The Swell Protocol: Invite-Bound Hybrid Key Agreement, a Double Ratchet with an ML-KEM Epoch Ratchet, and a Signed Server Statement

Ilyas Akbar Almadani

Jiji Labs (PT JIJI LABS DIGITAL), Karanganyar, Indonesia
hello@jijilabsdigital.com

Predominantly AI-generated work

The text of this paper and much of the protocol's implementation and analysis were produced with AI assistance under human direction and review (see Section 7).

Abstract. We describe the Swell Protocol, the end-to-end encryption protocol of the Swell messenger, as fixed by draft 3 of its specification. First contact uses an out-of-band invite code instead of a server-held prekey directory: a hybrid X25519/ML-KEM-768 handshake in the style of PQXDH, without prekey signatures, whose transcript is bound to the invite. Sessions then run the Double Ratchet with header encryption, extended by a post-quantum *epoch ratchet*: ML-KEM-768 exchanges carried inside the conversation renew an epoch key that is mixed into every message key, so that after a one-time state compromise confidentiality is restored against an adversary who is passive while healing, even if it later breaks every X25519 key. Sessions between existing contacts are started or repaired by *link frames* sealed under a static-static key. Groups fan out over pairwise sessions with identity-signed group states. Separately, the message relay publishes an hourly statement signed under a pinned key and kept in a hash-chained, append-only log meant to be mirrored by an external witness. We state the threat model precisely, including two admitted limits of forward secrecy, and we report symbolic analyses in ProVerif and Tamarin of simplified models, together with every attack those models found; there is no computational proof. The protocol has had internal review only; it has not undergone an independent security audit. We list the open problems and invite cryptanalysis.

Keywords: secure messaging · post-quantum · ML-KEM · Double Ratchet · post-compromise security · symbolic analysis

1 Introduction

Messaging protocols that resist "harvest now, decrypt later" adversaries now combine a classical Diffie–Hellman handshake with a post-quantum key-encapsulation mechanism (KEM), as in PQXDH [33]. A second, harder

goal is *post-quantum post-compromise security*: after an adversary has copied a device's state once, the conversation should become confidential again even if the adversary can later break the classical key exchanges. A ratchet that heals only through Diffie–Hellman steps cannot achieve this, since an adversary able to compute discrete logarithms can follow every later step.

This paper describes the Swell Protocol, which addresses both goals in a setting with three distinctive features. (i) There is no prekey directory: a person makes a one-use invite code that is shared out of band and names where the inviter's one-use key bundle can be fetched, and the code's secrecy, not a prekey signature, authenticates the inviter's ephemeral keys. (ii) Messages travel through a store-and-forward relay (the *mailbox*) that the threat model treats as hostile. (iii) The design must repair sessions without any side channel, for example after a device restore, because session state is deliberately never backed up.

Sources and scope. Everything here is taken from the protocol specification, draft 3 (cited as "Spec §x"), the wire-format document ("Wire §x"), the formal-results report ("FR x", with the result numbers used there) and the clean-room record, all published with the code [29]. Where this paper interprets rather than restates, it says so.

Contributions.

- A full description of the protocol: an invite-bound hybrid handshake (Section 4.4); a Double Ratchet with header encryption whose message keys also depend on an ML-KEM-768 epoch ratchet with alternating key generation, confirmation and pacing (Sections 4.5 and 4.6); link frames for session start and repair (Section 4.7); and group states signed by an Ed25519-compatible signature under the X25519 identity key (Sections 4.8 and 4.9).
- A threat model with explicit limits (Section 3): forward secrecy fails in two precisely stated cases, healing assumes a passive adversary, and header metadata and identity authentication are classical only.
- Symbolic analyses in ProVerif 2.05 and Tamarin 1.12 (Section 6), with what each model leaves out and every attack found, including the two that changed the specification (missing post-quantum healing in draft 1; an over-strong forward-secrecy claim).
- A signed, hash-chained server statement with an external witness, and a precise account of what it does and does not establish (Section 5).
- Two implementations (Swift with CryptoKit, Kotlin with Bouncy Castle) checked against shared byte-exact test vectors, including ML-KEM-768 vectors from fixed seeds (Section 7).

What this paper does not claim. We give no computational proof; the symbolic results are evidence about simplified models of the design, not about the code. The protocol has had internal review only and no independent security audit; one is planned. We make no claim of metadata hiding beyond what Section 3 states and no deniability claim (group states are signed), and we do not compare the protocol's security favourably with any other protocol.

Table 1. Notation (Spec §4).

