TECHNISCHES WHITEPAPER

uOS

Ein post-quantensicherer, Ende-zu-Ende-verschlüsselter Arbeitsbereich für Menschen und Organisationen

Version 4.22 · Oktober 2026 · Alpha

UltimaOS

Inhalt

Zusammenfassung für Entscheider 3

TEIL I — WARUM UOS

1. Das Problem: Vertrauen kraft Richtlinie 4
2. Der Ansatz von uOS 4
3. uOS im Überblick 4

TEIL II — ARCHITEKTUR UND KRYPTOGRAFIE

4. Systemarchitektur 6
5. Identität und Konten 7
6. Kryptografie 13
7. Schutz von Inhalten 15
8. Privatsphäre beim Nachrichtenaustausch 16
9. Was geschützt ist — pro Anwendung 18

TEIL III — ORGANISATIONEN, WIEDERHERSTELLUNG UND KI

10. Organisationen 21
11. Wiederherstellung 24
12. Künstliche Intelligenz 24

TEIL IV — VERTRAUENSWÜRDIGKEIT UND NACHWEISE

13. Bedrohungsmodell 26
14. Transparenz und Überprüfung 28
15. Privatsphäre und Datenschutz 29
16. Umfang und Einschränkungen 30
17. Roadmap 31
18. Geschäftsgrundsätze 33
19. Fazit 33

ANHÄNGE

A. Primitive und Parameter 34
B. Schlüsselgrößen nach Suite 35
C. Netzwerkschnittstellen 35
D. Glossar 36

Zusammenfassung für Entscheider

uOS ist ein vollständiger Arbeitsbereich – Chat, E-Mail, Videokonferenzen, Dateien, Dokumente, Tabellen, Präsentationen, Boards, Kalender, Notizen, Aufgaben, KI-Assistenten, Schichtplanung und ein App-Studio –, bereitgestellt im Browser, mit einer einzigen Identität und einem einzigen Sicherheitsmodell. Der Grundgedanke ist einfach: **Der Betreiber eines Dienstes sollte nicht lesen können, was seine Nutzer erstellen.** In uOS werden Inhalte auf dem Gerät des Nutzers mit Schlüsseln, die allein der Nutzer besitzt, verschlüsselt, bevor sie unsere Server erreichen.

Drei Eigenschaften zeichnen uOS aus:

- **Post-Quanten-Kryptografie, Ende zu Ende.** Aktuelle Kontoschlüssel, jeder empfängerspezifische Inhaltsschlüssel, jede uOS-Sitzung und die Medien jedes Anrufs sind durch die von NIST standardisierten Post-Quanten-Algorithmen (ML-KEM und ML-DSA) geschützt, *kombiniert* mit bewährten klassischen Verfahren (X25519 und Ed25519). Heute abgefangene Inhalte können morgen nicht von einem Quantencomputer entschlüsselt werden, und eine Schwachstelle in nur einer der beiden Verfahrensfamilien bricht das System nicht. Seit September 2026 verwendet der Dienst für neue Schlüssel die stärkere Suite der Kategorie 5 (ML-KEM-1024, ML-DSA-87), und seit Oktober 2026 stellt er jedes bestehende Konto bei dessen nächster Anmeldung mit den Geheimnissen darauf um. Verbindungen zu uos.sh handeln mit aktuellen Browsern die hybride TLS-Gruppe `X25519MLKEM768` aus. Die Ausnahmen – Altkonten bis zu ihrer nächsten Anmeldung, Browser ohne hybrides TLS sowie die in Anhang A aufgeführten klassischen Primitive – werden in Abschnitt 16 benannt.
- **Ein Verschlüsselungsmodell für den gesamten Arbeitsbereich.** Wo andere Datenschutzprodukte ein E-Mail-Postfach oder einen Dokumenteneditor schützen, wendet uOS dieselbe Umschlagverschlüsselung auf Nachrichten, Dateien, Dokumente, Kalender, KI-Unterhaltungen, Boards, Organisations-Feeds und Dienstpläne an – und legt für jede Anwendung genau offen, was für den Server sichtbar bleibt.
- **Für Organisationen gebaut, nicht nur für Einzelpersonen.** Organisationsarbeitsbereiche ergänzen Rollen, eine Governance per Inhaberquorum, signierte Compliance-Profile, rechtliche Aufbewahrungspflichten (Legal Hold), Aufbewahrungsregeln, eine Kontrolle ausgehender E-Mails, eine Sicherheitsrichtlinie, die zum Öffnen des Arbeitsbereichs einen Sicherheitsschlüssel verlangen kann, und eine *Schlüsseltreuhand*, mit der eine Organisation ihre eigenen Geschäftsunterlagen wiederherstellen kann – ohne dem Betreiber irgendeinen Zugriff zu gewähren und ohne die privaten Arbeitsbereiche der Mitglieder zu berühren.

Dieses Papier richtet sich an zwei Lesergruppen. Entscheider finden das Produkt, das Vertrauensmodell und die Geschäftsgrundsätze in Teil I und IV. Sicherheitsprüfer finden die Architektur, die kryptografischen Protokolle, eine Abdeckungsmatrix pro Anwendung und ein Bedrohungsmodell in Teil II und III, die genauen Primitive und Parameter in den Anhängen. Wo eine Garantie Grenzen hat, steht die Grenze direkt neben der Garantie.

TEIL I — WARUM UOS

1. Das Problem: Vertrauen kraft Richtlinie

Die Software, auf die sich die meisten Menschen und Unternehmen verlassen – Office-Pakete, Cloud-Speicher, Team-Chat, E-Mail –, folgt einem gemeinsamen Muster: Der Anbieter betreibt den Code, speichert die Daten und verwahrt die Schlüssel. Verschlüsselung „im Ruhezustand" schützt vor einer gestohlenen Festplatte, nicht vor dem Anbieter. Alles Übrige beruht auf Richtlinien: Datenschutzerklärungen, Verträgen und Zertifizierungen.

Richtlinien werden in gutem Glauben aufgestellt, sind aber keine technische Grenze. Sie können sich mit einem neuen Eigentümer, einem neuen Geschäftsmodell oder einem neuen Gesetz ändern. Sie können durch einen Gerichtsbeschluss in einer anderen Rechtsordnung ausgehebelt werden. Sie halten keinen Innentäter mit Zugriff auf die Produktivumgebung auf, und sie überstehen keinen Sicherheitsvorfall. Für regulierte Organisationen ist die Folge konkret: Jede sensible Datei, die in einer gängigen Cloud gespeichert ist, ist grundsätzlich für jemanden außerhalb der Organisation lesbar.

Ein zweiter Handlungsdruck folgt einem festen Zeitplan. Die heute eingesetzten Public-Key-Verfahren – RSA, elliptische Kurven – werden von einem ausreichend großen Quantencomputer gebrochen werden. Angreifer müssen nicht warten: Heute aufgezeichneter Datenverkehr und gespeicherter Chiffretext können später entschlüsselt werden („harvest now, decrypt later"). Sicherheitsbehörden in Europa und den Vereinigten Staaten haben Zeitpläne für die Migration zur Post-Quanten-Kryptografie festgelegt; Daten, die lange vertraulich bleiben müssen, benötigen diesen Schutz schon heute.

2. Der Ansatz von uOS

uOS ersetzt Vertrauen kraft Richtlinie durch Vertrauen kraft Konstruktion:

1. **Verschlüsseln vor dem Hochladen.** Inhalte werden im Browser des Nutzers verschlüsselt. Der Server speichert und übermittelt Chiffretext, den er nicht öffnen kann, weil er die Schlüssel nie erhält.
2. **Schlüssel gehören den Menschen.** Die privaten Schlüssel eines Kontos werden auf dem Gerät des Nutzers aus zufälligem Seed-Material erzeugt und mit Geheimnissen entsperrt, die nur der Nutzer kennt. Der Betreiber kann ein Konto weder zurücksetzen noch wiederherstellen, noch kann er sich als dieses Konto ausgeben.
3. **Den Rest minimieren.** Was der Server sehen muss, um zu funktionieren – Routing, Reihenfolge, Zeitpunkte –, wird reduziert, wo immer das Produkt es zulässt: Absenderidentitäten werden versiegelt, Dateigrößen aufgefüllt, Zeitstempel vergröbert, Präsenzinformationen sind kurzlebig, und nirgendwo im Produkt laufen Tracking, Werbung oder Analysewerkzeuge.
4. **Genau sagen, was abgedeckt ist und was nicht.** Eine präzise, veröffentlichte Grenze ist für einen Prüfer nützlicher als ein weitreichendes Versprechen. Abschnitt 9 führt für jede Anwendung auf, was verschlüsselt ist und was der Server sehen kann.

3. uOS im Überblick

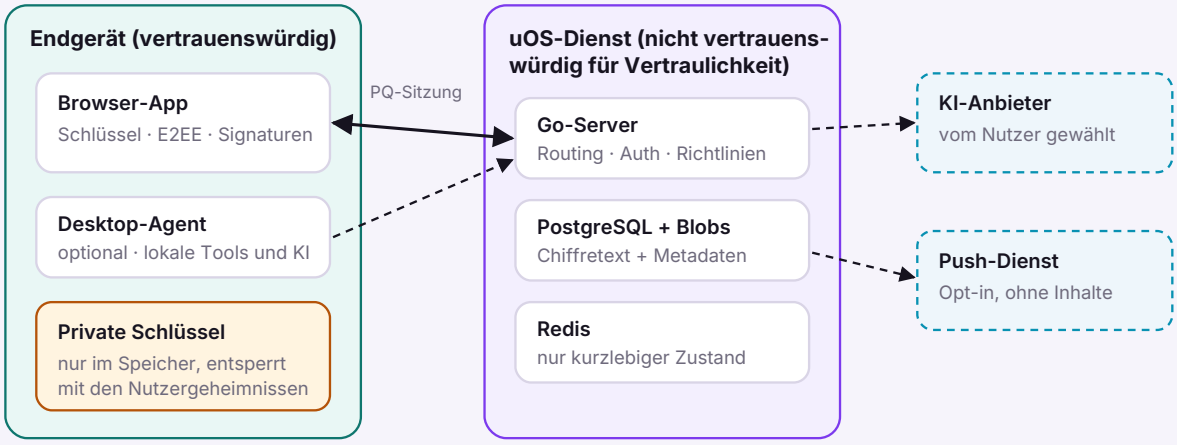
uOS läuft in jedem modernen Browser auf Desktop- und Mobilgeräten, als Arbeitsbereich im Desktop-Stil mit Fenstern oder als mobiles Vollbild-Layout. Es ist in 28 europäischen Sprachen verfügbar.

Bereich	Anwendungen
Kommunikation	uChat (Direkt-, Gruppen-, Kanal- und vertrauliche Chats), uMail, uMeet (Videokonferenzen), Kontakte, uLine (ein Feed innerhalb jedes Arbeitsbereichs)
Produktivität	Dateien, uDoc, uSheet, uPresent, Boards (Kanban), Kalender, uNotes, uTodo, Projekte
Organisation	uShift (Schichtplanung, Abwesenheiten, Zeiterfassung), Verwaltung, Plattformeinstellungen und Compliance
KI	Vivian (Assistent mit Werkzeugen), uSages (Gespräche mit historischen Persönlichkeiten), uAssistant (integrierte Hilfe), Desktop-Agent
Erstellen & Entdecken	Studio (Apps erstellen und veröffentlichen), Marktplatz, uBrowse und uNews, das uOS-Handbuch

Plattformen. Eine Person hat ein *Kernkonto* – ihre kryptografische Identität – und kann mehreren *Plattformen* angehören. Eine Plattform ist ein isolierter Arbeitsbereich mit eigenen Mitgliedern, Rollen, Apps, eigenem Branding und eigenen Einstellungen. Es gibt vier Arten: den persönlichen zentralen Arbeitsbereich, Organisationsplattformen (gewinnorientiert und gemeinnützig) sowie neutrale *Kollaborationsplattformen* für die organisationsübergreifende Zusammenarbeit. Der private Arbeitsbereich einer Person und der Arbeitsbereich ihrer Organisation sind bewusst getrennt, auch darin, wer den Inhalt jemals wiederherstellen kann (Abschnitt 10.4).

TEIL II — ARCHITEKTUR UND KRYPTOGRAFIE

4. Systemarchitektur



Durchgezogen: aller Nutzerverkehr, E2E-verschlüsselte Inhalte in einer authentifizierten PQ-Sitzung. Gestrichelt: optionale Pfade.

Abbildung 1 — Verschlüsselung und Schlüsselverwahrung liegen auf dem Gerät des Nutzers. Der Dienst speichert und übermittelt Chiffretext.

Client. Eine Angular-Single-Page-Anwendung, die die gesamte Schlüsselverwaltung und Verschlüsselung im Browser durchführt. Die privaten Kontoschlüssel liegen für die Dauer einer Sitzung im Arbeitsspeicher und werden standardmäßig nie in den Browserspeicher geschrieben. Dauerhafter Browserspeicher ist ein Gerätetresor mit den Schlüsseln für Vorwärtssicherheit (Forward Secrecy) dieses Geräts (Abschnitt 8.2), verschlüsselt unter einem nicht extrahierbaren Browserschlüssel, und – nur wenn der Nutzer einen Browser sich an ihn erinnern lässt – die versiegelte Kopie aus Abschnitt 5.6; der übrige Browserspeicher wird bei jedem Laden geleert. Anwendungsdaten werden – verschlüsselt – auf dem Server gehalten, damit sich ein Nutzer von jedem Gerät aus anmelden kann.

Server. Ein Go-Dienst, der Sitzungen authentifiziert, Nachrichten weiterleitet, Plattformmitgliedschaft und Organisationsrichtlinien durchsetzt und Daten speichert. Er gilt als *hinsichtlich der Vertraulichkeit nicht vertrauenswürdig*: Das System ist so ausgelegt, dass ein vollständig kompromittierter Server Chiffretext und Metadaten liefert, aber keine Inhalte.

Speicherung. PostgreSQL enthält Chiffretext, öffentliche Schlüssel und betriebliche Metadaten; verschlüsselte Dateiinhalte werden als opake Blobs gespeichert. Redis enthält ausschließlich kurzlebigen Zustand – Handshake-Challenges, Präsenz, Zähler für die Ratenbegrenzung. Sitzungstokens und Signaturschlüssel werden im Arbeitsspeicher des Servers gehalten, nie in Redis oder auf der Festplatte; ein Neustart beendet jede Sitzung.

Transport. Alles, was ein Nutzer tut, läuft über einen einzigen authentifizierten WebSocket (`/ws`), der mit einem Post-Quanten-Handshake aufgebaut wird (Abschnitt 6.3). Blockiert ein Netzwerk WebSockets, weicht der Client auf einen HTTPS-Long-Poll-Kanal aus, der denselben Handshake durchführt und dieselben verschlüsselten, signierten Frames überträgt. Die verbleibende HTTP-Schnittstelle ist klein und in Anhang C aufgeführt: Anmeldung und Abfrage der Schlüsselableitung, Anmeldung mit einer uID oder ei-

nem gemerkten Browser, PIN-Prüfungen, Übertragungen bei der Kontorotation, ticketgebundener Massen-Upload von Chiffretext, öffentliche Einladungs- und Plattformseiten sowie veröffentlichte Studio-Websites. Diejenigen dieser Anfragen, die etwas Sensibles übertragen – die Anmeldeabfrage und ihre Antwort, die Anmeldung mit uID und mit gemerktem Browser, das Entsperren, PIN-Prüfungen, das Ändern der Geheimnisse und die Rotation –, sind selbst mit dem hybriden KEM (ML-KEM-1024 + X25519) für einen Serverschlüssel versiegelt, der von der fest hinterlegten (gepinnten) Serveridentität signiert ist; sie hängen daher auch nicht von TLS ab.

Desktop-Agent. Ein optionales Programm auf dem Computer des Nutzers, das sich *ausgehend* mit dem Dienst verbindet, sodass keine Ports geöffnet werden. Es ermöglicht dem KI-Assistenten, unter ausdrücklichen Berechtigungen mit lokalen Dateien und Befehlen zu arbeiten, und kann KI-Modelle lokal ausführen.

5. Identität und Konten

5.1 Registrierung und Entsperrung

Ein uOS-Konto ist ein Satz von Schlüsselpaaren, kein Benutzername mit Passwort auf einem Server. Bei der Registrierung erzeugt der Browser mit einem kryptografischen Zufallszahlengenerator zufälliges Seed-Material – einen 32-Byte-ML-DSA-Seed und einen 64-Byte-ML-KEM-Seed. Aus diesem *Seed-Bündel* leitet er die Post-Quanten- und die klassischen Signatur- und Verschlüsselungsschlüsselpaare des Kontos ab, jeweils mit domänengetrennter Schlüsselableitung.

