

Aionda Mail Security Whitepaper

Zero-Knowledge · Post-Quantum · Made in Germany

Version 1.0 — March 2026

Aionda GmbH
Stuttgart, Germany

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

PUBLIC DOCUMENT

Contents

1	Aionda Mail Security Whitepaper	5
1.1	Table of Contents	5
1.2	1. Executive Summary	5
1.3	2. Threat Model	6
1.3.1	2.1 What Aionda Mail Protects Against	6
1.3.2	2.2 What Aionda Mail Does NOT Protect Against	6
1.3.3	2.3 Design Principle	7
1.4	3. Architecture Overview	7
1.4.1	3.1 Component Overview	8
1.5	4. Zero-Knowledge Authentication (OPAQUE)	8
1.5.1	4.1 Why Not Password Hashing?	8
1.5.2	4.2 How OPAQUE Works	8
1.5.3	4.3 Implementation	9
1.5.4	4.4 SRP Migration	9
1.6	5. Vault Master Key & Shamir Secret Sharing	9
1.6.1	5.1 Master Key Generation	9
1.6.2	5.2 Shamir Secret Sharing (2-of-3)	10
1.6.3	5.3 Share Protection	10
1.6.4	5.4 Session Storage Protection	10
1.7	6. Post-Quantum Hybrid KEM	11
1.7.1	6.1 Why Post-Quantum?	11
1.7.2	6.2 Hybrid Approach	11
1.7.3	6.3 Encapsulation Process	11
1.7.4	6.4 Decapsulation Process	11
1.7.5	6.5 Key Sizes	12
1.7.6	6.6 Library	12
1.8	7. Email Encryption Pipeline	12
1.8.1	7.1 Incoming Email (SMTP → Encrypted Storage)	12
1.8.2	7.2 Reading Email (Browser Decryption)	13
1.8.3	7.3 Sending Email	14
1.8.4	7.4 Attachments	14
1.8.5	7.5 Email Threading (Zero-Knowledge)	14
1.9	8. Encrypted API Transport Layer	14
1.9.1	8.1 Problem	14
1.9.2	8.2 Solution: End-to-End Encrypted API	14
1.9.3	8.3 Key Properties	15
1.9.4	8.4 What CloudFlare Sees	15
1.10	9. Folder Sharing Cryptography	15
1.10.1	9.1 Sharing Model	15
1.10.2	9.2 Sharing Flow	15
1.10.3	9.3 Permission Model	16
1.10.4	9.4 Cross-Vault Copy (Re-Wrapping)	16
1.11	10. Side-Channel Protections	17
1.11.1	10.1 Bucket Padding	17
1.11.2	10.2 Compression-Before-Encryption	17
1.11.3	10.3 Threading Privacy	17
1.12	11. Key Management & Lifecycle	17

1.12.1	11.1	Key Hierarchy	17
1.12.2	11.2	Key Storage	18
1.12.3	11.3	Key Fingerprints	18
1.13	12.	Recovery Mechanism	18
1.13.1	12.1	Recovery Key (BIP39 Mnemonic)	18
1.13.2	12.2	Recovery Key Derivation	19
1.13.3	12.3	What the Server Stores	19
1.13.4	12.4	Recovery Flow	19
1.13.5	12.5	Rate Limiting	19
1.13.6	12.6	No Password Recovery	19
1.14	13.	Passwordless Authentication (Passkeys)	19
1.14.1	13.1	WebAuthn PRF Extension	19
1.14.2	13.2	How It Works	20
1.14.3	13.3	Multiple Passkeys	20
1.15	14.	Guardian: MITM Protection & Response Signing	20
1.15.1	14.1	The Problem with Web Applications	20
1.15.2	14.2	Architecture Overview	20
1.15.3	14.3	Response Signature Verification (Ed25519)	21
1.15.4	14.4	Ed25519 Public Key Management	21
1.15.5	14.5	TLS Certificate Verification (Firefox)	22
1.15.6	14.6	Self-Certificate Fetching (Anti-Spoofing)	22
1.15.7	14.7	Security Status Indicators	23
1.15.8	14.8	Threat Coverage	23
1.15.9	14.9	Limitations	23
1.16	15.	Enterprise Email Archive (Blockchain)	23
1.16.1	15.1	Overview	23
1.16.2	15.2	Hash Chain Architecture	24
1.16.3	15.3	Tamper Detection	24
1.16.4	15.4	Company Archive Key (CAK) — Zero-Knowledge Encryption	25
1.16.5	15.5	What Gets Encrypted	26
1.16.6	15.6	Audit Trail	26
1.16.7	15.7	Legal Hold & Retention	27
1.16.8	15.8	Forensic Export	27
1.17	16.	What the Server Sees — and What It Does Not	27
1.17.1	16.1	The Server CAN See	27
1.17.2	16.2	The Server CANNOT See	27
1.17.3	16.3	Cryptographic Guarantee	28
1.18	17.	Algorithm Reference	28
1.18.1	17.1	Complete Algorithm Table	28
1.18.2	17.2	Security Levels	29
1.19	18.	Comparison with Other Providers	29
1.20	19.	Limitations & Honest Boundaries	30
1.20.1	19.1	Web Application Trust Model	30
1.20.2	19.2	Metadata Visibility	31
1.20.3	19.3	Email Processing Log	31
1.20.4	19.4	External Email Security	31
1.20.5	19.5	No Key Escrow	31
1.21	20.	Roadmap	32
1.22		Document History	32