$\mathsf{DH}(a, B)$	X25519 [34] with scalar a and u-coordinate B; an all-zero output is rejected by both sides, for every DH.
$\mathsf{AEAD}(k, n, ad, m)$	AES-256-GCM [22], 12-byte nonce, output ciphertext‖16-byte tag.
$\mathsf{HKDF}(s, ikm, info, L)$	HKDF-SHA-256 [32] (Extract with salt s, then Expand); $\mathsf{Expand}(prk, info, L)$ is Expand alone.
KEM	ML-KEM-768 [41]; $\lvert\mathsf{EK}\rvert = 1184$, $\lvert\mathsf{ct}\rvert = 1088$, $\lvert\mathsf{ss}\rvert = 32$; the private key is stored as its 64-byte seed $d\|z$. KEM.Enc validates EK as FIPS 203 requires; KEM.Dec uses implicit rejection.
$\mathsf{Info}(\ell, f_1, \ldots, f_n)$	`"Swell"`‖`0x00`‖`u8 2`‖`u8` $\lvert\ell\rvert \,\|\|\, \ell \,\|\|\, (\texttt{u16}\,\lvert f_i\rvert \,\|\|\, f_i)_i$; every label has exactly one use (Spec §9).
$\mathsf{H}(t, x)$	SHA-256 [40](`"Swell"`‖`0x00`‖`u8 2`‖`u8` $\lvert t\rvert \,\|\|\, t \,\|\|\, x$) ("SHA256Tagged").
$\mathsf{ID}_I, \mathsf{ID}_J$; $\mathsf{id}_I, \mathsf{id}_J$	identity public keys (32 bytes) and secrets of I and J; $\widehat{\mathsf{ID}} = \texttt{0x05}\|\mathsf{ID}$ is the 33-byte serialized form.
$(\mathsf{epk}, \mathsf{EPK})$, $(\mathsf{dk}, \mathsf{EK})$	the inviter's one-use X25519 and ML-KEM-768 key pairs in a bundle.
(e, E), $(\mathsf{rsk}, \mathsf{RPK})$	the joiner's ephemeral key pair and first ratchet key pair.
T	handshake transcript hash; RK, CK root and chain keys; PQ_e epoch key of epoch e.

2 Background and Notation

Building blocks. X3DH [37] derives a session key from several Diffie–Hellman values between long-term identity keys, a signed prekey and an ephemeral key. PQXDH [33] adds an ML-KEM shared secret to the key-derivation input; its authentication remains classical. The Double Ratchet [42] combines a Diffie–Hellman ratchet (a new key pair at each turn of the conversation) with symmetric chains that derive one key per message; its header-encryption variant hides the ratchet public key and counters. Post-compromise security (PCS) was formalised by Cohn-Gordon, Cremers and Garratt [18]. Revision 4 of the Double Ratchet document adds the Sparse Post-Quantum Ratchet (SPQR), built on a sparse continuous key agreement protocol such as the ML-KEM Braid [45], and the Triple Ratchet, which combines it with the Double Ratchet [42,21]; the Swell epoch ratchet takes only the idea of an ML-KEM ratchet whose secrets are mixed into message keys (Spec §2, Spec §7.6). ML-KEM is standardised in FIPS 203 [41]. Analyses of all these designs are discussed in Section 9.

Notation. Table 1 fixes notation. ‖ is concatenation; integers are big-endian unless stated; `i32`, `i64` are signed two's-complement; a field written $\langle x \rangle$ is `i32 length`‖x (Spec §4).

Roles. I is the inviter, or, on the link path, the device whose serialized identity key is lower in unsigned byte order; J is the joiner or the other device (Spec,

"Words"). The relay is the *mailbox*: an onion and TLS service holding queues of envelopes (Wire §6).

3 Threat Model and Security Goals

3.1 Adversaries

Following Spec §3, we consider: (A-net) a network observer; (A-mbx) the mailbox operator, who logs everything and may drop, delay, reorder or replay; (A-q) a later quantum adversary holding recordings, modelled as knowing every X25519 private key; (A-theft) a thief who copies one device's complete state once; (A-contact) a hostile contact. Adversaries may combine these capabilities where a goal says so.

3.2 Goals

G1 Secrecy. Message content is secret from A-net, A-mbx and A-q if X25519 *or* ML-KEM-768 is secure for the handshake (Spec §3; FR 1, 4, 5, 17, 2-1–2-5).

G2 Forward secrecy, with two limits. A message stays secret after a later compromise of either endpoint once its message key mk has been deleted, *except* in two cases (Spec §3, Spec §12):

(L1) The sender's current ratchet private key lives until its next DH step. A second compromise inside the healing window of an earlier, unhealed compromise can therefore expose a message already sent (attack A1, Section 6.3). The published Double Ratchet behaves the same way.

(L2) Skipped message keys (at most 2000 per relationship, for at most 15 days, Spec §8.1) are kept on purpose, and a compromise exposes the messages they belong to (FR gap G6).

"Deleted" means removed from the application's store; physical erasure from database pages, logs or flash is not promised (Spec §12).

G3 Classical PCS. Against an adversary who is passive while healing, messages are secret again one DH round trip after a one-time compromise (FR 13).

G4 Post-quantum PCS. Against an adversary who is passive while healing and also knows every X25519 private key, messages are secret again from the first epoch whose ML-KEM key is generated after the compromise, i.e. at the latest at the second epoch change (Spec §3, Spec §7.6; FR 2-P2). Epoch changes are paced (Section 4.6), so this takes at least two round trips and may take up to the pacing limits.

G5 Mutual authentication. A session between two identity keys can only be established by holders of both private keys (Spec §6.3). This is machine-checked for one invite-path session with random identity keys (FR 2-2, 2-3) and for unbounded link handshakes under persistent keys (FR 3-2); for the invite path under persistent keys it is open (FR 3-7). First contact is trust on first use: the identity key is taken from the invite, and only comparing safety numbers shows it was the intended one.

G6 Mailbox privacy (limited). Apart from each frame's per-contact `channelId` and ciphertext type, and the joiner's identity key in an invite-path HELLO (which lets A-mbx link one person's invite redemptions, D7), no key, identifier or counter appears in clear: a DATA envelope has a pseudo-random header nonce and an encrypted header, and a link frame is fully sealed (Spec §3).

3.3 Out of scope and admitted limits

The following are *not* protected (Spec §3; the IP-address item is from Wire §6), and every security statement about the protocol must carry them.