Damit das Konto von jedem Gerät aus nutzbar ist, wird das Seed-Bündel in umhüllter Form auf dem Server gespeichert. Der Nutzer wählt zwei Geheimnisse – **Secret 1** (mindestens 12 Zeichen) und **Secret 2**, eine Passphrase (mindestens 20 Zeichen) – und eine Kontokennung in Form einer E-Mail-Adresse. Der Browser leitet mit Argon2id einen Schlüsselverschlüsselungsschlüssel (KEK) ab und umhüllt damit das Seed-Bündel:

```
KEK     = Argon2id(NFKC(S1) ‖ 0 ‖ NFKC(S2) ‖ 0 ‖ NFKC(identifier) ‖ 0, salt₃₂, m = 256 MiB, t =
3, p = 2)
wrapped = XChaCha20-Poly1305(KEK, nonce₂₄, seed bundle, AAD = account id ‖ KDF version ‖ secret
epoch)
```

Jede Eingabe wird vor der Ableitung Unicode-normalisiert (NFKC) und von führenden und abschließenden Leerzeichen befreit, sodass dieselben Geheimnisse das Konto auf jedem Betriebssystem und mit jeder Tastatur entsperren; die Felder werden durch Null-Byte-Trennzeichen verbunden. Der Server speichert das umhüllte Bündel, das Salt, die öffentlichen Schlüssel und einen gepfefferten HMAC der Kennung – niemals die Geheimnisse, einen Passwort-Prüfwert oder die Kennung im Klartext. Bei der Anmeldung ruft der Browser das infrage kommende umhüllte Bündel ab, leitet den KEK lokal ab und hebt die Umhüllung auf. Eine unbekannte Kennung erhält eine Attrappe derselben Form, die deterministisch aus der Kennung und einem Servergeheimnis abgeleitet wird; wiederholte Abfragen liefern daher dieselbe Antwort und verraten nicht, ob ein Konto existiert.

Das umhüllte Bündel wird nur an jemanden herausgegeben, der die Geheimnisse nachweist. Die Abfrage liefert nur das Argon2id-Salt des Kontos. Aus der Argon2id-Ausgabe leitet der Browser neben dem KEK einen separaten Anmeldeschlüssel `auth_key = HKDF-SHA256(Argon2id output, "uOS/login-auth/v1"`

`|| account id)` ab; der Server speichert nur `SHA-256("uOS/login-verifier/v1" || auth_key)` und gibt das umhüllte Bündel über eine mit dem hybriden KEM versiegelte Anfrage nur heraus, wenn der vorgelegte Anmeldeschlüssel übereinstimmt. Falsche Schlüssel, unbekannte Konten und Attrappen erhalten dieselbe Ablehnung, und Versuche sind pro Konto und Adresse (5 pro Minute) sowie pro Konto insgesamt (30 pro Minute) begrenzt, sodass ein Fremder den Kontoinhaber nicht aussperren kann. Die Kenntnis einer Kennung bietet daher kein Ziel für Offline-Rateangriffe mehr: Offline-Raten erfordert die Datenbank des Servers. Der Mechanismus verwendet ausschließlich symmetrische Primitive und fügt daher keine quantenanfällige Annahme hinzu. Er ist kein OPAQUE: Der Server sieht den Anmeldeschlüssel während einer Anmeldung. Konten wechseln bei ihrer nächsten Anmeldung automatisch auf diese Anmeldeschranke.

Folgen dieses Designs:

- Der Betreiber **kann ein Konto weder zurücksetzen noch wiederherstellen**. Es gibt keinen Link „Passwort vergessen“; die Wiederherstellung liegt in der Hand des Nutzers (Abschnitt 11).
- Wer eine Kopie der Datenbank besitzt, kann weiterhin offline raten: Jeder Rateversuch für die beiden Geheimnisse kostet eine Argon2id-Auswertung mit 256 MiB. Neue Geheimnisse folgen NIST SP 800-63B: Secret 1 hat mindestens 12 Zeichen und Secret 2 mindestens 20, die beiden unterscheiden sich voneinander und von der Kennung, und es gibt keine Zusammensetzungsregeln. Der Browser weist jedes Geheimnis zurück, das in einer Liste der eine Million häufigsten Passwörter aus Datenlecks (SecLists, MIT-Lizenz) vorkommt; die Liste wird als Bloom-Filter ausgeliefert und vollständig auf dem Gerät geprüft, auch ohne Leerzeichen und ohne abschließende Ziffern oder Sonderzeichen; nichts über das Geheimnis wird irgendwohin gesendet. Bestehenden Konten wird empfohlen, aber nicht vorgeschrieben, kurze Geheimnisse zu verstärken.
- Private Schlüssel existieren nur im Arbeitsspeicher einer entsperrten Sitzung, es sei denn, der Nutzer lässt einen Browser sich an das Konto erinnern (Abschnitt 5.6).

Weitere Wege zum Bündel. Zwei weitere Wege geben das Seed-Bündel frei, ohne dass die Geheimnisse eingetippt werden: ein gemerkter Browser (Abschnitt 5.6) und eine *uID*, ein FIDO2-Sicherheitsschlüssel oder Passkey, den der Nutzer hinzugefügt hat (Abschnitt 5.7). Beide öffnen eine versiegelte Kopie des Bündels im Browser und führen dann denselben Anmelde-Handshake aus; keiner lässt den Server das Bündel sehen oder von sich aus eine Sitzung ausstellen. Die drei Geheimnisse funktionieren immer weiter, und die Einrichtung der Wiederherstellung, der Wiederherstellungstest und das Ändern der Geheimnisse verlangen sie weiterhin.

5.2 PIN und besonders schutzbedürftige Aktionen

Eine separate PIN, die auf dem Server gegen einen Argon2id-Hash geprüft wird, sichert sensible Aktionen ab – etwa das Entsperren einer zurückkehrenden Sitzung, das Einschalten und erneute Öffnen eines gemerkten Browsers (Abschnitt 5.6), das Hinzufügen einer uID (Abschnitt 5.7), das Ändern der Geheimnisse und das Rotieren der Identität –, und zwar zusätzlich zu den Signaturen der Sitzung. Weil das Hinzufügen einer uID die PIN verlangt, kann eine gestohlene oder unbeaufsichtigte Sitzung nicht unbemerkt einen eigenen Schlüssel registrieren. Die uID selbst wird nicht mit dieser PIN entsperrt: Der Authenticator prüft seine eigene PIN oder ein biometrisches Merkmal, und eine Änderung der Konto-PIN lässt die uIDs bestehen.

Begrenzung der Versuche. Jede PIN-Abfrage – das Entsperren, das erneute Öffnen und Einschalten eines gemerkten Browsers, das Hinzufügen einer uID, das Ändern der Geheimnisse, die Rotation sowie das Prüfen und Ändern der PIN in den Einstellungen – zählt auf dem Server gegen einen einzigen Zähler pro

Konto, sodass sich Rateversuche nicht auf verschiedene Abfragen verteilen lassen. Jeder Versuch wird gezählt, bevor die PIN geprüft wird, sodass parallele Rateversuche dem Zähler nicht zuvorkommen können. Die ersten drei Versuche dürfen unmittelbar aufeinander folgen; nach der dritten falschen PIN in Folge muss jeder weitere Versuch 2, 4, 8, 16, 32, 64 und schließlich 128 Sekunden nach dem vorherigen warten, und der Browser zeigt die verbleibenden Sekunden an. Die zehnte falsche PIN in Folge sperrt die PIN: Jede PIN-Abfrage wird dann abgelehnt, selbst mit der richtigen PIN, und aufgehoben wird die Sperre nur durch eine erneute Anmeldung mit den drei Geheimnissen. Eine richtige PIN vor diesem Punkt setzt den Zähler zurück. Ein gemerkter Browser auf der PIN-Stufe hat zusätzlich eine eigene, niedrigere Grenze: Nach fünf falschen PINs in Folge wird er vergessen (Abschnitt 5.6).

5.3 Rotation und Ändern der Geheimnisse

Nutzer können ihre Geheimnisse ändern oder ihre gesamte Identität rotieren. Eine Rotation erzeugt eine neue Schlüsselepoche, umhüllt jeden Inhaltsschlüssel, den das Konto öffnen kann, neu für den neuen Schlüssel und lässt jeden Verweis auf den alten Schlüssel auf den neuen zeigen – in einer einzigen Transaktion, die 24 Stunden lang fortgesetzt werden kann. Werte, die für einen früheren Schlüssel des Kontos versiegelt sind – etwa Daten, die nach der Umstellung auf v3-hybrid noch für seinen v2-Schlüssel versiegelt sind –, werden mit diesem früheren Schlüssel geöffnet und ebenfalls neu umhüllt, damit eine spätere Rotation sie nicht zurücklässt; mit dem neuen Schlüssel wird nie etwas geöffnet. Die Kontaktdatensätze des Kontos, die kein Konto nennen, sondern nur einen schlüsselabhängigen Hash aus Plattform und Schlüssel des Inhabers, werden im selben Commit neu verschlüsselt und auf den Hash des neuen Schlüssels verschoben. Eine verpflichtende PIN, eine Prüfung der Wiederherstellungsanteile und eine Prüfstufe beim Deployment, die die Rotation und auch die Kontolöschung gegen das aktuelle Datenbankschema prüft (Abschnitt 14.3), schützen den Vorgang. Das Ändern der Geheimnisse ersetzt das umhüllte Bündel und macht alte Geheimnisse und altes Wiederherstellungsmaterial ungültig.

Konten, die vor dem Zufalls-Seed-Modell erstellt wurden und deren Schlüssel sich direkt aus ihren Geheimnissen ergeben, werden **bei ihrer nächsten Anmeldung aktualisiert**. Bevor sich der Arbeitsbereich öffnet, fragt ein nicht überspringbarer Schritt die PIN ab und führt eine Rotation auf ein neues, zufälliges Seed-Bündel in der aktuellen Suite durch. Das umhüllte Bündel, der Anmelde-Prüfwert (Login-Verifier), die Einträge im Schlüsseltransparenz-Protokoll und jeder Inhaltsschlüssel werden in einem einzigen Commit ersetzt, und schlägt der Schritt fehl, ändert sich nichts. Ein Betreiber kann zusätzlich ein Datum festlegen, ab dem ein solches Konto nichts anderes mehr tun kann als diese Aktualisierung.

5.4 Serveridentität

Clients authentifizieren den Server, nicht nur umgekehrt. Die Post-Quanten-Signaturschlüssel des Servers sind im Client **gepinnt**: Bei jedem Handshake signiert der Server die Challenge des Clients, und der Client lehnt jeden Serverschlüssel ab, dessen Fingerabdruck nicht gepinnt ist. Jeder Frame, den der Server danach sendet, wird signiert und verifiziert. Ein Angreifer im Netzwerk, der TLS bricht oder terminiert, kann sich daher *gegenüber einem Client, der bereits unveränderten Code ausführt,* nicht als uOS-Server ausgeben. Die Pins sind jedoch Teil des Web-Clients, und der Web-Client wird selbst über TLS ausgeliefert: Wer TLS in dem Moment kontrolliert, in dem die App geladen wird, kann anderen Code ausliefern, einschließlich anderer Pins. Die Pins schützen einen unverändert bezogenen Client – ein zwischengespeichertes Release, einen verifizierten Build oder eine native App – und schützen für sich allein nicht vor der Auslieferung bösartigen Codes (Abschnitt 13.2).

5.5 Kontakte, Schlüsselwechsel und Verifizierung

Wenn ein Kontakt seine Schlüssel rotiert, verteilt der Server einen Schlüsselwechsel-Datensatz, der *sowohl* mit dem alten als auch mit dem neuen Schlüssel des Kontakts signiert ist, und Clients warnen, bevor sie an den neuen Schlüssel senden; ein Server kann keine Rotation eines Schlüssels fälschen, den er nicht besitzt. Beim Erstkontakt ist das anders: Wenn Alice einem neuen Kontakt schreibt oder eine Datei mit Bob teilt, verschlüsselt ihr Client für den Schlüssel, den der Server für Bob ausliefert. Ein bösartiger Server könnte für jeden Kontakt, dessen Schlüssel nicht verifiziert wurde, einen eigenen Schlüssel ausliefern. **Schlüsseltransparenz** schließt den größten Teil dieser Lücke. Jede Schlüsselepoche eines Kontos (Registrierung, Rotation, Suite-Upgrade) wird in derselben Datenbanktransaktion an ein nur erweiterbares Merkle-Protokoll nach RFC 6962 angehängt, dessen Baumköpfe (tree heads) der Server mit seinem gepinnten Identitätsschlüssel signiert. Bevor ein Client für einen Kontaktschlüssel verschlüsselt, verlangt er einen Inklusionsnachweis, dass der Schlüssel der neueste Eintrag dieses Kontos ist, weist Schlüsselwechsel zurück, die das Protokoll nicht zeigt, und prüft jeden Baumkopf auf Konsistenz mit jedem früheren Baumkopf, den er gesehen hat – einschließlich der Baumköpfe, die seine Kontakte in verschlüsselten Chat-Umschlägen melden. Bei jeder Anmeldung prüft ein Client sein eigenes Konto: Jeder Eintrag muss sein aktueller Schlüssel sein oder über Schlüsselwechsel-Datensätze, die mit beiden Schlüsseln signiert sind, mit ihm verknüpft sein, und ausgehend vom zuletzt verifizierten Baumkopf (signiert in seinen verschlüsselten Einstellungen gespeichert) berechnet er die Wurzel über alle seither hinzugekommenen Einträge neu, sodass ein Schlüssel, den der Server für ein Konto registriert, dem Kontoinhaber angezeigt wird. Dies gilt konstruktionsbedingt für jeden Verschlüsselungspfad. Die Verschlüsselungsfunktionen des Clients akzeptieren nur einen Empfänger, der über die verifizierte Abfrage ermittelt wurde, die eigenen Schlüssel der Sitzung, Schlüssel, die der Browser gerade erzeugt hat, oder eine kurze Liste benannter Schlüssel, die keinem Konto gehören, und eine Prüfung beim Build weist jede andere Verwendung zurück. Die Pfade sind Chat, Anhänge, E-Mail, Dateien, Dokumente, Tabellen, Präsentationen, Projekte, Boards, Kalender, Meetings, uLine, uShift, Studio und die App-Datenbank des Studios, Profilfreigaben, Treuhand-Anteilsinhaber und Rotation. Ein Geräte-Prekey (Vorab-Schlüssel) wird nur verwendet, wenn sein Gerät von einem im Protokoll nachgewiesenen Kontoschlüssel signiert ist und das Gerät den Prekey signiert hat. Offer und Answer eines Anrufs werden nur von einem nachgewiesenen Identitätsschlüssel akzeptiert. Kann der Schlüssel eines Empfängers nicht nachgewiesen werden, lassen Gruppenoperationen diesen Empfänger aus und weisen darauf hin; Operationen, bei denen sonst jemand seinen Zugang verlieren würde, etwa Treuhandanteile und Neuverschlüsselungen, brechen stattdessen ab. Auch die Treuhandschlüssel von Organisationen stehen im Protokoll, als Erklärungen, die der Inhaber signiert hat, der sie festgelegt hat (Abschnitt 10.4). Das Protokoll kann nicht beweisen, dass ein Schlüssel nicht existiert, und bindet Konten nicht an Personen: Ein Server, der verschiedenen Nutzern verschiedene Protokolle zeigt, wird erst entdeckt, wenn deren Sichten aufeinandertreffen, und ein neu erfundenes Konto, das sich als eine bestimmte Person ausgibt, erfordert weiterhin den Vergleich einer symmetrischen *Sicherheitsnummer* über einen unabhängigen Kanal.

5.6 Gemerkte Browser

Standardmäßig erfordert jeder Besuch die drei Geheimnisse. Ein Nutzer kann stattdessen einen Browser sich an das Konto erinnern lassen, auf einer von zwei Stufen, die er in der Systemleiste wählt: **PIN in diesem Browser** (beim erneuten Öffnen wird nur die Konto-PIN abgefragt; gültig für 90 Tage) oder **Angemeldet bleiben** (beim erneuten Öffnen erfolgt die Anmeldung direkt; gültig für 30 Tage, mit deutlicher Warnung). Beide Stufen lassen sich nur mit der PIN einschalten.