1.23 Contact 32

1. Aionda Mail Security Whitepaper

Version 1.0 — March 2026 Aionda GmbH, Stuttgart, Germany

1.1 Table of Contents

1. Executive Summary
 2. Threat Model
 3. Architecture Overview
 4. Zero-Knowledge Authentication (OPAQUE)
 5. Vault Master Key & Shamir Secret Sharing
 6. Post-Quantum Hybrid KEM
 7. Email Encryption Pipeline
 8. Encrypted API Transport Layer
 9. Folder Sharing Cryptography
 10. Side-Channel Protections
 11. Key Management & Lifecycle
 12. Recovery Mechanism
 13. Passwordless Authentication (Passkeys)
 14. Guardian: MITM Protection & Response Signing
 15. Enterprise Email Archive (Blockchain)
 16. What the Server Sees — and What It Does Not
 17. Algorithm Reference
 18. Comparison with Other Providers
 19. Limitations & Honest Boundaries
 20. Roadmap
-

1.2 1. Executive Summary

Aionda Mail is a zero-knowledge, post-quantum encrypted email service operated by Aionda GmbH in Stuttgart, Germany. The service combines disposable email addresses (DEAs) with a fully encrypted mailbox — a combination not offered by any other provider.

Core security properties:

- **Zero-Knowledge Architecture:** All encryption and decryption happens exclusively in the user's browser. The server never has access to plaintext email content in the secure mailbox, passwords, or encryption keys.
- **Post-Quantum Security:** Hybrid Key Encapsulation Mechanism (X25519 + ML-KEM-1024) protects all data against both classical and quantum computer attacks.
- **Zero-Knowledge Authentication:** OPAQUE protocol (RFC 9807) ensures passwords are never transmitted to or stored on the server — not even as hashes.
- **Shamir Secret Sharing (2-of-3):** The vault master key is split into three shares protected by password, passkey, and recovery key. Any two shares reconstruct the master key.

- **Perfect Forward Secrecy:** Every API request uses a unique, one-time cryptographic keypair. Compromising one request does not affect any other.
- **MITM Protection (Guardian):** The browser extension independently verifies all server responses via Ed25519 signatures and detects man-in-the-middle attacks through TLS certificate verification — even against corporate proxies and compromised CDNs.
- **GoBD-Compliant Email Archive:** Enterprise accounts benefit from a tamper-evident hash chain (SHA3-256 blockchain) with end-to-end encrypted content (Hybrid KEM), complete audit trail, legal hold, and configurable retention — compliant with German GoBD regulations.
- **No Password Recovery:** If the password and all recovery methods are lost, the data is irrecoverable. This is by design — it proves the server cannot access user data.

Jurisdiction: German law (DSGVO/GDPR), no data sharing with foreign intelligence agencies.

1.3 2. Threat Model

1.3.1 2.1 What Aionda Mail Protects Against

Threat	Protection
Server compromise (database leak, insider access)	All email content encrypted with keys the server never possesses
Network eavesdropping (ISP, Wi-Fi, CDN)	End-to-end encrypted API transport via Hybrid KEM
CloudFlare inspection	API requests are encrypted before leaving the browser; CloudFlare sees only ciphertext. Guardian extension detects response tampering via Ed25519 signatures
Corporate MITM proxies (ZScaler, Fortinet, etc.)	Guardian extension detects proxy certificates via issuer blocklist (Firefox)
Quantum computer attacks (“harvest now, decrypt later”)	ML-KEM-1024 (NIST FIPS 203) provides post-quantum resistance
Password database theft	OPAQUE stores only cryptographic records, not password hashes
Offline brute-force attacks on passwords	OPAQUE prevents offline attacks; server-side rate limiting prevents online attacks
Email size analysis	Bucket padding obscures actual email sizes
Compression side-channels (CRIME/BREACH)	Bucket padding applied after compression
User enumeration	Deterministic fake responses for non-existent accounts

