoll gespeichert
- Nur Metadaten werden protokolliert (Absenderadresse, Zeitstempel, Zustellstatus)
- Die Vault-Verschlüsselung bleibt der einzige Speichermechanismus

Wichtig: Dieses Verarbeitungsprotokoll ist unabhängig vom verschlüsselten Vault. E-Mails im Vault sind immer mit Hybrid KEM verschlüsselt, unabhängig von der Protokolleinstellung. Das Verarbeitungsprotokoll existiert als Legacy-Funktion aus dem E-Mail-Weiterleitungssystem und bietet betriebliche Transparenz. Nutzer, die eine strikte Zero-Knowledge-Speicherung für alle E-Mails benötigen, sollten diese Option deaktivieren.

23.4 Sicherheit externer E-Mails

E-Mails, die an oder von Nicht-Aionda-Adressen gesendet/empfangen werden, durchlaufen die Standard-E-Mail-Infrastruktur (SMTP). Obwohl sie verschlüsselt im Vault gespeichert sind, war der E-Mail-Inhalt während des Transits sichtbar, sofern nicht PGP-Verschlüsselung verwendet wurde.

23.5 Kein Key Escrow

Es gibt keinen Master Key, keine Hintertür und keinen Wiederherstellungsmechanismus, der der Aionda GmbH zur Verfügung steht. Wenn ein Nutzer sein Passwort und alle Wiederherstellungsmethoden verliert, sind seine Daten dauerhaft verloren. Dies ist eine bewusste Designentscheidung, die die Integrität des Zero-Knowledge-Modells beweist.

---

24. Roadmap

Meilenstein	Status	Ziel
Zero-Knowledge-Architektur	Abgeschlossen	—
Post-Quantum Hybrid KEM (ML-KEM-1024)	Abgeschlossen	—
OPAQUE-Authentifizierung (RFC 9807)	Abgeschlossen	—
Shamir Secret Sharing (2-of-3)	Abgeschlossen	—
Verschlüsselte API-Transportschicht	Abgeschlossen	—
Passkey/WebAuthn PRF Unterstützung	Abgeschlossen	—
Ende-zu-Ende-verschlüsselter Kalender	Abgeschlossen	—
Ende-zu-Ende-verschlüsselte Aufgaben	Abgeschlossen	—
GoBD-konformes E-Mail-Archiv (Blockchain)	Abgeschlossen	—
Guardian MITM-Schutz (Ed25519)	Abgeschlossen	—
TLS-Zertifikatsverifizierung (Firefox)	Abgeschlossen	—

---

25. Geräteanmeldung per QR-Code

25.1 Bedienung und Gültigkeit

Ein bereits angemeldeter, entsperrter Client zeigt **Neues Gerät anmelden (QR-Code)** an. Das neue Gerät scannt die Einladung. Beide Geräte zeigen eine achtstellige Vergleichszahl; der Benutzer muss sie vergleichen und die Übertragung am bestehenden Gerät ausdrücklich bestätigen. Web, iOS und macOS können die Einladung anzeigen; der empfangende Client muss das Kopplungsprotokoll und dessen vertrauenswürdiges Transportprofil unterstützen.

Die Einladung läuft nach **15 Minuten (900 Sekunden)** ab. Diese feste Frist ist eine Produktentscheidung, kein Industriestandard. Lesen und Wiederholen verlängern sie nicht. Eine abgeschlossene Einladung kann kein weiteres Gerät anmelden. Abbruch oder Ablauf machen eine unvollständige Übertragung ungültig; ein bereits erfolgreich angemeldetes Gerät wird dadurch nicht abgemeldet.

25.2 QR-Inhalt und verschlüsselte Übertragung

Der QR-Code `AIONDA_DEVICE_PAIRING_V2` enthält eine Kopplungs-UUID, den Benutzernamen, die Dienst-Origin, Erstellungs- und Ablaufzeit, einen SHA-256-Fingerabdruck über die kurzlebigen öffentlichen Quellschlüssel und Einladungsmetadaten sowie Transportprofil und Authentifizierungs-Origin. Er enthält **weder Kontopasswort noch Wiederherstellungswörter, Zugangstoken oder Postfach-Hauptschlüssel**. Der Benutzername ist lesbar: Den QR-Code nicht veröffentlichen oder weiterleiten.