Der Browser hält dann das entpackte Seed-Bündel doppelt versiegelt: innen mit einem nicht extrahierbaren WebCrypto-AES-256-GCM-Schlüssel, der den Browser nie verlässt, außen mit XChaCha20-Poly1305 unter einem Schlüssel, der (HKDF-SHA256) aus einem zufälligen 32-Byte-Anteil des Servers und auf der PIN-Stufe aus einer Argon2id-Ableitung der PIN gebildet wird. Zum erneuten Öffnen weist der Browser über zwei versiegelte Endpunkte vor der Anmeldung den Besitz eines zweiten nicht extrahierbaren Schlüssels nach (HMAC über eine einmalige Challenge) und auf der PIN-Stufe die PIN; erst dann gibt der Server seinen Anteil heraus, und der normale Anmelde-Handshake läuft unverändert ab. Der Server stellt in diesem Schritt nie eine Sitzung aus und sieht das Seed-Bündel nie.

- Ein gestohlenes Gerät allein lässt sich auf der PIN-Stufe nicht offline öffnen: Der Anteil des Servers fehlt, und fünf falsche PINs löschen die Registrierung.
- Die Serverdatenbank allein enthält nur die Anteile und die Nachweisschlüssel, die ohne die versiegelte Kopie des Browsers nichts öffnen.
- Gestohlenes Gerät **und** Datenbank zusammen erlauben ein Offline-Erraten der PIN; auf der Stufe „Angemeldet bleiben“ öffnet das Gerät allein das Konto, bis die Registrierung abläuft oder widerrufen wird.

Jeder gemerkte Browser kann aus einer anderen Sitzung heraus vergessen werden; das Sperren des Kontos, das Ändern der Geheimnisse und das Rotieren der Identität vergessen alle. Organisationen können gemerkte Browser für ihre Mitglieder einschränken: nicht erlaubt, nur mit PIN oder uID, oder PIN, uID oder angemeldet bleiben (Abschnitt 10.5).

Eine uID als Faktor. Auf der PIN-Stufe kann der Faktor statt der PIN die Berührung einer der uIDs des Kontos sein (Abschnitt 5.7). Der Browser bewahrt in seinem Datensatz ein zufälliges 32-Byte-Salt auf und fragt genau diesen einen Berechtigungsnachweis nach einer PRF-Ausgabe für `SHA-256("uOS/uid/device-unlock/v1|" || salt)`; der äußere Schlüssel wird zu `HKDF-SHA256(server share || PRF output, info = "uOS/device-unlock/v1" || registration id)`. Weil die Eingabe pro Browser gewählt wird, hat der Faktor nichts mit dem Anmeldeschlüssel der uID zu tun und unterscheidet sich zwischen zwei Browsern, die mit derselben uID gemerkt werden. Zum erneuten Öffnen sendet der Browser eine WebAuthn-Assertion über die Challenge des Entsperrvorgangs selbst; der Server prüft sie wie bei einer Anmeldung mit uID, einschließlich des Signaturzählers, bevor er seinen Anteil herausgibt. Gestohlenes Gerät und Datenbank zusammen lassen dann keine PIN zum Erraten übrig: Der Faktor ist ein 32-Byte-Geheimnis, das nur der Authenticator nach seiner eigenen Nutzerverifikation berechnet. Falsche Assertions werden nicht gezählt, weil nur der Authenticator eine gültige erzeugen kann; ein zurücklaufender Zähler wird abgelehnt und protokolliert. Das Einschalten verlangt weiterhin die Konto-PIN; eine PIN-Änderung lässt eine solche Registrierung bestehen, während das Entfernen dieser uID, eine Sperre, eine Änderung der Geheimnisse und eine Rotation sie vergessen. Der Server speichert nur, dass der Faktor eine uID ist und welche.

5.7 uID-Schlüssel

Eine *uID* ist ein WebAuthn-Berechtigungsnachweis, den der Nutzer bei sich trägt: ein FIDO2-Sicherheitsschlüssel mit der Erweiterung `hmac-secret` oder ein Passkey, dessen Anbieter die PRF-Erweiterung unterstützt. Sie öffnet eine Kopie des Seed-Bündels, die unter einem Schlüssel versiegelt ist, den nur der Authenticator nach der Nutzerverifikation auf dem Authenticator (seiner PIN oder einem biometrischen Merkmal) erneut berechnen kann. Danach läuft der normale Anmelde-Handshake unverändert ab, wie bei einem gemerkten Browser; der Server stellt aus einer uID nie eine Sitzung aus und sieht das Seed-Bündel nie.

Jede uID ist ein auffindbarer Berechtigungsnachweis (discoverable credential) mit verpflichtender Nutzerverifikation und PRF-Erweiterung; ein Authenticator, der keine PRF-Ausgabe liefert oder keinen auffindbaren Berechtigungsnachweis speichert, wird abgelehnt, und nichts wird aufbewahrt. Die Relying Party ist der Host der Installation (`uos.sh` beim Dienst uos.sh), sodass ein Berechtigungsnachweis nur diesem Ursprung antwortet: Eine nachgeahmte Domain erhält weder eine Assertion noch eine PRF-Ausgabe. Die Attestierung ist `none`, daher wird jedes Authenticator-Modell akzeptiert, und der Server erfährt nicht, welches. Das WebAuthn-User-Handle ist ein zufälliger 32-Byte-Wert pro Konto, nie die Konto-ID. Akzeptierte Algorithmen sind Ed25519, ECDSA P-256 und RSA (PKCS#1 v1.5); ein Konto hat höchstens zehn uIDs.

```
PRF_INPUT = SHA-256("uOS/uid/prf/v1")                              fixed for every credential
prf       = WebAuthn PRF(credential, PRF_INPUT)                    inside the authenticator, after
user verification
K         = HKDF-SHA256(prf, salt = cred_salt32, info = "uOS/uid/v1" ‖ credential id)
box       = XChaCha20-Poly1305(K, nonce24, account id ‖ seed bundle, AAD = "uOS/uid/bundle/v1|" ‖
credential id)
```

Die PRF-Eingabe ist fest, weil eine Anmeldung mit einem auffindbaren Berechtigungsnachweis das Konto vor der Assertion nicht kennt. Die PRF-Ausgabe unterscheidet sich trotzdem pro Berechtigungsnachweis, und das zufällige Salt pro Berechtigungsnachweis, das der Server hält, sowie die Credential-ID in der HKDF-Info trennen die Boxen zusätzlich und binden jede Box an ihre Zeile.

Dieses Gerät oder Sicherheitsschlüssel bzw. Telefon. Beim Hinzufügen einer uID wählt der Nutzer, wo sie liegt. *Dieses Gerät* bedeutet Touch ID, Face ID, Windows Hello oder die Displaysperre von Android: ein Passkey auf dem Gerät oder in dessen synchronisiertem Passwortmanager. *Sicherheitsschlüssel oder Telefon* bedeutet einen FIDO2-Schlüssel wie einen YubiKey oder Nitrokey oder ein anderes Gerät, etwa ein Telefon, das über einen QR-Code erreicht wird. Die PRF-Ausgabe ist in beiden Fällen Pflicht, denn sie ist es, die die Schlüssel auf dem Gerät entpackt: Ohne sie gibt es keinen Schlüssel zur Box. Ein Passkey-Speicher, der keine PRF-Ausgabe liefert – manche Browser halten Passkeys in einem eigenen Speicher, dem sie fehlt –, wird mit einem Hinweis auf einen anderen Browser oder einen Sicherheitsschlüssel abgelehnt, statt das Bündel unter etwas Schwächerem zu versiegeln. Passkeys im iCloud-Schlüsselbund oder im Google Passwortmanager werden mit den anderen Geräten dieses Kontos synchronisiert, während ein Hardwareschlüssel den Schlüssel nie verlässt; beide sind phishing-resistent, aber ein synchronisierter Passkey ist nur so sicher wie das Konto, das ihn synchronisiert. Der Server unterscheidet beide derzeit nicht: Er wertet die Backup-Flags in den Authenticator-Daten nicht aus und erfährt wegen der Attestierung `none` das Modell nicht. Eine Organisation, die eine uID verlangt (Abschnitt 10.5), kann daher noch keinen Hardwareschlüssel vorschreiben; die Oberfläche weist darauf hin, dass Organisationen einen solchen bevorzugen können.

Das Hinzufügen einer uID erfolgt über den angemeldeten WebSocket und verlangt die Konto-PIN (Abschnitt 5.2). Der Browser erzeugt den Berechtigungsnachweis, versiegelt das Bündel mit der PRF-Ausgabe und sendet die Box mit dem Attestierungsobjekt; der Server verbraucht die einmalige Challenge und prüft Zeremonientyp, Ursprung und Cross-Origin-Kennzeichen, den rpId-Hash, die Flags für Nutzeranwesenheit, Nutzerverifikation und angehängte Berechtigungsdaten sowie den Schlüsselalgorithmus, bevor er die Zeile speichert. **Die Anmeldung** besteht aus zwei versiegelten Anfragen vor der Anmeldung: einer einmaligen Challenge, die 120 Sekunden gilt, und dann der Assertion. Der Server verbraucht die Challenge, prüft Zeremonientyp, Ursprung und Cross-Origin-Kennzeichen, den rpId-Hash, Nutzeranwesenheit und -verifikation, die Signatur über `authenticatorData ‖ SHA-256(clientDataJSON)` mit dem gespeicher-

ten Schlüssel, das User-Handle, den Signaturzähler (ein Zähler, der nicht wächst, wird abgelehnt und als möglicher Klon protokolliert; Passkeys, die stets null melden, werden akzeptiert) und ob das Konto aktiv ist, und gibt erst dann das Salt und die Box zurück. Der Browser leitet K aus der PRF-Ausgabe derselben Assertion ab, öffnet die Box und führt den normalen Handshake aus. Beide Anfragen sind wie die übrigen Endpunkte vor der Anmeldung versiegelt (Anhang C) und pro Adresse und pro Berechtigungsnachweis begrenzt. Falsche PINs des Authenticators erreichen den Server nie: Der Authenticator zählt sie selbst.

Eine Sperre entfernt alle uIDs des Kontos, denn eine uID, die jemand mit einer gestohlenen Sitzung und der PIN hinzugefügt hat, darf den Notschalter nicht überstehen; eine Änderung der Geheimnisse und eine Rotation entfernen sie in derselben Transaktion, weil die Box das alte Bündel enthält. Eine Änderung der Konto-PIN lässt sie bestehen. Das Entfernen der letzten uID sperrt niemanden aus, da die drei Geheimnisse weiter funktionieren.

Hinweis zur Post-Quanten-Sicherheit. uID-Assertions sind klassische Signaturen (Ed25519, ECDSA, RSA), authentifizieren aber nur eine einmalige Challenge: Eine spätere Fälschung bringt nichts. Die Vertraulichkeit des Seed-Bündels beruht auf symmetrischen Primitiven – der auf HMAC-SHA-256 beruhenden PRF des Authenticators, HKDF-SHA256 und XChaCha20-Poly1305 – und bei der Übertragung auf der hybriden Versiegelung der Anfragen vor der Anmeldung mit ML-KEM-1024 und X25519. Ist die Box geöffnet, sind Identität und Sitzung die ML-DSA- und ML-KEM-Schlüssel des Kontos im normalen Handshake.

6. Kryptografie

6.1 Grundsätzlich hybrid

uOS kombiniert für jede asymmetrische Operation einen Post-Quanten- und einen klassischen Algorithmus. Beide müssen erfolgreich sein; keiner wird allein akzeptiert:

- **Schlüsselvereinbarung:** ML-KEM (FIPS 203) mit X25519. Das gemeinsame Geheimnis wird mit HKDF über beide Geheimnisse und beide Chiffretexte abgeleitet.
- **Signaturen:** ML-DSA (FIPS 204) mit Ed25519. In den v3-Suiten ist jede Signatur zusammengesetzt, und beide Hälften müssen gültig sein. In v2 tragen Sitzungsrahmen und Gerätesignaturen beide, wenn für das Konto ein Ed25519-Schlüssel hinterlegt ist; andere Identitätssignaturen (Einladungen, Chatnachrichten, Agent-Aktionen und Ähnliches) sind reine ML-DSA-65-Signaturen.
- **Symmetrische Verschlüsselung:** XChaCha20-Poly1305 mit 256-Bit-Schlüsseln und zufälligen 192-Bit-Nonces, die auch gegen Quantenangriffe stark bleiben.
- **Anmeldeschlüssel:** Eine uID (Abschnitt 5.7) antwortet mit einer klassischen WebAuthn-Signatur (Ed25519, ECDSA P-256 oder RSA). Das ist die einzige asymmetrische Operation ohne Post-Quanten-Hälfte, bedingt durch das FIDO2-Ökosystem: Sie authentifiziert nur eine einmalige Anmelde-Challenge und schützt keine Daten, und das Seed-Bündel, das sie freigibt, ist durch symmetrische Schlüssel und die hybride Versiegelung dieser Anfragen geschützt.

Die hybride Konstruktion ist der Ansatz, den europäische Sicherheitsbehörden (ANSSI; BSI TR-02102-1, die außerdem ML-KEM-1024 empfiehlt) für die Übergangsphase zur Post-Quanten-Kryptografie empfehlen: Eine unerwartete Schwäche der jungen Post-Quanten-Algorithmen fängt die klassische Hälfte auf, und Quantenangriffe auf die klassische Hälfte fängt die Post-Quanten-Hälfte auf. Zusätzlich schützt TLS

die Verbindung. Sobald die App läuft, hängt die Vertraulichkeit von Inhalten und sensiblen Anfragen nicht von TLS ab; die Auslieferung der App selbst dagegen schon (Abschnitt 13.2). Deshalb handelt uos.sh mit Browsern, die es unterstützen, zusätzlich hybrides Post-Quanten-TLS (`X25519MLKEM768`) aus.

6.2 Kryptosuiten

Jeder Schlüssel, jeder Chiffretext und jede Signatur gehört zu einer **Suite**. Eine Bereitstellung akzeptiert eine feste Menge von Suiten und schreibt genau eine; nichts wird ausgehandelt, daher gibt es nichts herabzustufen.

Suite	Schlüsselkapselung	Signaturen	AEAD	KDF-Hash	Verwendung
v2	ML-KEM-768 + X25519	ML-DSA-65 (+ Ed25519 bei Sitzungsrahmen und Gerätesignaturen)	XChaCha20-Poly1305	SHA-256	nur für noch nicht umgestellte Konten akzeptiert
v3-hybrid	ML-KEM-1024 + X25519	ML-DSA-87 + Ed25519	XChaCha20-Poly1305	SHA-384	**von uos.sh seit September 2026 geschrieben**
v3-cnsa	ML-KEM-1024	ML-DSA-87	AES-256-GCM	SHA-384	integriert und per KAT getestet; für dedizierte Installationen

Die v3-hybrid-Werte sind *zusammengesetzt* (Composite): Die klassische Hälfte ist in jedem öffentlichen Schlüssel, jedem Chiffretext und jeder Signatur enthalten, sodass kein Feld ohne sie existieren kann, und wer sie entfernt, erhält einen fehlerhaften statt eines schwächeren Werts. Die Konstruktion ist eine Eigenentwicklung von uOS. Eine zusammengesetzte Signatur ist eine ML-DSA-87-Signatur mit einem suitespezifischen Kontext-String, gefolgt von einer Ed25519-Signatur über eine mit dem Suite-Präfix versehene Nachricht, sodass keine der beiden Hälften als Signatur eines anderen Protokolls gültig ist, das denselben Schlüssel verwendet. Der KEM folgt dem Design von X-Wing – Chiffretext und öffentlicher Schlüssel der klassischen Hälfte fließen in den Kombinierer (Combiner) ein –, verwendet aber HKDF-SHA384 und ist im Wire-Format nicht mit X-Wing kompatibel; die Signaturen beanspruchen keine Konformität mit den Composite-Entwürfen von IETF LAMPS. Die Angleichung an diese Standards steht auf der Roadmap. Jedes Schlüsselableitungs-Label ist an seine Suite gebunden, sodass ein Chiffretext nicht unter der Ableitung einer anderen Suite geöffnet werden kann. Neue Konten, neue Schlüssel und alles, was der Server signiert, verwenden v3-hybrid. Ein Konto, das noch v2 verwendet, wird bei seiner nächsten Anmeldung mit den drei Geheimnissen dorthin umgestellt, aus demselben Seed-Bündel, sodass man sich nichts Neues merken muss; bis dahin akzeptiert uos.sh noch seine v2-Schlüssel. Die Suite *v3-cnsa* richtet sich an Kunden, die das US-amerikanische Profil CNSA 2.0 einhalten müssen, und ist ausschließlich für deren eigene oder dedizierte Infrastruktur vorgesehen; sie ist bereits integriert und gegen die gemeinsamen Testvektoren geprüft, und eine erste dedizierte Installation ist der nächste Schritt.

