C6P/IslandAccord/Convro Protocol Threat Model

Security Analysis for Auditors and Grant Reviewers

Convro Protocol Team

Version 1.0 — January 2026

Contents

1 Executive Summary

1.1 What is C6P/IslandAccord/Convro?

C6P (Convro 6 Protocol) is an end-to-end encrypted direct messaging protocol designed for device-to-device secure communication. The protocol consists of:

- **IslandAccord v1**: An authenticated prekey handshake using 3DH + optional 4DH with OTP
- **DM Ratchet**: A symmetric double-ratchet providing per-message forward secrecy
- **Device-based identities**: Ed25519 signatures and X25519 Diffie-Hellman key exchange

C6P is built on three foundational principles:

1. **Server is not trusted for secrecy**: The server routes messages and manages state but never derives or learns session keys or message content
2. **Deterministic crypto by design**: Nonces are derived deterministically from per-message key material, enabling reproducible behavior and eliminating randomness-related vulnerabilities
3. **Fail-closed as invariant**: Any validation failure, decryption failure, or state inconsistency causes immediate abort with no partial state updates

1.2 Scope of This Threat Analysis

This document analyzes threats to:

- Session establishment (handshake replay, MITM, OTP exhaustion)
- Message transport security (replay, reordering, counter manipulation)
- Server state machine integrity (invalid transitions, concurrency races)
- Cryptographic correctness (key derivation, AEAD, nonce uniqueness)

1.3 Explicitly Out of Scope

This threat model does **not** address:

- Endpoint compromise (malicious OS, malware, keyloggers)
- Physical device theft with unlocked screen
- Social engineering attacks on users
- Denial-of-service attacks (covered separately in operational security)
- Metadata analysis beyond protocol-level binding
- Quantum adversaries (acknowledged limitation; post-quantum migration planned for v2)

1.4 Key Security Properties

C6P guarantees the following within its defined threat model:

- **Confidentiality**: Only intended recipients can decrypt messages
- **Authentication**: Peers verify each other via Ed25519 signatures
- **Replay protection**: Both handshake and messages protected against replay attacks
- **State machine integrity**: Server enforces strict state transitions with fail-closed validation
- **OTP single-use guarantee**: One-time prekeys consumed exactly once via atomic state transitions
- **Deterministic reproducibility**: All cryptographic operations are deterministic and testable

2 System Overview & Trust Boundaries

2.1 2.1 Actors

Initiator (A) The device creating a DM session. Possesses long-term identity keys (`IK_sig`, `IK_dh`) and generates a fresh ephemeral key (`A_EK`) for each session.

Responder (B) The device accepting a DM session. Possesses long-term identity keys and rotating prekeys (SPK with 30-day rotation, OTP single-use).

Server (S) Message router and state authority. Stores offers/accepts as opaque blobs, enforces state machine transitions, manages OTP reservation lifecycle. **Never derives cryptographic secrets.**

Network Attacker Passive observer or active man-in-the-middle on transport layer. Can observe encrypted envelopes, metadata (session IDs, timestamps, sizes), and network topology.

2.2 2.2 Trust Boundaries

Client-Server Boundary - Server is **authoritative** for: routing, state machine enforcement, OTP scarcity management - Server is **not trusted** for: secrecy, cryptographic validation (signatures, KC tags) - Clients validate all cryptographic proofs locally; server failures are fail-closed

Client-Network Boundary - All message payloads encrypted end-to-end (ChaCha20-Poly1305 AEAD) - Metadata visible: session ID, message counter, timestamp, envelope size - Transport security (TLS) provides confidentiality against passive network observers but does not prevent malicious server from observing metadata

Internal Server Boundary (Database) - Session state (`dm_sessions`): sessionId, device IDs, state enum, offer/accept blobs (immutable), expiry timestamps - OTP state (`prekeys_otp`): otpId, status (AVAILABLE/RESERVED/PENDING_CONSUMPTION/CONSUMED), reservation TTL - State transitions are atomic; partial writes cause transaction rollback

2.3 2.3 Protected Assets

Session Secrecy Root key, chain keys, per-message keys must remain secret to both endpoints. Compromise reveals future messages within that session but not past messages (forward secrecy).

Forward Secrecy Per-message keys derived from chain key, which advances after each message. Compromise of current state does not reveal past messages.

OTP Scarcity One-time prekeys provide enhanced forward secrecy for session establishment. Server must guarantee each OTP consumed exactly once.

Session State Correctness State machine integrity prevents: replayed offers, double accepts, invalid transitions. Server enforces uniqueness of `(initiatorDeviceId, responderDeviceId, sessionId)` tuples.

Replay Resistance Consumed counter sets prevent duplicate message acceptance. Skip-window (2048 messages) bounds out-of-order acceptance without weakening replay guarantees.