Der Empfänger gleicht den QR-Fingerabdruck mit den vom Relay gelieferten öffentlichen Schlüsseln ab. X25519 + ML-KEM-1024 erzeugt ein gemeinsames Geheimnis. HKDF-SHA-256 leitet rollenspezifische, an das Protokolltranskript gebundene Schlüssel ab; AES-256-GCM verschlüsselt und authentifiziert Anfrage, Bereitschaft, Freigabe und Empfangsbestätigung. Konto, Umgebung, Empfänger und Gültigkeit sind an den Austausch gebunden. Ungültige Fingerabdrücke, Transportbindungen oder authentifizierte Nachrichten führen zu einem Fehler statt zu einem Klartext-Rückfall.

Nach der Bestätigung überträgt die Quelle einen **neuen Geräte-Zugangstoken und den Postfach-Hauptschlüssel** im verschlüsselten Freigabepaket, nicht das bestehende Kontopasswort oder

die Wiederherstellungswörter. Der Token wird mit OPAQUE registriert. Der Server vermittelt Chiffretexte und verwaltet Vorgangsstatus und Registrierungsdaten; der Empfänger entschlüsselt das Paket. Er erhält echten Postfachzugriff. Die Freigabe ist daher sicherheitskritisch. Ein Token-Widerruf verhindert weitere Authentifizierungen, löscht aber keine bereits auf dem Gerät vorhandenen Daten oder Schlüssel.

25.3 Browser-Risiken und Schutzmaßnahmen

Die Web-Kopplung zeigt einen lokalisierten Risikohinweis und verlangt eine aktuelle Kontoprüfung: Der Server akzeptiert die Erstellung nur innerhalb von 120 Sekunden nach erfolgreicher, kontogebundener OPAQUE- oder Passkey-Prüfung. Das Postfach muss entsperrt sein. Die Browserquelle ist an ihre Sitzung gebunden; der Empfänger nutzt ein festes vertrauenswürdiges Authentifizierungsprofil statt eines beliebigen Endpunkts aus dem QR-Code.

Als Quelle empfehlen wir die native iOS- oder macOS-App. Ende-zu-Ende-Verschlüsselung schützt den Austausch nur, solange beide Endpunkte vertrauenswürdigen Code ausführen. Ein TLS-inspektierender Proxy wie ZScaler, eine schädliche Erweiterung oder manipulierte Web-Auslieferung können Browser-Code verändern und Geheimnisse vor der Verschlüsselung lesen. HTTPS, der QR-Code und übereinstimmende Vergleichszahlen machen einen kompromittierten Endpunkt nicht sicher. Guardian kann installiert und aktiviert zusätzliche Prüfungen liefern, ist aber keine Voraussetzung und wird für diese Sicherheitszusage nicht angenommen. Unerwartete Anfragen niemals bestätigen; die Zahlen auf den beiden eigenen Geräten vergleichen.

---

26. Quelloffener MCP-Client: das Protokoll zum Nachprüfen

Der MCP-Client von Aionda Mail ist quelloffen unter der **Apache License 2.0** veröffentlicht:

- Quelltext: `https://github.com/AiondaDotCom/aionda-mail-mcp-client`
- Paket: `@aiondadotcom/mail-mcp-client` auf npm

Er verbindet KI-Agenten über das Model Context Protocol mit dem Postfach. Er läuft auf dem Rechner des Nutzers, nicht auf unseren Servern, und enthält die vollständige clientseitige Kryptografie eines Aionda-Mail-Clients. Damit liegt das Protokoll, das sonst nur in den Clients steckt, für jeden lesbar offen.