Gemeinsame Known-Answer-Testvektoren legen jedes Primitiv und jede Kodierung implementierungsübergreifend fest: Go erzeugt sie, der TypeScript-Client reproduziert jeden deterministischen Wert Byte für Byte, und der Dart-Code der Mobil-App prüft die v2-Vektoren.

6.3 Sitzungs-Handshake

1. Der Client öffnet `/ws` und sendet seinen Identitätsschlüssel.
2. Der Server antwortet mit einer zufälligen Challenge, seinem Identitätsschlüssel und einer Signatur über die Challenge und den Schlüssel des Clients. Der Client prüft den Schlüssel gegen seine Pins und verifiziert die Signatur.
3. Der Client signiert die Challenge und den Schlüssel des Servers und sendet Kapselungsmaterial für den hybriden KEM seiner Suite.
4. Beide Seiten leiten mit domänengetrenntem HKDF einen Sitzungsschlüssel ab. Von da an wird jeder Frame verschlüsselt und signiert und trägt eine Sequenznummer.

Eine v2-Identität erhält den v2-Handshake; eine v3-Identität erhält den v3-Schlüssel des Servers und den zusammengesetzten KEM. Handshakes sind pro Adresse und pro Konto ratenbegrenzt.

7. Schutz von Inhalten

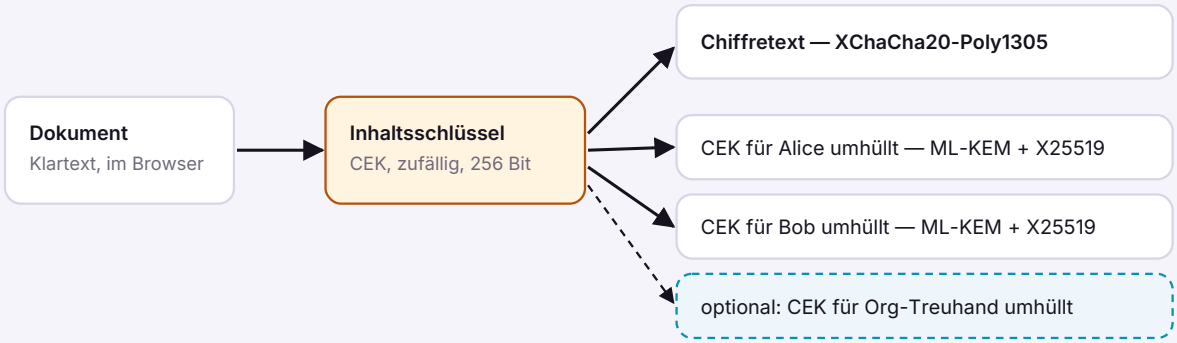


Abbildung 2 — Inhalte werden einmal verschlüsselt; ihr Schlüssel wird für jeden Leser separat umhüllt. Kommt ein Leser hinzu, wird der Schlüssel erneut umhüllt; wird einer entfernt, wird seine Umhüllung gelöscht. Der Server speichert nur die rechte Spalte.

7.1 Umschlagverschlüsselung

Inhalte in uOS werden unter einem zufälligen Inhaltsschlüssel verschlüsselt, und dieser Schlüssel wird für jede Person, die sie lesen darf, *umhüllt* – mit dem hybriden KEM gekapselt. Vier Muster decken das Produkt ab:

- **Schlüssel pro Ressource** für Dateien, Dokumente, Tabellen, Präsentationen, Boards, Projekte und Kalendereinträge. Beim Teilen wird der Schlüssel für den neuen Leser umhüllt; beim Widerruf wird die Umhüllung gelöscht, und der Server setzt die Berechtigung beim Download durch. Der Widerruf eines Lesers löst eine **Neuverschlüsselung** der Ressource aus: Der Client des Eigentümers erzeugt einen neuen Inhaltsschlüssel, verschlüsselt den aktuellen Inhalt neu (bei Dateien ein neuer Blob, bei Boards jedes Feld, jeder Kommentar und jeder Anhang), umhüllt den Schlüssel für genau die verbleibenden Leser – und gegebenenfalls für den Treuhandschlüssel –, und der Server wendet Widerruf, Chiffretext und Umhüllungen in einer einzigen Transaktion an, die eine Schlüsselversion pro Ressource hochzählt.

Speichervorgänge und Live-Kollaborations-Frames unter einer alten Schlüsselversion werden abgewiesen. Ein entfernter Leser kann weiterhin Kopien lesen, die er vor dem Widerruf erhalten hat, aber nichts, was danach gespeichert wird. Board-Felder sind über Associated Data an ihr Board und ihr Feld gebunden, sodass Chiffretext nicht zwischen Datensätzen verschoben werden kann.

- **Kopien pro Empfänger** für Chat, E-Mail und Benachrichtigungen, bei denen jeder Empfänger eine einzeln verschlüsselte Kopie erhält.
- **Arbeitsbereichsschlüssel** für Inhalte, die für jedes Mitglied einer Plattform bestimmt sind, etwa den uLine-Feed. Browser erzeugen jede *Schlüsselepoche*, verteilen sie an die Mitglieder und rotieren den Schlüssel, wenn ein ehemaliges Mitglied noch die aktuelle Epoche besitzt.
- **Bereichsschlüssel** in uShift, wo die Bereiche Personal, HR und alle Mitglieder jeweils eigene Epochenschlüssel haben und persönliche Datensätze für den Bereich und für jede beteiligte Person versiegelt werden.

Die Live-Kollaboration in uDoc und uSheet verschlüsselt jedes Update und jede Awareness-Nachricht unter dem Dokumentschlüssel mit frischen Nonces; der Server weist Kollaborations-Frames im Klartext zurück.

7.2 Große Dateien

Große Dateien werden als Chiffretext über einen ticketgebundenen HTTP-Pfad hochgeladen: Der Client erhält über die WebSocket-Verbindung ein vom Server signiertes, kurzlebiges Ticket und lädt damit die verschlüsselte Datei hoch. Dateigrößen werden auf Stufen in Zweierpotenzen aufgefüllt und Datei-Zeitstempel auf die Stunde vergröbert, sodass die Speicher-Metadaten weniger über die Inhalte verraten.

8. Privatsphäre beim Nachrichtenaustausch

8.1 Versiegelter Absender (Sealed Sender)

In uChat reist die Identität des Absenders *innerhalb* des verschlüsselten Umschlags jedes Empfängers, signiert mit dem Identitätsschlüssel des Absenders und gebunden an die Unterhaltung, eine Zufallszahl (Nonce) und die Sendezeit. Jeder Empfänger prüft die Signatur gegen das Key-Transparency-Log; eine Kopie, die die Prüfung nicht besteht, wird markiert und niemandem zugeordnet, und eine Bearbeitung zählt nur, wenn der ursprüngliche Verfasser sie signiert hat. Die Nachrichtentabelle speichert keinen Absender: Das Bearbeiten oder Löschen einer Nachricht für alle wird durch ein Geheimnis pro Nachricht autorisiert, das nur die Geräte des Verfassers ableiten können und von dem der Server nur einen Hashwert aufbewahrt. Nachrichten, die vor der Einführung dieses Verfahrens gesendet wurden, tragen keine Signatur; sie werden wie bisher angezeigt und können nicht mehr bearbeitet werden. Die Zustellung wird mit Einmal-Zustellungstokens autorisiert, die über RSA-Blindsignaturen nach RFC 9474 bezogen werden (RSABSSA-SHA384-PSS-Randomized, eigens dafür bestimmter 3072-Bit-Schlüssel), sodass der Server prüfen kann, ob ein Token gültig und noch nicht verbraucht ist, ohne es dem Konto zuordnen zu können, das es bezogen hat. Tokens tragen eine vom Server vergebene, auf volle Stunden ausgerichtete Ablaufzeit von höchstens 12 Stunden; Einträge über verbrauchte Tokens, die nur zur Verhinderung von Wiedereinspielung (Replay) aufbewahrt werden, werden nach 24 Stunden gelöscht. Blockieren und das Melden von Missbrauch funktionieren auf Grundlage dieses Modells. Benachrichtigungen halten nicht fest, wer wen kontaktiert hat – nur Kontaktanfragen und Beitrittshinweise speichern einen Absender, weil genau diese

Beziehung ihr Gegenstand ist –; ein Datenbank-Trigger erzwingt dies für jeden schreibenden Zugriff, und Benachrichtigungen werden 30 Tage nach ihrer Erstellung oder 7 Tage nach dem Lesen gelöscht. Serverprotokolle der Nachrichtenzustellung nennen den Absender nie zusammen mit der Unterhaltung.

Der versiegelte Absender beseitigt den gespeicherten Absendergraphen; er verbirgt nicht, wer zu einem bestimmten Zeitpunkt verbunden ist. Der Server sieht weiterhin, welche authentifizierte Sitzung wann eine Nachricht übermittelt, und er kennt die Mitglieder jeder Unterhaltung. Abschnitt 13.2 führt auf, was sichtbar bleibt.

8.2 Vorwärtssicherheit

Jedes Messaging-Gerät eines Nutzers registriert einen eigenen, vom Konto signierten Schlüssel und veröffentlicht signierte, ablaufende Prekeys (ML-KEM + X25519). Nachrichten werden für die Geräte-Prekeys des Empfängers verschlüsselt, sodass eine spätere Kompromittierung eines Kontoschlüssels Nachrichten auf dem Übertragungsweg nicht offenlegt. Damit der Verlauf auf allen Geräten eines Nutzers lesbar bleibt, legt ein Gerät von jeder empfangenen Nachricht eine Kopie auf dem Server ab, neu verschlüsselt für den Kontoschlüssel. Der gespeicherte Verlauf ist daher durch den Kontoschlüssel geschützt, nicht durch Vorwärtssicherheit: Wer das Konto entsperrt – etwa indem er dessen Geheimnisse anhand des umhüllten Bündels errät (Abschnitt 5.1) –, kann den gesamten gespeicherten Verlauf lesen. Vorwärtssicherheit schützt Nachrichten während der Zustellung. Ein Schild-Abzeichen im Chat zeigt an, wann Vorwärtssicherheit aktiv ist.

8.3 Chatklassen

Organisationsplattformen unterscheiden **Direktchats**, **Gruppenchats**, **Kanäle** und **vertrauliche Chats**. Kanäle können nur von Administratoren erstellt werden und sind als treuhandfähig gekennzeichnet; vertrauliche Chats sind als solche gekennzeichnet, können nie der Treuhand unterstellt werden und ihre Klasse nach dem Festlegen nicht mehr ändern.

8.4 Präsenz

Der Online-Status wird mit einer Lebensdauer von 90 Sekunden im Arbeitsspeicher gehalten; Zeitstempel der letzten Aktivität verfallen nach 24 Stunden, „Zuletzt gesehen"-Zeitstempel nach 30 Tagen. Nichts zur Präsenz wird in der Datenbank gespeichert.

8.5 Meetings

uMeet betreibt seine Verbindungsserver selbst. Jeder Anruf, auch Anrufe, die aus einem Chat in uChat gestartet werden, nutzt einen Medienrouter (SFU) innerhalb des uOS-Servers, der das Video jedes Teilnehmers in bis zu drei Qualitätsstufen weiterleitet, sodass jeder Bildschirm nur die Größe erhält, die er anzeigt. Ein Relay (TURN), ebenfalls Teil des uOS-Servers, überträgt Anrufe für Teilnehmer, deren Netzwerk direkte Verbindungen blockiert. Es verweigert jede Weiterleitung an private, Loopback- oder interne Adressen. Kein Dritter ist an einem Anruf beteiligt.

Router und Relay übertragen Medien, die sie nicht lesen können. Jeder Teilnehmer verschlüsselt jeden Audio- und Videoframe mit einem **Absenderschlüssel**, bevor er den Browser verlässt (AES-256-GCM, wobei die ersten Bytes des Codecs lesbar bleiben, damit der Router Frames weiterleiten kann). Ein Teilnehmer sendet seinen Schlüssel jedem anderen Teilnehmer einzeln, verschlüsselt mit dem hybriden KEM für den per Schlüsseltransparenz verifizierten Schlüssel dieser Person und signiert mit seinem Identitäts-

schlüssel. Tritt jemand bei, schaltet jeder Absender seinen Schlüssel per Ratchet weiter, sodass der Neuankömmling nichts entschlüsseln kann, was vorher gesagt wurde. Verlässt jemand das Meeting oder wird entfernt, wechselt jeder Absender zu einem frischen Zufallsschlüssel. Chat im Anruf, Reaktionen und gehobene Hände werden unter denselben Schlüsseln verschlüsselt. Der Server setzt Entscheidungen des Gastgebers durch (Einlass aus dem Warteraum, Entfernen, Sperren, Stummschalten), indem er verschlüsselte Medien weiterleitet oder zurückhält. Er erfährt nie, was gesagt oder gezeigt wurde. Das Pegelsignal, das verraten würde, wer spricht, wird nicht an den Server gesendet; der Browser ermittelt anhand des entschlüsselten Audios, wer spricht.

Ein Absenderschlüssel authentifiziert die Gruppe, nicht den einzelnen Frame: Wer den Schlüssel eines Teilnehmers besitzt, könnte grundsätzlich Frames dieses Teilnehmers fälschen, wie beim SFrame-Design der IETF. Die Schlüssel selbst sind durch Signaturen an Identitäten gebunden.

Gäste. Ein Gastgeber oder Co-Gastgeber kann Personen ohne uOS-Konto mit einem Gastlink einladen. Der geheime Teil des Links steht hinter `#`, sodass der Browser ihn nie mit der Seitenanfrage sendet. Der Server speichert nur dessen Hash, ein Ablaufdatum und ein Nutzungslimit. Der Browser eines Gastes erzeugt temporäre Identitäts- und Verschlüsselungsschlüssel für genau dieses eine Meeting und öffnet eine Verbindung, die nichts außer diesem Meeting erreichen kann. Gäste warten immer, bis ein Gastgeber oder Co-Gastgeber sie einlässt, und erst dann senden die Teilnehmer ihnen ihre Absenderschlüssel. Von da an sind Medien und Nachrichten des Gastes Ende-zu-Ende-verschlüsselt wie die aller anderen. Die Schlüssel eines Gastes stehen nicht im Schlüsseltransparenz-Protokoll, und der Name eines Gastes ist nur das, was er eingegeben hat; deshalb kennzeichnet uMeet Gäste als nicht verifiziert: Der Gastgeber entscheidet, wer sie sind. Aus demselben Grund kann ein Gast die anderen Teilnehmer nicht verifizieren. Lässt der Gastgeber den Link von uOS per E-Mail versenden, sieht der Server diesen Link, der jemanden jedoch nur bis in den Warteraum bringt.

9. Was geschützt ist — pro Anwendung

Die Tabelle gibt für jede Anwendung an, was Ende-zu-Ende-verschlüsselt ist und was der Server sehen kann. „Sichtbar“ heißt nicht veröffentlicht: Es heißt, dass der Server die Angabe für den Betrieb benötigt und sie offenlegen könnte, wenn er kompromittiert oder dazu gezwungen würde.

