Aionda Mail Security Whitepaper

Zero-Knowledge · Post-Quantum · Made in Germany

Version 1.4 — June 2026

Aionda GmbH
Stuttgart, Germany

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

PUBLIC DOCUMENT

Contents

1. Resume executif

Aionda Mail est un service de messagerie chiffre Zero-Knowledge et post-quantique, opere par Aionda GmbH a Stuttgart, en Allemagne. Le service combine des adresses e-mail jetables (DEAs) avec une boite aux lettres entierement chiffree — une combinaison qu'aucun autre fournisseur ne propose.

Proprietes de securite fondamentales :

- **Architecture Zero-Knowledge :** Tout le chiffrement et le dechiffrement s'effectue exclusivement dans le navigateur de l'utilisateur. Le serveur n'a jamais acces au contenu des e-mails dans la boite aux lettres securisee, aux mots de passe ou aux cles de chiffrement.
- **Securite post-quantique :** Le mecanisme hybride d'encapsulation de cles (X25519 + ML-KEM-1024) protege toutes les donnees contre les attaques des ordinateurs classiques et quantiques.
- **Authentification Zero-Knowledge :** Le protocole OPAQUE (RFC 9807) garantit que les mots de passe ne sont jamais transmis ni stockes sur le serveur — meme pas sous forme de hachages.
- **Shamir Secret Sharing (2-of-3) :** La cle maitre du coffre-fort est divisee en trois parts protegees par le mot de passe, la passkey et la cle de recuperation. Deux parts quelconques suffisent a reconstruire la cle maitre.
- **Perfect Forward Secrecy :** Chaque requete API utilise une paire de cles cryptographiques unique et a usage unique. La compromission d'une requete n'affecte aucune autre.
- **Protection MITM (Guardian) :** L'extension de navigateur verifie independamment toutes les reponses du serveur via des signatures Ed25519 et detecte les attaques de type « homme du milieu » grace a la verification des certificats TLS — meme face aux proxys d'entreprise et aux CDN compromis.
- **Archive e-mail conforme GoBD :** Les comptes d'entreprise beneficient d'une chaine de hachage inviolable (blockchain SHA3-256) avec contenu chiffre de bout en bout (Hybrid KEM), une piste d'audit complete, une mise en suspens juridique et une retention configurable — conforme aux reglementations allemandes GoBD.
- **Pas de recuperation de mot de passe :** En cas de perte du mot de passe et de toutes les methodes de recuperation, les donnees sont irrecuperables. C'est un choix de conception — cela prouve que le serveur ne peut pas acceder aux donnees.

Juridiction : Droit allemand (RGPD/DSGVO), aucun partage de donnees avec des services de renseignement etrangers.

---

2. Modele de menace

2.1 Contre quoi Aionda Mail protege

Menace	Protection
Compromission du serveur (fuite de base de donnees, acces interne)	Tout le contenu des e-mails est chiffre avec des cles que le serveur ne possede jamais
Ecoute reseau (FAI, Wi-Fi, CDN)	Transport API chiffre de bout en bout via Hybrid KEM

Menace	Protection
Inspection CloudFlare	Les requetes API sont chiffrees avant de quitter le navigateur ; CloudFlare ne voit que du texte chiffre. L'extension Guardian detecte les alterations de reponses via des signatures Ed25519
Proxys MITM d'entreprise (ZScaler, Fortinet, etc.)	L'extension Guardian detecte les certificats de proxy via une liste de blocage des emetteurs (Firefox)
Attaques par ordinateur quantique (« recolter maintenant, dechiffrer plus tard »)	ML-KEM-1024 (NIST FIPS 203) offre une resistance post-quantique
Vol de base de donnees de mots de passe	OPAQUE ne stocke que des enregistrements cryptographiques, pas de hachages de mots de passe
Attaques hors ligne par force brute sur les mots de passe	OPAQUE empeche les attaques hors ligne ; la limitation de debit cote serveur empeche les attaques en ligne
Analyse de la taille des e-mails	Le Bucket Padding masque les tailles reelles des e-mails
Canaux auxiliaires de compression (CRIME/BREACH)	Le Bucket Padding est applique apres la compression
Enumeration d'utilisateurs	Reponses factices deterministes pour les comptes inexistants

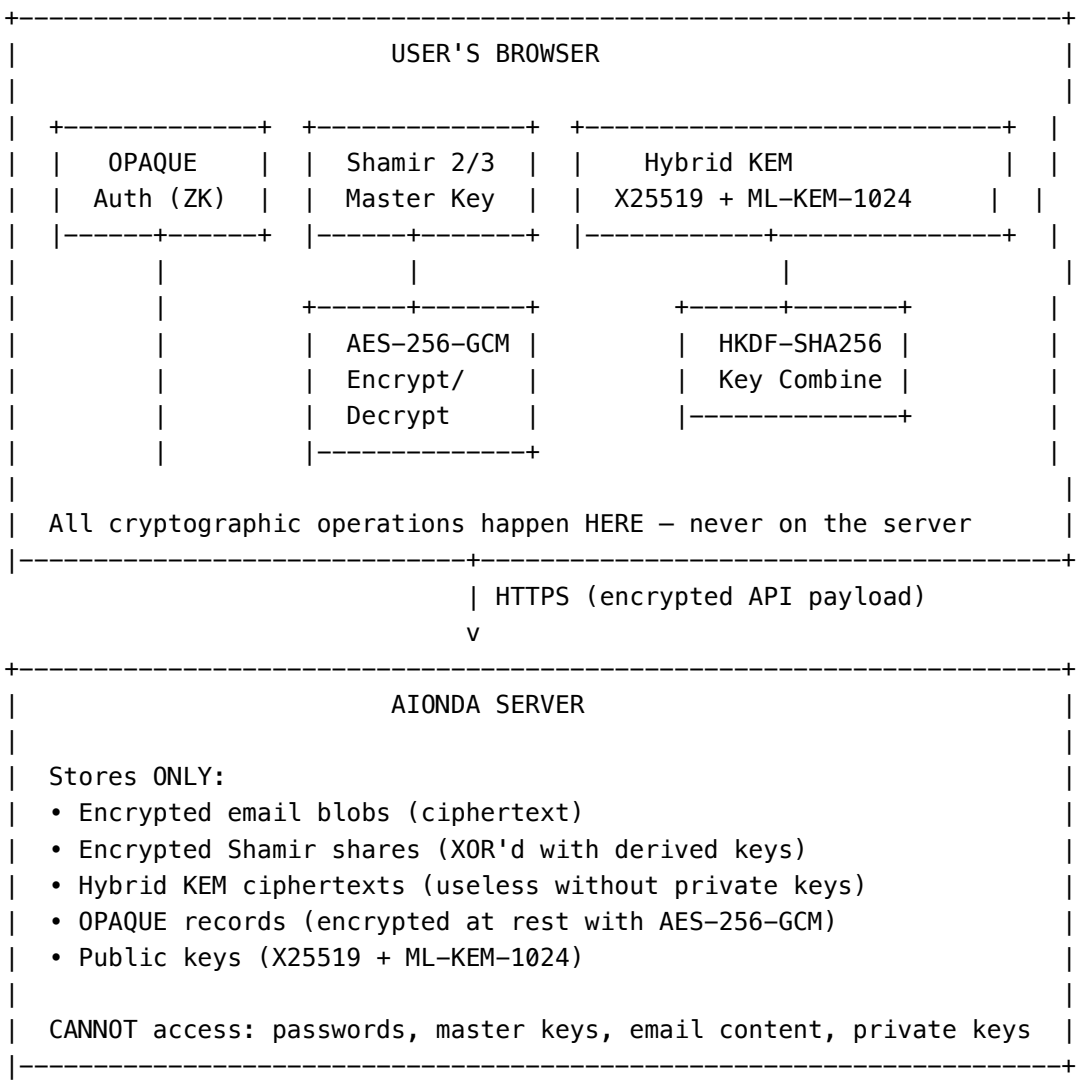
2.2 Contre quoi Aionda Mail ne protege PAS

Limitation	Explication
Appareil compromis	Si un logiciel malveillant controle le navigateur, il peut lire le contenu dechiffre
Metadonnees vers des destinataires externes	Les e-mails vers Gmail/Outlook circulent en clair apres avoir quitte les serveurs d'Aionda (sauf si PGP est utilise)
Metadonnees des e-mails sur les serveurs	Les horodatages, les adresses IP et les tailles des e-mails chiffres sont visibles par le serveur
Confiance dans la livraison web	Le navigateur telecharge du JavaScript depuis les serveurs d'Aionda a chaque visite (voir la section 18 pour les mesures d'attenuation)
Cryptanalyse sous contrainte physique	Aucun systeme cryptographique ne protege contre la coercition physique

2.3 Principe de conception

Aionda Mail suit le modele du **« serveur honnete »** : le systeme est concu de sorte que meme un serveur totalement compromis — ou un operateur malveillant — ne puisse dechiffrer les donnees des utilisateurs. La securite ne repose pas sur la confiance envers l'operateur. Elle repose sur les mathematiques.

3. Vue d'ensemble de l'architecture



3.1 Vue d'ensemble des composants

Composant	Objectif	Emplacement
OPAQUE (RFC 9807)	Authentification par mot de passe Zero-Knowledge	Client + Serveur
Shamir Secret Sharing	Protection de la cle maitre du coffre-fort (seuil 2-of-3)	Client uniquement
Hybrid KEM (X25519 + ML-KEM-1024)	Encapsulation de cles post-quantique pour les e-mails	Client + Serveur
AES-256-GCM	Chiffrement symetrique authentifie	Client + Serveur
HKDF-SHA256	Derivation de cles a partir de secrets partages hybrides	Client + Serveur

Composant	Objectif	Emplacement
BIP39	Encodage de la cle de recuperation (mnemonique de 24 mots)	Client uniquement
WebAuthn PRF	Deverrouillage du coffre-fort par passkey	Client uniquement
Bucket Padding	Protection contre les canaux auxiliaires	Client + Serveur

---

4. Authentification Zero-Knowledge (OPAQUE)

4.1 Pourquoi pas le hachage de mot de passe ?

Les services traditionnels stockent des hachages de mots de passe (bcrypt, Argon2). Bien que meilleure que le texte en clair, cette approche presente des faiblesses fondamentales :

- Le serveur voit le mot de passe lors de la connexion (meme brievement en RAM)
- Les hachages de mots de passe peuvent etre attaques par force brute hors ligne si la base de donnees est volee
- Le serveur pourrait etre modifie pour enregistrer les mots de passe

OPAQUE elimine ces trois problemes. Le mot de passe ne quitte jamais le navigateur — ni en clair, ni sous forme de hachage, sous aucune forme.

4.2 Fonctionnement d'OPAQUE

OPAQUE (RFC 9807) est un protocole d'echange de cles authentifie par mot de passe asymetrique (aPAKE). Il utilise un mecanisme de defi-reponse cryptographique ou le serveur peut verifier que l'utilisateur connait le bon mot de passe sans jamais apprendre ce qu'il est.

Inscription (une seule fois) :

```
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)
```

Connexion (a chaque session) :

```
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)
```

Proprietes essentielles :

- Le mot de passe est verifie **cote client** a l'etape 3 — le serveur ne le voit jamais
- Le serveur stocke un **enregistrement OPAQUE**, qui n'est pas un hachage de mot de passe et ne peut pas etre attaque par force brute hors ligne
- Les enregistrements OPAQUE sont en outre **chiffres au repos** avec AES-256-GCM a l'aide d'une cle cote serveur
- **Protection contre l'enumeration d'utilisateurs** : les comptes inexistants recoivent des reponses factices deterministes avec un timing identique
- **Limitation de debit** : maximum 3 tentatives d'authentification par session, delai d'expiration de session de 120 secondes

4.3 Implementation

- **Bibliotheque :** `@serenity-kit/opaque` (basee sur WASM, qualite production)
- **Composant serveur :** microservice dedie aux operations cryptographiques OPAQUE
- **Format base64 :** base64url (compatible URL, sans remplissage) pour la compatibilite du protocole
- **Journalisation d'audit :** tous les evenements d'authentification sont journalises avec horodatages et adresses IP

4.4 Migration SRP

Les comptes existants utilisant SRP-6a sont automatiquement migres vers OPAQUE lors de la prochaine connexion. Apres la migration, le verificateur SRP est definitivement supprime. La migration est unidirectionnelle — les comptes ne peuvent pas revenir a SRP.