26.1 Was der Quelltext offenlegt

- **Entsperren:** Der Zugangsschlüssel `tmv_…` verlässt den Rechner nie. Lokal wird daraus K = SHA-256(`tmv_…`) abgeleitet. Mit K öffnet der Client einen 60 Byte großen Umschlag (AES-256-GCM, die Zusatzdaten binden die Schlüssel-UUID) und erhält den Vault Master Key. (Bei KI-Mitarbeitern ist K selbst der Master Key; einen Umschlag gibt es dort nicht.) Der Server speichert nur diesen Umschlag und einen Hash des API-Schlüssels.
- **Private Schlüssel:** Mit dem Master Key entschlüsselt der Client lokal die privaten Schlüssel des Postfachs (X25519 und ML-KEM-1024).
- **Mail-Inhalte:** Betreff, Absender, Empfänger, Text, Kopfzeilen, Anhänge und Ordnernamen kommen als AES-256-GCM-Chiffrat beim Client an. Der Schlüssel einer Mail ist entweder ein zufälliger Einmalschlüssel, per hybridem KEM an die öffentlichen Schlüssel gekapselt (X25519 + ML-KEM-1024, kombiniert über HKDF-SHA-256, Kapitel 6), oder ein mailspezifischer Schlüssel HKDF-SHA-256(Master Key, Mail-UUID).
- **Größenverschleierung:** Bucket Padding vor der Verschlüsselung (Kapitel 14).

- **Freigaben:** Ordnerschlüssel werden aus dem Master Key abgeleitet und an die öffentlichen Schlüssel der Empfänger gekapselt. Der Client gibt den Master Key nie weiter; eine Mail, die noch direkt unter dem Master Key liegt, lehnt er beim Einliefern in einen geteilten Ordner ab.

26.2 Warum das das Protokoll belegt

1. **Der Server liefert nur Chiffrat.** Der MCP-Client erhält vom Server ausschließlich verschlüsselte Inhalte, gekapselte Schlüssel und öffentliche Schlüssel und muss alles selbst entschlüsseln. Der Quelltext zeigt, dass weder der Zugangsschlüssel noch der Master Key oder ein privater Schlüssel zum Server gehen. Hätte der Server den Klartext, bräuchte der Client diese Kryptografie nicht.
2. **Ende-zu-Ende mit Post-Quanten-Verfahren.** Ohne den privaten ML-KEM-1024-Schlüssel des Nutzers lässt sich keine Mail öffnen. Weil ML-KEM-1024 mit X25519 kombiniert ist, muss ein Angreifer beide Verfahren brechen - auch mit einem künftigen Quantencomputer.
3. **Dieselben Formate in allen Clients.** Web-App, iOS-/macOS-App, Android-App und MCP-Client lesen dieselben gespeicherten Daten. Bytegenaue Testvektoren (Known-Answer-Tests) im Repository werden gegen den Web-Client und die Apple-Apps geprüft. Ein Client mit einem anderen Verfahren könnte dieselben Mails nicht lesen.
4. **Selbst nachprüfbar.** Wer den Netzwerkverkehr der Web-App oder einer nativen App mit einem TLS-Proxy mitschneidet, sieht dasselbe Chiffrat in denselben Feldern; in der Web-App zusätzlich im Umschlag der verschlüsselten API-Transportschicht (Kapitel 8).
5. **Nachweisbare Herkunft.** Jede Version wird aus GitHub Actions heraus veröffentlicht, mit npm-Provenance-Nachweis im öffentlichen Sigstore-Transparenzprotokoll. Ein installiertes Paket lässt sich mit `npm audit signatures` seinem Commit zuordnen. Die CI testet auf Ubuntu 22.04, 24.04 und 26.04 mit Node 22 und 24 und verlangt mindestens 90 % Testabdeckung.

26.3 Was er nicht belegt

- Den Quelltext des Servers und der übrigen Clients. Insbesondere könnte ein kompromittierter Server einem Browser anderen JavaScript-Code ausliefern (Abschnitt 23.1).
- Eingehende Mails erreichen den Server per SMTP und werden beim Empfang verschlüsselt (Abschnitt 7.1). Gesendete Mails muss der Server zum Zustellen lesen können (Abschnitt 7.3).
- Was ein KI-Agent liest, verarbeitet dessen Anbieter im Klartext.

Dokumentenhistorie

Version	Datum	Änderungen
1.0	März 2026	Erstveröffentlichung
1.1	April 2026	Neues Kapitel 10: Vault Drive (Per-File-Key-Architektur, Sharing mit Key-Rewrap)