Anwendung	Ende-zu-Ende-verschlüsselt	Für den Server sichtbar
uChat	Nachrichtentext, Anhänge, Thread-Titel, Identität des Absenders	Mitgliedschaft in Unterhaltungen, Zeitstempel, Größen, Chatklasse, ein undurchsichtiges Autorisierungs-Tag pro Nachricht
uMail	Betreff und Text, pro Empfänger	Absender, Empfänger, Ordner, Gelesen- und Stern-Markierungen, Zeitstempel
uMeet	Titel und Beschreibung des Meetings; jeder Audio- und Videoframe, Chat im Anruf, Reaktionen und gehobene Hände (Schlüssel pro Absender, übermittelt mit dem hybriden KEM)	Wer wen wann anruft; wer an einem Meeting teilnimmt; Netzwerkadressen; Größe und Zeitpunkt der verschlüsselten Medien, die der Router weiterleitet

Anwendung	Ende-zu-Ende-verschlüsselt	Für den Server sichtbar
Dateien	Inhalte, Datei- und Ordnernamen, in Bildern und Scans erkannter Text (die Texterkennung läuft im Browser; der Text ist unter dem Dateischlüssel verschlüsselt)	Ordnerstruktur, aufgefüllte Größe, auf die Stunde vergröberter Zeitstempel, Freigabebeziehungen, ob eine Datei erkannten Text hat und welche Modellversion ihn erzeugt hat
uDoc, uSheet, uPresent	Inhalte, Titel und Live-Bearbeitungen	Eigentümer, Leser, Zeitstempel
Boards	Alle Felder privater Boards	Struktur und Kartenreihenfolge; öffentliche Boards sind bewusst lesbar
Kalender	Titel, Beschreibung, Ort, Besprechungsvorschläge	Beginn und Ende, Wiederholung, Teilnehmer
uNotes, uTodo	Titel, Texte, Ordner- und Kategorienamen	Reihenfolge, Fälligkeitsdaten, Erledigt-Status
Kontakte	Die Kontaktliste selbst (Abfragen über schlüsselabhängige Hashes, verschlüsselte Einträge)	Offene Kontaktanfragen
Vivian, uSages, uAssistant, Projekte	Unterhaltungen, Denkprozesse (Reasoning), Werkzeugergebnisse, Anhänge, Projektinhalte	Zeitstempel, gewähltes Modell; Inhalte werden während einer Anfrage an den gewählten KI-Anbieter gesendet (Abschnitt 12)
uLine	Beiträge, unter dem Arbeitsbereichsschlüssel	Autor, Zeitpunkt, Antwortstruktur
uShift	Dienstpläne, Verträge, Anträge, Wünsche, Zeiteinträge	Teams, Rollen, Zeiträume, Antragsstatus, Genehmigungskette, Erinnerungszeiten
Einstellungen	Präferenzen	Sprache, Design, Farben, Skalierung und zuletzt genutzte Plattform, die vor dem Entsperren benötigt werden
Profil	—	Name, Benutzername, Bio, Standort und Avatar, die Nutzer nach eigener Wahl für andere veröffentlichen

Anwendung	Ende-zu-Ende-verschlüsselt	Für den Server sichtbar
Studio	Build-Unterhaltungen mit der KI (Eingaben, Antworten, Denkprozesse, Werkzeugergebnisse), Projektnamen und -beschreibungen, unter einem Schlüssel, den nur die Mitglieder des Projekts besitzen; Werte, die jede App für einen Nutzer speichert; Titel, Text und Daten von App-Mitteilungen und Erinnerungen, vom Absender signiert; die Einträge, die eine App über ihre App-Datenbank teilt, unter einem Schlüssel je Bereich (alle auf der Plattform, die die App installiert haben, eine App-Rolle oder ein App-Raum), den nur die Personen in diesem Bereich besitzen und der ersetzt wird, wenn jemand den Zugriff verliert; Nachrichten und gemeinsamer Zustand in den Echtzeit-Räumen einer App, unter dem Schlüssel des Raumbereichs (ein neuer, nachdem jemand gegangen ist; Nachrichten unter einem alten weist der Server zurück)	App-Code, Manifeste und Assets (bewusst: Die KI bearbeitet sie auf dem Server, und sie werden geprüft und an andere ausgeliefert); wer an einem Projekt arbeitet; Name und Beschreibung einer App, sobald sie getestet oder eingereicht wird; Eingaben und Code werden während einer Anfrage an den gewählten KI-Anbieter gesendet (Abschnitt 12); bei der App-Datenbank Bereich, Sammlungsnamen, Eintrags-IDs, Versionen, Größen und Zeiten, wer eine App-Rolle hat, und dass die Plattform entscheidet, wer zu einem Bereich gehört; bei Räumen Raumnamen, wer in welchem Raum ist, wann und wie viel gesendet wird, und Einladungstexte; bei Mitteilungen die App, die Empfänger, die Stufe und die Zeit sowie, wann jede Erinnerung fällig ist
uBrowse, uNews	—	Öffentliche Inhalte, auf Anfrage abgerufen

Studio-Apps. In Studio erstellte Apps laufen in einem abgeschotteten Frame mit undurchsichtigem Ursprung unter der Content-Security-Policy von uOS und können daher keine externen Server erreichen; alles andere, was sie nutzen, läuft über den uOS-Host im Browser. Der Host gibt jeder App einen Speicher pro Nutzer, die App-Datenbank und die Räume aus der Tabelle oben sowie Dialoge und Hinweise, die uOS selbst mit dem Namen der App zeichnet. Alles Weitere – Mitteilungen senden, Erinnerungen setzen, Kalender, Aufgaben, Kontaktliste, Dateien, E-Mails, Dokumente und die Daten der anderen Apps des Nutzers – muss die App in ihrem Manifest angeben; bevor sie andere erreicht, gibt ein Mitglied mit dem Recht „Apps verwalten“ sie zusammen mit dem Code frei; und jeder Nutzer muss es auf einer Zustimmungskarte erlauben, die an genau diesen Code und dieses Manifest gebunden ist; eine Änderung fragt erneut. Der Host entschlüsselt solche Daten im Browser und übergibt sie der App in ihrer Sandbox. Mitteilungen werden für den im Schlüsseltransparenz-Protokoll nachgewiesenen Schlüssel jedes Empfängers oder einmal für einen Bereich der App-Datenbank versiegelt und vom Absender signiert, sodass der Server sie weder lesen noch fälschen kann; er sieht die App, die Empfänger, die Stufe und die Zeit sowie, wann eine Erinnerung fällig ist. Zum Erstellen bietet Studio einen Code-Arbeitsbereich, einen Versionsverlauf, der wie der Code lesbar gespeichert wird, einen Komponentenbaukasten im uOS-Stil und sechs Start-Apps; sein KI-Assistent ist optional (Abschnitt 12).

uMail ist uOS-intern. Es stellt nur zwischen uOS-Konten zu; es sendet keine E-Mails an Internet-E-Mail-Adressen und empfängt keine von dort, sodass keine uMail-Nachricht über SMTP läuft. „Externe Empfänger“ in der Richtlinie einer Organisation für ausgehende E-Mails (Abschnitt 10.3) sind uOS-Konten außerhalb der Plattform dieser Organisation. Die einzigen E-Mails, die der Dienst versendet, sind Einladungen und optionale Sicherheitshinweise (Abschnitt 15), die der Server verfasst.

Benachrichtigungen werden als opake Wecksignale gesendet; das Gerät ruft den Inhalt selbst ab und entschlüsselt ihn. Web Push muss ausdrücklich aktiviert werden (Opt-in) und überträgt keine Nachrichteninhalte.

TEIL III — ORGANISATIONEN, WIEDERHERSTELLUNG UND KI

10. Organisationen

10.1 Mitgliedschaft und Einladungen

Jedes Mitglied tritt über eine Einladung bei, und jede Einladung ist von ihrem Ersteller signiert; die Einladungskette wird aufgezeichnet, sodass der Weg jedes Mitglieds in einen Arbeitsbereich zurechenbar ist. Auf Organisationsplattformen können nur Inhaber, Administratoren und HR einladen. Die Rollen reichen von Inhaber und Superuser über Administrator und Moderator bis zu Mitglied und Gast, mit Berechtigungen pro Plattform.

Was jede Rolle darf, legt eine Rechte-Matrix pro Plattform fest, die Inhaber und Inhaber des Rechts *Rollen verwalten* bearbeiten können: Rollen verwalten, Plattform verwalten, Treuhand verwalten, Sicherheit verwalten, Apps verwalten, Inhalte moderieren, Mitglieder anzeigen und Mitglieder einladen. Inhaber haben immer alle Rechte, und der Server prüft die Matrix bei jeder Anfrage. *Sicherheit verwalten* – standardmäßig Inhaber, Superuser und Administrator – steuert die Sicherheitsrichtlinie der Organisation (Abschnitt 10.5) und braucht weder das Plattform- noch das Treuhandrecht; die Compliance-Kontrollen aus Abschnitt 10.3 brauchen *Plattform verwalten*, der Treuhandschlüssel *Treuhand verwalten*.

10.2 Governance

Kritische Änderungen, etwa Übertragungen der Inhaberschaft und Legal Holds, können ein Quorum von Inhabern erfordern. Jede Zustimmung ist eine ML-DSA-Signatur, jeweils von einem anderen Inhaber, die der Server prüft und für spätere Audits zusammen mit der Aktion aufbewahrt.

Jede Änderung der Sicherheitsrichtlinie (Abschnitt 10.5) wird in derselben Transaktion in eine Audit-Tabelle der Richtlinie geschrieben (`platform_policy_audit` : das handelnde Konto, die Richtlinie, der alte und der neue Wert, die Zeit) und im Protokoll der Verwaltungs-App neben Rollenänderungen, Inhaberwechseln und Verifizierungen angezeigt.

10.3 Compliance-Kontrollen

Das **Compliance-Profil** einer Organisation ist ein signiertes, versioniertes Dokument mit einer Historie, die nur ergänzt werden kann (append-only). Es steuert, welche Anwendungen Mitglieder nutzen können (auf dem Server durchgesetzt, nicht nur in der Oberfläche), Nur-Ansicht- und Download-Beschränkungen, Wasserzeichen in Dateien, Aufbewahrungsregeln, die über eine geprüfte Löschwarteschlange ausgeführt werden, sowie eine Richtlinie für ausgehende E-Mails, die E-Mails an Empfänger außerhalb der Organisation zulässt, davor warnt oder sie blockiert. Ein **Legal Hold**, per Inhaberquorum eingerichtet und aufgehoben, blockiert das Löschen der erfassten Inhaltstypen. Organisationen erfassen ihre Rechtsordnung; deutsche Organisationen bestätigen ihre Vereinbarung mit dem Betriebsrat.

10.4 Schlüsseltreuhand für Organisationen

Organisationen müssen Geschäftsunterlagen wiederherstellen können, wenn ein Mitarbeiter ausscheidet oder den Zugang verliert; Privatpersonen müssen sich darauf verlassen können, dass niemand sonst das kann. uOS bietet beides, getrennt voneinander:

- Eine Organisation kann einen **Treuhandschlüssel** erstellen. Er wird im Browser des Inhabers erzeugt, der ihn erstellt – der daher in diesem Moment den vollständigen privaten Schlüssel sieht –, und seine private Hälfte wird mit Shamir-Secret-Sharing unter benannten Inhabern und Anteilsinhabern aufgeteilt, wobei jeder Anteil für den jeweiligen Anteilsinhaber verschlüsselt wird. Beim Rotieren des Treuhandschlüssels wird jeder erfasste Inhaltsschlüssel unter einem neuen Schlüssel neu umhüllt.
- Für die Inhaltstypen, die die Organisation auswählt – Dateien, Dokumente, Tabellen, Boards, Kalender, Notizen, Aufgaben und E-Mail –, wird der Inhaltsschlüssel zusätzlich für den Treuhandschlüssel umhüllt (Abbildung 2).
- Die Wiederherstellung erfordert ein Quorum von Anteilsinhabern in ihren eigenen Browsern. Der Betreiber hält keinen Anteil und kann nicht mitwirken.

Treuhand gibt es nur auf Organisationsplattformen. Auf persönlichen und Kollaborationsplattformen fehlt sie – sie ist dort nicht bloß abgeschaltet –, und sie erfasst nie Direkt-, Gruppen- oder vertrauliche Chats. Mitglieder werden informiert, wenn Treuhand greift. Jede Datei, jedes Dokument, jede Tabelle, jedes Board, jeder Kalendereintrag, jede Notiz, jede Aufgabe und jede E-Mail, deren Inhaltsschlüssel für die Treuhand umhüllt ist, trägt in Listen und im geöffneten Element die Kennzeichnung „Durch Ihre Organisation wiederherstellbar". Erstellungsformulare zeigen „Wird durch Ihre Organisation wiederherstellbar sein", wenn die Plattform den Typ erfasst, und Chat-Kanäle tragen die Kennzeichnung „Treuhandfähig".

Der Treuhandschlüssel ist signiert und protokolliert. Jeder Treuhandschlüssel wird als Erklärung (Statement) veröffentlicht, die Plattform, Schlüssel, Schwellenwert, Anteilsinhaber, Zeitpunkt und den Hash der vorherigen Erklärung nennt. Die Erklärung wird mit dem Identitätsschlüssel des Inhabers signiert, der den Schlüssel festgelegt hat, und im Schlüsseltransparenz-Protokoll erfasst, und zwar in derselben Transaktion, die den Schlüssel speichert. Bevor die App eines Mitglieds etwas für die Treuhand umhüllt, verlangt sie drei Dinge:

- einen Inklusionsnachweis, dass die Erklärung der neueste Treuhandeintrag der Plattform ist;
- eine Signatur, die sich verifizieren lässt;
- den Nachweis, dass der signierende Schlüssel ein Kontoschlüssel im Protokoll ist.

Sie weicht nie auf einen Schlüssel aus, den der Server liefert. Bei jeder Anmeldung prüfen die Apps der Inhaber die gesamte Kette der Erklärungen der Plattform und zeigen jede Änderung an, die sie nicht selbst vorgenommen haben, samt der Person, die sie signiert hat. Um Treuhandkopien für einen eigenen Schlüssel zu erhalten, müsste ein Server ein Inhaberkonto anlegen und damit signieren, und beides erschiene im Protokoll, wo die echten Inhaber es sehen. Schlüssel, die vor Einführung der Erklärungen festgelegt wurden, erhalten keine neuen Treuhandkopien, bis ein Inhaber sie bestätigt.

10.5 Sicherheitsrichtlinie

Organisationsplattformen haben eine Sicherheitsrichtlinie mit zwei Regeln. Inhaber und das Recht *Sicherheit verwalten* lesen und ändern sie; jede Änderung wird protokolliert (Abschnitt 10.2).

Gemerkte Browser: nicht erlaubt, nur mit PIN oder uID, oder PIN, uID oder angemeldet bleiben (die Voreinstellung). Die Obergrenze eines Mitglieds ist die strengste Einstellung aller Organisationsplattformen, denen es angehört. Der Server setzt sie durch, wenn ein Browser registriert und wenn er wieder geöffnet wird, und löscht Registrierungen oberhalb der Grenze; der Client vergisst solche Browser nach der Anmeldung (Abschnitt 5.6).

uID zum Öffnen dieser Plattform verlangt. Ist diese Regel eingeschaltet, lässt sich die Plattform nur aus einer Sitzung nutzen, die mit einer uID begonnen hat. Der Server kann nicht sehen, wie ein Seed-Bündel geöffnet wurde; er markiert eine Sitzung deshalb nur aufgrund dessen, was er selbst geprüft hat:

1. Eine Anmeldung mit uID (`/api/core/uid/open`) und das Wiederöffnen eines gemerkten Browsers, dessen Faktor eine uID ist (`/api/core/device_unlock/open`), prüfen beide eine nutzerverifizierte Assertion einer uID, die zum Konto gehört. Beide geben dann einen *uID-Nachweis* zurück: 32 Zufallsbytes, die zwei Minuten lang unter SHA-256("uOS/uid/proof/v1|" ‖ Nachweis) in Redis liegen und an das Konto gebunden sind.
2. Der Browser legt den Nachweis einmal bei `/api/core/auth_proof` vor. Nachdem die Identitätssignaturen geprüft sind und das Konto als aktiv erkannt ist, verbraucht der Server ihn (Lesen und Löschen in einem Schritt, ob er passt oder nicht) und legt die nur im Arbeitsspeicher gehaltene Sitzung mit der Methode *uid* nur dann an, wenn der Nachweis für dieses Konto ausgestellt wurde. Ein wiederverwendeter, abgelaufener, fremder oder erfundener Nachweis ergibt eine gewöhnliche Sitzung; die Anmeldung selbst scheitert daran nie.
3. Der WebSocket-Handshake trägt die Kennung dieser HTTP-Sitzung; der HTTPS-Fallback legt sie als Bearer-Token vor. Der Socket übernimmt die Markierung nur, wenn diese Sitzung besteht, zu der Identität gehört, die den Handshake gerade signiert hat, und markiert ist. Die Kennung verknüpft nur; authentifiziert wird durch die Identitätssignatur. Ein erneuter Verbindungsaufbau behält die Markierung, solange die HTTP-Sitzung besteht (24 Stunden).

Die Regel wird pro Plattform durchgesetzt. Jedes Ereignis einer nicht markierten Sitzung wird nach der Plattform beurteilt, die es nennt – seiner `platform_id` , sonst der Plattform, für die der Socket geöffnet wurde –, und für eine Plattform mit der Regel mit `uid_required` abgewiesen. Handler, die die Plattform über eine Ressource bestimmen, etwa eine Datei oder ein Board, weisen sie in derselben Mitgliedschaftsprüfung ab. Was der Nutzer braucht, um wieder hineinzukommen, bleibt erreichbar: die Plattformliste, die Anwesenheit, die eigenen Einstellungen und das eigene Profil, die Verwaltung von uIDs und gemerkten Browsern, das Key-Transparency-Protokoll und Benachrichtigungen. Die anderen Plattformen des Nutzers sind nicht betroffen, und der Client zeigt statt der Apps der Plattform einen Hinweis. Anmeldungen mit den drei Geheimnissen, das Wiederöffnen mit der PIN oder auf der Stufe „angemeldet bleiben“ und Sitzungen des Desktop-Agenten (Geräte) sind keine uID-Sitzungen; ein Agent kann auf einer solchen Plattform daher nicht arbeiten. Der Server hält die Regel höchstens 15 Sekunden im Cache.