---

5. Cle maitre du coffre-fort & Shamir Secret Sharing

5.1 Generation de la cle maitre

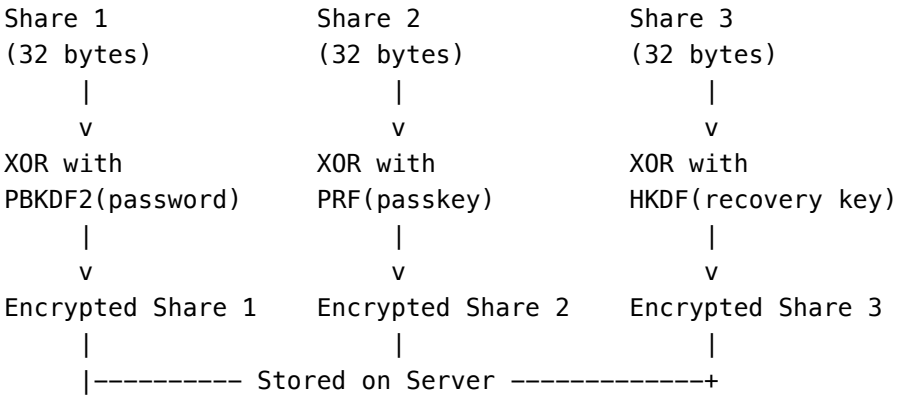
Lorsqu'un utilisateur active la boite aux lettres chiffree, une **cle maitre de 256 bits (32 octets)** est generee a l'aide du generateur de nombres aleatoires cryptographiquement sur du navigateur (`crypto.getRandomValues`).

Cette cle maitre est la racine de tout le chiffrement. Elle ne quitte jamais le navigateur en clair. Elle n'est stockee nulle part — ni dans le navigateur, ni sur le serveur, sous aucune forme.

5.2 Shamir Secret Sharing (2-of-3)

La cle maitre est divisee en trois parts a l'aide du schema **Shamir's Secret Sharing** sur le corps de Galois GF(2^8) avec le polynome irreductible AES (x^8 + x^4 + x^3 + x + 1).

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



Propriete de seuil : deux quelconques des 3 parts suffisent a reconstruire la cle maitre par interpolation de Lagrange. Le serveur ne stocke que les parts chiffrees — et ne peut en dechiffrer aucune.

5.3 Protection des parts

Chaque part est combinee par XOR avec une cle derivee d'un facteur d'authentification different :

Part	Protegee par	Derivation de cle
Part 1	Mot de passe	PBKDF2-SHA256, 600 000 iterations, sel aleatoire de 32 octets
Part 2	Passkey (FIDO2)	Extension WebAuthn PRF, liee au materiel
Part 3	Cle de recuperation	HKDF-SHA3-256 avec sel lie au compte

Scenarios de reconstruction :

- **Connexion normale :** Mot de passe (Part 1) + Passkey (Part 2) -> Cle maitre
- **Passkey perdue :** Mot de passe (Part 1) + Cle de recuperation (Part 3) -> Cle maitre
- **Changement de mot de passe :** Passkey (Part 2) + Cle de recuperation (Part 3) -> Cle maitre

5.4 Protection du stockage de session

Meme au sein d'une session de navigateur, la cle maitre n'est jamais stockee en clair :

1. Une cle AES-256 ephemere est generee
2. La cle maitre est chiffree avec cette cle ephemere
3. Seul le blob chiffre est place dans le `sessionStorage`
4. La cle ephemere n'existe qu'en memoire JavaScript (liberee par le ramasse-miettes a la fermeture de l'onglet)

---

6. KEM hybride post-quantique

6.1 Pourquoi le post-quantique ?

Les ordinateurs quantiques executant l'algorithme de Shor pourraient casser la cryptographie a cle publique classique (RSA, ECDH, X25519) en temps polynomial. Bien que des ordina-

teurs quantiques a grande echelle n'existent pas encore, la menace des attaques **« recolter maintenant, dechiffrer plus tard »** est reelle : des adversaires pourraient stocker des donnees chiffrees aujourd'hui et les dechiffrer lorsque les ordinateurs quantiques seront disponibles.

6.2 Approche hybride

Aionda Mail utilise un **mecanisme hybride d'encapsulation de cles** combinant :

- **X25519** (Curve25519 ECDH) — securite classique eprouvee, niveau de securite de 128 bits
- **ML-KEM-1024** (NIST FIPS 203, anciennement Kyber-1024) — securite post-quantique, niveau de securite NIST 5

L'approche hybride offre une **defense en profondeur** : la cle combinee est sure tant qu'**au moins un** des deux algorithmes reste incasse.

6.3 Processus d'encapsulation

```
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 Processus de decapsulation

```
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 Tailles des cles

Parametre	Taille	Standard
X25519 public key	32 octets	RFC 7748
X25519 private key	32 octets	RFC 7748
ML-KEM-1024 public key	1 568 octets	NIST FIPS 203
ML-KEM-1024 private key	3 168 octets	NIST FIPS 203
ML-KEM-1024 ciphertext	1 568 octets	NIST FIPS 203
Secret partage combine	32 octets	Sortie HKDF-SHA256

6.6 Bibliotheque

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

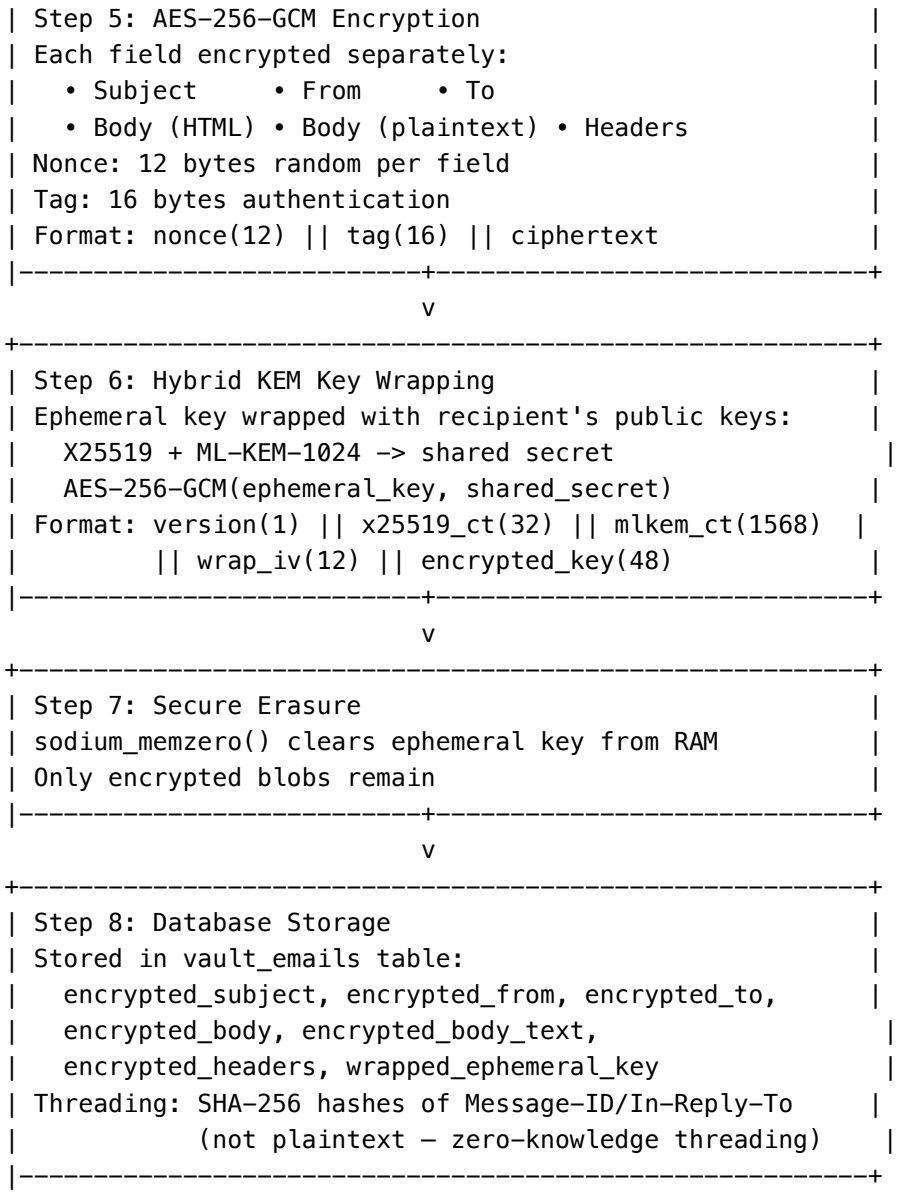
---

7. Pipeline de chiffrement des e-mails

7.1 E-mail entrant (SMTP -> Stockage chiffre)

Lorsqu'un e-mail arrive sur le serveur SMTP d'Aionda Mail :

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



7.2 Lecture d'un e-mail (dechiffrement dans le navigateur)

Le processus inverse se deroule entierement dans le navigateur :

1. Recuperation de l'e-mail chiffre depuis le serveur via l'API chiffree
2. Analyse de la cle ephemere encapsulee (extraction du texte chiffre X25519 + texte chiffre ML-KEM)
3. **Decapsulation Hybrid KEM** a l'aide des cles privees du coffre-fort -> secret partage
4. Dechiffrement de la cle ephemere AES-256
5. **Dechiffrement AES-256-GCM** de chaque champ (objet, expediteur, destinataire, corps, en-tetes)
6. Reordonnancement des octets : format serveur (nonce || tag || ct) -> format WebCrypto (nonce || ct || tag)
7. Suppression du Bucket Padding (detection des octets magiques 0xDEAD)
8. Decompression si gzip (detection des octets magiques 0x1F8B)

9. Decodage UTF-8 vers texte en clair

7.3 Envoi d'un e-mail

Lors de la composition et de l'envoi d'un e-mail :

1. Le client chiffre le contenu de l'e-mail avec un protocole de defi-reponse
2. Le serveur recoit le payload chiffre, le dechiffre de maniere ephemere (en memoire uniquement), et l'envoie via SMTP
3. Le serveur retourne les en-tetes MIME generes au client (chiffres)
4. Le client chiffre une copie avec la cle maitre du coffre-fort et la stocke dans le dossier Envoyes
5. Le texte en clair ephemere cote serveur est immediatement detruit — jamais ecrit sur disque

7.4 Pieces jointes

Chaque piece jointe est chiffree independamment :

- Cle ephemere AES-256 separee par piece jointe
- Encapsulation Hybrid KEM separee par piece jointe
- Nom de fichier et type MIME chiffres separement
- Pas de compression pour les formats deja compresses (JPEG, ZIP, PDF, etc.)

7.5 Regroupement des e-mails (Zero-Knowledge)

Le regroupement des e-mails (classement des e-mails lies) utilise uniquement des **hachages SHA-256** des en-tetes Message-ID et In-Reply-To. Le serveur ne voit jamais les chaines Message-ID reelles — il peut regrouper les e-mails par egalite de hachage sans en connaitre le contenu.

---

8. Couche de transport API chiffree

8.1 Problematique

Meme avec HTTPS, certains intermediaires peuvent inspecter le trafic :

- **CloudFlare** (CDN/protection DDoS) termine le TLS et peut voir les requetes en clair
- **Les proxys d'entreprise** peuvent effectuer une inspection TLS
- **Les parametres API** (comme `?cmd=read_email&id=123`) exposent des metadonnees

8.2 Solution : API chiffree de bout en bout

Toutes les communications API sont en outre chiffrees de bout en bout entre le navigateur et le serveur applicatif, a l'interieur du tunnel HTTPS :

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

Phase 1: Key Exchange (once per session)
----------------------------------------
GET /get_encryption_keys          ->        Return 20 pre-generated
                                           Hybrid KEM keypairs
```

```
                                  ←         {uuid, x25519_pub, mlkem_pub}

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

8.3 Proprietes essentielles

- **Usage unique :** chaque paire de cles API est utilisee exactement une fois, puis definitivement invalidee
- **Perfect Forward Secrecy :** la compromission d'une cle de requete n'affecte aucune autre requete
- **Liees a la session :** les cles sont revendiquees par une session specifique et ne peuvent pas etre reutilisees par une autre
- **Pool de cles :** le serveur maintient environ 100 000 paires de cles pre-generees
- **Reapprovisionnement automatique :** le client demande automatiquement de nouvelles cles lorsque le cache tombe en dessous de 10
- **Duree de vie des cles :** les cles revendiquees expirent apres 24 heures
- **Bidirectionnel :** la requete ET la reponse sont chiffrees — le serveur ne retourne jamais de texte en clair

8.4 Ce que voit CloudFlare

Avec cette architecture, CloudFlare (ou tout proxy terminant le TLS) ne voit que :

- `POST /e` — un point d'entree unique et opaque
- Un blob binaire de donnees chiffrees
- Aucun nom de commande API, aucun parametre, aucun identifiant d'e-mail, aucune donnee utilisateur

---

9. Cryptographie du partage de dossiers

9.1 Modele de partage

Les utilisateurs peuvent partager des dossiers chiffres avec d'autres utilisateurs d'Aionda Mail. Le mecanisme de partage utilise le Hybrid KEM pour chiffrer une cle specifique au dossier pour chaque destinataire.