1.3.2 2.2 What Aionda Mail Does NOT Protect Against

Limitation	Explanation
Compromised device	If malware controls the browser, it can read decrypted content

- | | | |
|--|---|---|
| | ← | |
| 3. Finalize registration
(export_key available) | → | 4. Store encrypted OPAQUE
record (AES-256-GCM at rest) |

Login (every session):

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

Key properties:

- Password is verified **client-side** in step 3 — the server never sees it
- The server stores an **OPAQUE record**, which is not a password hash and cannot be brute-forced offline
- OPAQUE records are additionally **encrypted at rest** with AES-256-GCM using a server-side key
- **User enumeration protection**: Non-existent accounts receive deterministic fake responses with identical timing
- **Rate limiting**: Maximum 3 authentication attempts per session, 120-second session timeout

1.5.3 4.3 Implementation

- **Library**: @serenity-kit/opaque (WASM-based, production-grade)
- **Server component**: Dedicated microservice for OPAQUE cryptographic operations
- **Base64 format**: base64url (URL-safe, no padding) for protocol compatibility
- **Audit logging**: All authentication events logged with timestamps and IP addresses

1.5.4 4.4 SRP Migration

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

1.6 5. Vault Master Key & Shamir Secret Sharing

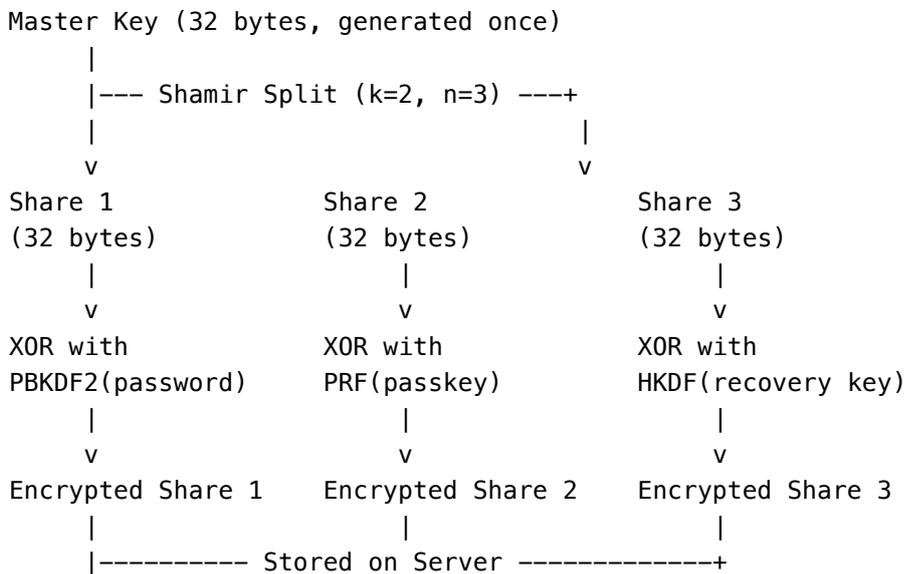
1.6.1 5.1 Master Key Generation

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

This master key is the root of all encryption. It never leaves the browser in plaintext. It is never stored anywhere — not in the browser, not on the server, not in any form.

1.6.2 5.2 Shamir Secret Sharing (2-of-3)

The master key is split into three shares using **Shamir’s Secret Sharing** scheme over the Galois Field $GF(2^8)$ with the AES irreducible polynomial $(x^8 + x^4 + x^3 + x + 1)$.



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

1.6.3 5.3 Share Protection

Each share is XOR’d with a key derived from a different authentication factor:

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

Reconstruction scenarios:

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

1.6.4 5.4 Session Storage Protection

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

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

1.7 6. Post-Quantum Hybrid KEM

1.7.1 6.1 Why Post-Quantum?

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

1.7.2 6.2 Hybrid Approach

Aionda Mail uses a **hybrid Key Encapsulation Mechanism** combining:

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

The hybrid approach provides **defense-in-depth**: the combined key is secure as long as **at least one** of the two algorithms remains unbroken.