Version	Datum	Änderungen
1.2	April 2026	Abschnitt 10.11: Externes Teilen — öffentliche Share-Links mit Hybrid-KEM-Hüllen, unlock_token-Flow, Argon2id-Passwort-Modus, URL-Fragment für link_only, manipulationssichere Audit-Chain (EXT_SHARE_*) und Kanal-Wahl durch Eigentümer
1.3	Mai 2026	Neues Kapitel 11: Aionda Chat — Post-Quantum-E2EE-Echtzeit-Messaging via AAR (Aionda Async Ratchet), eine hybride X25519 + ML-KEM-1024-Variante von Signal X3DH + Double Ratchet. Forward Secrecy pro Nachricht, Post-Compromise Security und Integration in die Enterprise-Audit-Chain. Folgekapitel auf 12-22 umnummeriert. PT-BR- und PT-PT-Übersetzungen ergänzt. Doppel-Numbering-Bug im PDF-Rendering behoben (LaTeX-Auto-Nummerierung deaktiviert; Kapitelnummern stammen jetzt nur noch aus den Headings).
1.4	Juni 2026	Neues Kapitel 12: Aionda Video-Anrufe — Post-Quantum-E2EE-Gruppenkonferenzen. MLS (RFC 9420) Gruppen-Schlüsseleinigung mit der Cipher-Suite `AIONDA_PQ_CALL_V1` (X-Wing = ML-KEM-768 + X25519, ML-DSA-65-Signaturen), SFrame (RFC 9605) Per-Frame-AES-256-GCM-Medienverschlüsselung mit Schlüssel aus dem MLS-Exporter, SAS-Bindung und ein Zero-Knowledge-SFU-Relay. Folgekapitel auf 13-23 umnummeriert.

Version	Datum	Änderungen
1.5	Juli 2026	Neues Kapitel 13: Aionda Aufgaben - Ende-zu-Ende-verschlüsselte Aufgabenverwaltung. AES-256-GCM-Schlüssel pro Aufgabe und pro Liste, gewrappt mit dem Master Key des Vaults (HKDF-SHA-256 mit modulspezifischer Domänentrennung), verschlüsselte Titel, Notizen, Teilschritte, Listennamen und Wiederholungsregeln, eine ehrliche Darstellung der bewusst im Klartext gehaltenen Planungsmetadaten, die Liste gekennzeichneter E-Mails auf Basis des vorhandenen Stern-Kennzeichens, Schlüsseltrennung im Supervision-Modus sowie serverseitige Längenprüfung gegen stille Trunkierung von Chiffraten. Folgekapitel auf 14-24 umnummeriert.
1.6	Oktober 2026	Abschnitt 15.3 auf das Gemessene zurückgebaut (den serverseitigen Fingerprint und die „Erkennung unautorisierter Schlüsseländerungen” gab es nicht); Entwurf der signierten Schlüsselwechsel beschrieben (ML-DSA-65-Signierschlüssel je Postfach, Fingerabdruck v2 über KEM und Signierschlüssel, kanonische Wechselakte vom bisherigen Schlüssel signiert, Master-Key-Nachweis beim Einrichten, serverseitige Prüfung vor dem Speichern, Kette mit den Empfängerschlüsseln) – Server-Umsetzung in Abnahme, zum Zeitpunkt dieser Änderung nicht ausgeliefert. Abschnitt 2.2: Schlüsseltausch durch den Server als Grenze benannt, einschliesslich der Lenkung auf einen älteren signierten Stand.
1.7	13 Oktober 2026	25: Geräteanmeldung per QR-Code (Web, iOS, macOS).

Version	Datum	Änderungen
1.8	13 Oktober 2026	3.2: Native iOS-/macOS-Clients, gemeinsame Swift-/SwiftUI-Architektur und Vertrauensgrenzen eingebetteter WebKit-Komponenten.
1.9	8 November 2026	26: Quelloffener MCP-Client (Apache-2.0) - offengelegtes Protokoll, Begründung der Nachprüfbarkeit, Herkunftsnachweis, Grenzen. 22: Zeile „Open-Source Client“ angepasst.
1.10	8 November 2026	3.3: Android-Client (Kotlin/Jetpack Compose, Rust-Kryptokern, Android Keystore, abgesicherte HTML-Ansicht; Google Play und eigenes F-Droid-Repository). 3.4: Übersicht aller Clients mit Bezugsquelle, Ort der Kryptografie und Quelltextstatus.

Kontakt

Aionda GmbH Stephan Ferraro Stuttgart, Germany

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

---

Dieses Dokument beschreibt die Sicherheitsarchitektur von Aionda Mail, Stand 13. Oktober 2026. Kryptografische Systeme entwickeln sich weiter — dieses Dokument wird aktualisiert, wenn sich die Architektur ändert.