Die Regel lässt sich nur aus einer uID-Sitzung einschalten; so hat die Person, die sie setzt, gezeigt, dass sie die Plattform weiterhin öffnen kann. Eine Ausnahme für Inhaber gibt es nicht, und solange die Regel gilt, braucht auch jede weitere Änderung der Richtlinie eine uID-Sitzung. Sie ist eine Zugangsregel, die der Server durchsetzt, keine kryptografische Schranke: Ein Mitglied, das sich anders angemeldet hat, hält seine Schlüssel weiterhin im Browser, und die Mitteilungen, die der Server an einen Nutzer schickt (Broadcasts), filtert sie nicht. Uploads über HTTP brauchen Tickets, die über den WebSocket ausgegeben werden, und diese unterliegen der Regel. Die Regel gibt es nur auf Organisationsplattformen.

11. Wiederherstellung

Da der Betreiber Konten nicht wiederherstellen kann, richten Nutzer die Wiederherstellung selbst ein und werden direkt nach der Registrierung dazu aufgefordert:

- **Shamir-Wiederherstellungsanteile.** Der Browser teilt die Geheimnisse des Kontos in *n* Anteile auf, von denen beliebige *k* sie rekonstruieren – 2 von 3, 3 von 5 (empfohlen) oder 4 von 7. Anteile können vertrauenswürdigen Personen übergeben (für deren Schlüssel verschlüsselt), ausgedruckt, als QR-Codes aufbewahrt oder in einem Passwortmanager abgelegt werden. Der Server speichert Metadaten über die Anteile, nie die Anteile selbst; die Notiz des Nutzers, wo ein Anteil liegt, ist für den eigenen Schlüssel des Nutzers versiegelt.
- **Wiederherstellungsdatei.** Eine herunterladbare Datei mit den Anteilen und einer Kopie des umhüllten Seed-Bündels, sodass das Konto auch dann wiederhergestellt werden kann, wenn der Server es nicht mehr bereitstellen könnte. Da sie alle Anteile enthält, genügt die Datei allein, um das Konto wiederherzustellen; der Nutzer kann sie mit einer Passphrase von mindestens 20 Zeichen schützen (Argon2id, 256 MiB), die nach Entscheidung des Betreibers optional ist und beim Herunterladen angeboten wird. Einzelne Anteile werden ungeschützt heruntergeladen, da ein Anteil allein nichts verrät. Beim Import wird die Datei gegen den öffentlichen Schlüssel des Kontos geprüft und abgelehnt, wenn irgendetwas nicht übereinstimmt.

Ein Konto, dessen Geheimnisse und Wiederherstellungsmaterial beide verloren sind, kann nicht wiederhergestellt werden – und das ist so gewollt. Ein Dienst, der Ihre Identität für Sie wiederherstellen kann, kann sich auch als Sie ausgeben.

12. Künstliche Intelligenz

uOS bietet KI im gesamten Arbeitsbereich: den Assistenten Vivian mit Werkzeugen für Kalender, Dateien und Dokumente; uSages-Personas; die Hilfe uAssistant; ein Assistentenfenster in jeder App; den Build-Assistenten im Studio; und den Desktop-Agenten für lokale Arbeit.

KI-Unterhaltungen, Denkprozesse, Werkzeugergebnisse und Anhänge werden für den Schlüssel des Nutzers **Ende-zu-Ende-verschlüsselt gespeichert**; Build-Unterhaltungen im Studio werden unter einem Projektschlüssel verschlüsselt, den nur die Mitglieder des Projekts besitzen und der für jedes Mitglied nach einer Prüfung per Schlüsseltransparenz verpackt wird. Um eine Antwort zu erzeugen, muss jedoch der relevante Kontext an ein Sprachmodell gesendet werden. Für diese Anfrage leitet der Server den Kontext im Arbeitsspeicher weiter, ohne ihn zu speichern oder zu protokollieren, und der **vom Nutzer oder von der Organisation gewählte Anbieter** verarbeitet ihn im Klartext. Dass die Weiterleitung weder speichert noch protokolliert, ist eine Eigenschaft der Serversoftware – eine Richtlinie, keine kryptografische Garantie: Ein kompromittierter Server könnte den Kontext von KI-Anfragen aufzeichnen. Wird ein Modell lokal über den Desktop-Agenten ausgeführt, entfällt die Weiterleitung vollständig. KI ist optional: Eine Organisation kann die KI-Funktionen für ihre Plattform abschalten, und jeder Nutzer entscheidet bei der ersten Anmeldung oder später in den Einstellungen, ob seine Inhalte überhaupt an einen KI-Anbieter gesendet werden dürfen; der Server lehnt jede Anfrage ab, die Inhalte an einen Anbieter senden würde, solange nicht beides es erlaubt. Anbieter sind pro Plattform und pro Nutzer konfigurierbar – gehostete Anbieter in den USA, der EU und anderswo, ein eigener Endpunkt oder ein Modell, das **lokal** über den Desktop-Agenten läuft; in diesem Fall verlässt nichts den Rechner des Nutzers. Die Oberfläche zeigt an, welcher Anbieter eine Anfrage erhält.

Der Desktop-Agent führt lokale Aktionen nur innerhalb der Pfade und Befehlsregeln aus, die der Nutzer erlaubt, mit Bestätigung bei sensiblen Vorgängen. Er besitzt einen eigenen Geräteschlüssel, wird mit einem Einmal-Token gekoppelt und verifiziert jeden Frame, den der Server sendet. Bei der Kopplung pinnt er den Identitätsschlüssel des Kontos seines Besitzers: Der Kopplungscode, den der Nutzer aus dem Browser kopiert, enthält einen Fingerabdruck dieses Schlüssels, sodass ein vom Server untergeschobener Schlüssel abgelehnt wird. Werkzeuge, die schreiben, löschen oder ausführen – Shell-Befehle, das Erstellen und Bearbeiten von Dateien, Abrufe von Adressen im lokalen Netzwerk –, laufen erst, nachdem der Besitzer genau diese Aktion im Browser genehmigt hat; der Browser signiert die Genehmigung (Werkzeug, Arbeitsverzeichnis, jedes Argument, Kennungen, Entscheidung, Zeitpunkt) mit dem Identitätsschlüssel des Kontos, und der Agent verifiziert sie gegen den gepinnten Schlüssel und lehnt veraltete, wiedereingespielte, veränderte oder unsignierte Genehmigungen ab. Wechsel des Arbeitsverzeichnisses und Anfragen, eine Webadresse im lokalen Browser zu öffnen, werden auf dieselbe Weise signiert, und der Pin folgt einer Schlüsselrotation nur über Einträge, die mit beiden Schlüsseln signiert sind. Der Server leitet Genehmigungen weiter, kann sie aber nicht erstellen.

Automatische Genehmigung. Für längere Aufgaben bietet der Bestätigungsdialog zwei Möglichkeiten, nicht mehr nachzufragen, solange eine KI-Anfrage dieses Tabs läuft. *Pro Anfrage* genehmigt der Nutzer weitere Aktionen einer Art – Befehle, Dateiänderungen oder Zugriffe auf das lokale Netzwerk – für diese eine KI-Anfrage, höchstens 15 Minuten lang; das endet mit der Anfrage, und eine Bestätigung gilt nur so lange als Teil davon, wie sie die einzige laufende KI-Anfrage dieses Tabs ist. *Alle Aktionen automatisch genehmigen, bis ich es beende* genehmigt jede Art für jede KI-Anfrage, die dieser Tab startet, ohne Zeitgrenze; das wird nur im Arbeitsspeicher des Tabs gehalten, nie im Browserspeicher, und endet, wenn der Nutzer es beendet, sich abmeldet oder sperrt, das Konto oder die Plattform wechselt oder den Tab schließt. In beiden Modi signiert der Browser jede Genehmigung weiterhin selbst und protokolliert sie als automatisch. Manche Bestätigungen fragen immer nach oder werden abgelehnt: Ein Schreibzugriff auf einen Ort, der später von selbst Programme startet – Startdateien der Shell, Autostart-Einträge, Launch Agents, geplante Aufgaben, Git-Hooks und Ähnliches –, zeigt immer den Dialog; Befehle, die einem gesperrten Muster entsprechen, und Aktionen, die der Browser nicht lesen kann, werden abgelehnt, bevor einer der Modi überhaupt herangezogen wird; und eine Bestätigung, die eintrifft, während keine KI-Anfrage dieses Tabs läuft – aus einem anderen Tab oder von einem anderen Gerät, aus einer Automatisierung oder vom Server erfunden –, zeigt immer den Dialog.

Agent-Updates werden nur installiert, wenn ihr Manifest mit dem in den Agenten einkompilierten uOS-Release-Schlüssel (ML-DSA-87 + Ed25519) signiert ist, und führen nie zu einem Downgrade. Nur lesende Werkzeuge innerhalb des Arbeitsverzeichnisses laufen weiterhin ohne Genehmigung, und ihre Ergebnisse erreichen den Server und die KI; Verzeichnisse mit Zugangsdaten (zum Beispiel `~/.ssh`) werden nie gelesen.

TEIL IV — VERTRAUENSWÜRDIGKEIT UND NACHWEISE

13. Bedrohungsmodell

13.1 Wogegen uOS ausgelegt ist

- **Kompromittierung des Dienstes.** Eine vollständige Kopie der Datenbank und des Blob-Speichers liefert Chiffretext, öffentliche Schlüssel und Metadaten – keine privaten Schlüssel und keine Inhalte der verschlüsselten Kategorien (Abschnitt 9).
- **Innentäter.** Ein Mitarbeiter des Betreibers mit vollem Zugriff auf die Produktionsumgebung erhält dasselbe.
- **Rechtlicher Zwang zur Herausgabe gespeicherter Daten.** Der Betreiber kann nur herausgeben, was er besitzt: Kontometadaten, Plattformmitgliedschaften, Einladungsdatensätze und die Metadaten aus Abschnitt 9. Verschlüsselte Inhalte kann er nicht herausgeben. Inhalte der Organisations-Treuhand können nur die Anteilsinhaber der Organisation wiederherstellen.
- **Angreifer im Netzwerk**, die Datenverkehr beobachten oder aufzeichnen, einschließlich eines künftigen Angreifers, der das heutige TLS brechen kann: Sitzungen und Inhalte sind durch die Post-Quanten-Schichten geschützt, und ein laufender Client authentifiziert den Server anhand gepinnter Schlüssel. (Ein Angreifer, der TLS kontrolliert, während die App geladen wird, ist eine andere Bedrohung: siehe die Auslieferung manipulierter Clients in 13.2.)
- **Harvest now, decrypt later.** Inhalte und Sitzungsschlüssel sind durch hybride Post-Quanten-Schlüsselkapselung geschützt.
- **Richtlinienänderung des Betreibers.** Gespeicherte Inhalte bleiben unabhängig von künftigen Richtlinien vertraulich, weil der Betreiber keine Schlüssel besitzt.
- **Phishing bei der Anmeldung mit uID.** Eine uID antwortet nur dem Ursprung und der Relying Party von uOS; eine nachgeahmte Website erhält weder eine Assertion noch die PRF-Ausgabe, die das Seed-Bündel öffnet (Abschnitt 5.7). Die Datenbank allein enthält nur Boxen, die den Authenticator brauchen, die öffentlichen Schlüssel der Authenticatoren und Salts.

13.2 Was gemindert, aber nicht beseitigt wird

- **Auslieferung manipulierter Clients.** Eine Webanwendung wird von ihrem Betreiber über TLS ausgeliefert. Ein Betreiber – jemand, der ihn dazu zwingt, oder jeder, der in diesem Moment TLS kontrolliert – könnte einem Nutzer veränderten Code ausliefern, einschließlich anderer gepinnter Schlüssel. Jedes Release-Manifest ist mit dem uOS-Release-Schlüssel signiert, und ein Prüfer kann kontrollieren, ob das, was ein Host ausliefert, damit übereinstimmt (Abschnitt 14.1); dadurch wird Manipulation erkennbar, doch ein Nutzer, der nicht prüft, ist bei diesem Aufruf nicht geschützt.
- **Metadaten.** Der Server sieht, wer zu welcher Unterhaltung, welchem Arbeitsbereich und welchem Meeting gehört und wann Aktivität stattfindet. Versiegelter Absender, Padding, vergröberte Zeitstempel und kurzlebige Präsenzdaten verringern dies; sie beseitigen es nicht.
- **Schlüsselverteilung.** Das Unterschieben fremder Schlüssel wird durch das Schlüsseltransparenz-Protokoll erkannt (Abschnitt 5.5), außer wenn der Server verschiedenen Nutzern, deren Sichten nie zusammentreffen, unterschiedliche Protokolle zeigt oder im Namen einer Person ein neues Konto erfindet; der Vergleich von Sicherheitsnummern deckt beides ab. Treuhandschlüssel von Organisationen haben denselben Schutz und dieselbe Grenze: Ein Treuhandschlüssel, der von einem erfundenen Inhaberkonto signiert wurde, wird von den echten Inhabern erkannt, nicht verhindert (Abschnitt 10.4).

- **Offline-Raten von Geheimnissen.** Ein Angreifer mit einer Kopie der Datenbank kann offline versuchen, das umhüllte Bündel oder den Anmelde-Prüfwert zu erraten (Abschnitt 5.1). Argon2id macht jeden Rateversuch teuer; ein erratbares Geheimnis kann es nicht retten. Konten, die sich seit Einführung der Anmeldeschranke nicht angemeldet haben, geben ihr Bündel bis zu ihrer nächsten Anmeldung weiterhin heraus.
- **Desktop-Agent.** Ein kompromittierter Server kann weiterhin die nur lesenden Werkzeuge des Agenten innerhalb des Arbeitsverzeichnisses auslösen (Abschnitt 12). Ohne die Signatur des Besitzers kann er keine Befehle ausführen, keine Dateien ändern und keine Webadressen öffnen. Solange der Besitzer die automatische Genehmigung eingeschaltet hat (Abschnitt 12), könnte ein kompromittierter Server der laufenden KI-Anfrage Aktionen der genehmigten Arten hinzufügen und sie vom Browser signieren lassen, ausgenommen Schreibzugriffe auf Autostart-Orte, gesperrte Muster und unlesbare Aktionen.
- **Phishing.** Wer an die Geheimnisse eines Nutzers gelangt, kann sich als dieser anmelden. Eine uID ist dem nicht in dieser Weise ausgesetzt (Abschnitt 13.1).
- **Eine gestohlene uID samt PIN.** Wer eine uID besitzt und ihre PIN kennt – oder ein entsperrtes Smartphone mit einem synchronisierten Passkey hat –, hat wie mit den drei Geheimnissen vollen Zugriff auf das Konto, bis der Nutzer diese uID aus einer anderen Sitzung entfernt, das Konto sperrt oder die Geheimnisse ändert. Ohne ihre PIN oder ihr biometrisches Merkmal berechnet der Authenticator nichts und sperrt sich nach seiner eigenen Fehlversuchsgrenze selbst.
- **Gemerkte Browser.** Ein Browser, den der Nutzer sich merken lässt, ist so stark wie seine Stufe: Auf der PIN-Stufe erlauben ein gestohlenes Gerät und eine Kopie der Serverdatenbank zusammen ein Offline-Erraten der PIN (nicht, wenn der Faktor eine uID ist); auf der Stufe „Angemeldet bleiben“ ist das entsperrte Gerät selbst der Schlüssel, bis die Registrierung abläuft oder vergessen wird (Abschnitt 5.6).
- **„uID verlangt“ ist eine Zugangsregel.** Eine Organisation, die eine uID verlangt, verlässt sich darauf, dass der Server sie durchsetzt (Abschnitt 10.5). Ein kompromittierter Server könnte eine Sitzung ohne uID auf die Plattform lassen; den Inhalt könnte er trotzdem nicht lesen, denn er bleibt für die Schlüssel der Mitglieder verschlüsselt.
- **Urheberschaft im Chat im Moment der Änderung.** Der Server speichert keinen Verfasser, sieht aber, wenn ein Verfasser eine Nachricht für alle bearbeitet oder löscht, welche Sitzung die Urheberschaft dieser Nachricht nachweist.
- **KI-Anfragen.** Der gewählte KI-Anbieter sieht den Kontext jeder Anfrage, und dass die Weiterleitung nichts protokolliert, ist eine Richtlinie des Servers (Abschnitt 12).
- **Verfügbarkeit.** Ein kompromittierter oder unter Zwang stehender Dienst kann die Bereitstellung von Daten verweigern. Lesen kann er sie nicht.