3 Attacker Model

3.1 3.1 Attacker Capabilities

Passive Observer (Network) - Observe all encrypted traffic (ciphertext, AAD, session IDs, timestamps) - Record traffic for later analysis (harvest-now-decrypt-later) - Perform traffic analysis (timing, frequency, message sizes)

Active Man-in-the-Middle (Network) - Intercept, modify, replay, drop, or delay messages - Inject fake messages - Attempt downgrade attacks (protocol version, crypto suite) - Cannot break cryptographic primitives

Malicious Server - Control prekey bundle delivery (serve stale/incorrect SPKs) - Deny message delivery or delay arbitrarily - Attempt to cause state machine violations - Reorder messages within transport constraints - Observe all metadata (session IDs, device IDs, timestamps, message counts) - Cannot decrypt message content (lacks DH private keys)

Malicious Peer - Participate in protocol with intent to cause failures - Attempt replay attacks with previously valid messages - Send malformed envelopes to trigger parsing/validation errors - Cannot impersonate other users without their private keys

Compromised Device (Post-Handshake) - Extract all keys stored on compromised device - Decrypt future messages sent to/from that device - Cannot decrypt past messages (forward secrecy protects if keys already deleted) - Cannot compromise other devices in the same user account

3.2 3.2 What Attackers CANNOT Do

Break Cryptographic Primitives - Cannot find X25519 or Ed25519 private keys from public keys - Cannot forge Ed25519 signatures without private key - Cannot break ChaCha20-Poly1305 AEAD encryption - Cannot find HKDF collisions or reverse HKDF-Expand

Extract Keys from Uncompromised Device - Keys stored in platform secure storage (iOS Keychain with `kSecAttrAccessibleWhenUnlockedThisDeviceOnly`, Android KeyStore with hardware backing) - Biometric/passcode protection required for key access - Memory zeroization applied after cryptographic operations

Forge Cryptographic Proofs - Cannot generate valid SPK signatures without responder's `IK_sig` private key - Cannot generate valid offer signatures without initiator's `IK_sig` private key - Cannot compute correct key confirmation tags (KC1/KC2) without completing DH exchange

4 Security Goals

C6P achieves the following security properties:

1. **Confidentiality of DM content**: Messages encrypted with per-message AEAD keys derived from session chain keys; server and network observers cannot decrypt
2. **Authentication of peers**: Mutual authentication via Ed25519 signatures (SPK signature, offer signature) and bidirectional key confirmation (KC1/KC2)
3. **Replay protection (handshake + messages)**: Server enforces uniqueness of `(initiatorDeviceId, responderDeviceId, sessionId)` tuple; message layer enforces consumed counter sets per stream
4. **State machine integrity**: Server validates all transitions; clients validate cryptographic proofs; fail-closed design prevents partial state updates
5. **OTP single-use guarantee**: Atomic state transitions (RESERVED → PENDING_CONSUMPTION → CONSUMED) with database constraints prevent OTP reuse
6. **Deterministic reproducibility**: All key derivations, nonce generation, and AAD construction are deterministic; enables comprehensive test vector validation across implementations

5 Threat Enumeration Methodology

This threat model uses a **STRIDE-inspired** approach adapted for end-to-end encrypted protocols:

- **Spoofing**: Identity verification via signatures, key confirmation
- **Tampering**: AEAD tags, transcript binding, immutable server blobs
- **Repudiation**: Not a goal (signatures provide non-repudiation; deniability deferred to future version)
- **Information Disclosure**: Confidentiality via E2EE; metadata minimization as best-effort
- **Denial of Service**: Partially addressed (rate limits, TTLs, bounded skip-window); full DoS out of scope
- **Elevation of Privilege**: Server cannot derive secrets; strict role enforcement

Threats are grouped by protocol phase:

- **Handshake phase** (open/accept): Offer replay, OTP exhaustion, MITM, cross-device attacks
- **Server state machine**: Invalid transitions, concurrency races, terminal state violations
- **Message transport**: Replay, counter manipulation, reordering
- **Replay/concurrency**: Duplicate detection, skip-window enforcement, atomic persistence

Each threat includes:

- Attacker type and goal
- Attack path (step-by-step)
- Affected protocol phase
- Specific mitigation from protocol spec (with section references)
- Residual risk assessment
- Current status (prevented/detected/out-of-scope)

6 Threat Scenarios

6.1 Threat 1: Replayed Handshake Offer

Attacker: Network or Malicious Server **Goal:** Cause initiator to believe multiple sessions are established or trigger duplicate processing

Attack Path:

1. Attacker captures valid offer from initiator A to responder B
2. Attacker replays offer to server or directly to responder
3. Attempt to cause server to create duplicate session or responder to process offer twice

Affected Phase: Handshake (open)

Mitigation in Convro:

- Server enforces unique constraint on `(initiatorDeviceId, responderDeviceId, sessionId)` tuple (§4.2 island-accord-state-machine.md)
- Server implements idempotency: duplicate offer with matching blob returns existing session state; differing blob rejected with `C6P.HANDSHAKE.REPLAYED_OFFER` (§6.1 island-accord-state-machine.md)
- Server stores `offerBlob` as immutable after first write (§4.3 island-accord-state-machine.md)

Residual Risk: None — replay prevented by database uniqueness constraint and idempotent server behavior