9.2 Flux de partage

```
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 Modele de permissions

Permission	Capacite
`readonly`	Lire les e-mails du dossier (dechiffrement uniquement)
`einliefern`	Soumettre de nouveaux e-mails dans le dossier
`bearbeiten`	Modifier le contenu du dossier
`antworten`	Repondre aux e-mails du dossier
`vollzugriff`	Acces complet, y compris le transfert de propriete

9.4 Copie inter-coffres (Re-Wrapping)

Lorsqu'un destinataire copie un e-mail d'un dossier partage vers son propre coffre-fort, la cle de l'e-mail doit etre **re-encapsulee** pour sa propre paire de cles Hybrid KEM. Cette operation est realisee entierement cote client :

1. Dechiffrement de la cle du dossier partage a l'aide des cles privees du destinataire
2. Dechiffrement de la cle ephemere de l'e-mail a l'aide de la cle du dossier
3. Re-encapsulation de la cle ephemere avec les propres cles publiques du destinataire
4. Stockage de la copie re-encapsulee dans le coffre-fort du destinataire

Le serveur facilite le transfert mais ne voit jamais aucune cle en clair.

---

10. Vault Drive — Stockage de documents chiffre

Vault Drive est le stockage de documents Zero-Knowledge d'Aionda Mail. Contrairement aux e-mails, qui utilisent des cles ephemeres par message, Drive utilise une **architecture a cle par fichier** : chaque document recoit sa propre cle AES-256-GCM de 32 octets, generee cote client et enveloppee avec la cle maitre.

10.1 Pourquoi des cles par fichier ?

L'architecture a cle par fichier offre trois avantages cles :

1. **Partage granulaire :** Des documents individuels peuvent etre partages sans exposer la cle maitre. Le serveur re-enveloppe simplement la cle du fichier pour le destinataire — le contenu reste inchange.
2. **Isolation en cas de compromission :** Si une seule cle de fichier est compromise (p.ex. via un lien partage), tous les autres documents restent proteges.
3. **Partage efficace sans re-chiffrement :** Lors du partage, le contenu n'a pas besoin d'etre re-chiffre — tous les destinataires lisent le meme texte chiffre avec une cle re-enveloppee.

10.2 Flux de telechargement montant

```
Client                                                  Serveur
------                                                  -------

1. Generer une cle de fichier aleatoire :
   fileKey = randomBytes(32)

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

3. Envelopper la cle de fichier avec la cle maitre :
   wrappedKey = AES-256-GCM(fileKey, masterKey, nonce4)

4. Envoyer le paquet chiffre :
   POST /e {
```

```
  encContent, encName, encMime,
  wrappedKey, wrappedKeyNonce
}
```

```
5. Stocker dans vault_files + vault_file_c
   (textes chiffres purs, aucun materiel d
```

10.3 Flux de telechargement descendant

```
1. Recuperer chunk + wrapped_key depuis le serveur
2. Deballer la cle de fichier :
   fileKey = AES-256-GCM-decrypt(wrappedKey, masterKey, nonce)
3. Dechiffrer contenu, nom, MIME :
   content = AES-256-GCM-decrypt(encContent, fileKey, nonce1)
```

Le dechiffrement s'effectue dans un **Web Worker** qui ne conserve la cle maitre que temporairement en memoire. Apres l'operation, la memoire est ecrasee.

10.4 Partage — re-enveloppement de cle uniquement

L'avantage central des cles par fichier apparait lors du partage : le contenu n'est **pas** re-chiffre. Seule la cle de fichier de 32 octets est re-enveloppee avec le KEM hybride du destinataire.

```
Proprietaire                                    Destinataire
------------                                    ------------

1. Deballer la cle de fichier :
   fileKey = AES-GCM-decrypt(wrappedKey, masterKey)

2. Charger les cles publiques du destinataire :
   recipient.x25519_pub, recipient.mlkem_pub

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

4. Envelopper la cle de fichier avec sharedSecret :
   shareWrappedKey = AES-256-GCM(fileKey,
                                 sharedSecret, nonce)

5. Enregistrer le partage :
   INSERT INTO vault_file_shares {
     file_uuid, recipient_account_id,
     x25519_ephemeral, mlkem_ciphertext,
     share_wrapped_key, permissions
   }
```

```
6. Charger l'enregistrement + chunk
7. Decapsulation KEM hybride :
   hybridDecapsulate(x25519_ct, mlkem_ct,
     own_x25519_priv, own_mlkem_priv)
   -> sharedSecret
8. Deballer la cle de fichier :
```

```
   fileKey = AES-GCM-decrypt(
     shareWrappedKey, sharedSecret)
9. Dechiffrer le MEME texte chiffre :
   content = AES-GCM-decrypt(
     encContent, fileKey)
```

Le proprietaire n'a pas besoin de la cle maitre du destinataire — seulement des cles publiques KEM hybrides, que le serveur stocke en clair.

10.5 Partage de dossier (recursif)

Lorsqu'un dossier est partage, le client traite recursivement tous les documents et sous-dossiers contenus. Chaque fichier recoit son propre enregistrement de partage contenant la cle de fichier du document respectif, re-enveloppee pour le destinataire. Le dossier lui-meme n'a pas de cle de dossier — la structure est liee via des UUIDs parents.

Important : L'etape d'encapsulation KEM est effectuee **par operation**, pas par fichier. Tous les fichiers d'un partage par lot utilisent le meme `sharedSecret` — ce qui rend l'operation efficace sans reduire la securite (chaque fichier a toujours sa propre cle).

10.6 Modele d'autorisations

Autorisation	Peut effectuer
Lecture	Telecharger, previsualiser, lire les metadonnees
Acces complet	+ renommer, televerser une nouvelle version, deplacer dans le dossier partage, televerser de nouveaux documents
Impossible	Supprimer (proprietaire uniquement), sortir du dossier partage (proprietaire uniquement)

Les autorisations sont stockees dans l'enregistrement `vault_file_shares` et appliquees cote serveur. La cryptographie protege le contenu — le controle d'acces protege les operations.

10.7 Renommage, ecrasement et mises a jour de metadonnees

Pour les documents partages, les operations de renommage et d'ecrasement sont effectuees directement sur l'enregistrement `vault_files` — pas sur les enregistrements de partage individuels. Tous les destinataires voient immediatement le nouveau nom ou contenu, car ils referencent le meme texte chiffre.

Lors d'un **ecrasement** (nouvelle version), une **nouvelle cle de fichier** est generee, l'ancien texte chiffre est remplace, et tous les enregistrements de partage existants sont re-enveloppes cote client avec la nouvelle cle. Le proprietaire doit charger les cles publiques de tous les destinataires pour cette operation.

10.8 Previsualisation et dechiffrement en memoire

Les images, PDF et autres documents sont dechiffres **exclusivement en memoire** pour la previsualisation. Il n'y a pas de cache disque avec des donnees en clair. Le client utilise le **VaultDataLayer**, un cache IndexedDB chiffre qui stocke les textes chiffres de maniere persistante et ne les dechiffre qu'a la demande.

Les miniatures ne sont **jamais** generees cote serveur — le serveur ne voit jamais le contenu.

Les miniatures sont generees cote client a partir de l'original dechiffre et egalement mises en cache chiffrees.

10.9 Ce que le serveur voit

Visible par le serveur	Non visible
Contenu chiffre (octets aleatoires)	Contenu en clair
Nom de fichier chiffre (octets aleatoires)	Nom de fichier en clair
Type MIME chiffre (octets aleatoires)	Type de fichier (image, PDF, texte…)
Taille du fichier (paddee aux limites de buckets)	Taille originale reelle
Horodatage du televersement	Cle de fichier, cle maitre
ID du compte proprietaire	Contenus des sous-dossiers
UUID du dossier parent (pour la structure)	Noms des dossiers (egalement chiffres)
Enregistrements de partage avec textes chiffres KEM	Cle maitre du destinataire, cle de fichier deballee

10.10 Application des quotas

L'application des quotas se fait cote serveur sur la base de la **taille chiffree du fichier** (y compris le padding de bucket). Le serveur ne connait pas la taille reelle en clair — le surcout de padding est intentionnellement paye par l'utilisateur pour empecher la fuite de taille.

Limites : - **Gratuit :** 100 Mo de stockage total, max. 10 Mo par document - **Plus :** 1 Go de stockage total, max. 100 Mo par document

10.11 Partage externe — liens de partage publics

Les documents Vault Drive peuvent etre partages avec des destinataires qui **n'ont pas de compte Aionda Mail**, via des liens de partage publics servis depuis le domaine isole `mail.aionda.com`. La fonctionnalite preserve le principe zero-knowledge en traitant chaque partage comme une **enveloppe cryptographique independante**, decouplee de la cle maitre du coffre du proprietaire.

Modes de protection :

Mode	Facteur de confiance	Cas d'usage
`link_only`	Fragment d'URL (jamais envoye au serveur)	Confort — toute personne disposant du lien peut acceder
`password`	Cle derivee par Argon2id	Transfert hors bande du mot de passe (SMS, appel telephonique)
`recipient_pubkey`	KEM hybride (X25519 + ML-KEM-1024)	Securise post-quantique lorsque le destinataire a preenregistre des cles publiques

10.11.1 Creation du partage (client proprietaire)

```
1. fileKey := AES-GCM-decrypt(vault_files.wrapped_key, masterKey)
2. shareKey := randomBytes(32)                          // ephemere, par partage
3. wrappedFileKey := AES-GCM(fileKey, shareKey)         // nouvelle enveloppe
4. unlockVerifier := HMAC-SHA256(shareKey,
                     "unlock:" || shareUuid)            // preuve de connaissance
5. Si protection par mot de passe :
     salt := randomBytes(16)
     pwKey := Argon2id(password, salt, t=3, m=64MB, p=1)
     wrappedShareKey := AES-GCM(shareKey, pwKey)
     -> Lien : https://mail.aionda.com/s/<shareUuid>
   Sinon (link_only) :
     -> Lien : https://mail.aionda.com/s/<shareUuid>#<base64(shareKey)>
       (fragment d'URL — les navigateurs ne transmettent jamais les fragments au serveur)

6. Metadonnees chiffrees (par partage, avec shareKey) :
     encFilename, encMime, encMessage  // toutes AES-GCM
7. POST /e  (transport API chiffre, voir §8)
   -> le serveur stocke l'enregistrement de partage — ne voit jamais le texte clair
```

La distribution du lien est choisie par le proprietaire — pas par Aionda Mail. Apres la creation, le proprietaire decide du canal : copier le lien, QR code, brouillon `mailto:` dans son propre client, Signal, SMS, AirDrop ou tout autre canal hors bande. Aionda Mail ne voit, n'envoie et ne journalise jamais la distribution sortante — le serveur sait uniquement quelles share-UUIDs existent. Cela importe pour deux raisons : (a) le proprietaire choisit le canal qu'il juge le plus sur (politique de conformite, messagerie preferee, scan QR en face a face), et (b) pour les partages proteges par mot de passe, cela permet la **separation des canaux** — lien via canal A, mot de passe via canal B — afin qu'un canal compromis seul ne donne jamais acces.

10.11.2 Acces au partage (destinataire, sans compte) Le destinataire visite `https://mail.aionda.com/s/` La Share Page est un **bundle separe** du Manager, avec son propre build reproductible, un manifeste Subresource Integrity et un endpoint `/.well-known/integrity.json` pour la verification hors ligne.