- Traffic analysis: mailbox timing, sizes and volume. A frame carrying an ML-KEM key or ciphertext moves to a larger padding bucket, so epoch changes are visible as size.
- The operator learns the device's IP address unless the optional Tor mode is on (Wire §6); call media and platform push services do not use Tor.
- A compromised device, a contact who keeps what they received, dropped messages, push tokens and Tor guard exposure, implementation bugs.
- A leaked invite: whoever redeems it first becomes the contact.
- An *active* adversary after a compromise, who injects its own ratchet or KEM public keys and remains in the middle until safety numbers are compared again or a new invite is made (attacks A2, A6). Every ratchet has this limit.
- Theft of the identity secret (derived from the 12-word recovery phrase), after which the thief can start sessions as that person.
- Header metadata (ratchet public key, counters, epoch) after a compromise plus a quantum break: header keys are not mixed with epoch keys (deviation D11).
- Identity authentication and link frames are classical. A quantum adversary who records an invite or a link exchange can later impersonate the identity but cannot read earlier messages; PQXDH [33] has the same limit.
- Linkability of identities: one identity key per person (deviation D14); the mailbox sees the joiner's identity key in clear in an invite-path HELLO (D7).

4 The Protocol

4.1 Primitives, encodings and domain separation

The primitives are those of Table 1; there are no new primitives. Every HKDF *info* passes through Info and every tagged hash through H, both carrying the domain version 2 (Spec §4). The labels are listed in Spec §9: HKDF labels `"handshake"`, `"root"`, `"chain"`, `"message"`, `"pq-message"`, `"pq-epoch"`, `"header-enc"`, `"header-iv"`, `"header-nonce"`, `"safety"`, `"link"`; hash tags `"bundle"`, `"transcript"`, `"upgrade"`, `"pq-kem"`, `"session"`, `"direct"`, `"hello"`; AEAD associated-data prefixes `"SWL2-H"`, `"SWL2-M"`, `"SWL3-L"`. Three older raw-ASCII infos (identity, report and backup keys) do not use Info. Comparisons of secret-derived values are constant time; a failure of the platform random source stops the protocol (Spec §4).

4.2 Identity

Each person has one identity key, derived from 16 bytes of account entropy (shown as 12 recovery words) (Spec §5.1):

$$\mathsf{id} = \mathsf{HKDF}(0^{32},\ \mathit{entropy}_{16},\ \texttt{"swell identity v1"},\ 32),$$
$$\mathsf{ID} = \mathrm{X25519}(\mathsf{id}, 9), \qquad \widehat{\mathsf{ID}} = \texttt{0x05}\|\mathsf{ID}.$$

An identity public key u is *valid* iff its encoding is canonical (top bit clear, value below $2^{255} - 19$) and it is not of small order (tested by $\mathsf{DH}(s, u) \neq 0$ for a non-zero scalar s). Invalid keys are refused wherever they appear: invite, bundle, HELLO, group roster, link frame. The safety number of a pair is $\mathsf{HKDF}(0^{32},\ \widehat{\mathsf{ID}}_{lo}\|\widehat{\mathsf{ID}}_{hi},\ \mathsf{Info}(\texttt{"safety"}),\ 60)$ with the two keys in unsigned byte order, printed as twelve 5-digit groups (each 5-byte block, big-endian, mod 10^5) and encoded in a QR code whose scan is an equality check (Spec §11).

4.3 Bundles and invite codes

For every invite (or link offer) I draws fresh $(\mathsf{epk}, \mathsf{EPK})$ and $(\mathsf{dk}, \mathsf{EK})$ and forms the 1257-byte bundle (Spec §5.3)

$$B = \texttt{u8 2} \,\|\, \mathsf{ID}_I \,\|\, \mathsf{EPK} \,\|\, \mathsf{EK} \,\|\, \texttt{i64}\ \mathit{expiresAtMs},$$

expiring after 24 hours (invite) or 7 days (link offer). Every code is one use, with one exception: the code of a single published service contact, whose maker keeps up to three 7-day bundles and accepts a HELLO for any of them (Spec §5.3); for those sessions the bundle secrets, and with them handshake forward secrecy, last until bundle expiry. *The bundle carries no signature* (deviation D1): its authenticity comes from the invite, which names ID_I and travels out of band, or from the link key (Section 4.7). A substituted bundle yields no session key to the substituter, because dh_2 below requires id_I.

An invite code is `swell:` followed by the unpadded base64url encoding of a 162-byte payload (always exactly 216 characters; padding, whitespace and non-canonical encodings are refused) (Spec §5.4):

$$\text{v5 (mailbox):}\quad \texttt{u8 5} \,\|\, \langle \mathit{mailboxOnion} \rangle \,\|\, \mathit{queueId}_{32} \,\|\, \mathit{sealKey}_{32} \,\|\, \langle \widehat{\mathsf{ID}_I} \rangle$$
$$\text{v6 (direct onion):}\quad \texttt{u8 6} \,\|\, \langle \mathit{onion} \rangle \,\|\, \mathit{secret}_{32} \,\|\, \langle \widehat{\mathsf{ID}_I} \rangle \,\|\, \mathit{authKey}_{32}$$

For v5 the bundle waits as the first envelope of the invite queue, sealed as $n\|\mathsf{AEAD}(\mathit{sealKey}, n, \mathit{queueId}, B)$; the redeemer reads it with a non-destructive PEEK. For v6 the bundle is fetched unsealed from the inviter's client-authorised onion service. The *binding* that enters the transcript is *queueId* (v5) or $\mathsf{H}(\texttt{"direct"}, \mathit{secret})$ (v6) (Spec §5.4).