Status: Prevented

6.2 Threat 2: Modified Offer Blob

Attacker: Network MITM or Malicious Server **Goal:** Alter offer contents to change session parameters or inject attacker keys

Attack Path:

1. Attacker intercepts offer from A to B
2. Attacker modifies fields: `ephemeralDhPub`, `transcriptHash`, `kc1`, or `offerSignature`
3. Server stores modified offer; responder receives tampered blob

Affected Phase: Handshake (open/accept)

Mitigation in Convro:

- Offer signature binds `transcriptHash` and all session parameters (§8.2 island-accord-crypto.md: signature over SHA-256 of label + transcript_hash + version + suite + sessionId + deviceIds)
- Responder verifies Ed25519 signature against initiator's `IK_sig_pub` (§8.3 island-accord-crypto.md)
- Responder independently recomputes transcript hash from offer fields and compares (§6.3 island-accord-crypto.md)
- Any mismatch causes immediate abort with `C6P.HANDSHAKE.INITIATOR_SIGNATURE_INVALID` or `C6P.HANDSHAKE.TRANSCRIPT_MISMATCH`

Residual Risk: None — cryptographic binding prevents tampering; server cannot forge valid signatures

Status: Prevented

6.3 Threat 3: SessionId Collision Attempt

Attacker: Malicious Peer or Malicious Server **Goal:** Force two different sessions to share the same sessionId, potentially enabling cross-session attacks

Attack Path:

1. Attacker generates offer with sessionId matching existing session between A and B
2. Attacker submits to server for storage
3. Attempt to cause state confusion or cross-session message delivery

Affected Phase: Handshake (open)

Mitigation in Convro:

- Server enforces `UNIQUE(initiatorDeviceId, responderDeviceId, sessionId)` database constraint (§9.1 island-accord-state-machine.md)
- SessionId is 8 bytes (64 bits), generated randomly by initiator from CSPRNG
- Collision probability negligible: 2^64 space, birthday bound at ~2^32 sessions between same device pair
- Server rejects duplicate tuple with `C6P.HANDSHAKE.STATE_VIOLATION` (§6.1 island-accord-state-machine.md)

Residual Risk: None — database uniqueness constraint enforced atomically

Status: Prevented

6.4 Threat 4: Accept Replay with Modified kc2

Attacker: Network MITM or Malicious Server **Goal:** Replay accept with incorrect key confirmation to cause initiator to activate session with wrong keys

Attack Path:

1. Responder sends valid accept with `kc2`
2. Attacker captures accept and modifies `kc2` value
3. Server stores modified accept; initiator receives tampered blob

Affected Phase: Handshake (accept)

Mitigation in Convro:

- Server stores `acceptBlob` as immutable after first write (§4.3 island-accord-state-machine.md)
- Accept idempotency: if session already ACTIVE and stored kc2 matches provided kc2, return OK; if kc2 differs, reject with `C6P.HANDSHAKE.STATE_VIOLATION` (§6.2 island-accord-state-machine.md)

- Initiator computes expected `kc2 = HMAC-SHA256(kc_key, kc_payload || "RESP")` and compares byte-for-byte (§9.2 island-accord-crypto.md)
- Mismatch causes abort with `C6P.HANDSHAKE.KEY_CONFIRMATION_FAILED`; session remains PENDING_HANDSHAKE locally

Residual Risk: None — key confirmation cryptographically binds both parties to derived keys

Status: Prevented

6.5 Threat 5: Cross-Device Accept Attempt

Attacker: Malicious Peer **Goal:** Accept an offer intended for device B1 using credentials from device B2 (same user, different device)

Attack Path:

1. Offer sent to responder device B1
2. Attacker (controlling device B2 of same user) fetches offer via API or database access
3. Attacker attempts to generate accept using B2's private keys

Affected Phase: Handshake (accept)

Mitigation in Convro:

- Server validates `responderDeviceId` in accept matches `responderDeviceId` in stored offer (§3.2 precondition 2, island-accord-state-machine.md)
- Reject with `C6P.HANDSHAKE.DEVICE_BINDING_MISMATCH` if mismatch
- Offer transcript binds `B_deviceId` (§6.2 field 7, island-accord-crypto.md)
- Accept requires loading SPK/OTP private keys for exact device referenced in offer (§10.2 responder checklist, island-accord-crypto.md)
- Transcript recomputation will fail if different device keys used (§6.3 island-accord-crypto.md)

Residual Risk: None — device binding enforced at both server and crypto layers

Status: Prevented

6.6 Threat 6: OTP Reuse Attempt

Attacker: Malicious Initiator or Malicious Server **Goal:** Use the same OTP for multiple session establishments, weakening forward secrecy

Attack Path:

1. Initiator A1 fetches bundle containing OTP
2. Server reserves OTP (status: RESERVED)
3. A1 calls open() with otpId, server moves OTP to PENDING_CONSUMPTION
4. Attacker A2 attempts to call open() with same otpId before A1's accept completes

Affected Phase: Handshake (open)

Mitigation in Convro:

- Server OTP lifecycle enforces single-use via atomic state transitions (§5 island-accord-state-machine.md)
- Once OTP moved to PENDING_CONSUMPTION, it cannot be returned by bundle fetch to another initiator (§4.4 invariant, island-accord-state-machine.md)
- Second open() attempt with same otpId fails: OTP status no longer RESERVED → reject with `C6P.HANDSHAKE.OTP_MISSING` or `C6P.KEYS.KEY_NOT_FOUND` (§3.1 precondition 6, island-accord-state-machine.md)
- OTP linked to session via `linkedSessionDbId` in same transaction as open() (§5.3 island-accord-state-machine.md)

Residual Risk: None — atomic state transitions and database constraints prevent reuse

Status: Prevented

6.7 Threat 7: OTP Race Between Initiators

Attacker: Multiple Malicious Initiators **Goal:** Two initiators fetch bundle simultaneously and both receive the same OTP

Attack Path:

1. Initiators A1 and A2 call GET `/v1/prekeys/bundle` simultaneously for responder B
2. Server selects same OTP for both requests before reservation commit
3. Both initiators receive identical otpId and otpPub
4. Both attempt to call open() with same OTP

Affected Phase: Prekey fetch + handshake (open)

Mitigation in Convro:

- Bundle fetch atomically reserves OTP: SELECT one AVAILABLE OTP + UPDATE status to RESERVED in single transaction (§5.2 island-accord-state-machine.md)
- Database isolation (READ COMMITTED or SERIALIZABLE) ensures only one transaction sees OTP as AVAILABLE
- Second concurrent fetch either: (a) receives different OTP, or (b) receives bundle with no OTP if pool exhausted
- OTP binding at open() time validates status is RESERVED and not expired (§5.3 island-accord-state-machine.md)

Residual Risk: Limited — race condition mitigated by atomic reservation; worst case one initiator fails at open() (acceptable)

Status: Prevented

6.8 Threat 8: OTP Replay After Expiration

Attacker: Malicious Initiator or Network **Goal:** Reuse an OTP that was previously reserved but expired without being consumed

Attack Path:

1. Initiator fetches bundle with OTP; server reserves OTP (status: RESERVED, expiresAt = now + 10min)
2. Initiator never calls open(); reservation expires
3. Server cleanup job moves OTP to EXPIRED status (§5.5 island-accord-state-machine.md)
4. Attacker attempts to call open() referencing expired otpId

Affected Phase: Handshake (open)

Mitigation in Convro:

- Server validates OTP state at open() time: status must be RESERVED and `now < expiresAt` (§3.1 precondition 6, island-accord-state-machine.md)
- Expired OTP fails validation → reject with `C6P.HANDSHAKE.OTP_MISSING`
- Recommendation: never reuse OTP IDs even after expiry; refill by minting new rows (§5.5 island-accord-state-machine.md)

Residual Risk: None — expired OTPs rejected at open()

Status: Prevented

6.9 Threat 9: OTP Prefetch Hoarding

Attacker: Malicious Initiator **Goal:** Reserve all available OTPs for a responder without consuming them, causing DoS for legitimate initiators

Attack Path:

1. Attacker calls GET `/v1/prekeys/bundle` repeatedly for target responder
2. Server reserves OTPs (status: RESERVED) with 10-minute TTL
3. Attacker never calls open(); OTPs locked during reservation window
4. Legitimate initiators receive bundles with no OTP → reduced forward secrecy

Affected Phase: Prekey fetch

Mitigation in Convro:

- OTP reservation TTL limited to 10 minutes (§7 island-accord-state-machine.md: `OTP_RESERVATION_TTL`)
- Expired reservations cleaned up by scheduled job (§5.5 island-accord-state-machine.md)
- Server-side rate limiting on `/fetch-bundle` per initiator device (§7 island-accord-state-machine.md: recommended `OPEN_RATE_LIMIT = 10/min`)
- OTP pool replenishment strategy: responder uploads new OTPs periodically (§4.5.2 threat-model-v1.md)

Residual Risk: Limited — short-term OTP exhaustion possible; mitigated by TTL cleanup and rate limits; full DoS prevention out of protocol scope

Status: Partially Mitigated (DoS mitigation relies on operational controls)

6.10 Threat 10: Invalid State Transition Injection

Attacker: Malicious Client or Compromised Server **Goal:** Force session into illegal state (e.g., PENDING → EXPIRED while accept in flight, or ACTIVE → PENDING)

Attack Path:

1. Session in PENDING state; responder about to accept
2. Attacker sends forged state transition request (e.g., manual DB update or API call to cancel/expire)
3. Session state changed to terminal (EXPIRED/CANCELLED) after accept already processed
4. Attempt to cause accept to be lost or state inconsistency

Affected Phase: Server state machine

Mitigation in Convro:

- Server state machine defines valid transitions explicitly (§2 island-accord-state-machine.md)
- Terminal states (ACTIVE, REJECTED, EXPIRED, CANCELLED, ABORTED) are final; no transitions allowed back to PENDING (§1.1 island-accord-state-machine.md)
- Accept can only succeed if state is PENDING (§3.2 precondition 4, island-accord-state-machine.md)
- Atomic transaction: check state → write accept → transition PENDING→ACTIVE; rollback on any failure (§3.2 effects, island-accord-state-machine.md)
- Any attempt to transition from terminal state rejected with `C6P.HANDSHAKE.STATE_VIOLATION`

Residual Risk: None — state machine strictly enforced with atomic operations

Status: Prevented

6.11 Threat 11: Double Accept Concurrency

Attacker: Malicious Responder or Concurrent Bug **Goal:** Process two accepts for the same session, potentially with different kc2 values

Attack Path:

1. Responder (or attacker controlling responder) calls accept() twice concurrently with same sessionId
2. First accept sets acceptBlob and transitions PENDING → ACTIVE
3. Second accept attempts to overwrite acceptBlob or transition again

Affected Phase: Handshake (accept)

Mitigation in Convro:

- Accept is idempotent: if session already ACTIVE and provided kc2 matches stored accept, return OK; if differs, reject (§6.2 island-accord-state-machine.md)
- `acceptBlob` is immutable after first write (§4.3 island-accord-state-machine.md)
- Database transaction isolation ensures only one accept can transition PENDING→ACTIVE (§3.2 atomic transaction, island-accord-state-machine.md)
- Second concurrent accept either: (a) sees state already ACTIVE and follows idempotency rule, or (b) transaction conflict causes retry/abort

Residual Risk: None — idempotency and atomic transitions prevent double accept

Status: Prevented

6.12 Threat 12: Offer Delivery After Terminal State

Attacker: Malicious Server or Race Condition **Goal:** Deliver offer to responder after session already transitioned to terminal state (EXPIRED, CANCELLED)

Attack Path:

1. Session created in PENDING; offer deliverable
2. Session expires or initiator cancels → state becomes EXPIRED/CANCELLED
3. Server or WebSocket reconnect delivers stale offer to responder
4. Responder attempts to accept expired/cancelled session

Affected Phase: Message delivery + handshake (accept)

Mitigation in Convro:

- Offer deliverable only when `state == PENDING` (§4.5 delivery invariant, island-accord-state-machine.md)
- Terminal states stop delivery immediately (§4.5 island-accord-state-machine.md)
- Accept precondition checks state is PENDING; if terminal, reject with `C6P.HANDSHAKE.STATE_VIOLATION` (§3.2 precondition 4, island-accord-state-machine.md)
- Client-side: responder should cache session state and reject accept if local state already terminal

Residual Risk: None — state checks at accept time prevent processing of stale offers

Status: Prevented

6.13 Threat 13: Message Replay on WS Reconnect

Attacker: Malicious Server or Network **Goal:** Replay previously delivered message after WebSocket reconnection

Attack Path:

1. Initiator sends message with counter=5
2. Server delivers message; responder accepts and marks counter=5 consumed
3. WebSocket disconnects and reconnects
4. Server (or attacker) replays message with counter=5

Affected Phase: Message transport (DM ratchet)

Mitigation in Convro:

- Receiver maintains consumed counter set per `(sessionId, streamId)` (§2.1 c6p-replay-and-skip-window.md)
- Replay detection: if counter already consumed, reject with `REPLAY_DUPLICATE_COUNTER` (§3.5 c6p-replay-and-skip-window.md)

- Consumed state persisted atomically with ratchet state (§6.1 c6p-replay-and-skip-window.md)
- Replay rejected even if AEAD verifies (§3.5 hard rule, c6p-replay-and-skip-window.md)

Residual Risk: None — consumed set prevents replay regardless of transport reconnects

Status: Prevented

6.14 Threat 14: Counter Desync Attack

Attacker: Network or Malicious Server **Goal:** Cause receiver's expected counter to desynchronize by dropping messages selectively

Attack Path:

1. Sender sends messages counter=1,2,3,4,5
2. Attacker drops counter=3
3. Receiver accepts 1,2 (in-order), then receives 4 (out-of-order, within skip-window)
4. Receiver marks 4 consumed but recv_expected=3
5. Attacker attempts to inject forged message with counter=3 or cause permanent desync

Affected Phase: Message transport (DM ratchet)

Mitigation in Convro:

- Skip-window allows out-of-order acceptance within bounded range (2048 messages) (§6.1 c6p-replay-and-skip-window.md)
- Counter 4 accepted and marked consumed; recv_expected remains at 3 (§3.2 c6p-replay-and-skip-window.md)
- When counter=3 eventually arrives, it can be accepted (if within window and not yet consumed)
- If counter=3 never arrives and more than 2048 messages progress, counter=3 becomes permanently skipped
- Per-message key derivation is deterministic per counter; missing messages do not break future messages (§7.2 c6p-replay-and-skip-window.md)
- Bounded skip-window prevents unbounded cache growth (§6.3 c6p-replay-and-skip-window.md)

Residual Risk: Limited — messages can be permanently lost if skipped beyond window; this is acceptable tradeoff for availability