1.7.3 6.3 Encapsulation Process

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

1.7.4 6.4 Decapsulation Process

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

1.7.5 6.5 Key Sizes

Parameter	Size	Standard
X25519 public key	32 bytes	RFC 7748
X25519 private key	32 bytes	RFC 7748
ML-KEM-1024 public key	1,568 bytes	NIST FIPS 203
ML-KEM-1024 private key	3,168 bytes	NIST FIPS 203
ML-KEM-1024 ciphertext	1,568 bytes	NIST FIPS 203
Combined shared secret	32 bytes	HKDF-SHA256 output

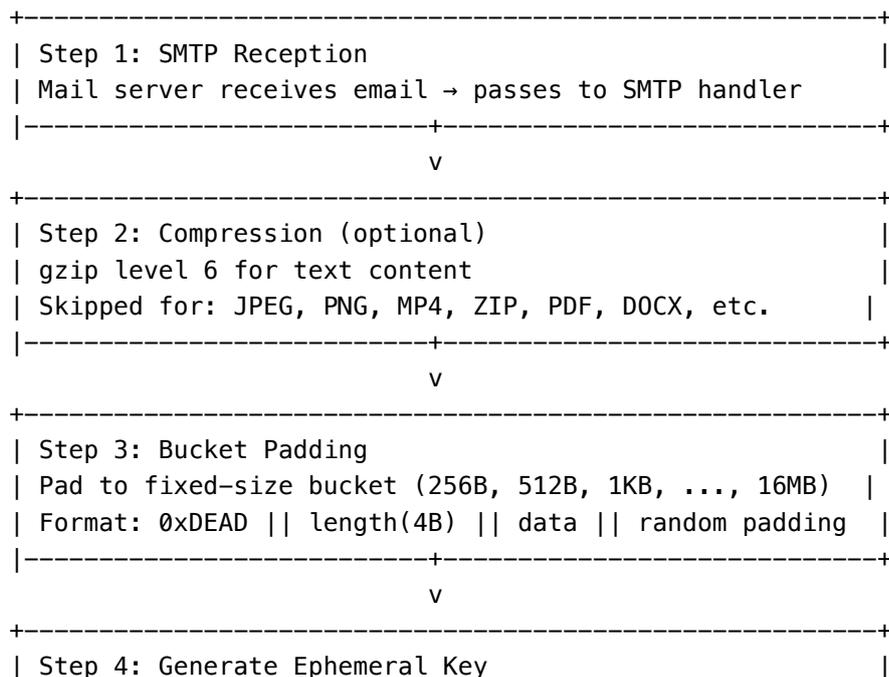
1.7.6 6.6 Library

- **ML-KEM-1024:** @noble/post-quantum (audited, pure JavaScript implementation)
- **X25519:** WebCrypto API (`crypto.subtle.deriveBits`)
- **HKDF:** @noble/hashes (RFC 5869 compliant)

1.8 7. Email Encryption Pipeline

1.8.1 7.1 Incoming Email (SMTP → Encrypted Storage)

When an email arrives at Aionda Mail's SMTP server:



```

| 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) |
+-----+

```

1.8.2 7.2 Reading Email (Browser Decryption)

The reverse process happens entirely in the browser:

1. Fetch encrypted email from server via encrypted API
2. Parse wrapped ephemeral key (extract X25519 ciphertext + ML-KEM ciphertext)
3. **Hybrid KEM decapsulation** using vault private keys □ shared secret
4. Unwrap ephemeral AES-256 key
5. **AES-256-GCM decryption** of each field (subject, from, to, body, headers)
6. Reorder bytes: server format (nonce || tag || ct) □ WebCrypto format (nonce || ct || tag)
7. Remove bucket padding (detect 0xDEAD magic bytes)

8. Decompress if gzip (detect 0x1F8B magic bytes)
9. UTF-8 decode to plaintext

1.8.3 7.3 Sending Email

When composing and sending an email:

1. Client encrypts email content with a challenge-response protocol
2. Server receives encrypted payload, decrypts ephemerally (in-memory only), sends via SMTP
3. Server returns generated MIME headers to client (encrypted)
4. Client encrypts a copy with vault master key and stores in Sent folder
5. Ephemeral server-side plaintext is immediately discarded — never written to disk

1.8.4 7.4 Attachments

Each attachment is encrypted independently:

- Separate ephemeral AES-256 key per attachment
- Separate Hybrid KEM key wrapping per attachment
- Filename and MIME type encrypted separately
- No compression for already-compressed formats (JPEG, ZIP, PDF, etc.)