```
1. Bootstrap de la Share Page (le client genere un keypair KEM hybride ephemere)
2. share_fetch_meta -> { wrapped_file_key, salt?, encrypted_message, ... }
3. Phase d'unlock — produit un unlock_token a usage unique :
     mode password :   le serveur verifie la derivation Argon2id -> unlock_token
     mode link_only :  le client calcule HMAC-SHA256(shareKey, "unlock:"||uuid)
                       le serveur verifie contre unlockVerifier stocke
                       -> unlock_token
4. fileKey := AES-GCM-decrypt(wrappedFileKey, shareKey)
5. share_download_chunk (par chunk, unlock_token requis)
6. Le client hache le texte clair, appelle share_confirm_download
```

Le `unlock_token` unifie le flow pour les trois modes de protection et permet un rate-limiting et un audit logging centralises — l'acces link_only exige la preuve de connaissance du fragment, les bots et crawlers sans fragment ne peuvent donc pas declencher de telechargement.

10.11.3 Politiques imposees Chaque partage doit declarer les elements suivants a la creation (tous appliques cote serveur) :

- **`expires_at`** — obligatoire, 7 jours par defaut, maximum 90 jours
- **`max_downloads`** — compteur obligatoire, 10 par defaut
- **Rate-limiting** — les tentatives Argon2 sur les partages proteges par mot de passe sont limitees par hash d'IP et par partage
- **`revoked_at`** — le proprietaire peut revoquer tout partage en un clic ; les tentatives d'unlock et telechargements ulterieurs renvoient une reponse unifiee « partage indisponible » (meme erreur qu'expire/epuise — pas d'oracle d'enumeration)

10.11.4 Chaine d'audit infalsifiable Tous les evenements d'acces pertinents sont ajoutes a la hash chain existante `enterprise_audit_log` (SHA3-256, voir §18.2). Les types d'action suivants sont reserves au partage externe :

`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`.

Les hashes d'IP et user-agent utilisent un **sel rotatif quotidien** (`vault_drive_external_share_daily_salts`, purge apres 30 jours) — les hashes historiques deviennent non reversibles apres rotation du sel (conformite RGPD), tout en conservant une correlation court terme pour le proprietaire.

10.11.5 Rotation de cle (rewrap du partage) Le proprietaire peut faire pivoter un partage a tout moment sans reuploader le contenu :

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

Les destinataires utilisant l'ancien lien recoivent « partage indisponible ». Le proprietaire distribue le nouveau lien. Les evenements d'acces des deux partages restent lies via `linked_to_uuid` dans la chaine d'audit.

10.11.6 Ce que fait la revocation — et ce qu'elle ne fait pas La revocation **stoppe la livraison future cote serveur** du partage. Elle **ne revoque pas** retroactivement les share keys, file keys ou chunks deja livres. Un destinataire qui a deja telecharge le texte clair — ou qui a capture le fragment du lien — peut continuer a utiliser ce materiel localement. Pour les cas exigeant une assurance plus forte, combiner : `expires_at` court, `max_downloads` bas, protection par mot de passe et l'extension navigateur Guardian chez le destinataire.

10.11.7 Ce que voit le serveur

Le serveur voit	Le serveur ne voit PAS
`share_uuid`, `file_uuid`, `account_id` (proprietaire)	Nom de fichier, type MIME, contenu, message en clair
`wrapped_file_key`, `wrapped_share_key` (chiffres)	`share_key` (link_only : vit uniquement dans le fragment d'URL, cote client)

Le serveur voit	Le serveur ne voit PAS
`unlock_verifier` (sortie HMAC, non reversible)	Mot de passe, cle derivee Argon2
`protection_mode`, `expires_at`, `max_downloads`, `allow_preview`	Identite du destinataire (uniquement hash sale IP/UA, purge apres 30 jours)
`encrypted_filename`, `encrypted_mime`, `encrypted_message` (chiffres)	Leurs contenus en clair
`sender_display_name` en clair (le destinataire le voit avant l'unlock)	
Entrees de la chaine d'audit pour chaque evenement d'acces	

---

11. Aionda Chat — Messagerie E2EE post-quantique

Aionda Chat est la surface de messagerie temps réel intégrée à Aionda Mail. Contrairement aux e-mails — qui utilisent des cles ephemeres a usage unique par message — les conversations de chat sont longues et necessitent **forward secrecy** et **post-compromise security** pour chaque message individuel. Pour cela, nous avons concu un protocole dedie : **AAR (Aionda Async Ratchet)** — une variante post-quantique du X3DH + Double Ratchet de Signal, dans laquelle chaque etape Diffie-Hellman est remplacee par un KEM hybride (X25519 + ML-KEM-1024).

11.1 Pourquoi un protocole dedie ?

E-mail et chat ont des profils de securite fondamentalement differents :

Propriete	E-mail	Chat
Cadence	Sporadique (minutes/heures)	Temps reel (secondes)
Duree de session	Message unique	Continue, jours a annees
Unite de forward secrecy	Par message	Par message **au sein** d'une longue session
Recuperation post-compromise	Non requise (cle one-shot)	Requise — chaque nouveau message restaure d'un compromis passe
Asynchronisme	Eleve (destinataire souvent hors ligne)	Eleve (destinataire souvent hors ligne)
Dynamique de groupe	Liste de destinataires statique	Ajout/retrait dynamique

Une conversation de chat qui reutiliserait simplement la pipeline e-mail devrait soit (a) bruler une nouvelle paire de cles ephemeres a chaque frappe (inacceptablement lent), soit (b) partager une seule cle de conversation statique (pas de forward secrecy). Aucune n'est acceptable. AAR resout cela en maintenant un etat de cle a ratchet continu par pair — chaque message fait progresser l'etat de maniere irreversible.

11.2 Briques de construction

```
AAR utilise exactement quatre primitives :

   KEM hybride        X25519 + ML-KEM-1024  (encapsulation de cles)
   AEAD               AES-256-GCM           (chiffrement des messages)
   KDF                HKDF-SHA256           (derivation de cles)
   Signatures         Ed25519               (liaison d'identite, signature SPK)

Aucune nouvelle hypothese cryptographique n'est introduite — chaque primitive
est aussi utilisee ailleurs dans le systeme et a ete auditee independamment.
```

11.3 KeyBundle — le materiel de handshake asynchrone

Chaque utilisateur publie un **KeyBundle** sur le serveur lors de l'activation initiale du chat. Le bundle permet aux pairs de commencer a envoyer des messages meme si l'utilisateur est hors ligne.

```
KeyBundle (par compte, hybride de bout en bout) :

   Cle d'identite (IK)        — long terme
      |-- IK.x25519_pub
      |-- IK.mlkem_pub
      |-- IK.ed25519_pub      — utilisee pour signer la SPK

   Signed Pre-Key (SPK)       — rotation hebdomadaire
      |-- SPK.x25519_pub
      |-- SPK.mlkem_pub
      |-- Signature Ed25519   — signe la concatenation, lie SPK a IK

   One-Time Pre-Keys (OPK)    — pool d'environ 100, consommees atomiquement
      |-- OPK.x25519_pub
      |-- OPK.mlkem_pub
```

Stockage serveur : Seules les moities publiques de chaque cle sont stockees, plus signatures et metadonnees. Les moities privees ne quittent jamais le navigateur d'origine. Le serveur applique un `UPDATE … LIMIT 1 SET consumed_ts = NOW()` atomique lorsqu'une OPK est recuperee, garantissant la semantique d'usage unique sous des requetes concurrentes.

Quand le pool d'OPK descend sous 20, le client le reapprovisionne avec un nouveau lot — cela empeche le handshake asynchrone de se degrader vers un mode degenere sans entropie one-time.

11.4 Handshake style PQXDH (X3DH-PQ)

Pour demarrer une nouvelle conversation avec Bob, Alice recupere le bundle de Bob (`IK_B`, `SPK_B`, une `OPK_B`) et la signature Ed25519 de Bob sur `SPK_B`. Alice verifie la signature, puis effectue quatre encapsulations KEM hybrides :

```
1. Verifier la signature Ed25519 sur SPK_B  — lie SPK a l'identite long terme

2. Generer la cle ephemere d'Alice (EK_A) :
   EK_A.x25519, EK_A.mlkem  (one-time, jetee immediatement apres le handshake)
```

```
3. Quatre encapsulations KEM hybrides :
   ss1 = HKEM(IK_A_priv  ⊗ SPK_B_pub)    — Identite Alice -> SPK Bob
   ss2 = HKEM(EK_A_priv  ⊗ IK_B_pub)     — Ephemere Alice -> Identite Bob
   ss3 = HKEM(EK_A_priv  ⊗ SPK_B_pub)    — Ephemere Alice -> SPK Bob
   ss4 = HKEM(EK_A_priv  ⊗ OPK_B_pub)    — Ephemere Alice -> OPK Bob

   (chaque ss_i est lui-meme la combinaison HKDF d'un secret X25519
    ET d'un secret ML-KEM — voir Section 6.4)

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

5. Oublier toute matiere ephemere privee ; SK ensemence le ratchet.
```

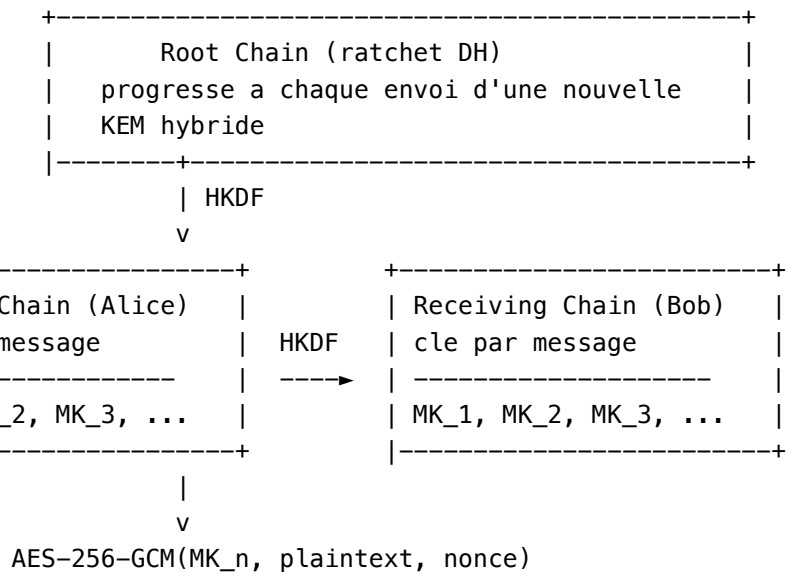
La construction garantit :

- **Authentification mutuelle** via la `SPK_B` signee Ed25519 et la liaison `IK_A`
- **Forward secrecy** meme si `IK_B` est compromis ulterieurement — `ss3` et `ss4` utilisent des cles ephemeres des deux cotes
- **Securite post-quantique** car chaque `ss_i` inclut un terme ML-KEM — un attaquant collectant le trafic d'aujourd'hui pour un futur ordinateur quantique ne peut pas reconstruire les quatre secrets
- **Resistance au rejeu** par consommation atomique des OPK — le declencheur cote serveur de Bob refuse de liberer la meme OPK deux fois

L'identifiant de l'OPK consommee par Alice est lie au parametre `info` de HKDF — Bob ne peut donc reproduire la meme cle racine qu'avec la meme OPK, et une seule fois.

11.5 Double Ratchet — cles par message

Apres le handshake, les deux cotes font progresser un etat **Double Ratchet** pour chaque message :



Chaque evenement de chat porte la **cle publique de ratchet** actuelle de l'expediteur (paire hybride, ~1,6 Ko). Quand le destinataire recoit un message dont la cle de ratchet differe de la derniere vue, les deux parties effectuent une nouvelle KEM hybride, derivent une nouvelle cle racine et reinitialisent leurs chaines. C'est le **ratchet DH** dans la terminologie Signal — sauf que chaque etape est un KEM hybride complet, pas un seul Diffie-Hellman.

Garanties par message :

- **Forward secrecy :** la suppression de `MK_n` rend le message `n` indechiffrable, meme avec l'etat long terme complet au moment de la capture
- **Post-compromise security :** une fois qu'une etape DH-ratchet fraiche a eu lieu apres un compromis, tous les messages suivants sont proteges de l'attaquant
- **Livraison hors ordre :** les cles de message sont mises en cache pour les messages tardifs (plafonnees a 1000 cles par session pour borner la memoire)

11.6 Conteneur de message

Chaque evenement de chat televerse sur le serveur est un texte chiffre autonome :

```
MessageContainer (encode base64, stocke dans chat_events.payload) :

   +-------------------------------------------------------------+
   | ratchet_pub      :  Cle publique KEM hybride (etape actuelle)|
   | prev_chain_len   :  Nombre de messages dans la chaine preced.|
   | msg_number       :  Compteur dans la chaine actuelle         |
   | ciphertext       :  AES-256-GCM(MK_n, plaintext || header)   |
   | tag              :  Tag d'authentification AES-256-GCM       |
   |-------------------------------------------------------------+