Status: Mitigated (bounded desync acceptable; integrity maintained)

6.15 Threat 15: Malicious Server Message Reordering

Attacker: Malicious Server **Goal:** Deliver messages out of order to cause confusion, trigger skip-window exhaustion, or bypass application-level ordering

Attack Path:

1. Sender sends messages counter=1,2,3,4,5
2. Malicious server reorders: delivers 5,4,3,2,1
3. Receiver accepts all if within skip-window but application logic may assume ordering

Affected Phase: Message transport

Mitigation in Convro:

- Protocol allows out-of-order delivery within skip-window by design (§3.2 c6p-replay-and-skip-window.md)
- Each message independently validated; no dependency on prior messages for decryption (§7.1 c6p-replay-and-skip-window.md)
- Application layer responsible for enforcing ordering if required (e.g., display messages sorted by counter)
- Extreme reordering (>2048 messages) causes rejection with `REPLAY_COUNTER_TOO_FAR` (§3.3 c6p-replay-and-skip-window.md)

Residual Risk: Limited — protocol does not guarantee ordering; application must handle if required

Status: Out of Scope (ordering is application concern; protocol provides tools via counters)

7 Non-Goals & Explicit Limitations

7.1 9.1 Compromised Endpoint

Threat: Malicious OS, malware, keylogger, or debugger running on device

Impact: Attacker gains access to all keys stored on device, can decrypt all future messages, can impersonate user

Mitigation Status: Out of scope for protocol. Rely on:

- Platform security (iOS Secure Enclave, Android StrongBox)
- Biometric/passcode protection
- App sandboxing
- User device hygiene

Note: Forward secrecy protects past messages if keys already deleted, but future messages compromised until device cleaned or keys rotated.

7.2 9.2 Malicious OS / Physical Access

Threat: Attacker with physical access to unlocked device or compromised operating system

Impact: Full access to user's messages, keys, and account

Mitigation Status: Out of scope. C6P cannot protect against:

- Unlocked device left unattended
- Compromised iOS/Android system
- Rooted/jailbroken devices (optional detection possible)

Recommendation: User education, device lock policies, remote wipe capabilities (application-layer feature)

7.3 9.3 Side-Channel Attacks

Threat: Timing attacks, power analysis, cache timing, electromagnetic emanation

Impact: Potential key leakage via measurement of execution time or power consumption

Mitigation Status: Partially addressed:

- Constant-time crypto libraries used (libsodium, ring)
- Constant-time comparisons for tags/MAC validation
- Power analysis out of scope (requires hardware-level defenses)

Residual Risk: Advanced physical side-channel attacks not prevented; rely on hardware security modules for high-value keys if required.

7.4 9.4 Denial of Service

Threat: Flooding, OTP exhaustion, computational DoS via expensive operations

Impact: Service unavailable to legitimate users

Mitigation Status: Partially addressed:

- Rate limiting (server-side policy)
- OTP reservation TTL (10 minutes)
- Skip-window bounds (2048) prevent unbounded cache growth
- AEAD validation fails fast on invalid tags

Residual Risk: Full DoS prevention out of protocol scope; rely on network-level defenses, CDN, rate limiting infrastructure.

7.5 9.5 Metadata Leakage

Threat: Traffic analysis reveals communication patterns (who talks to whom, when, frequency)

Impact: Metadata exposes social graph and behavior patterns

Mitigation Status: Minimal in v1:

- Session IDs are random (not derived from user identities)
- Message sizes visible (padding not implemented in v1)
- Timing visible (cover traffic not implemented in v1)

Recommendation: High-threat users should combine C6P with Tor or similar anonymity networks.

Note: C6P focuses on content confidentiality; metadata protection deferred to future versions and external tools.

7.6 9.6 Quantum Adversaries

Threat: Future quantum computers break X25519 ECDH via Shor's algorithm (harvest-now-decrypt-later)

Impact: Encrypted traffic recorded today could be decrypted in future with sufficiently powerful quantum computer

Mitigation Status: Not addressed in v1. Acknowledged limitation:

- C6P v1 is not quantum-safe
- Post-quantum prekeys (hybrid X25519 + Kyber) planned for v2
- Key rotation limits exposure window

Residual Risk: High-value targets should assume quantum vulnerability; plan for v2 migration.

8 Determinism & Fail-Closed Philosophy

8.1 9.5.1 Why Deterministic Nonces Are Safe

C6P uses deterministic nonce derivation, which is secure because:

1. **Per-message keys**: Every message has unique AEAD key derived via HKDF from chain key + counter
2. **Injective binding**: Nonce derivation binds `(session_id, suite_id, message_type, stream_id, counter)` via HKDF-Expand with domain-separated label `C6P_NONCE_V1`
3. **Replay prevention**: Consumed counter sets ensure no counter reused within a session/stream
4. **No key reuse**: The pair `(suite_key, nonce)` is never reused by construction

Security Property: Even if nonce is deterministic, each message uses a fresh AEAD key. The security of ChaCha20-Poly1305 depends on unique `(key, nonce)` pairs, which C6P guarantees via unique per-message keys.