1.8.5 7.5 Email Threading (Zero-Knowledge)

Email threading (grouping related emails) uses only **SHA-256 hashes** of Message-ID and In-Reply-To headers. The server never sees the actual Message-ID strings — it can group emails by hash equality without knowing the content.

1.9 8. Encrypted API Transport Layer

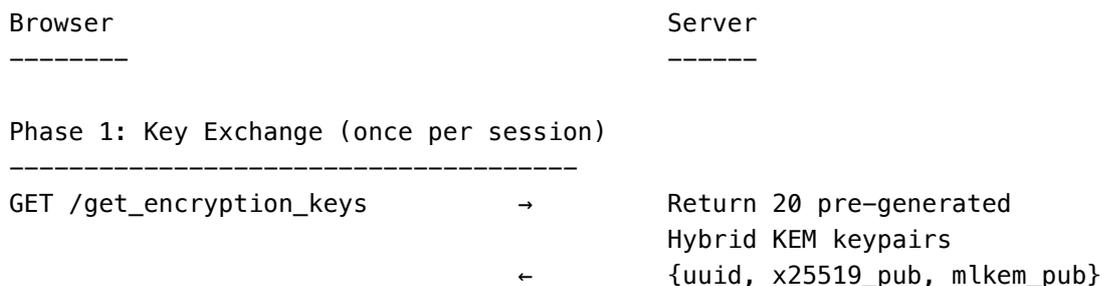
1.9.1 8.1 Problem

Even with HTTPS, certain intermediaries can inspect traffic:

- **CloudFlare** (CDN/DDoS protection) terminates TLS and can see plaintext requests
- **Corporate proxies** may perform TLS inspection
- **API parameters** (like `?cmd=read_email&id=123`) leak metadata

1.9.2 8.2 Solution: End-to-End Encrypted API

All API communication is additionally encrypted end-to-end between the browser and the application server, inside the HTTPS tunnel:



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

1.9.3 8.3 Key Properties

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

1.9.4 8.4 What CloudFlare Sees

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

- POST /e — a single, opaque endpoint
- A binary blob of encrypted data
- No API command names, no parameters, no email IDs, no user data

1.10 9. Folder Sharing Cryptography**1.10.1 9.1 Sharing Model**

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

1.10.2 9.2 Sharing Flow

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 folderKey

1.10.3 9.3 Permission Model

Permission	Capability
<code>readonly</code>	Read folder emails (decrypt only)
<code>einliefern</code>	Submit new emails into folder
<code>bearbeiten</code>	Edit folder contents
<code>antworten</code>	Reply to emails within folder
<code>vollzugriff</code>	Full access including ownership transfer

1.10.4 9.4 Cross-Vault Copy (Re-Wrapping)

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

1. Decrypt shared folder key using recipient's private keys
2. Decrypt email's ephemeral key using folder key
3. Re-encapsulate ephemeral key with recipient's own public keys

4. Store re-wrapped copy in recipient's vault

The server facilitates the transfer but never sees any plaintext key material.

1.11 10. Side-Channel Protections

1.11.1 10.1 Bucket Padding

Problem: Encrypted email sizes can leak information. An attacker observing ciphertext lengths could infer content (e.g., a 50-byte email is likely “OK, thanks” while a 500KB email contains attachments).

Solution: All data is padded to fixed-size “buckets” before encryption:

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]

Example: A 523-byte email is padded to 1,024 bytes. An observer sees only “1KB email” — not the actual 523-byte size.

1.11.2 10.2 Compression-Before-Encryption

Data is compressed with gzip (level 6) **before** encryption. This is the only correct order:

- Compression after encryption would fail (encrypted data has maximum entropy)
- The bucket padding after compression prevents CRIME/BREACH-style attacks that exploit compression ratios

1.11.3 10.3 Threading Privacy

Email threads use SHA-256 hashes of Message-ID headers instead of plaintext identifiers. The server can group related emails by hash equality without knowing the actual message identifiers.