Le header dans le plaintext AEAD contient : login expediteur,
content_type (text/quote/file-reference), timestamp,
reply_to_event_uuid optionnel.
```

Le conteneur est opaque pour le serveur — y compris le type de message, le format du nom d'affichage de l'expediteur, et tout contexte de reponse cite. La base de donnees d'Aionda stocke le conteneur tel quel dans `chat_events.payload`.

11.7 Conversations de groupe — Sender Keys par destinataire

Pour les conversations de salon (3+ participants), AAR utilise un modele de **fan-out par destinataire**. L'expediteur derive une chaine sender-key par paire de pair et chiffre un message une fois par destinataire. Chaque destinataire recoit donc un texte chiffre personnellement adresse, dechiffrable uniquement avec les cles derivees de sa propre session AAR avec l'expediteur.

Compromis : Le cout du fan-out est `O(N)` en destinataires — acceptable pour les tailles d'equipe typiques (≤25 participants). Pour des groupes plus grands, un futur mode base sur MLS est prevu (voir Roadmap). Le modele actuel est preferable a une seule cle de groupe partagee parce que : (a) il preserve la forward secrecy par paire, (b) il ne necessite pas de rotation de cles au-dela du niveau bilateral lors du retrait d'un membre, et (c) il herite des garanties post-quantiques de l'AAR par paire sans modification.

11.8 Conteneur de conversation initial

Quand Alice demarre un nouveau salon avec Bob, Carol et Dan, le premier message a chaque participant est emballe comme un **conteneur de handshake initial** qui transporte (a) le materiel de handshake X3DH consomme contre le bundle de ce participant et (b) le premier message lui-meme. Le destinataire dispatche sur le champ discriminateur :

```
{
  "type": "initial",
  "initialMessage": { /* metadonnees du handshake, OPK id consommee */ },
  "encryptedMessage": "<base64 MessageContainer>"
}
```

Les messages suivants dans le meme salon reutilisent la session AAR etablie et ne contiennent que la partie `encryptedMessage`.

11.9 Transport — API chiffree + Mercure SSE + WebSocket

Le plan de chat utilise trois canaux orthogonaux :

Canal	Direction	Objectif	Plaintext vu par le serveur
API chiffree `/e`	Client -> Serveur	Envoi de message, recuperation d'historique, operations key-bundle (11 endpoints)	Aucun — tous les 11 endpoints chat utilisent le meme transport hybrid-KEM documente en Section 8
Mercure SSE	Serveur -> Client	Push de livraison de nouveaux evenements (`chat.event_received`, `chat.read_receipt`)	Aucun — le serveur pousse le meme MessageContainer opaque qu'il a stocke
WebSocket `/chat/ws`	Bidirectionnel	Backfill de reconnexion (`sync_request`), heartbeat, presence	Booleen de presence (en ligne/hors ligne) et chaine de login uniquement — jamais le contenu des messages

La presence est tenue dans une carte en memoire sur `aionda_chat_realtime` ; un seul endpoint HTTP intranet expose une recherche `login -> online` pour le selecteur d'equipe afin que les utilisateurs puissent voir si un pair est joignable. Aucun historique, aucun etat de lecture, aucun contenu ne traverse jamais le canal de presence.

11.10 Accuses de lecture

Les accuses de lecture sont un parametre opt-in par compte (`chat_participants.send_read_receipts`). S'ils sont actives, marquer un message comme lu publie un evenement `chat.read_receipt` contenant :

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

Notez que `last_read_event_uuid` n'est **pas** le contenu du message — c'est un identifiant alloue par le serveur deja connu de lui. Aucune information supplementaire ne fuit au-dela de « Alice a maintenant vu les messages jusqu'a l'evenement X. » Les destinataires qui desactivent les accuses de lecture (`send_read_receipts = false`) n'emettent jamais de tels evenements, et le serveur applique le toggle.

11.11 Integration de la chaine d'audit

Pour les comptes Enterprise, chaque operation de chat est ajoutee au journal d'audit infalsifiable existant (`enterprise_audit_log`, chaine de hash SHA3-256 — voir §18). Les types d'action reserves sont :

`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`.

L'enregistrement d'audit ne contient que l'UUID de conversation, le login de l'acteur et un timestamp — **jamais** le payload chiffre, les cles de ratchet ou les contenus de messages.

11.12 Ce que le serveur voit

Visible pour le serveur	Non visible
`conversation_uuid`, liste des participants (par `mail_account.name`)	Contenu des messages en clair
Payloads `MessageContainer` chiffres (textes chiffres opaques)	Cles de ratchet, cles de chaine, cles de message
Moities publiques de IK, SPK, OPK ; signatures Ed25519 sur SPK	Moities privees de toute paire de cles
Compteur et timestamp de consommation OPK	Quelle OPK a ete consommee pour quelle conversation (correlation empechee par la liaison HKDF-info uniquement cote client)
Cadence de conversation (timestamps des lignes `chat_events`)	Chaines de reponses citees, pieces jointes dans le conteneur
Booleen de presence (en ligne/hors ligne)	Indicateurs de frappe (peer-to-peer, n'atteignent jamais le serveur)
Entrees de la chaine d'audit pour les comptes Enterprise	Contenus en clair des entrees d'audit

11.13 Audits cryptographiques

Le protocole AAR a ete implemente a partir de zero en TypeScript et a subi trois audits independants assistes par IA, documentes dans `typescript/manager/chat/crypto/AUDIT.md`, `AUDIT_SECOND_OPINION.md` et `AUDIT_THIRD_OPINION.md`. Les conclusions de chaque tour ont ete integrees avant le deploiement public. Les surfaces principales auditees etaient : symetrie du handshake, reservation OPK sous concurrence, deep-cloning de l'etat immuable du ratchet, et le framing AEAD du `MessageContainer`.

11.14 Limites d'implementation

La base de code chat vit dans `typescript/manager/chat/` (~9 000 lignes de TypeScript) et `includes/classes/Api/Services/Chat/` (couche service PHP). Artefacts d'audit cles :

```
crypto/aar-types.ts            — declarations de types uniquement, aucune logique
crypto/aar-keybundle.ts        — creation, rotation, serialisation du KeyBundle
crypto/aar-x3dh.ts             — Handshake (chemins initiateur + repondeur)
crypto/aar-double-ratchet.ts   — Root chain, sending chain, receiving chain
```

La separation propre entre logique de protocole (`crypto/`) et transport/UI (`chat-store.ts`, `chat-panel.ts`, ...) garantit que le coeur cryptographique peut etre re-audite sans derive de portee UI.

---

12. Appels video Aionda — visioconference E2EE post-quantique

Aionda Mail inclut des appels video de groupe chiffres de bout en bout (suite cryptographique `AIONDA_PQ_CALL_V1`). Le relais de medias ne voit jamais l'audio ou la video en clair — il ne transfere que des trames chiffrees. Les cles de groupe sont negociees par un handshake **hybride post-quantique**.

12.1 Objectifs de securite & modele de confiance

Dans le perimetre — le protocole fournit :

- **Confidentialite** des medias audio et video, de bout en bout.
- **Authenticite** du groupe — chaque participant est cryptographiquement membre du meme groupe MLS, verifie hors-bande via la Short Authentication String (Section 12.5).
- **Forward secrecy** — les medias passes restent secrets meme si les cles actuelles sont compromises ulterieurement.
- **Post-compromise security** — le groupe recouvre la confidentialite apres le retrait d'un membre compromis et le renouvellement des cles du groupe.
- **Confidentialite post-quantique** — l'etablissement du groupe utilise le KEM hybride X-Wing.
- **Independance du serveur** — une SFU, un serveur de signalisation ou une couche de stockage malveillants ou compromis ne peuvent ni lire ni injecter de medias.

Hors perimetre (explicitement *non* defendu) :

- L'analyse de trafic — le timing des paquets, les tailles des paquets, le nombre de participants et la duree de l'appel sont observables (Section 12.9).
- La compromission du terminal (voir l'hypothese de confiance ci-dessous).
- Les logiciels malveillants au niveau du systeme d'exploitation ou la saisie physique de l'appareil.

Hypothese de confiance sur le terminal. Le modele suppose un navigateur et un systeme d'exploitation non compromis. Les trames sont scellees a l'interieur de la pipeline media du navigateur (`RTCRtpScriptTransform`), mais un terminal compromis peut lire les medias en clair *avant* le chiffrement ou *apres* le dechiffrement. Le modele de confiance de la livraison web et ses mesures d'attenuation (signature des reponses Guardian) sont couverts en Section 17 et Section 22.

12.2 Pourquoi un protocole dedie ?

La visioconference temps reel a deux exigences que les couches e-mail et chat n'ont pas :

1. **Chiffrement par trame au debit de la ligne.** Les trames audio/video doivent etre chiffrees

individuellement afin qu'une Selective Forwarding Unit (SFU) puisse les router, les supprimer et les reordonner sans jamais les dechiffrer. C'est le role de **SFrame** (RFC 9605).

2. **Negociation dynamique de cle de groupe.** Les participants rejoignent et quittent l'appel en cours. Chaque changement de composition doit renouveler la cle du groupe avec forward secrecy et post-compromise security. C'est le role de **MLS** (RFC 9420).

La SFU cote serveur (`mediasoup`) est traitee comme une **infrastructure non fiable** : c'est un routeur de paquets a connaissance nulle, exactement comme la couche de stockage pour les e-mails.

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

Le profil MLS substitue les primitives classiques d'une suite MLS standard par des hybrides post-quantiques.

Role	Algorithme	Notes
HPKE KEM	**X-Wing** (ML-KEM-768 + X25519)	Hybrid; `draft-connolly-cfrg-xwing-kem`
Signatures	**ML-DSA-65**	NIST FIPS 204 (Dilithium), Level 3
KDF	HKDF-SHA-256	RFC 5869
AEAD (HPKE + SFrame)	**AES-256-GCM**	AES-256 offre une marge confortable contre les attaques de recherche quantique connues (Grover)

Aucun downgrade. Il n'existe aucun repli purement classique. Si le handshake post-quantique ne peut pas etre mene a terme, l'appel echoue de maniere fermee (fail closed) — un downgrade est une erreur dure, jamais un affaiblissement silencieux.

Maturite de X-Wing. X-Wing est actuellement un draft IETF/CFRG (`draft-connolly-cfrg-xwing-kem`). Le profil actuel l'adopte comme KEM hybride prefere ; de futures revisions du protocole pourront migrer vers un KEM hybride finalise et standardise par le CFRG/NIST, tout en preservant inchangees les couches MLS et SFrame.

Agilite cryptographique. L'identifiant de la suite cryptographique (`0xFF01`) determine entierement le KEM, l'algorithme de signature, le KDF et l'AEAD. Une future suite `AIONDA_PQ_CALL_V2` peut remplacer n'importe quelle primitive sans modifier les couches superieures du protocole ; les suites sont selectionnees par identifiant et ne sont jamais negociees silencieusement vers le bas.

12.4 Negociation de cle de groupe (MLS, RFC 9420)

Chaque participant est represente dans l'arbre a ratchet MLS par un **KeyPackage** (cle KEM publique X-Wing + identifiant ML-DSA-65). Le groupe evolue a travers des *epochs* :

- **Welcome / Commit** — un membre rejoignant est ajoute avec un secret de chemin encapsule par X-Wing ; l'arbre a ratchet avance vers une nouvelle epoch.
- **Renouvellement de cle a chaque changement de composition.** Tout changement — une arrivee, un depart, un remplacement d'appareil ou une rotation de cle d'identite —

fait avancer le groupe vers une nouvelle epoch. C'est ce qui assure la forward secrecy (un nouveau participant ne peut lire les medias passes) et la post-compromise security (un membre retire ne peut lire les medias futurs).

- **Remove-Commit.** Lorsqu'un participant quitte (ou qu'un pair perime/zombie est detecte), le proprietaire emet un Remove-Commit qui renouvelle la cle du groupe, de sorte qu'un membre parti ne puisse pas dechiffrer les trames ulterieures meme s'il a conserve l'ancien materiel de cle.

La construction hybride est concue de sorte que la compromission d'un **seul** composant — ML-KEM-768 *ou* X25519 — ne revele pas a elle seule le secret de groupe.

12.5 Identite & authentification

- **Identifiants.** Chaque membre porte un **Credential** MLS qui lie une identite de participant a sa cle de signature ML-DSA-65. Chaque message de handshake MLS (Welcome / Commit / Update) est signe avec cette cle, de sorte que la SFU ou le serveur de signalisation ne peut pas falsifier les operations de groupe.
- **Short Authentication String (SAS).** Le lien de la salle transporte un **secret pre-partage** a haute entropie dans le fragment d'URL, qui n'est jamais envoye au serveur en clair. L'exportateur du groupe MLS et ce secret sont melanges ensemble dans une Short Authentication String, dont le client **epingle** l'empreinte. Une divergence — par exemple un serveur de signalisation ou une SFU tentant d'inserer un participant malveillant — declenche un **abandon dur sans repli souple**. Cela lie le groupe cryptographique au secret de salle hors-bande et fait echec a un man-in-the-middle cote serveur.
- **Facteur d'authentification.** Pour une salle invite, *la possession du lien de salle equivaut a la possession de l'identifiant de salle* — le secret de salle est le seul facteur d'authentification. Les participants Aionda authentifies echangent en outre un jeton de signalisation signe et limite au compte (Section 12.8).

12.6 Chiffrement des medias (SFrame, RFC 9605)

Chaque piste media est chiffree trame par trame **avant** d'atteindre la SFU. Le cycle de vie de la cle :