The v5 HELLO frame, which is placed in the same queue, carries $\mathsf{H}(\texttt{"hello"}, \mathit{sealKey})$ rather than *sealKey*, which I checks in constant time (Spec §5.5). This is a fix from internal review (finding B27 H1): an earlier

J redeems (Spec §6.1)

obtain B; check $|B|$=1257, version,
  ID_I = key named by invite, valid,
  $now \le exp \le now + 7\text{d}\,1\text{h}$
$(\mathsf{e}, \mathsf{E}), (\mathsf{rsk}, \mathsf{RPK}) \leftarrow_{\$} \text{X25519}$
$(\mathsf{ct}, \mathsf{ss}) \leftarrow \mathsf{KEM.Enc}(\mathsf{EK})$
$\mathsf{dh}_1 \leftarrow \mathsf{DH}(\mathsf{id}_J, \mathsf{EPK})$
$\mathsf{dh}_2 \leftarrow \mathsf{DH}(\mathsf{e}, \mathsf{ID}_I)$
$\mathsf{dh}_3 \leftarrow \mathsf{DH}(\mathsf{e}, \mathsf{EPK})$
$\mathsf{T} \leftarrow \mathsf{H}(\texttt{"transcript"}, \mathsf{ID}_I \| \mathsf{ID}_J \| \mathsf{EPK} \| \mathsf{EK}$
  $\| \mathsf{E} \| \mathsf{ct} \| binding \| exp)$
$hs \leftarrow \mathsf{HKDF}(\mathsf{T},\ \texttt{0xFF}^{32} \| \mathsf{dh}_1 \| \mathsf{dh}_2$
  $\| \mathsf{dh}_3 \| \mathsf{ss},\ \mathsf{Info}(\texttt{"handshake"}), 128)$
$\mathsf{RK}_0 \| hk_a \| nhk_b \| \mathsf{PQ}_0 \leftarrow hs$
init ratchet as Alice; HELLO = msg 0, epoch 0
commit state + exact bytes; erase $\mathsf{e}, \mathsf{ss}, \mathsf{dh}_i, hs$
send HELLO; wait for I's first reply

I accepts (Spec §6.2)

parse HELLO prefix $(\mathsf{ID}_J, \mathsf{E}, \mathsf{ct})$;
  ID_J valid, $now \le exp + 10\,\text{min}$
$\mathsf{dh}_1 \leftarrow \mathsf{DH}(\mathsf{epk}, \mathsf{ID}_J)$
$\mathsf{dh}_2 \leftarrow \mathsf{DH}(\mathsf{id}_I, \mathsf{E})$
$\mathsf{dh}_3 \leftarrow \mathsf{DH}(\mathsf{epk}, \mathsf{E})$
$\mathsf{ss} \leftarrow \mathsf{KEM.Dec}(\mathsf{dk}, \mathsf{ct})$
$\mathsf{T},\ hs,\ \mathsf{RK}_0 \| hk_a \| nhk_b \| \mathsf{PQ}_0$ as for J
init ratchet as Bob; normal receive
on failure: drop silently
on success: commit state, contact $\widehat{\mathsf{ID}_J}$,
  message; erase $\mathsf{dk}, \mathsf{dh}_i, hs$
invite path: $\mathsf{ID}_J \neq \mathsf{ID}_I$; ACK queue,
  never parse a later HELLO
reply with DATA (STATE)

Fig. 1. The handshake. *exp* is the bundle's `expiresAtMs`. The HELLO envelope is `u8 2`$\|$`u8 1`$\|\mathsf{ID}_J\|\mathsf{E}\|\mathsf{ct}\|hnonce\|hct\|body$ (2266 bytes in the smallest bucket).

build placed *sealKey* itself in that frame, so the mailbox operator, who sees the queue, could have opened the bundle, redeemed the invite first and become the inviter's contact. Only internal test builds had the earlier frame (Spec §1). v5 codes are also delivered inside sessions to connect group members (Section 4.9) and through optional username-based chat requests (Wire §6e); in both cases the maker accepts a HELLO only from one named identity key. The username directory and chat requests are outside the scope of this paper.

4.4 Handshake

The pattern is X3DH without a one-time prekey plus an ML-KEM secret as in PQXDH (Spec §6). Figure 1 gives both sides.

All transcript fields have fixed length, and T is kept with the relationship. The HELLO is ratchet message $n = 0$ of epoch 0: its header is encrypted under hk_a with T in the associated data, and its body under the first message key of J's sending chain (Section 4.5). A wrong ct yields a pseudo-random secret by implicit rejection, so the header AEAD fails; there is no refusal message. J retries the exact committed bytes and never re-encrypts.

Authentication argument (Spec §6.3). J knows that only the holder of id_I can compute dh_2, hence open the HELLO and produce a reply that opens; I knows that only the holder of id_J can compute dh_1. Key confirmation for J comes with I's first reply. On the invite path "J" means "whoever holds the invite" (attack A4): a person who also controls the mailbox and knows a leaked code

Table 2. Initial ratchet state (Spec §7.1).