1.12 11. Key Management & Lifecycle

1.12.1 11.1 Key Hierarchy

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

```

1.12.2 11.2 Key Storage

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

1.12.3 11.3 Key Fingerprints

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

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

1.13 12. Recovery Mechanism

1.13.1 12.1 Recovery Key (BIP39 Mnemonic)

During vault setup, the user is presented with a **24-word recovery phrase** generated from 256 bits of entropy, encoded using the BIP39 standard:

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

1.13.2 12.2 Recovery Key Derivation

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)

1.13.3 12.3 What the Server Stores

The server stores **only the SHA3-256 hash** of a derived verification key. It does not store:

- The recovery words
- The entropy
- The verification key itself

1.13.4 12.4 Recovery Flow

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

1.13.5 12.5 Rate Limiting

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

1.13.6 12.6 No Password Recovery

There is no password reset via email. If a user loses their password AND all other authentication factors (passkey + recovery key), their data is permanently inaccessible. This is the fundamental proof that zero-knowledge works — if the service could recover user data, it could also read it.

1.14 13. Passwordless Authentication (Passkeys)

1.14.1 13.1 WebAuthn PRF Extension

Aionda Mail supports FIDO2 passkeys (hardware security keys, biometric authenticators) for passwordless login and vault unlock.

The **WebAuthn PRF (Pseudo-Random Function) extension** provides a deterministic 32-byte output bound to the specific passkey and credential. This output is used to protect Shamir Share 2.

Yes Green = Verified
No Red = MITM Detected
! Red = Missing Signatures

1.15.3 14.3 Response Signature Verification (Ed25519)

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

Signing process (server-side):

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

Verification process (browser extension):

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

Key properties:

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

1.15.4 14.4 Ed25519 Public Key Management

Public keys are shipped with the browser extension in `public_key.json`:

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

- **Key format:** SPKI DER (Subject Public Key Info, Distinguished Encoding Rules)

- **Key size:** 32 bytes (256-bit Ed25519 public key)
- **Signature size:** 64 bytes (fixed)
- **Rotation:** New keys are added before old keys expire; extension updates deliver new keys
- **No server trust:** Keys are embedded in the extension binary, not fetched from the server

1.15.5 14.5 TLS Certificate Verification (Firefox)

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

Verification flow:

1. Browser extension intercepts HTTPS response
2. Extract TLS certificate chain from browser's security info:
 - Leaf certificate fingerprint (SHA-256)
 - Issuer Distinguished Name (O=, CN=)
 - Subject (CN=)
3. Check against known MITM issuers (hardcoded blocklist):
ZScaler, Netskope, Fortinet, Palo Alto, Blue Coat,
Check Point, Barracuda, Sophos, WatchGuard, Cisco Umbrella
→ If match: MITM detected, show warning
4. Check against trusted issuers:
Google Trust Services, Cloudflare, Let's Encrypt,
DigiCert, Sectigo
→ If match AND subject matches expected domain: OK
5. If unknown issuer: Fetch server's own certificate fingerprint
 - Server connects to itself via external routing (prevents spoofing)
 - Response is Ed25519 signed (prevents MITM from lying about cert)
 - Compare issuer organization with browser's certificate issuer
 - If mismatch: MITM suspected, show warning

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

1.15.6 14.6 Self-Certificate Fetching (Anti-Spoofing)

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

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

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

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

1.15.7 14.7 Security Status Indicators

The extension displays a badge on the browser toolbar:

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

1.15.8 14.8 Threat Coverage

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

1.15.9 14.9 Limitations

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

1.16 15. Enterprise Email Archive (Blockchain)

1.16.1 15.1 Overview

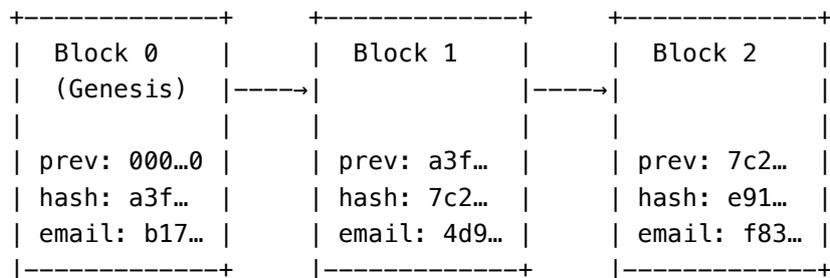
Aionda Mail’s Enterprise plan includes a **GoBD-compliant email archive** secured by a cryptographic hash chain (blockchain). Every archived email becomes an immutable block in a per-company chain. Any tampering — modification, deletion, or insertion of blocks — is cryptographically detectable.

The archive combines two independent security layers:

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

1.16.2 15.2 Hash Chain Architecture

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



Block hash calculation:

```

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

Properties:

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

1.16.3 15.3 Tamper Detection

The chain verification algorithm detects any form of tampering:

For each block (ordered by block_number ASC):

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

If ANY check fails → chain is broken at block N

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

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

1.16.4 15.4 Company Archive Key (CAK) — Zero-Knowledge Encryption

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

Company Owner's Browser

Server

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

 2. Derive wrapping key from password:


```
wrappingKey = HKDF-SHA256(
    password,
    salt = "trashmail-archive-{account_id}",
    info = "trashmail-archive-key-wrap",
    length = 32
)
```

 3. Wrap private keys:


```
AES-256-GCM(x25519_priv || mlkem_priv, wrappingKey)
```

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

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

1. Owner decrypts CAK private keys with their password
2. Owner re-wraps private keys with target employee's password-derived key

3. Server stores the re-wrapped copy on the employee's record
4. Each authorized employee has their own independently wrapped copy

The server never sees the CAK private keys in plaintext.

1.16.5 15.5 What Gets Encrypted

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

Encrypted metadata (AES-256-GCM with Hybrid KEM):

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

Encrypted email content (separate Hybrid KEM key wrapping):

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

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

1.16.6 15.6 Audit Trail

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

Action	When Logged
EMAIL_RECEIVED / EMAIL_SENT / DRAFT_ARCHIVED	Email archived
VIEW_EMAIL / VIEW_ATTACHMENT SEARCH_ARCHIVE	Employee reads archived email Search performed
EXPORT_EMAIL / EXPORT_REPORT VERIFY_CHAIN / CHAIN_VERIFIED_OK / CHAIN_VERIFIED_BROKEN	Data exported Integrity check
LEGAL_HOLD_SET / LEGAL_HOLD_RELEASED	Legal hold toggled
ARCHIVE_DECRYPT ADMIN_ACCESS	CAK used to decrypt content Administrative action

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

1.16.7 15.7 Legal Hold & Retention

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

1.16.8 15.8 Forensic Export

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

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

1.17 16. What the Server Sees — and What It Does Not

This section explicitly documents the zero-knowledge boundary.

1.17.1 16.1 The Server CAN See

Data	Why Visible	Mitigation
IP address	TCP/IP requirement	Use VPN/Tor if desired
Timestamps	Email reception time	Inherent to email protocol
Encrypted email blobs	Stored for retrieval	AES-256-GCM encrypted, key unknown to server
Padded ciphertext sizes	Storage requirement	Bucket padding hides actual sizes
Recipient DEA address	Routing requirement	DEA is disposable, not the real address
Account existence	Authentication flow	User enumeration protection deployed
Public keys	Required for encryption by server	Public by definition, not sensitive
Encrypted Shamir shares	Storage for user	XOR'd with keys server doesn't know
OPAQUE records	Authentication protocol	Not password hashes, encrypted at rest

1.17.2 16.2 The Server CANNOT See

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

1.17.3 16.3 Cryptographic Guarantee

Even with full access to:

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

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

1.18 17. Algorithm Reference

1.18.1 17.1 Complete Algorithm Table

Component	Algorithm	Parameters	Standard
Password authentication	OPAQUE	RFC 9807, aPAKE	RFC 9807
Password key derivation	PBKDF2-SHA256	600,000 iterations, 32B salt, 32B output	NIST SP 800-132
Vault encryption	AES-256-GCM	256-bit key, 96-bit nonce, 128-bit tag	NIST SP 800-38D
Classical key exchange	X25519	Curve25519, 256-bit	RFC 7748
Post-quantum KEM	ML-KEM-1024	Kyber-1024, NIST Level 5	NIST FIPS 203
Hybrid key derivation	HKDF-SHA256	64B IKM, info=“trashmail-hybrid-kem-v1”	RFC 5869
Secret sharing	Shamir SSS	k=2, n=3, GF(2 ⁸)	Shamir (1979)

Component	Algorithm	Parameters	Standard
Recovery key encoding	BIP39	256-bit entropy, 24 words	BIP-0039
Recovery key derivation	HKDF-SHA3-256	Account-bound salt	NIST FIPS 202
Recovery key verification	SHA3-256	32-byte output	NIST FIPS 202
OPAQUE record encryption	AES-256-GCM	Server-side at-rest encryption	NIST SP 800-38D
Passkey vault unlock	WebAuthn PRF	HMAC-based, hardware-bound	WebAuthn Level 2
Compression	gzip	Level 6	RFC 1952
Bucket padding	Custom	17 sizes (256B–16MB), 0xDEAD magic	–
Response signing	Ed25519	256-bit key, 512-bit signature	RFC 8032
Archive hash chain	SHA3-256	Per-block hash, sequential linking	NIST FIPS 202
Archive key wrapping (CAK)	HKDF-SHA256 + AES-256-GCM	Password-derived wrapping key	RFC 5869 / NIST SP 800-38D
Certificate verification	SHA-256	TLS cert fingerprint comparison	–
Email threading	SHA-256	Hash of Message-ID	NIST FIPS 180-4