```
MLS group secret
      |  MLS Exporter (RFC 9420 §8)
      v
SFrame base key
      |  HKDF(base_key, KID = sender || epoch)
      v
per-sender key
      |  AES-256-GCM(nonce = frame counter, aad = sframe_header)
      v
per-frame ciphertext || tag      ->  wire = sframe_header || ciphertext || tag
```

- La cle SFrame est derivee de l'**exportateur MLS**, donc elle n'existe que sur les membres authentifies du groupe — jamais sur le serveur.
- Le chiffrement s'execute dans un Worker `RTCRtpScriptTransform` dedie (l'API standardisee Encoded-Transform, disponible dans Chromium et Safari/WebKit), de sorte que les trames sont scellees sur la pipeline media du navigateur.
- Le codec est VP8 ; la SFU effectue le routage des keyframes uniquement sur les metadonnees — elle n'inspecte jamais la charge utile chiffree.

12.7 Protection contre le rejeu

Chaque trame chiffree porte un **compteur** SFrame monotone dans son en-tete authentifie, et est liee a l'epoch MLS courante. Les recepteurs suivent le compteur le plus eleve vu par emetteur (un magasin de compteurs cote reception persistant). Une trame avec un compteur duplique ou regresse, ou liee a une epoch perimee, est rejetee — un texte chiffre capture ne peut pas etre rejoue dans la session.

12.8 Signalisation et modele d'acces

- **La signalisation** s'effectue sur un WebSocket authentifie (`aionda_signaling`). Chaque session est protegee par un **JWT** a courte duree de vie, emis par le serveur applicatif apres que le client a echange le secret de la salle via l'API chiffree `/e` (voir Section 8). Le jeton est symetrique (HS256) et n'est valide qu'entre le serveur applicatif et l'unique service de signalisation ; il autorise l'etablissement de la session et ne transporte jamais de medias ni de secrets de groupe. Le serveur de signalisation relaie les messages MLS Welcome/Commit mais ne peut pas les lire.
- **Lien de salle = justificatif au porteur.** Quiconque detient le lien complet (UUID de salle + secret du fragment) peut rejoindre — le modele est intentionnellement « celui qui a le lien ». Les identifiants de base de donnees ne sont jamais exposes ; les salles sont adressees uniquement par des UUID non devinables.
- **Controle par le proprietaire.** Le proprietaire de la salle (verifie cote serveur par la propriete du compte) peut marquer une salle comme *terminee* pour bloquer les futures connexions, et peut retirer un participant present via un MLS Remove-Commit (Section 12.4).

12.9 Ce que la SFU / le serveur voit — et les limites des metadonnees

L'infrastructure PEUT voir	L'infrastructure NE PEUT PAS voir
Tailles des trames chiffrees, timing, metadonnees de routage	Le contenu audio ou video (les trames sont scellees par AES-256-GCM)
Texte chiffre du handshake MLS (Welcome/Commit)	Les secrets de groupe ou les cles SFrame (derivees cote client depuis l'exportateur MLS)
UUID de salle, nombre de participants, sujet du JWT	Le secret du lien de salle (reside dans le fragment d'URL)
Candidats ICE / adresses de transport	Tout texte en clair qui lui permettrait de rejoindre ou dechiffrer la session

Metadonnees que le systeme ne cache pas intentionnellement : les adresses IP exposees a l'infrastructure ICE / STUN / TURN, le timing des paquets, les tailles des paquets, le nombre de participants et la duree de l'appel. La resistance a l'analyse de trafic n'est pas un objectif de conception (Section 12.1).

12.10 Proprietes de securite (resume)

Propriete	Garantie
Confidentialite	Seuls les membres actuels du groupe MLS possedent les cles SFrame.

Propriete	Garantie
Authentification	Tous les messages MLS sont signes avec ML-DSA-65 ; le groupe est lie au secret de salle via la SAS epinglee.
Forward secrecy	La compromission des secrets actuels ne revele pas les medias passes.
Post-compromise security	Le renouvellement des cles restaure la confidentialite apres le retrait d'un membre.
Independance du serveur	La compromission du serveur de signalisation, de la SFU ou du stockage n'expose pas les medias.
Post-quantique	L'etablissement du groupe repose sur le KEM hybride X-Wing (ML-KEM-768 + X25519).

Ces proprietes decrivent la *conception* du protocole. Un audit cryptographique independant par un tiers du sous-systeme d'appel est prevu ; tant qu'il n'est pas publie, les garanties ci-dessus constituent une auto-evaluation.

---

13. Protections contre les canaux auxiliaires

13.1 Bucket Padding

Problematique : les tailles des e-mails chiffres peuvent reveler des informations. Un attaquant observant les longueurs de texte chiffre pourrait en deduire le contenu (par exemple, un e-mail de 50 octets est probablement « OK, merci » tandis qu'un e-mail de 500 Ko contient des pieces jointes).

Solution : toutes les donnees sont remplies jusqu'a des « buckets » de taille fixe avant le chiffrement :

```
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]
```

Exemple : un e-mail de 523 octets est rempli jusqu'a 1 024 octets. Un observateur ne voit qu'un « e-mail de 1 Ko » — pas la taille reelle de 523 octets.

13.2 Compression avant chiffrement

Les donnees sont compressees avec gzip (niveau 6) **avant** le chiffrement. C'est le seul ordre correct :

- La compression apres le chiffrement echouerait (les donnees chiffrees ont une entropie maximale)
- Le Bucket Padding apres la compression empeche les attaques de type CRIME/BREACH qui exploitent les taux de compression

13.3 Confidentialite du regroupement

Les fils d'e-mails utilisent des hachages SHA-256 des en-tetes Message-ID au lieu d'identifiants en clair. Le serveur peut regrouper les e-mails lies par egalite de hachage sans connaitre les identifiants de messages reels.

---

14. Gestion et cycle de vie des cles

14.1 Hierarchie des cles

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

14.2 Stockage des cles

Cle	Emplacement de stockage	Protection
Cle maitre	Nulle part (reconstruite a la demande a partir des parts Shamir)	Shamir 2-of-3
Cles privees du coffre-fort	Serveur (chiffrees)	AES-256-GCM avec cle maitre
Cles publiques du coffre-fort	Serveur (en clair)	Non sensibles — publiques par definition
Cles ephemeres d'e-mail	Serveur (encapsulees)	Encapsulation Hybrid KEM
Enregistrements OPAQUE	Serveur (chiffres au repos)	AES-256-GCM avec cle serveur

Cle	Emplacement de stockage	Protection
Parts Shamir chiffrees	Serveur	XOR avec les cles derivees du mot de passe/passkey/recuperation
Cles de transport API	Serveur (pool pre-genere)	Usage unique, duree de vie 24h

14.3 Empreintes de cles

Chaque paire de cles du coffre-fort possede une empreinte SHA-256 stockee sur le serveur. Cela permet :

- Une piste d'audit des rotations de cles
- La detection de changements de cles non autorises
- La verification de l'integrite des cles cote client

---

15. Mecanisme de recuperation

15.1 Cle de recuperation (mnemonique BIP39)

Lors de la configuration du coffre-fort, l'utilisateur recoit une **phrase de recuperation de 24 mots** generee a partir de 256 bits d'entropie, encodee selon le standard BIP39 :

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

15.2 Derivation de la cle de recuperation

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

15.3 Ce que le serveur stocke

Le serveur ne stocke **que le hachage SHA3-256** d'une cle de verification derivee. Il ne stocke pas :

- Les mots de recuperation
- L'entropie
- La cle de verification elle-meme

15.4 Processus de recuperation

1. L'utilisateur saisit la phrase de recuperation de 24 mots

2. Le client derive l'entropie -> HKDF-SHA3-256 -> SHA3-256 -> hachage de verification
3. Le client envoie le hachage de verification au serveur (jamais la cle en clair)
4. Le serveur compare avec le hachage stocke
5. En cas de correspondance : toutes les methodes 2FA sont desactivees, l'utilisateur configure une nouvelle authentification
6. La cle de recuperation est revoquee apres un seul usage

15.5 Limitation de debit

- Maximum 3 tentatives de verification par heure
- Verrouillage de 60 minutes apres depassement de la limite
- Usage unique : la cle de recuperation est definitivement revoquee apres une utilisation reussie

15.6 Pas de recuperation de mot de passe

Il n'y a pas de reinitialisation du mot de passe par e-mail. Si un utilisateur perd son mot de passe ET tous les autres facteurs d'authentification (passkey + cle de recuperation), ses donnees sont definitivement inaccessibles. C'est une decision de conception intentionnelle qui prouve l'integrite du modele Zero-Knowledge — si le serveur pouvait recuperer les donnees, il pourrait aussi les lire.

---

16. Authentification sans mot de passe (Passkeys)

16.1 Extension WebAuthn PRF

Aionda Mail prend en charge les passkeys FIDO2 (cles de securite materielles, authentificateurs biometriques) pour la connexion sans mot de passe et le deverrouillage du coffre-fort.

L'**extension WebAuthn PRF (Pseudo-Random Function)** fournit une sortie deterministe de 32 octets liee a la passkey et au credential specifiques. Cette sortie est utilisee pour proteger la Part Shamir 2.

16.2 Fonctionnement

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

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

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

16.3 Passkeys multiples

Les utilisateurs peuvent enregistrer plusieurs passkeys (par exemple, Touch ID MacBook, Face ID iPhone, YubiKey). Chaque passkey protege independamment sa propre copie de la Part 2. Une seule passkey combinee au mot de passe suffit a deverrouiller le coffre-fort.

---

17. Guardian : protection MITM & signature des reponses

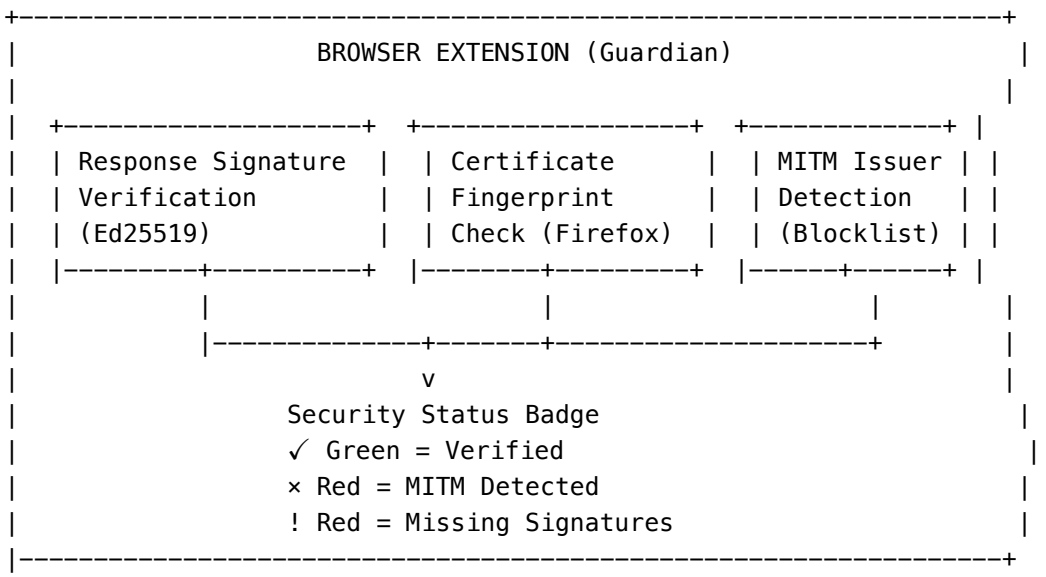
17.1 Le probleme des applications web

Toute application web a un probleme de confiance inherent : le navigateur telecharge du JavaScript depuis le serveur a chaque visite. Un attaquant de type « homme du milieu » (MITM) — qu'il s'agisse d'un CDN compromis, d'un proxy d'entreprise ou d'un FAI malveillant — pourrait theoriquement injecter du code modifie qui exfiltre les cles de chiffrement.

Aionda Mail repond a ce probleme avec le **module Guardian**, un composant d'extension de navigateur (disponible pour Chrome et Firefox) qui verifie independamment l'integrite du serveur.

17.2 Vue d'ensemble de l'architecture

Le module Guardian fonctionne au sein du Service Worker de l'extension de navigateur — completement independant du JavaScript de l'application web. Il effectue trois types de verification :



17.3 Verification de la signature des reponses (Ed25519)

Chaque reponse API du serveur d'Aionda Mail est signee cryptographiquement a l'aide d'**Ed25519** (Edwards-curve Digital Signature Algorithm).

Processus de signature (cote serveur) :

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>
```

Processus de verification (extension de navigateur) :

```
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
```

Proprietes essentielles :

- **Protection contre la relecture :** la fenetre de 5 minutes sur l'horodatage empeche la relecture d'anciennes reponses
- **Detection d'alteration :** toute modification du corps de la reponse invalide la signature
- **Isolation des cles :** les cles publiques sont integrees dans l'extension (non telechargees depuis le serveur)
- **Separation des environnements :** les cles de developpement (`dev-2026-01`) ne peuvent pas etre utilisees sur les URL de production et vice versa

17.4 Gestion des cles publiques Ed25519

Les cles publiques sont distribuees avec l'extension de navigateur dans `public_key.json` :

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

- **Format de cle :** SPKI DER (Subject Public Key Info, Distinguished Encoding Rules)
- **Taille de cle :** 32 octets (cle publique Ed25519 de 256 bits)
- **Taille de signature :** 64 octets (fixe)
- **Rotation :** de nouvelles cles sont ajoutees avant l'expiration des anciennes ; les mises a jour de l'extension livrent les nouvelles cles
- **Aucune confiance envers le serveur :** les cles sont integrees dans le binaire de l'extension, non recuperees depuis le serveur

17.5 Verification des certificats TLS (Firefox)

Sur Firefox, le module Guardian effectue une verification supplementaire des certificats TLS a l'aide de l'API `browser.webRequest.getSecurityInfo()` (non disponible sur Chrome en raison des limitations de Manifest V3).