13.3 Außerhalb des Umfangs

Ein kompromittiertes Gerät oder ein kompromittierter Browser des Nutzers, physischer Zwang gegen den Nutzer sowie Schwächen der zugrunde liegenden Standards selbst. Symmetrische Primitive mit 256-Bit-Schlüsseln gelten als sicher gegen Quantenangriffe; ändert sich diese Einschätzung, ist der Suite-Mechanismus (Abschnitt 6.2) der Migrationspfad.

14. Transparenz und Überprüfung

14.1 Überprüfung des Clients

Alles, was den Nutzer schützt, läuft im Browser und kann dort untersucht werden. Der Quellcode des Clients liegt im Repository github.com/uOS-sh/uOS unter der UltimaOS Source-Available License 1.0; das Studio-SDK und die Studio-App-Vorlagen stehen unter der MIT-Lizenz. Der Code ist einsehbar (source-available), aber nicht Open Source. Das Repository ist derzeit privat: Ausgewiesene Sicherheitsexperten, Auditoren und Forscher erhalten auf Anfrage Lesezugriff, und nach der Beta soll das Repository für alle geöffnet werden. Wer Zugriff hat, darf den Code lesen, bauen und testen, um die Aussagen dieses Papiers zu prüfen; ihn als Dienst zu betreiben, erfordert unsere schriftliche Genehmigung. Über die Untersuchung hinaus wird jedes Release deterministisch gebaut – wiederholte Builds derselben Quellen erzeugen identische Dateien – und mit einem Manifest veröffentlicht, das den SHA-384-Hash jeder Datei enthält (`/release-manifest.json`), während die Seite ihren Code mit Subresource Integrity lädt. Jedes Manifest ist mit dem uOS-Release-Schlüssel signiert – einem hybriden Schlüssel aus ML-DSA-87 + Ed25519, der außerhalb der ausliefernden Infrastruktur aufbewahrt wird und dessen öffentliche Hälfte im Repository liegt und dessen Schlüssel-ID hier abgedruckt ist (`666MWqG6o0cwguPHVNYgspQDJ9bLtMvUTk/UxeA1YoY=`), sodass sich eine Kopie des Schlüssels, die man aus dem Repository oder von uns erhält, überprüfen lässt. Ein Prüfwerkzeug im Repository (`scripts/verify-release.mjs`) prüft das signierte Manifest eines Hosts gegen diesen Schlüssel und vergleicht die ausgelieferten Dateien Byte für Byte. Der verbleibende Schritt ist die Veröffentlichung des Manifest-Hashes jedes Releases in einem unabhängig betriebenen Protokoll, sodass die Auslieferung unterschiedlicher Releases an unterschiedliche Nutzer öffentliche Spuren hinterlässt.

14.2 Der Server

Der Quellcode des Servers ist nicht öffentlich. In dieser Architektur ist der Server nicht der Vertrauensanker des Nutzers: Die Garantien aus Abschnitt 13.1 gelten gegenüber einem vollständig böswilligen Server. Was der Server annehmen und senden kann, ist durch den Client und sein Protokoll festgelegt, die Prüfer untersuchen können (Abschnitt 14.1). Sicherheitsforscher, Auditoren und Wissenschaftler können für eine Prüfung einen eingegrenzten Zugang zum Server-Quellcode beantragen.

14.3 Entwicklungsdisziplin

Jedes Release durchläuft automatisierte Prüfstufen, bevor es in die Produktion gelangt: Known-Answer-Tests für jedes Primitiv über alle Implementierungen hinweg; eine Abdeckungsprüfung (`scripts/check-rotation-coverage.sh`), die beide Deployment-Skripte gegen das aktuelle Schema der Zieldatenbank ausführen und die fehlschlägt, wenn eine Spalte mit einem Schlüssel oder mit für ein Konto verschlüsselten Daten von der Identitätsrotation nicht behandelt wird, wenn eine Anweisung der Rotation nicht mehr zum Schema passt oder wenn die Kontolöschung nicht dieselben Tabellen abdeckt; sowie die Testsuiten für Authentifizierung, Kryptografie, Server und HTTP. Das Produktions-Release wird verweigert, wenn eine Prüfstufe fehlschlägt. Der Client wird auf einer unterstützten Version seines Frameworks gehalten – Stand Oktober 2026 Angular 21 –, und zum Datum dieses Papiers lagen für die an den Browser ausgelieferten Abhängigkeiten keine bekannten Sicherheitswarnungen vor (`npm audit --omit=dev` meldete 0).

14.4 Betriebsmodelle

Organisationen wählen, wo uOS läuft:

- **Die Plattform uos.sh**, von uns betrieben, auf der Organisationen neben privaten Nutzern ihre eigenen Plattformen anlegen.
- **In eigenen Räumlichkeiten** (On-Premises), von der Organisation auf ihrer eigenen Infrastruktur betrieben, mit demselben Code, demselben Client und denselben Release-Manifesten – und, wo erforderlich, mit der CNSA-Suite.

Die Prüfkette funktioniert in beiden Fällen gleich: Jedes Release wird mit einem Manifest seiner Dateien veröffentlicht, sodass eine Organisation, die uOS selbst betreibt, prüfen kann, dass das, was sie bereitstellt, genau dem veröffentlichten Release entspricht, und ihre Nutzer prüfen können, was ihr Browser erhält. Bei einer On-Premises-Bereitstellung betreibt die Organisation, nicht wir, den Server; die Ende-zu-Ende-Garantien aus Abschnitt 13.1 gelten dann in gleicher Weise gegenüber diesem Betreiber.

15. Privatsphäre und Datenschutz

- **Minimale Registrierung.** Keine Telefonnummer, kein bürgerlicher Name, keine Identitätsprüfung. Eine E-Mail-Adresse ist optional; wird sie angegeben, wird sie unter einem Serverschlüssel verschlüsselt gespeichert und nur für die Sicherheits- und Servicenachrichten verwendet, die der Nutzer aktiviert.
- **Kein Tracking.** Keine Tracking-Cookies, keine Werbekennungen, keine Analysewerkzeuge Dritter, kein Geräte-Fingerprinting und keine externen Schriftarten oder Skripte in der Anwendung.
- **Personenbezogene Daten, genau benannt.** Nach der DSGVO können öffentliche Schlüssel, Verbindungsmetadaten und kurzlebige Logs personenbezogene Daten sein. uOS minimiert sie konstruktionsbedingt und wahrt die Betroffenenrechte für die Daten, die es besitzt.
- **Löschen eines Kontos.** Ein Nutzer kann sein Konto in den Einstellungen löschen. Eine einzige Transaktion entfernt das Konto und seine Daten, einschließlich seiner eigenen versiegelten Kontaktdatensätze (gefunden über die Hashes aller Schlüssel, die das Konto je hatte), seiner eigenen Empfängerkopien von Nachrichten und der für das Konto versiegelten Kopien in Kalenderterminen und Terminanfragen, Meeting-Einladungen und Chat-Thread-Titeln anderer Personen; seine verschlüsselten Dateiinhalte werden unmittelbar danach aus dem Speicher entfernt. Kopien, die andere Personen erhalten haben, bleiben bei ihnen. Wer eine Plattform besitzt, muss zuerst die eigenen Plattformen löschen; bis dahin wird die Löschung abgelehnt, und nichts wird entfernt. Die Prüfstufe aus Abschnitt 14.3 stellt sicher, dass die Löschung jede Tabelle abdeckt, die auch die Rotation abdeckt.
- **Hosting.** Der Dienst uos.sh wird von Infomaniak in der Schweiz gehostet, einem Land, für das ein Angemessenheitsbeschluss der Europäischen Kommission nach der DSGVO besteht. Unterauftragsverarbeiter sind in einem veröffentlichten, versionierten Verzeichnis aufgeführt: der Hosting-Anbieter, KI-Anbieter nur, wenn ein Nutzer sie auswählt, und Push-Dienste nur, wenn ein Nutzer sie aktiviert. Organisationen können uOS stattdessen in eigenen Räumlichkeiten betreiben (Abschnitt 14.4). Eine 2025 vorgeschlagene Revision der Schweizer Überwachungsverordnung (VÜPF) würde Identifikations- und Vorratsdatenspeicherungspflichten auf weitere Online-Dienste ausdehnen. Unter jeder Rechtsordnung kann der Betreiber nur herausgeben, was er besitzt – die Metadaten aus Abschnitt 9 –, weil bei der Registrierung keine Identität erhoben wird und Inhalte mit Schlüsseln verschlüsselt sind, die der Betreiber nicht hat. Wir verfolgen die Revision und werden den Hosting-Standort des Dienstes anpassen, falls ihre endgültige Fassung dies erfordert.

- **Anmeldeschlüssel.** Zu jeder uID speichert der Server die Credential-ID, ihren öffentlichen Schlüssel und Algorithmus, den Signaturzähler, ein zufälliges Salt, die versiegelte Box, die Bezeichnung des Nutzers und die Daten des Hinzufügens und der letzten Nutzung, außerdem pro Konto ein zufälliges User-Handle. Er erfährt, wie viele uIDs ein Konto hat, wie sie heißen und wann und von welcher Adresse jede benutzt wurde. Die PRF-Ausgabe, den Schlüssel der Box, das Seed-Bündel und die PIN des Authenticators erhält er nie; wegen der Attestierung `none` erfährt er das Modell des Authenticators nicht. Bei einer Anmeldung mit uID hält Redis höchstens zwei Minuten lang den Hash des einmaligen Nachweises, an das Konto gebunden, und die Sitzung im Arbeitsspeicher vermerkt nur, dass sie mit einer uID begonnen hat.
- **Rechtliche Anfragen.** Anfragen werden auf Zuständigkeit und Rechtsgrundlage geprüft, bevor irgendetwas herausgegeben wird, und aggregierte Zahlen werden für einen öffentlichen Transparenzbericht erfasst.

16. Umfang und Einschränkungen

uOS befindet sich in der Alpha. Die folgenden Einschränkungen sind bekannt, und an ihnen wird gearbeitet; sie werden hier genannt, damit kein Prüfer sie selbst entdecken muss.

- **Ältere Konten.** Eine kleine Zahl von Konten, die vor dem Identitätsmodell mit zufälligem Seed angelegt wurden, verwendet bis zu ihrer nächsten Anmeldung, die sie aktualisiert (Abschnitt 5.3), noch die alte Ableitung, bei der die Schlüssel direkt aus den Geheimnissen folgen. Bis dahin bleiben ihre Daten dem Offline-Raten der ursprünglichen Geheimnisse ausgesetzt. Inhalte, die sie früher erstellt haben und die jemand vor der Aktualisierung kopiert hat, bleiben in dieser Kopie offengelegt.
- **Ältere Boards und Beiträge.** Private Boards, die vor der Verschlüsselung auf Feldebene angelegt wurden, bleiben für den Server lesbar, bis ihr Eigentümer oder ein Administrator sie das nächste Mal öffnet. Dann werden sie automatisch verschlüsselt, einschließlich Kommentaren, Anhängen und Aktivitäten. Öffentliche Boards bleiben bewusst lesbar. Frühe uLine-Beiträge werden versiegelt, wenn ihr Autor den Feed das nächste Mal öffnet, und Studio-Unterhaltungen und -Projektnamen aus der Zeit vor ihrer Verschlüsselung, wenn der Autor des Projekts es das nächste Mal öffnet.
- **Meetings.** Meetings erfordern einen aktuellen Browser, der Medien-Frames verschlüsseln kann (Chrome, Edge, Firefox oder Safari). Ein Meeting fasst bis zu 50 Teilnehmer. Gäste ohne Konto sind nicht verifiziert (Abschnitt 8.5); wo WebSockets blockiert sind, nutzen sie denselben HTTPS-Long-Poll-Kanal wie Konten.
- **TLS.** uos.sh bietet die hybride TLS-Gruppe `X25519MLKEM768` an, die aktuelle Versionen von Chrome, Edge, Firefox und Safari verwenden. Ältere Browser fallen auf einen klassischen Schlüsselaustausch zurück. Sobald die App läuft, hängen weder Inhalte noch sensible Anfragen von TLS ab, weil die darüberliegenden Post-Quanten-Schichten beide schützen. Die Auslieferung der App selbst hängt dagegen davon ab (Abschnitt 13.2).
- **Client-Überprüfung.** Release-Manifeste sind signiert und überprüfbar, werden aber noch nicht in einem unabhängigen Protokoll veröffentlicht, und Browser prüfen sie nicht automatisch (Abschnitt 14.1).
- **Mobile Apps.** Die nativen Apps für iOS und Android befinden sich in Entwicklung und unterstützen die Suite v2; die Webanwendung ist heute der unterstützte Weg, uOS auf Smartphones zu nutzen.
- **Persönliche Profile** sind aufgrund einer bewussten Produktentscheidung für den Server lesbar, weil Nutzer sie für andere veröffentlichen.

- **Klassische Primitive.** Zustellungstokens für den versiegelten Absender verwenden RSA-Blindsignaturen nach RFC 9474; ihre Blindheit gilt bedingungslos, und ein Quantenangreifer könnte Tokens fälschen (mit Auswirkungen auf die Missbrauchsgrenzen), nicht aber ihre Unverknüpfbarkeit aufheben. Web Push verwendet die klassische Verschlüsselung des Web-Push-Standards und transportiert keine Inhalte. uID-Assertions sind klassische WebAuthn-Signaturen; sie authentifizieren eine einmalige Challenge und schützen keine Inhalte (Abschnitt 5.7).
- **Unterstützung für uIDs.** Eine uID braucht die WebAuthn-PRF-Erweiterung: Chrome und Edge ab 116 (Sicherheitsschlüssel mit `hmac-secret`, Passkeys im Google Passwortmanager und in Windows Hello in aktuellen Versionen), Safari ab 18 (Passkeys im iCloud-Schlüsselbund) und Firefox ab 135; ältere Browser und Authenticatoren werden beim Hinzufügen abgelehnt. Eine entfernte uID bleibt auf dem Authenticator, sofern der Browser ihm nicht mitteilen kann, den Berechtigungsnachweis zu vergessen; eine solche verwaiste uID öffnet nichts.
- **Studio-App-Daten.** Die Schlüsselhinterlegung der Organisation (Abschnitt 10.4) deckt die App-Datenbank noch nicht ab, sodass geteilte App-Einträge verloren gehen, wenn die letzte Person geht, die den Schlüssel eines Bereichs besitzt. Wer neu berechtigt ist, erhält den Schlüssel eines Bereichs erst, nachdem jemand, der ihn besitzt, die App öffnet. Mitteilungen an eine Person zeigen dem Server, dass diese Person die App installiert hat, und Erinnerungen wiederholen sich nur in festen Abständen.
- **Noch kein unabhängiges Audit.** Das kryptografische Design und die Implementierung wurden noch nicht von einem unabhängigen Auditor geprüft.

17. Roadmap

Die nächsten Schritte, nach Priorität geordnet. Alles in dieser Liste ist geplant und noch nicht verfügbar, sofern der jeweilige Punkt nichts anderes sagt.

1. Ein unabhängiges Sicherheitsaudit des kryptografischen Designs und der Implementierung, mit Veröffentlichung des Berichts.
2. Abschluss der Umstellung aller Konten auf v3-hybrid. Jede Anmeldung mit den drei Geheimnissen stellt das Konto bereits um (Abschnitt 6.2); als Nächstes folgt ein Datum, ab dem ein noch nicht umgestelltes Konto nichts anderes mehr tun kann als diese Umstellung, und danach das Entfernen der v2-Pfade zum Schreiben und Anmelden. Alte Signaturen lassen sich weiterhin prüfen und unter v2-Schlüsseln verbliebene Daten weiterhin öffnen, sodass der Verlauf lesbar bleibt.
3. Veröffentlichung des signierten Manifests jedes Releases in einem unabhängig betriebenen Protokoll sowie automatische Überprüfung durch den Desktop-Agenten oder eine Browsererweiterung.
4. Gossip-Abgleich der Schlüsseltransparenz mit unabhängigen Zeugen (Witnesses).
5. Angleichung der zusammengesetzten Konstruktionen an die IETF-Standards (eine künftige Suite).
6. Native mobile Apps auf der Suite v3-hybrid mit der Anmeldeschranke (heute sprechen sie v2) sowie eine erste dedizierte Installation mit der Suite v3-cnsa, die bereits integriert und per KAT getestet ist.
7. Studio Pro Stufe 2, Verbindungen (gebaut): Eine freigegebene Studio-App erreicht die externen Dienste, die ihr Manifest angibt, jeden vom Prüfer freigegeben und von jedem Nutzer auf der Einwilligungskarte erlaubt. Apps, die solche Dienste aus dem Browser aufrufen, laufen in einem Frame, der von einer eigenen Domain (`frame.uos-apps.net`) mit einer Content Security Policy pro freigegebener Version ausgeliefert wird, sodass die Richtlinie von uOS selbst geschlossen bleibt und uos.sh von diesen Anfragen nichts sieht; die Frame-Seite enthält keine App-Daten, erhält den Code per Fensternachricht

von der uOS-Oberfläche und prüft seinen Hash gegen die freigegebene Version. Dienste, die ein serverseitiges Token brauchen, lassen sich stattdessen über ein Relais auf uos.sh erreichen, das aus bleibt, bis ein Plattform-Admin es pro App einschaltet, begrenzt und nur mit Zählern protokolliert; der Server kann solche Anfragen lesen, was die Einwilligungskarte sagt. Tokens liegen in einem Tresor, den der Server nicht öffnen kann: versiegelt für den eigenen Schlüssel des Nutzers oder, wenn geteilt, für den Schlüssel einer App-Rolle, und vom Nutzer markiert, falls sie durch das Relais dürfen. Noch geplant: die Konnektor-Art von Geheimnis für die Geräte der Stufe 3.