Benefits of Determinism:

- Test vectors are reproducible across implementations
- No reliance on RNG for nonce generation (reduces attack surface)
- Easier to audit and verify correctness

8.2 9.5.2 Fail-Closed Design Principle

C6P enforces fail-closed behavior at every layer:

Encoding Layer Invalid hex/base64url encoding → immediate rejection with `C6P.ENC.*` error code. No "best-effort" parsing.

Cryptographic Layer Signature verification failure, AEAD tag mismatch, transcript mismatch → immediate abort. No partial session creation.

State Machine Layer Invalid state transition, expired offer, replayed offer → rejection with `C6P.HANDSHAKE.STATE_VIOLATION`. No "fixup" attempts.

Ratchet Layer Replayed counter, counter outside skip-window, AEAD open failure → message rejected, no state advancement. Consumed set unchanged.

Why Fail-Closed?

1. **Security over availability**: Better to reject ambiguous input than risk state corruption
2. **Predictable behavior**: Deterministic failures enable reliable testing and debugging
3. **Attack detection**: Failed validations indicate active attack or serious bug; must be observable
4. **No silent degradation**: Partial failures escalate to full failure; prevents attacker from gradually weakening security

---

8.3 9.5.3 Unknown Inputs Cause Abort

If any input is unknown, unexpected, or outside spec:

- Unknown `suite_id` → `C6P.AEAD.UNSUPPORTED_SUITE`
- Unknown `message_type` or `stream_id` → `C6P.WIRE.INVALID_ENVELOPE`
- Unknown protocol version → `C6P.HANDSHAKE.VERSION_UNSUPPORTED`
- Malformed encoding → `C6P.ENC.*`

No Fallbacks: C6P does not attempt to "guess" or "default" unknown values. Unknown inputs are treated as potential attacks or unacceptable version skew.

8.4 9.5.4 State Never Advances on Failure

Sender Invariant: Send counter incremented only after message sealed and state persisted atomically.

Receiver Invariant: Counter marked consumed only after AEAD open succeeds and state persisted atomically.

Server Invariant: Session state transitions only after all validation passes and database transaction commits.

Crash Safety: If process crashes mid-operation, state remains at last committed checkpoint. Operations are idempotent where possible.

9 Conclusion & Security Posture

9.1 10.1 Summary of Security Posture

C6P (Convro 6 Protocol) provides strong end-to-end encrypted messaging within its defined threat model:

Achieved Security Properties:

- **Confidentiality**: Messages encrypted with per-message AEAD keys; server and network cannot decrypt
- **Authentication**: Mutual authentication via Ed25519 signatures and bidirectional key confirmation
- **Forward secrecy**: Per-message keys provide forward secrecy; compromise of current state does not reveal past messages
- **Replay resistance**: Handshake and message replay prevented via server uniqueness constraints and consumed counter sets
- **State integrity**: Strict state machine enforcement with atomic transitions and fail-closed validation
- **Determinism**: All cryptographic operations deterministic and reproducible

Threat Coverage:

- Network attackers (passive and active MITM) cannot decrypt or forge messages
- Malicious server cannot decrypt messages or violate state machine invariants
- Malicious peers cannot replay messages or bypass authentication
- OTP scarcity enforced via atomic state transitions
- Concurrent operations handled safely via database isolation and idempotency

---

9.2 10.2 Acknowledged Limitations

C6P explicitly does **not** protect against:

- Endpoint compromise (malicious OS, malware, physical access to unlocked device)
- Social engineering (user accepts fake identity fingerprint)
- Quantum adversaries (post-quantum migration planned for v2)
- Full denial-of-service prevention (relies on operational controls)
- Metadata analysis beyond protocol-level binding (timing, message sizes, session linkability)

These limitations are inherent to the threat model and do not represent protocol failures.

9.3 10.3 Residual Risks

Within Scope (Acceptable Tradeoffs):

- **Bounded message loss**: Skip-window (2048) means messages more than 2048 ahead of recv_expected are rejected; acceptable for availability
- **OTP exhaustion DoS**: Short-term OTP pool exhaustion possible despite rate limits; acceptable as DoS is out of protocol scope

- **Metadata leakage**: Session IDs, message sizes, timestamps visible to network/server; acceptable as focus is content confidentiality

Out of Scope (Acknowledged):

- Quantum attacks on X25519 (harvest-now-decrypt-later threat)
- Platform compromise defeating key storage protections
- Advanced side-channel attacks requiring physical access

9.4 10.4 Conclusion

The C6P protocol is secure within its defined threat model.

The protocol provides strong cryptographic guarantees for confidentiality, authentication, and integrity of direct messages between devices. The deterministic design, fail-closed validation, and server state machine enforcement create a robust defense against network attackers, malicious servers, and malicious peers.

Security properties are achieved without requiring server trust for secrecy, making C6P suitable for deployment scenarios where server operators are not fully trusted or where defense-in-depth is required.