Flux de verification :

```
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
```

Pourquoi la validation basee sur l'emetteur plutot que le pinning ? CloudFlare (utilise comme CDN) effectue une rotation des certificats feuille entre les serveurs de bordure. Le pinning de certificat traditionnel (correspondance exacte des empreintes) provoquerait des faux positifs. La validation basee sur l'emetteur est plus robuste : l'autorite de certification emettrice est stable meme lorsque les certificats feuille changent.

17.6 Recuperation automatique du certificat (anti-usurpation)

Le point d'entree de certificat du serveur utilise une technique anti-usurpation ingenieuse :

```
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
```

Le serveur demande essentiellement « quel certificat le monde exterieur voit-il pour mon domaine ? » — et signe la reponse pour que l'extension puisse lui faire confiance.

17.7 Indicateurs d'etat de securite

L'extension affiche un badge dans la barre d'outils du navigateur :

Badge	Couleur	Signification
✓	Vert	Toutes les reponses verifiees — signatures valides
!	Orange	Utilisation d'une cle de signature obsolete (rotation en attente)

Badge	Couleur	Signification
×	Rouge	MITM detecte — echec de la verification de signature
!	Rouge	Signatures manquantes — reponses non signees
[Shield]	Bleu	Mode protege — aucune verification effectuee pour l'instant

17.8 Couverture des menaces

Attaque	Methode de detection	Navigateur
Proxy MITM d'entreprise (ZScaler, Fortinet)	Liste de blocage des emetteurs de certificats	Firefox
Reponses API modifiees	Verification de signature Ed25519	Chrome + Firefox
Attaques par relecture	Fenetre d'horodatage de 5 minutes	Chrome + Firefox
Compromission du CDN (CloudFlare)	Discordance de la signature de reponse	Chrome + Firefox
Substitution de certificat	Comparaison de l'emetteur + auto-verification du serveur	Firefox
Confusion cles dev/prod	Identifiants de cles lies a l'environnement	Chrome + Firefox

17.9 Limites

- **Chrome Manifest V3 :** impossible d'inspecter les certificats TLS — seule la verification de signature de reponse est disponible
- **Extension requise :** les utilisateurs sans l'extension ne beneficient pas des protections Guardian
- **Ed25519 n'est pas post-quantique :** la verification de signature utilise la cryptographie classique. Un ordinateur quantique suffisamment puissant pourrait theoriquement forger des signatures Ed25519.

---

18. Archive e-mail d'entreprise (Blockchain)

18.1 Vue d'ensemble

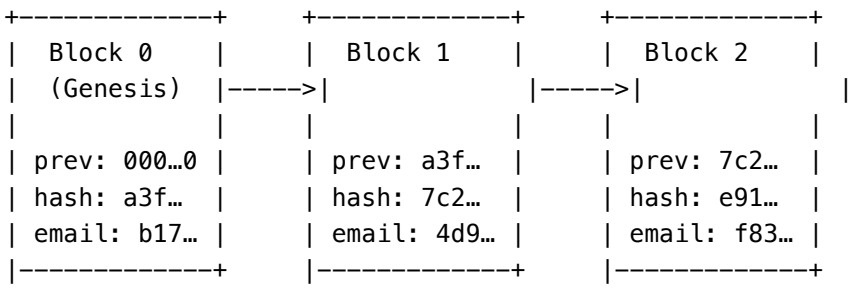
Le plan Entreprise d'Aionda Mail inclut une **archive e-mail conforme GoBD** securisee par une chaine de hachage cryptographique (blockchain). Chaque e-mail archive devient un bloc immuable dans une chaine propre a l'entreprise. Toute alteration — modification, suppression ou insertion de blocs — est detectee cryptographiquement.

L'archive combine deux couches de securite independantes :

1. **Chaine de hachage (SHA3-256) :** garantit l'integrite et l'immutabilite — prouve qu'aucun e-mail n'a ete modifie ou supprime apres l'archivage
2. **Chiffrement Hybrid KEM (CAK) :** garantit la confidentialite — le serveur ne peut pas lire le contenu des e-mails archives

18.2 Architecture de la chaine de hachage

Chaque e-mail archive devient un bloc dans une chaine sequentielle et inviolable :



Calcul du hachage de bloc :

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

Proprietes :

- **Algorithme de hachage :** SHA3-256 (NIST FIPS 202)
- **Bloc de genese :** `prev_block_hash` = 64 zeros, `block_number` = 0
- **Numerotation sequentielle :** imposee par la contrainte `UNIQUE KEY (company_uuid, block_number)` en base de donnees
- **Une chaine par entreprise :** isolation complete entre les entreprises
- **Hachage de l'e-mail :** `SHA3-256(sender || recipient || timestamp || size)` — preuve d'integrite des donnees originales de l'e-mail

18.3 Detection d'alteration

L'algorithme de verification de la chaine detecte toute forme d'alteration :

```
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
```

Tentative d'alteration	Detection
Modifier le contenu de l'e-mail	`email_hash` change -> echec du recalcul de `block_hash`
Modifier les metadonnees (expediteur, domaine, horodatage)	Incluses dans l'entree du hachage -> discordance de `block_hash`
Supprimer un bloc	Le `prev_block_hash` du bloc suivant devient orphelin
Inserer un bloc	Rompt le `block_number` sequentiel + la chaine `prev_block_hash`
Reordonner les blocs	La contrainte `UNIQUE KEY` + la verification sequentielle l'empechent
Remplacer toute la chaine	Le hachage du bloc de genese differerait de toute sauvegarde externe

Le resultat de verification indique le numero exact du bloc ou l'alteration a ete detectee, avec les valeurs de hachage attendues et reelles pour l'analyse forensique.

18.4 Cle d'archive d'entreprise (CAK) — Chiffrement Zero-Knowledge

Le contenu de l'archive est chiffre de bout en bout a l'aide d'une **cle d'archive d'entreprise** — une paire de cles Hybrid KEM (X25519 + ML-KEM-1024) generee cote client par le proprietaire de l'entreprise.

```
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`

Distribution de la cle aux autres employes (Administrateur, Responsable conformite) :

1. Le proprietaire dechiffre les cles privees CAK avec son mot de passe
2. Le proprietaire re-encapsule les cles privees avec la cle derivee du mot de passe de l'employe cible
3. Le serveur stocke la copie re-encapsulee sur l'enregistrement de l'employe
4. Chaque employe autorise dispose de sa propre copie encapsulee independamment

Le serveur ne voit jamais les cles privees CAK en clair.

18.5 Ce qui est chiffre

Lorsqu'un e-mail est archive, deux couches de chiffrement sont appliquees :

Metadonnees chiffrees (AES-256-GCM avec 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"
}
```

Contenu de l'e-mail chiffre (encapsulation Hybrid KEM separee) :

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

Application Zero-Knowledge : apres le chiffrement, les champs de metadonnees en clair dans la base de donnees (sender_address, recipient_address, domaines) sont remplaces par leurs hachages SHA3-256. Le serveur ne stocke que des hachages — les valeurs originales n'existent que dans les blobs chiffres.

18.6 Piste d'audit

Chaque action sur l'archive est journalisee dans une **chaine d'audit independante** (egalement enchainee par hachages SHA3-256) :

Action	Quand elle est journalisee
`EMAIL_RECEIVED` / `EMAIL_SENT` / `DRAFT_ARCHIVED`	E-mail archive
`VIEW_EMAIL` / `VIEW_ATTACHMENT`	Un employe lit un e-mail archive

Action	Quand elle est journalisee
`SEARCH_ARCHIVE`	Recherche effectuee
`EXPORT_EMAIL` / `EXPORT_REPORT`	Donnees exportees
`VERIFY_CHAIN` / `CHAIN_VERIFIED_OK` / `CHAIN_VERIFIED_BROKEN`	Verification d'integrite
`LEGAL_HOLD_SET` / `LEGAL_HOLD_RELEASED`	Mise en suspens juridique basculee
`ARCHIVE_DECRYPT`	CAK utilisee pour dechiffrer le contenu
`ADMIN_ACCESS`	Action administrative

Chaque entree d'audit enregistre : l'acteur (UUID + role), l'adresse IP, l'identifiant de session, le hachage de l'e-mail cible et si la chaine etait valide au moment de l'acces.

18.7 Mise en suspens juridique & retention

- **Duree de retention :** configurable par entreprise (par defaut : 10 ans), calculee par e-mail comme `archived_at + retention_years`
- **Mise en suspens juridique :** des e-mails individuels peuvent etre places sous mise en suspens juridique, empechant leur suppression jusqu'a la levee. Inclut le motif, l'acteur et l'horodatage.
- **Conformite GoBD :** la combinaison d'une chaine de hachage immuable, d'une piste d'audit complete, d'une retention configurable et d'une mise en suspens juridique satisfait les exigences du GoBD allemand (Grundsaetze zur ordnungsmaessigen Fuehrung und Aufbewahrung von Buechern, Aufzeichnungen und Unterlagen in elektronischer Form sowie zum Datenzugriff).

18.8 Export forensique

Les utilisateurs autorises (Proprietaire, Administrateur) peuvent exporter l'etat complet de la chaine pour une verification independante :

- Donnees completes de la chaine avec tous les hachages de blocs
- Resultat de verification (valide/rompu, numero de bloc rompu le cas echeant)
- Valeurs de hachage attendues et reelles pour l'analyse forensique
- 50 dernieres entrees du journal d'audit
- Format JSON pour une re-verification externe avec toute implementation SHA3-256

---

19. Ce que le serveur voit — et ce qu'il ne voit pas

Cette section documente explicitement la frontiere Zero-Knowledge.

19.1 Le serveur PEUT voir

Donnees	Raison de la visibilite	Mesure d'attenuation
Adresse IP	Exigence TCP/IP	Utilisation d'un VPN/Tor possible si souhaite
Horodatages	Heure de reception de l'e-mail	Inherent au protocole e-mail
Blobs d'e-mails chiffres	Stockes pour la recuperation	Chiffres avec AES-256-GCM, cle inconnue du serveur
Tailles de texte chiffre rembourrees	Exigence de stockage	Le Bucket Padding masque les tailles reelles
Adresse DEA du destinataire	Exigence de routage	La DEA est jetable, ce n'est pas l'adresse reelle
Existence du compte	Flux d'authentification	Protection contre l'enumeration d'utilisateurs deployee
Cles publiques	Requises pour le chiffrement par le serveur	Publiques par definition, non sensibles
Parts Shamir chiffrees	Stockage pour l'utilisateur	XOR avec des cles que le serveur ne connait pas
Enregistrements OPAQUE	Protocole d'authentification	Pas des hachages de mots de passe, chiffres au repos

19.2 Le serveur NE PEUT PAS voir

Donnees	Raison de l'invisibilite
Contenu de l'e-mail (objet, corps, en-tetes)	Chiffre avec des cles ephemeres encapsulees via Hybrid KEM
Mot de passe de l'utilisateur	OPAQUE — le mot de passe n'est jamais transmis
Cle maitre	Reconstruite uniquement dans le navigateur a partir des parts Shamir
Cles privees du coffre-fort	Chiffrees avec la cle maitre avant le stockage
Cles ephemeres d'e-mail	Encapsulees avec Hybrid KEM, le serveur n'a pas les cles privees
Cle de recuperation / mnemonique	Seul le hachage SHA3-256 de la cle derivee est stocke
Sorties PRF de la passkey	Liees au materiel, ne quittent jamais l'authentificateur
Noms de dossiers	Chiffres avec des cles specifiques aux dossiers
Signatures d'e-mail	Chiffrees avec la cle maitre
Contenu des requetes API	Chiffre via la couche de transport /e
Contenu des reponses API	Chiffre avant la transmission

19.3 Garantie cryptographique

Meme avec un acces complet a :

- La base de donnees complete
- Tout le trafic reseau
- Le code source et la configuration du serveur
- Tous les enregistrements OPAQUE et les cles du serveur

…un attaquant **ne peut pas** dechiffrer un seul e-mail sans le mot de passe de l'utilisateur (ou la passkey + la cle de recuperation). Ce n'est pas une politique — c'est une impossibilite mathematique imposee par la conception cryptographique.