Field	J (Alice)	I (Bob)
DHs	$(\mathsf{rsk}, \mathsf{RPK})$	$(\mathsf{epk}, \mathsf{EPK})$
DHr	EPK	none
RK, CKs, NHKs	$\mathsf{KDF}_{\mathsf{RK}}(\mathsf{RK}_0, \mathsf{DH}(\mathsf{rsk}, \mathsf{EPK}))$	RK_0, none, nhk_b
HKs; NHKr	hk_a; nhk_b	none; hk_a
Ns, Nr, PN	0	0
PQ	epoch 0, send epoch 0, $\{0\colon \mathsf{PQ}_0\}$	same

can substitute the bundle, which only denies service, or redeem first, which makes them the contact; safety numbers are the check.

Key derivation as a combiner (D3). The DH outputs and ss are concatenated into the HKDF input keying material, and all public values, including EK, ct and the binding, enter through the salt T. In the symbolic models HKDF is a random oracle; a computational argument that this is a sound hybrid combiner [24,11] has not been made (Spec §15).

4.5 Double Ratchet with header encryption

Sessions run the Double Ratchet with header encryption as in sections 3–4 of the published specification [42], with the differences below (Spec §7). Initial states are in Table 2.

Key derivation (Spec §7.2).

$$\begin{aligned}
\mathsf{RK}' \| \mathsf{CK} \| NHK &= \mathsf{HKDF}(\mathsf{RK},\ dh,\ \mathsf{Info}(\texttt{"root"}),\ 96) \quad (\mathsf{KDF}_{\mathsf{RK}}) \\
\mathsf{CK}' &= \mathsf{Expand}(\mathsf{CK}, \mathsf{Info}(\texttt{"chain"}), 32) \quad (\mathsf{KDF}_{\mathsf{CK}}) \\
\mathsf{mk}_{dr} &= \mathsf{Expand}(\mathsf{CK}, \mathsf{Info}(\texttt{"message"}), 32) \quad (\mathsf{KDF}_{\mathsf{CK}}) \\
\mathsf{mk} \,\|\, bnonce &= \mathsf{HKDF}(\mathsf{PQ}_e, \mathsf{mk}_{dr}, \mathsf{Info}(\texttt{"pq-message"}, \mathtt{i32}\, e), 44) \quad (\mathsf{combine}) \\
hk_{enc} &= \mathsf{Expand}(hk, \mathsf{Info}(\texttt{"header-enc"}), 32) \quad (\text{header keys}) \\
hk_{iv} &= \mathsf{Expand}(hk, \mathsf{Info}(\texttt{"header-iv"}), 32)
\end{aligned}$$

where e is the epoch named in the message header. The message key and the body nonce *bnonce* are both derived (D5); a state rollback would therefore reuse an AES-GCM key–nonce pair, which is why session state is never backed up or restored (Spec §8.2).

Header and body (Spec §7.3–Spec §7.5). The plaintext header is exactly 44 bytes, $h = dh_pub_{32} \| \mathtt{i32}\ pn \| \mathtt{i32}\ n \| \mathtt{i32}\ epoch$, with all three integers non-negative and at most 2^{16} messages per chain. Then

$$\begin{aligned}
hnonce &= \mathsf{Expand}(hk_{iv}, \mathsf{Info}(\texttt{"header-nonce"}, h), 12), \\
hct &= \mathsf{AEAD}(hk_{enc},\ hnonce,\ \texttt{"SWL2-H"} \| prefix \| \mathsf{T},\ h), \\
body &= \mathsf{AEAD}(\mathsf{mk},\ bnonce,\ \texttt{"SWL2-M"} \| \mathsf{T} \| prefix \| hnonce \| hct,\ padded),
\end{aligned}$$

where *prefix* is `u8 ver`$\|$`u8 kind` (plus $\mathsf{ID}_J\|\mathsf{E}\|\mathsf{ct}$ for HELLO). A receiver recomputes *hnonce* from the opened header and drops the message on mismatch. Deriving the header nonce from the header (D6) gives a 96-bit value; at 2^{16} headers per key the collision probability is about 2^{-65} (Spec §15). The plaintext is padded as $\mathit{padded} = \texttt{i32}\ \ell \,\|\, \mathit{content} \,\|\, 0^*$ to the smallest of the buckets 1, 4, 16, 64 and 256 KiB; the receiver rejects a non-bucket size, an out-of-range ℓ and any non-zero padding byte (checked in constant time). *content* starts with a PQ field: none, $\mathsf{EK}(t)$, or $\mathsf{ct}(t)$ with $t > 0$. A DATA envelope is `u8 2`$\|$`u8 2`$\|\mathit{hnonce}\|\mathit{hct}\|\mathit{body}$, 90 bytes plus the bucket.

Receiving (Spec §8.3). The receiver works on a copy of the state. It tries each candidate header key (the distinct header keys of stored skipped keys, HKr, NHKr); exactly one must open *hct*. It then follows the published algorithm: a stored skipped key is used once and deleted; under HKr, duplicates ($n < Nr$) and gaps above MAX_SKIP = 1000 are dropped; under NHKr a DH ratchet step is run and the old DHs private key is deleted. It takes PQ_e for the header's epoch (dropping the message if that key is not held), opens the body, parses the padding strictly, applies the PQ field (Section 4.6), and commits state and message in one transaction before acknowledging the mailbox. At most 2000 skipped keys are stored per relationship (oldest evicted) for at most 15 days; a skipped key stores mk_{dr} and is combined with the epoch key named in the late message's header (Spec §8.1). A message that fails to open leaves the state untouched; Spec §8.4 tabulates the behaviour for each event, including crash recovery.