For threats outside the protocol's threat model (endpoint compromise, quantum adversaries, full DoS prevention), deployments should layer additional controls: platform security features, user education, operational monitoring, and eventual migration to post-quantum cryptography.

C6P v1 is production-ready for audit and deployment.

10 Appendix A: Threat Mapping Table

ID	Threat	Attacker	Phase	Mitigation	Status
1	Replayed handshake offer	Network/Server	Handshake	Server uniqueness constraint + idempotency	Prevented
2	Modified offer blob	Network/Server	Handshake	Ed25519 signature + transcript binding	Prevented
3	SessionId collision	Peer/Server	Handshake	UNIQUE database constraint + 64-bit random ID	Prevented
4	Accept replay with modified kc2	Network/Server	Handshake	Immutable acceptBlob + KC verification	Prevented
5	Cross-device accept	Peer	Handshake	Device binding validation + transcript check	Prevented
6	OTP reuse	Initiator/Server	Handshake	Atomic state transitions (RESERVED→PENDING→CONSUMED)	Prevented
7	OTP race between initiators	Initiators	Prekey fetch	Atomic OTP reservation + DB isolation	Prevented
8	OTP replay after expiration	Initiator	Handshake	Expiry check at open() + expired status	Prevented
9	OTP prefetch hoarding	Initiator	Prekey fetch	10min TTL + rate limiting + cleanup job	Mitigated
10	Invalid state transition	Client/Server	State machine	Explicit valid transitions + terminal state enforcement	Prevented
11	Double accept concurrency	Responder	Handshake	Idempotency + immutable acceptBlob + atomic transition	Prevented
12	Offer delivery after terminal	Server/Race	Delivery	Delivery only when PENDING + state check at accept	Prevented
13	Message replay on reconnect	Server/Network	Transport	Consumed counter set + atomic persistence	Prevented
14	Counter desync attack	Network/Server	Transport	Deterministic key derivation + bounded skip-window	Mitigated

ID	Threat	Attacker	Phase	Mitigation	Status
15	Malicious server reordering	Server	Transport	Out-of-order support + app-layer ordering	Out-of-scope

11 Appendix B: Glossary

AAD (Additional Authenticated Data) 63-byte structure binding protocol version, suite ID, session binding, stream ID, and counter to ciphertext; authenticated but not encrypted.

AEAD (Authenticated Encryption with Associated Data) Encryption scheme providing both confidentiality and authenticity; C6P uses ChaCha20-Poly1305.

Chain Key (CK) 32-byte symmetric key updated after each message; used to derive per-message keys; provides forward secrecy.

Consumed Counter Set Bounded data structure tracking which message counters have been accepted; prevents replay attacks.

Device ID 16-byte identifier derived from Ed25519 signing public key via SHA-256; uniquely identifies a device.

DH (Diffie-Hellman) Key exchange using X25519; C6P uses 3DH (three DH operations) or 4DH (with OTP).

Ephemeral Key (EK) Short-lived X25519 key pair generated per session; provides forward secrecy.

Fail-Closed Design principle: any validation failure causes immediate abort with no partial state updates.

Identity Key (IK) Long-term key pair; `IK_sig` (Ed25519 for signatures), `IK_dh` (X25519 for DH).

IslandAccord Authenticated prekey handshake protocol; establishes session keys via 3DH+OTP.

Key Confirmation (KC) Bidirectional proof that both parties derived the same session keys; uses HMAC-SHA256 tags (KC1 from initiator, KC2 from responder).

One-Time Prekey (OTP) Single-use X25519 public key providing enhanced forward secrecy; consumed exactly once via atomic server state transitions.

Root Key 32-byte master secret derived from handshake DH operations; used to derive initial chain keys.

Session Binding 32-byte value computed as SHA-256 of session metadata (session ID, device IDs); binds all cryptographic operations to session context.

Signed Prekey (SPK) X25519 public key signed by `IK_sig`; rotated every 30 days; binds ephemeral session keys to long-term identity.

Skip-Window Bounded buffer (2048 messages) allowing out-of-order message acceptance without weakening replay protection.

Stream Unidirectional message flow within session; DM sessions have two streams (i2r: initiator→responder, r2i: responder→initiator).

Transcript Hash SHA-256 hash of canonical handshake parameters; binds offer signature and key confirmation to exact handshake inputs.

---

12 Document Metadata

Document Version: 1.0 **Protocol Version:** C6P v1 / IslandAccord v1 **Date:** January 2026
Authors: Convro Protocol Team **Status:** Production / Audit-Ready

Source Documentation:

- `/docs/threat-model/threat-model-v1.md`
- `/docs/handshake/island-accord-crypto.md`
- `/docs/handshake/island-accord-state-machine.md`
- `/docs/crypto/c6p-nonce-policy.md`
- `/docs/crypto/c6p-replay-and-skip-window.md`
- `/docs/Sessions/sessions-overview.md`
- `/docs/identity/key-storage-and-hardening.md`

Intended Audience:

- Security auditors
- Grant reviewers
- Protocol implementers
- Security researchers

End of Document