---

20. Reference des algorithmes

20.1 Tableau complet des algorithmes

Composant	Algorithme	Parametres	Standard
Authentification par mot de passe	OPAQUE	RFC 9807, aPAKE	RFC 9807
Derivation de cle de mot de passe	PBKDF2-SHA256	600 000 iterations, sel de 32 o, sortie de 32 o	NIST SP 800-132
Chiffrement du coffre-fort	AES-256-GCM	Cle de 256 bits, nonce de 96 bits, tag de 128 bits	NIST SP 800-38D
Echange de cles classique	X25519	Curve25519, 256 bits	RFC 7748
KEM post-quantique	ML-KEM-1024	Kyber-1024, NIST Level 5	NIST FIPS 203
Derivation de cle hybride	HKDF-SHA256	IKM de 64 o, info="trashmail-hybrid-kem-v1"	RFC 5869
Partage de secret	Shamir SSS	k=2, n=3, GF(2^8)	Shamir (1979)
Encodage de la cle de recuperation	BIP39	Entropie de 256 bits, 24 mots	BIP-0039
Derivation de la cle de recuperation	HKDF-SHA3-256	Sel lie au compte	NIST FIPS 202
Verification de la cle de recuperation	SHA3-256	Sortie de 32 octets	NIST FIPS 202
Chiffrement des enregistrements OPAQUE	AES-256-GCM	Chiffrement au repos cote serveur	NIST SP 800-38D
Deverrouillage du coffre-fort par passkey	WebAuthn PRF	Base sur HMAC, lie au materiel	WebAuthn Level 2
Compression	gzip	Niveau 6	RFC 1952
Bucket Padding	Personnalise	17 tailles (256 o–16 Mo), magic 0xDEAD	—

Composant	Algorithme	Parametres	Standard
Signature des reponses	Ed25519	Cle de 256 bits, signature de 512 bits	RFC 8032
Chaine de hachage de l'archive	SHA3-256	Hachage par bloc, enchainement sequentiel	NIST FIPS 202
Encapsulation de cle d'archive (CAK)	HKDF-SHA256 + AES-256-GCM	Cle d'encapsulation derivee du mot de passe	RFC 5869 / NIST SP 800-38D
Verification de certificat	SHA-256	Comparaison d'empreinte de certificat TLS	—
Regroupement d'e-mails	SHA-256	Hachage du Message-ID	NIST FIPS 180-4
Cles de groupe d'appel video	MLS + X-Wing	Suite `AIONDA_PQ_CALL_V1` (0xFF01), ML-KEM-768 + X25519	RFC 9420 / draft-connolly-cfrg-xwing-kem
Signatures d'appel video	ML-DSA-65	Dilithium, NIST Level 3	NIST FIPS 204
Medias d'appel video	SFrame (AES-256-GCM)	Par trame, cle issue de l'exportateur MLS	RFC 9605

20.2 Niveaux de securite

Algorithme	Securite classique	Securite post-quantique
X25519	128 bits	Casse par l'algorithme de Shor
ML-KEM-1024	Equivalent 256 bits	NIST Level 5 (≈AES-256)
AES-256-GCM	256 bits	128 bits (algorithme de Grover)
SHA-256	256 bits	128 bits (algorithme de Grover)
SHA3-256	256 bits	128 bits (algorithme de Grover)
Hybrid KEM (combine)	128 bits (borne X25519)	Level 5 (borne ML-KEM)

---

21. Comparaison avec d'autres fournisseurs

Fonctionnalite	Aionda Mail	Tuta Mail	Proton Mail
Pays	Allemagne (Stuttgart)	Allemagne (Hanovre)	Suisse
Zero-Knowledge	Oui (OPAQUE + cryptographie cote client)	Oui	Oui
Post-quantique	Oui (ML-KEM-1024 + X25519 hybride)	Oui (base sur Kyber)	En developpement

Fonctionnalite	Aionda Mail	Tuta Mail	Proton Mail
Protocole de mot de passe	OPAQUE (RFC 9807) — le mot de passe ne quitte jamais le navigateur	bcrypt (mot de passe envoye au serveur via TLS)	Base sur SRP
Objet chiffre	Oui	Oui	Non
En-tetes chiffres	Oui	Partiel	Non
Noms de contacts chiffres	Oui (dans le coffre-fort)	Oui	Non
Adresses e-mail jetables	Oui (fonctionnalite principale, illimitees pour Plus)	Non	Oui (via SimpleLogin)
Extension de navigateur	Oui (Chrome + Firefox)	Non	Via SimpleLogin
Partage de dossiers	Oui (Hybrid KEM par destinataire)	Limite	Oui
Client open source	Non	Oui	Oui
Audit de securite	Prevu	Oui	Oui
Recuperation de mot de passe	Non (par conception)	Non (par conception)	Non (par conception)
Support des passkeys	Oui (FIDO2 + PRF)	Oui	Oui
Support PGP	Oui (entrant + sortant)	Non (protocole propre)	Oui (OpenPGP)
Archive e-mail conforme GoBD	Oui (chaine de hachage SHA3-256 + Hybrid KEM)	Non	Non
Detection MITM (ext. navigateur)	Oui (signatures Ed25519 + verification TLS)	Non	Non
Perfect Forward Secrecy (API)	Oui (cles ephemeres par requete)	Inconnu	Inconnu
Masquage de la taille des e-mails	Oui (Bucket Padding)	Inconnu	Non

---

22. Limites & transparence

22.1 Modele de confiance des applications web

Aionda Mail est une application web. A chaque chargement de page, le navigateur telecharge du JavaScript depuis les serveurs d'Aionda. Un attaquant sophistique qui compromettrait les serveurs pourrait theoriquement servir du JavaScript modifie qui exfiltre les cles.

Mesures d'attenuation actuelles :

- Hachages d'integrite de sous-ressources (SRI) sur toutes les balises script
- En-tetes Content Security Policy (CSP) restreignant les sources de scripts
- Tout le code cryptographique critique est inclus dans le bundle principal de l'application

- **Extension de navigateur Guardian** (Section 17) : la verification des signatures Ed25519 sur toutes les reponses API detecte les alterations cote serveur ; la verification des certificats TLS (Firefox) detecte les proxys MITM

Mesures d'attenuation prevues :

- Mise en cache par Service Worker pour le fonctionnement hors ligne (reduit la frequence de confiance au chargement)

22.2 Visibilite des metadonnees

Bien que le contenu des e-mails soit entierement chiffre, certaines metadonnees sont visibles par le serveur :

- Quand les e-mails ont ete recus (horodatages)
- Quelle adresse DEA a recu l'e-mail
- Taille approximative de l'e-mail (dans les limites des buckets)
- Schemas d'activite du compte

22.3 Journal de traitement des e-mails

A des fins de diagnostic, Aionda Mail inclut un **journal de traitement des e-mails** optionnel qui peut stocker temporairement le contenu brut des e-mails entrants. Cette fonctionnalite est configurable par adresse e-mail jetable (DEA) et peut etre activee ou desactivee dans les parametres de la DEA ("Journaliser le contenu des e-mails").

Lorsqu'il est active (par DEA) :

- Le message SMTP brut complet (en-tetes + corps) est stocke en clair sur le serveur
- Suppression automatique apres une courte periode de retention (moins de 7 jours)
- Accessible uniquement au proprietaire du compte via l'API authentifiee
- Objectif : depannage des problemes de livraison, verification du transfert, examen des decisions de filtrage anti-spam

Lorsqu'il est desactive :

- Aucun contenu d'e-mail n'est stocke dans le journal de traitement
- Seules les metadonnees sont journalisees (adresse de l'expediteur, horodatage, statut de livraison)
- Le chiffrement du coffre-fort reste le seul mecanisme de stockage

Important : Ce journal de traitement est independant du coffre-fort chiffre. Les e-mails stockes dans le coffre-fort sont toujours chiffres avec Hybrid KEM, quel que soit le parametre de journalisation. Le journal de traitement existe en tant que fonctionnalite heritee du systeme de transfert d'e-mails et offre une transparence operationnelle. Les utilisateurs necessitant un stockage strictement Zero-Knowledge pour tous les e-mails doivent desactiver cette option.

22.4 Securite des e-mails externes

Les e-mails envoyes ou recus depuis des adresses non-Aionda transitent par l'infrastructure de messagerie standard (SMTP). Bien qu'ils soient stockes chiffres dans le coffre-fort, le contenu de l'e-mail etait visible durant le transit, sauf si le chiffrement PGP etait utilise.

22.5 Pas de sequestre de cles

Il n'existe pas de cle maitre, de porte derobee ou de mecanisme de recuperation disponible pour Aionda GmbH. Si un utilisateur perd son mot de passe et toutes les autres methodes de recuperation, ses donnees sont definitivement perdues. C'est une decision de conception intentionnelle qui prouve l'integrite du modele Zero-Knowledge.

---

23. Feuille de route

Jalon	Statut	Objectif
Architecture Zero-Knowledge	Termine	—
Hybrid KEM post-quantique (ML-KEM-1024)	Termine	—
Authentification OPAQUE (RFC 9807)	Termine	—
Shamir Secret Sharing (2-of-3)	Termine	—
Couche de transport API chiffree	Termine	—
Support Passkey/WebAuthn PRF	Termine	—
Calendrier chiffre de bout en bout	Termine	—
Archive e-mail conforme GoBD (Blockchain)	Termine	—
Protection MITM Guardian (Ed25519)	Termine	—
Verification des certificats TLS (Firefox)	Termine	—

Historique du document

Version	Date	Modifications
1.0	Mars 2026	Publication initiale
1.1	Avril 2026	Nouveau chapitre 10 : Vault Drive (architecture a cle par fichier, partage via re-enveloppement de cle)
1.2	Avril 2026	Section 10.11 : Partage externe — liens publics avec enveloppes KEM hybrides, flow unlock_token, mode mot de passe Argon2id, fragment URL pour link_only, chaine d'audit infalsifiable (EXT_SHARE_*) et distribution du lien choisie par le proprietaire

Version	Date	Modifications
1.3	Mai 2026	Nouveau chapitre 11 : Aionda Chat — messagerie temps reel E2EE post-quantique via AAR (Aionda Async Ratchet), une variante hybride X25519 + ML-KEM-1024 du Signal X3DH + Double Ratchet. Forward secrecy par message, post-compromise security et integration a la chaine d'audit Enterprise. Chapitres suivants renumerotes 12-22. Traductions PT-BR et PT-PT ajoutees. Bug de double numerotation dans le rendu PDF corrige (numerotation auto LaTeX desactivee ; les numeros de chapitres proviennent uniquement des en-tetes).
1.4	juin 2026	Nouveau chapitre 12 : Appels video Aionda — visioconference de groupe E2EE post-quantique. Accord de cles de groupe MLS (RFC 9420) avec la suite cryptographique `AIONDA_PQ_CALL_V1` (X-Wing = ML-KEM-768 + X25519, signatures ML-DSA-65), chiffrement des medias image par image AES-256-GCM SFrame (RFC 9605) avec cle derivee de l'exportateur MLS, liaison SAS, et un relais SFU a connaissance nulle. Chapitres suivants renumerotes 13-23.

Contact

Aionda GmbH Stephan Ferraro Stuttgart, Allemagne

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

Ce document decrit l'architecture de securite d'Aionda Mail en date de juin 2026. Les systemes cryptographiques evoluent — ce document sera mis a jour au fur et a mesure que l'architecture change.