Sending (Spec §8.2). A single writer per relationship chooses the PQ field, advances the chain, encrypts, and stores the exact envelope in one transaction (with `synchronous = FULL`); retries resend those bytes.

4.6 The post-quantum epoch ratchet

State and roles (Spec §7.6). Each side holds *epoch* (the highest epoch whose key is known), $\mathit{sendEpoch} \in \{\mathit{epoch}, \mathit{epoch}-1\}$, the keys of *epoch* and up to three older epochs, an optional pending key generation $(\mathit{seed}, \mathsf{EK}, t)$, an optional pending ciphertext (ct, t), a count of messages sent in the current epoch and the time the epoch was confirmed. The key-generation side of epoch t is I for odd t and J for even t; the other side encapsulates. Epoch keys evolve as

$$\mathsf{PQ}_t = \mathsf{HKDF}\big(\mathsf{PQ}_{t-1},\ \mathsf{ss}_t,\ \mathsf{Info}(\texttt{"pq-epoch"}, \texttt{i32}\ t,\ \mathsf{H}(\texttt{"pq-kem"}, \mathsf{EK}_t\|\mathsf{ct}_t)),\ 32\big),$$

and $\mathsf{PQ}_{\mathit{sendEpoch}}$ is used in combine for every message sent. Figure 2 gives the send and receive rules.

Confirmation. The encapsulating side derives PQ_t immediately but keeps sending under $t-1$, because the key-generation side cannot read epoch t before it has decapsulated; it switches when it sees the peer use epoch t. The ciphertext

Send: choose PQ field

if frame not durable: none
if field would push a >16 KiB frame into a larger bucket: none
if pending $\mathsf{ct}(t) \wedge$ (first after Enc $\vee$ repeat owed)**:** $\mathsf{ct}(t)$
elif pending keygen $\mathsf{EK}(t) \wedge$ (first after keygen $\vee$ repeat owed)**:** $\mathsf{EK}(t)$
elif keygen side of $t{=}epoch{+}1$
$\wedge$ $sendEpoch{=}epoch$
$\wedge$ ($epoch{=}0 \vee \#sent{\geq}25$
$\vee$ 6 h since confirmed)**:**
$seed \leftarrow_{\$} \{0,1\}^{512}$; attach $\mathsf{EK}(t)$
header epoch $\leftarrow sendEpoch$

Receive (after the body opened)

if $h.epoch{=}epoch \wedge sendEpoch{<}epoch$**:**
$sendEpoch \leftarrow epoch$; drop pending ct
if $\mathsf{EK}(t)$, $t{=}epoch{+}1$, not keygen side, $sendEpoch{=}epoch$, no pending ct**:**
$(\mathsf{ct}, \mathsf{ss}) \leftarrow \mathsf{KEM.Enc}(\mathsf{EK})$; derive PQ_t
$epoch \leftarrow t$; keep $\mathsf{ct}(t)$ pending
elif $\mathsf{ct}(t)$, $t{=}epoch{+}1$, pending keygen t**:**
$\mathsf{ss} \leftarrow \mathsf{KEM.Dec}(seed, \mathsf{ct})$; derive PQ_t
erase $seed$; $epoch, sendEpoch \leftarrow t$
else ignore field (message still delivered)
delete keys of epochs $< epoch - 3$

Fig. 2. The epoch ratchet (Spec §7.6). "Durable" frames are outbox rows retried byte for byte until delivered; receipts, typing indicators, call signals, STATE and link-protocol frames never carry a PQ field and change no PQ state. Each new sending chain started while a $\mathsf{ct}(t)$ (or, in the same way, an $\mathsf{EK}(t)$) is pending sets a "repeat owed" mark; a pending field stops being sent at confirmation.

therefore travels under the old epoch. One EK and one ct are sent per epoch, normally in one message each (moving it to the 4 KiB bucket), repeated on message 0 of a new sending chain if lost.

Healing (Spec §7.6). A compromise reveals the epoch keys held (at most four) and a pending seed. The first epoch whose ML-KEM key pair is generated after the compromise is unknown to the adversary even with every X25519 key, provided the EK and ct of that epoch were delivered unmodified; every later epoch key descends from it. With the alternation of roles, that epoch is at most two epoch changes after the compromise: the change in progress may already be exposed through a stolen pending seed. This is the statement modelled as FR 2-P2.

Design choices that differ from Braid (D10, D11, D13). Whole EK and ct are sent in single messages rather than erasure-coded chunks; there is one epoch at a time with alternating key generation and a confirmation step; pacing is by message count and time. Epoch keys enter message keys only, not the root chain or header keys. One PQ_e serves both directions; per-message forward secrecy comes from mk_{dr} (Spec §15).