8. Eigenständige Websites im Studio (gebaut): ein Projekttyp für schlichte Websites, die zur Laufzeit nicht von uOS abhängen und beim eigenen Hosting des Kunden veröffentlicht werden – als ZIP-Download oder per SFTP, FTPS oder FTP über den Desktop-Agenten auf dem Computer des Entwicklers, sodass Zugangsdaten zum Hosting uos.sh nie erreichen: Der Veröffentlichungsschlüssel des Agenten wird mit einem Sicherheitscode bestätigt und durch die Signatur des Inhabers gebunden, jedes Ziel und jede Veröffentlichung wird mit dem Identitätsschlüssel des Kontos signiert und vom Agenten geprüft, Host-Schlüssel werden angeheftet, und das signierte Veröffentlichungsprotokoll hält fest, was live ist – mit einer Vorschau der Änderungen, einer Prüfung auf am Host geänderte Dateien und der Möglichkeit, zu einem früheren Stand zurückzukehren. Pro Formular kann die Website ihre Einträge statt an ein E-Mail-Skript an einen Ende-zu-Ende-verschlüsselten Formulareingang senden: Der Browser des Besuchers versiegelt jeden Eintrag (auf eine feste Größe aufgefüllt) für den v3-hybrid-Schlüssel des Eingangs, den der Browser eines Inhabers erzeugt und signiert hat; uos.sh speichert nur Chiffretext und eine Größenklasse, wendet eine Herkunfts-Allowlist, einen Proof of Work und Ratenbegrenzungen an, und die Inhaber öffnen die Einträge im Studio; wird ein Inhaber entfernt, wird der Schlüssel gewechselt, mit einer Übergangsfrist für noch nicht erneut veröffentlichte Websites. Ein Ziel kann auch direkt aus dem Browser zu GitHub, GitLab.com oder Netlify veröffentlichen – über eine kleine uOS-Veröffentlichungsseite auf einer eigenen Domain (`publish.uos-apps.net`), deren eigene Content Security Policy diese drei Schnittstellen erreichen darf, während die Richtlinie von uOS selbst geschlossen bleibt; das Zugriffstoken wird im Browser einmal für jede Person versiegelt, die auf das Ziel veröffentlichen darf, für deren im Schlüsseltransparenz-Protokoll nachgewiesenen Schlüssel, sodass uos.sh nur Chiffretext speichert und nur weiß, wer eine Kopie hat. Cloudflare Pages baut aus dem Git-Repository. Noch geplant: direkter Upload zu Cloudflare Pages.
9. uID Phase 2: PKI-Smartcards, nationale eID-Karten und PIV/CAC-Token als zweite Art von uID, über ein optionales Client-Zertifikat auf einem eigenen Pfad, das gegen konfigurierte Zertifizierungsstellen und Sperrlisten geprüft wird, zuerst für Organisationen, die bereits eine PKI betreiben.
10. Studio Pro Stufe 3, Geräte (gebaut, wartet auf das externe Sicherheits-Review): Ein im Konnektor-Modus mit einer Plattform gekoppelter Desktop Agent führt die Worker freigegebener Studio-Apps neben den Geräten aus (HTTP, WebSocket, MQTT, Modbus TCP) in einer Sandbox, unter einer Richtlinie, die der Geräteverwalter mit dem Kontoschlüssel signiert und die der Agent gegen den festgepinnten Eigentümerschlüssel prüft. Messwerte sind unter einem Stream-Schlüssel versiegelt, den nur die Inhaber des App-Bereichs öffnen; Befehle signiert das Mitglied und versiegelt sie für den Konnektorschlüssel, der den festgepinnten Mitgliedsschlüssel, die Rolle, den Worker-Hash, eine Nonce, einen Zähler und die Zeit prüft, mit Vier-Augen-Prinzip für gefährliche Aktionen; Alarme wertet der Konnektor aus, sie erreichen die Benachrichtigungen der Empfänger mit einem generischen Push; das Audit ist für die Admins der App versiegelt. Der Server leitet weiter und speichert Chiffrat, weiß, welche Konnektoren es gibt und wann sie sich melden, und löst einen Offline-Alarm aus, wenn einer verstummt.

18. Geschäftsgrundsätze

- **Die private Nutzung ist kostenlos**, ohne Werbung, Datenverkauf oder Verhaltensprofile – in jeder Phase des Produkts.
- **Organisationen zahlen pro Nutzerlizenz**, mit Konditionen für gemeinnützige Organisationen und Bildungseinrichtungen, auf der Plattform uos.sh oder für eine Bereitstellung in eigenen Räumlichkeiten.
- **Unsere Einnahmen hängen nie davon ab, Inhalte zu lesen.** Verschlüsselte Inhalte können wir nicht lesen; Geschäftsmodell und Sicherheitsmodell weisen in dieselbe Richtung.
- **Während der Alpha** können Organisationen uOS im Gegenzug für strukturiertes Feedback kostenlos nutzen.

19. Fazit

uOS geht von einer Position aus, die die meiste Software als Zielkonflikt behandelt: dass sich ein vollständiger, kollaborativer Arbeitsbereich bauen lässt, ohne dass sein Betreiber ihn lesen kann. Post-Quanten-Kryptografie, Schlüssel in der Hand der Nutzer, wo möglich versiegelte Metadaten und eine klare Darstellung dessen, was sichtbar bleibt, machen diese Position konkret. Wir laden Organisationen, Sicherheitsprüfer und frühe Nutzer ein, sie gemeinsam mit uns zu erproben.

Kontakt: patrick@ultimaos.com · https://ultimaos.com · Registrierung und Anmeldung unter https://uos.sh

ANHÄNGE

A. Primitive und Parameter

Primitive	Standard	Verwendung in uOS
ML-KEM-768 / ML-KEM-1024	FIPS 203	Schlüsselkapselung (v2 / v3), stets mit X25519
ML-DSA-65 / ML-DSA-87	FIPS 204	Signaturen (v2 / v3); mit Ed25519 in v3 stets, in v2 bei Sitzungsrahmen und Gerätesignaturen
X25519, Ed25519	RFC 7748, RFC 8032	Klassische Hälften jeder hybriden Operation
XChaCha20-Poly1305	RFC 8439; draft-irtf-cfrg-xchacha	Authentifizierte Verschlüsselung, 256-Bit-Schlüssel, zufällige 192-Bit-Nonces
HKDF-SHA256 / HKDF-SHA384	RFC 5869	Schlüsselableitung, domänengetrennte Labels, an die Suite gebunden; für die uID-Box `uOS/uid/v1` und für gemerkte Browser `uOS/device-unlock/v1`
Argon2id	RFC 9106	Umhüllungsschlüssel aus den Geheimnissen (256 MiB, t = 3, p = 2); PIN-Hash; Passphrase der Wiederherstellungsdatei
HMAC-SHA256	RFC 2104	Gepfefferte Suche nach Kennungen; Abfragen der Kontaktliste
AES-256-GCM	FIPS 197, SP 800-38D	Medien-Frames und Nachrichten im Anruf in uMeet, unter Schlüsseln pro Absender
SHA-384	FIPS 180-4	Release-Manifeste, Subresource Integrity
RSA-Blindsignaturen	RFC 9474 (RSABSSA-SHA384-PSS-Randomized)	Unverknüpfbare Zustellungstokens für den versiegelten Absender
ML-DSA-87 + Ed25519	FIPS 204, RFC 8032	Signaturen von Releases und Updates des Desktop-Agenten (uOS-Release-Schlüssel)
Merkle-Baum (SHA-256)	RFC 6962	Schlüsseltransparenz-Protokoll der Kontoschlüssel
Shamir-Secret-Sharing	—	Wiederherstellungsanteile; Treuhandschlüssel der Organisation
WebAuthn-Assertions (Ed25519, ECDSA P-256, RSA PKCS#1 v1.5)	W3C WebAuthn Level 3; FIDO2 CTAP 2	uID-Schlüssel: Authentifizierung einer einmaligen Anmelde- oder Entsperr-Challenge; klassisch, schützt keine Inhalte
WebAuthn-PRF (HMAC-SHA-256 im Authenticator)	WebAuthn-PRF-Erweiterung; CTAP `hmac-secret`	uID-Schlüssel: 32-Byte-Geheimnis zur Eingabe `SHA-256("uOS/uid/prf/v1")`, das die uID-Box verschlüsselt, oder zu `SHA-256("uOS/uid/device-unlock/v1|" ‖ salt)` für den Faktor eines gemerkten Browsers
TLS 1.2 / 1.3	RFC 5246 / RFC 8446	Transport und Auslieferung der App; hybrides `X25519MLKEM768` unter TLS 1.3 angeboten

Die RSA-Blindsignaturen schützen die *Unverknüpfbarkeit* von Zustellungstokens, nicht die Vertraulichkeit irgendwelcher Inhalte.

B. Schlüsselgrößen nach Suite

Wert	v2	v3-hybrid
Öffentlicher KEM-Schlüssel	1184 B (+ 32 B X25519)	1600 B, zusammengesetzt
KEM-Chiffretext pro Empfänger	1088 B (+ 32 B)	1600 B, zusammengesetzt
Öffentlicher Signaturschlüssel	1952 B (+ 32 B Ed25519)	2624 B, zusammengesetzt
Signatur	3309 B (+ 64 B)	4691 B, zusammengesetzt

Die Größen beziehen sich auf den eigentlichen Wert; jeder v3-Wert trägt zusätzlich ein Byte zur Kennzeichnung der Suite (so umfasst ein öffentlicher v3-hybrid-KEM-Schlüssel bei der Übertragung 1601 Byte). v2-Werte behalten ihre ursprüngliche Kodierung ohne Kennzeichnung, sodass bestehende Daten unverändert bleiben.

C. Netzwerkschnittstellen

Pfad	Zweck
`/ws`	Sämtliche Nutzeraktivität: verschlüsselte, signierte Frames in einer Post-Quanten-Sitzung
`/api/ws/fallback/*`	Dieselbe Sitzung über HTTPS-Long-Polling, wo WebSockets blockiert sind; die Gastvariante (`/api/ws/fallback/guest/*`) erreicht nur das eigene Meeting
`/api/core/hello` , `/api/core/auth_proof` , `/api/core/kdf_params`	Anmelde-Challenge, Nachweis und Salt-Abfrage (Abfrage mit dem hybriden KEM versiegelt); `auth_proof` verbraucht außerdem einen einmaligen uID-Nachweis (Abschnitt 10.5)
`/api/core/kdf_unlock`	Herausgabe des umhüllten Bündels gegen den Anmeldeschlüssel (versiegelt)
`/api/core/set-pin` , `/api/core/verify-pin` , `/api/core/change-pin`	PIN-Verwaltung und -Prüfungen (versiegelt)
`/api/core/device_unlock/challenge` , `/api/core/device_unlock/open`	Erneutes Öffnen eines gemerkten Browsers: einmalige Challenge, dann der Anteil des Servers gegen den Nachweis des Browsers und auf der PIN-Stufe die PIN oder eine uID-Assertion, bei einem uID-Faktor zusätzlich ein einmaliger uID-Nachweis (versiegelt; Abschnitte 5.6, 10.5)
`/api/core/uid/challenge` , `/api/core/uid/open`	Anmeldung mit einer uID: einmalige WebAuthn-Challenge, dann Salt, versiegelte Box und ein einmaliger uID-Nachweis gegen eine geprüfte Assertion (versiegelt; Abschnitte 5.7, 10.5)

Pfad	Zweck
`/api/account/*`	Entsperren, Ändern der Geheimnisse und Identitätsrotation (versiegelt), einschließlich ticketgebundener Übertragung von Chiffretext
`/api/uploads/{ticket}`	Massen-Upload von Chiffretext mit einem vom Server signierten Einmal-Ticket
`/api/invites/*`, `/api/platform/*`	Öffentliche Einladungs- und Plattformseiten
`/api/profile`, `/api/server/config`	Profil-Fallback; öffentliche Serverschlüssel und Push-Schlüssel
`/api/contact`, `/api/contact/challenge`	Kontaktformular mit Proof of Work
`/api/forms/challenge`, `/api/forms/submit`	Einträge von im Studio gebauten Website-Formularen: im Browser des Besuchers für den Schlüssel des Eingangs versiegelt; Herkunfts-Allowlist pro Formular, Proof of Work, Ratenbegrenzung, aufgefüllte Größen (ADR-006 §B7)
`frame.uos-apps.net/f/{app}/{hash}`, `/p/{ticket}` **(eigene Domain)**	Die App-Frame-Seite von Studio-Apps mit externen Verbindungen: eine feste Bootstrap-Seite mit der Content Security Policy der freigegebenen Version; nur für eine freigegebene Version (Hash) oder ein einmaliges Vorschau-Ticket ausgeliefert; erhält den Code per Fensternachricht von der uOS-Oberfläche und enthält keine Nutzerdaten (ADR-006 §6.2)
`publish.uos-apps.net` **(statische Seite, eigene Domain)**	Die Veröffentlichungsseite des Studios: eine feste Seite mit eigener Content Security Policy, die eine Website aus dem Browser des Nutzers als Commit zu GitHub oder GitLab.com überträgt oder bei Netlify bereitstellt; das Studio öffnet sie in einem verborgenen Frame, übergibt ihr Dateien und Zugriffstoken per Fensternachricht; sie speichert nichts und liefert keine uOS-Daten aus (ADR-006 §B10.2)
`/s/*`, `/sdk/*`	Veröffentlichte Studio-Sites; App- und Asset-Bundles
`/healthz`	Lebenszeichen (Liveness)
UDP/TCP 3478 (STUN/TURN), UDP 40000–40999 (Relay), UDP/TCP 50000 (SFU)	Konnektivität, Relay und Medienrouter für uMeet; ausschließlich verschlüsselte Medien, kurzlebige Relay-Zugangsdaten, ausgegeben über `/ws`

D. Glossar

Kernkonto (Core Account) – die kryptografische Identität einer Person. **Plattform** – ein isolierter Arbeitsbereich mit eigenen Mitgliedern und Einstellungen. **Seed-Bündel** – das Zufallsmaterial, aus dem die Schlüsselpaare eines Kontos abgeleitet werden. **KEK** – Schlüsselverschlüsselungsschlüssel (key-encryption key), aus den Geheimnissen des Nutzers abgeleitet. **CEK** – Inhaltsschlüssel (content-encryption key) einer einzelnen Ressource. **Suite** – ein fester Satz von Algorithmen und Parametern. **Epoche** – eine Generation eines Schlüssels, die bei der Rotation ersetzt wird. **Treuhand (Escrow)** – ein von der Organisation gehaltener Wiederherstellungsschlüssel für ausgewählte Geschäftsinhalte. **Versiegelter Absender (Sealed Sender)** – Zustellung ohne gespeicherte Absenderidentität. **Vorwärtssicherheit (Forward Secrecy)** – Schutz vergangener Nachrichten, falls ein langfristiger Schlüssel später kompromittiert wird. **uID** – ein FI-

DO2-Sicherheitsschlüssel oder Passkey mit PRF-Erweiterung, der für die Anmeldung eine versiegelte Kopie des Seed-Bündels öffnet (Abschnitt 5.7). **uID-Sitzung** – eine Sitzung, die mit einer vom Server selbst geprüften uID-Anmeldung begonnen hat; Organisationsplattformen können sie verlangen (Abschnitt 10.5).

uOS

https://ultimaos.com · patrick@ultimaos.com

Registrierung und Anmeldung unter https://uos.sh

UltimaOS