1.18.2 17.2 Security Levels

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

1.19 18. Comparison with Other Providers

Feature	Aionda Mail	Tuta Mail	Proton Mail
Country	Germany (Stuttgart)	Germany (Hannover)	Switzerland
Zero-Knowledge	Yes (OPAQUE + client-side crypto)	Yes	Yes
Post-Quantum	Yes (ML-KEM-1024 + X25519 Hybrid)	Yes (Kyber-based)	In development

Feature	Aionda Mail	Tuta Mail	Proton Mail
Password protocol	OPAQUE (RFC 9807) – password never leaves browser	bcrypt (password sent to server over TLS)	SRP-based
Subject encrypted	Yes	Yes	No
Headers encrypted	Yes	Partial	No
Contact names encrypted	Yes (in vault)	Yes	No
Disposable email addresses	Yes (core feature, unlimited for Plus)	No	Yes (via SimpleLogin)
Browser addon	Yes (Chrome + Firefox)	No	Via SimpleLogin
Folder sharing	Yes (Hybrid KEM per recipient)	Limited	Yes
Open source client	No	Yes	Yes
Security audit	Planned	Yes	Yes
Password recovery	No (by design)	No (by design)	No (by design)
Passkey support	Yes (FIDO2 + PRF)	Yes	Yes
PGP support	Yes (incoming + outgoing)	No (own protocol)	Yes (OpenPGP)
GoBD-compliant email archive	Yes (SHA3-256 hash chain + Hybrid KEM)	No	No
MITM detection (browser ext.)	Yes (Ed25519 signatures + TLS check)	No	No
Perfect Forward Secrecy (API)	Yes (per-request ephemeral keys)	Unknown	Unknown
Email size obfuscation	Yes (bucket padding)	Unknown	No

1.20 19. Limitations & Honest Boundaries

1.20.1 19.1 Web Application Trust Model

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

Current mitigations:

- Subresource Integrity (SRI) hashes on all script tags

- Content Security Policy (CSP) headers restrict script sources
- All critical cryptographic code is included in the main application bundle
- **Guardian browser extension** (Section 14): Ed25519 signature verification on all API responses detects server-side tampering; TLS certificate verification (Firefox) detects MITM proxies

Planned mitigations:

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

1.20.2 19.2 Metadata Visibility

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

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

1.20.3 19.3 Email Processing Log

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

When enabled (opt-in per DEA):

- The full raw SMTP message (headers + body) is stored in plaintext on the server
- Automatic deletion after a short retention period (less than 7 days)
- Accessible only to the account owner via authenticated API
- Purpose: troubleshooting delivery issues, verifying forwarding, reviewing spam filtering decisions

When disabled:

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

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

1.20.4 19.4 External Email Security

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

1.20.5 19.5 No Key Escrow

There is no master key, backdoor, or recovery mechanism available to Aionda GmbH. If a user loses their password and all recovery methods, their data is permanently lost. This is an

intentional design decision that proves the integrity of the zero-knowledge model.

1.21 20. Roadmap

Milestone	Status	Target
Zero-Knowledge Architecture	Completed	—
Post-Quantum Hybrid KEM (ML-KEM-1024)	Completed	—
OPAQUE Authentication (RFC 9807)	Completed	—
Shamir Secret Sharing (2-of-3)	Completed	—
Encrypted API Transport Layer	Completed	—
Passkey/WebAuthn PRF Support	Completed	—
End-to-End Encrypted Calendar	Completed	—
GoBD-Compliant Email Archive (Blockchain)	Completed	—
Guardian MITM Protection (Ed25519)	Completed	—
TLS Certificate Verification (Firefox)	Completed	—

1.22 Document History

Version	Date	Changes
1.0	March 2026	Initial publication

1.23 Contact

Aionda GmbH Stephan Ferraro Stuttgart, Germany

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

This document describes the security architecture of Aionda Mail as of March 2026. Cryptographic systems evolve — this document will be updated as the architecture changes.