4.7 Link frames: starting and repairing sessions between contacts

Two contacts hold each other's identity keys, which suffices to (re)start a session after an update, a restore (sessions are not in backups) or a lost state (Spec §6.4). The link key and frame are

$$LK = \mathsf{HKDF}\big(\mathit{chId},\ \mathsf{DH}(\mathsf{id}_{\mathrm{me}}, \mathsf{ID}_{\mathrm{them}}),\ \mathsf{Info}(\texttt{"link"}, \widehat{\mathsf{ID}}_{lo}, \widehat{\mathsf{ID}}_{hi}),\ 32\big),$$
$$ad = \texttt{"SWL3-L"} \,\|\, \texttt{u8 3} \,\|\, \mathit{chId} \,\|\, \widehat{\mathsf{ID}}_{\mathrm{sender}} \,\|\, \widehat{\mathsf{ID}}_{\mathrm{receiver}},$$
$$\mathit{frame} = \texttt{u8 3} \,\|\, n_{12} \,\|\, \mathsf{AEAD}\big(LK,\ n,\ ad,\ \mathit{padded}(\texttt{i64}\ \mathit{sentAtMs} \,\|\, f)\big),$$

where *chId* is the contact's 16-byte `channelId`, n is random, and f is one of OFFER(*offerId*, bundle), HELLO(*offerId*, HELLO envelope), RESET, or SYNC($\mathsf{H}(\texttt{"session"}, \mathsf{T})$ or 0^{32}); READY(*offerId*) travels inside the first DATA message each way. The associated data names sender and receiver, so a frame cannot be reflected to its sender or moved to another channel; Section 6.3 shows the attack that motivates this. The link-path binding is $\mathsf{H}(\texttt{"upgrade"}, \mathit{channelId} \| \widehat{\mathsf{ID}}_{lo} \| \widehat{\mathsf{ID}}_{hi} \| \mathit{offerId})$.

Steps. I (the lower key) alone makes offers: a bundle with a 7-day expiry, sent as OFFER. J runs the handshake of Figure 1 with the link binding and the contact's identity as expected inviter, with READY as the HELLO's inner message, and answers repeated OFFER bytes with the same HELLO bytes. I runs the accept procedure, requires ID_J to equal the contact's key, commits, and sends READY as its first DATA. J becomes active when that DATA opens.

Acceptance rules (engine level, Spec §6.4). A link frame is processed only if it arrives for the `channelId` of an established, unblocked contact, opens under that contact's link key, has *sentAtMs* within 15 days before and one hour after the receiver's clock, and comes from the right role (OFFER from I, HELLO from J). RESET and SYNC are ignored when more than five minutes older than the start of the receiver's session, or not later than the frame from the same sender that started that session. A pending handshake is replaced only by a strictly newer offer, and an active session is never replaced by an offer. The last two rules are fixes from internal review: without them, a RESET held back in the mailbox could end a fresh session, and a replayed older OFFER could end a handshake under way (B27 L1; Spec §1).

Repair. A device that receives DATA without a session sends RESET (at most daily). A J that holds an active session and receives a fresh OFFER infers that I lost its state (for example after a restore) and answers SYNC with its session hash; I, holding none, sends RESET and a new offer. After three well-formed DATA messages that do not open, a device sends SYNC; unequal hashes cause both sides to restart. After three refusals for gaps above MAX_SKIP from one contact, the receiver deletes the session and sends RESET (B27 L2). Messages sealed under a lost session are never re-sent as they are: queued plaintexts are sealed again under the next session (Spec §6.4).

No forward secrecy for link frames (D16). *LK* is static per pair and channel. A stolen identity secret lets the thief read past link frames (public keys and a ratchet HELLO, no message content) and forge new ones until the owner changes identity. The content of messages keeps the ratchet's forward secrecy.

4.8 SwellSig: signatures under the X25519 identity key

Group states are signed with *SwellSig*, an Ed25519 signature [30] that verifies under the Edwards form of the X25519 identity key and is produced from the X25519 scalar without any Ed25519 seed (Spec §5.2). Let $p = 2^{255} - 19$, L the Ed25519 group order, B the base point, a the clamped identity scalar, and A the encoding of $y = (u-1)(u+1)^{-1} \bmod p$ with sign bit 0 (the birational map of RFC 7748, §4.1). With $msg = \texttt{"swell-sig-v3"} \| \texttt{0x00} \| \texttt{u8}\,|c| \,\|\, c \,\|\, m$ for a context string c:

- *Sign*: *rseed* is the first 32 bytes of $\mathsf{SHA\text{-}512}(\texttt{"swell-sig-nonce"} \| \texttt{0x00} \| \mathsf{id} \| msg)$; (R, r) is the RFC 8032 key pair of *rseed*; $k = \mathsf{SHA\text{-}512}(R \| A \| msg) \bmod L$; $S = (r + k a_0) \bmod L$ where $a_0 = a \bmod L$ if aB has even x and $L - (a \bmod L)$ otherwise. An implementation that cannot read the sign computes both candidates and keeps the one that verifies.
- *Verify (strict)*: $S < L$; u valid and A of order exactly L; R canonically encoded and of order exactly L; accept if and only if $[S]B = R + [k]A$.

The strict checks place R and A in the prime-order subgroup, where the cofactored and cofactorless equations agree, so any RFC 8032 verifier gives the same answer after them; without them, signatures exist that one library accepts and another refuses [15]. The vector file contains four such hostile cases, all of which must be refused.

The same scalar is used for X25519 and for signing; separation rests on the `"swell-sig-v3"` prefix and the context string (D15). Joint security of DH and signature use of one key has been studied for other schemes [25,19]; we know of no analysis covering this construction, and we do not provide one. The specification notes that the design follows the shape of Signal's XEdDSA but was written from RFC 8032 and RFC 7748 alone and is not interoperable with it (Spec §5.2). The scalar and point arithmetic is hand-written and *not constant time* (D15, open item O17).

4.9 Groups

A group has no server-side state, no shared group key and no sender keys (Spec §18). Every group frame is sent to each other member through the pair's own session as ordinary DATA, so cost per message is linear in the group size. A group state lists the roster (members' serialized identity keys, sorted), a version, the hash of the previous state and the current *sequencer*; a signed state is $(state, \widehat{\mathsf{ID}}_{\text{signer}}, \text{SwellSig}_{\texttt{"swell-group-state-v1"}}(state))$. The creator signs version 1; each next version increments the version, links the previous hash and must be signed by the sequencer that the previous state names; anything else is ignored.

Members without a channel are connected by one-use v5 invites relayed through an admin; the maker accepts a HELLO only from the intended identity key. We have not analysed group consistency properties of the kind studied in [44]; no model covers groups.

4.10 Mailbox envelopes, transport, backups

A message frame is `u8 2`$\|$`u8 3`$\|$*channelId*$_{16}\|$`u8 type`$\|\langle$*ciphertext*$\rangle$ with type 34 (a bare DATA envelope) or 35 (a link frame) (Spec §5.6). The mailbox stores complete frames in queues (queue identifier SHA-256(`"swell-queue"`$\|$*secret*), creation behind a proof of work, 14-day time-to-live, at most 80 KiB per PUT) (Wire §6). Clients reach it either directly over TLS 1.3 with a pinned key, in which case the server sees the client's IP address, or, in the optional Tor mode, only through the mailbox's onion service [20], with no fallback while the mode is on (Wire §6).

Backups are encrypted under a key derived from the account entropy, in AES-256-GCM chunks whose nonces and associated data bind the chunk index and a final-chunk flag, so a changed, reordered or truncated file does not open (Spec §20). Backups contain *no* session state and no sealed outbound frame; a restored device repairs sessions with link frames. A device restored from a backup made between redeeming an invite and the inviter's first answer cannot complete that contact (Spec §6.4).

5 Signed Server Statement and Append-Only Log

The mailbox publishes a statement about itself, signed by a dedicated Ed25519 key that signs nothing else and whose public key is pinned in the apps (Wire §6f). On request (operation `0x27`) it returns $\langle st\rangle\|\sigma$ with

$$\sigma = \text{Ed25519.Sign}\big(sk_{\text{st}},\ \texttt{"swell-server-statement-v1\textbackslash n"} \,\|\, st\big),$$

where *st* is UTF-8 JSON made at most once per hour, containing a format version, the software version and source commit, a flag for uncommitted changes, the issue time, "facts" (message retention in days, username retention in days, whether IP addresses are kept only in memory, whether an access log is kept), counts (usernames on file, bytes of sealed mail waiting) and a source link. The apps check the signature over the bytes as received and reject a statement more than two days from the device clock; any failure is displayed as "could not check".

Chain and log (statement v2). Each statement carries a sequence number *seq* (1, 2, 3, ..., never reused) and *prev*, the record hash of statement $seq - 1$ (all zeros for the first):

$$h_{seq} = \text{SHA-256}\big(\texttt{"swell-server-statement-v1\textbackslash n"} \,\|\, st_{seq} \,\|\, \sigma_{seq}\big).$$

The mailbox appends each signed record to a file on persistent storage, synchronises it to disk before serving it, and serves any past record by number (operation

0x28, rate-limited). At start-up it re-checks every signature, sequence number and link and *fails closed*: if the log is missing, empty or contains a bad line, no statement is served. It never starts a new chain on its own; genesis requires an explicit setting and a newly created file. A *witness* tool fetches the current statement and every record since the last one it published, checks signatures and links, writes them into a git repository and sets aside any record that does not agree with what it saw before (Wire §6f).

What the statement establishes. (i) Authenticity: a statement that checks was signed by the holder of the pinned key. (ii) Under a public witness, a single linear history as seen by the witness: once the witness has recorded record *seq*, a different record with the same number signed by the same key is detected if it is ever served to the witness, and is otherwise visible only to parties who compare with the witness's copy. This is in the spirit of transparency logs [35,39], but with a plain hash chain rather than a Merkle tree, so checking consistency requires the chain of records rather than short proofs.

What it does not establish. A signature shows who said something, not that it is true (Wire §6f); the apps state this next to the result. It proves neither that the running server is built from the named commit nor that the stated facts (retention, IP handling, logging) hold; a key holder can sign false facts consistently. It does not prevent equivocation towards clients who never compare with the witness, and comparing what different devices saw is open. Key rotation and recovery after compromise of the statement key are unsolved. At the time of writing the chained format is specified with vectors but not yet deployed, pending compatibility with older Android clients, and the public witness is not yet operating (Wire §6f). The statement is outside the end-to-end protocol and no security goal of Section 3 depends on it.

6 Security Analysis

6.1 Informal arguments

G1. hs depends on $\mathsf{dh}_1, \mathsf{dh}_2, \mathsf{dh}_3$ and ss. If X25519 holds, dh_3 (two fresh ephemerals) is unknown to A-net and A-mbx; if ML-KEM-768 holds and EK is authentic, ss is unknown. On the invite path EK is authentic because the bundle reaches J out of band (v6) or sealed under *sealKey*, which the mailbox does not learn (v5, after the H1 fix). All later keys descend from RK_0 and PQ_0. This argument treats HKDF as a random oracle.

G2. Message keys are erased after use and chain keys are one-way. The two limits are inherent to the design: (L1) DHs must persist until the peer's next chain begins; (L2) skipped keys are retained for out-of-order delivery. The epoch key PQ_e is shared by many messages, so it cannot provide per-message forward secrecy; this is provided by mk_{dr} (D13).

G3–G4. Classical healing is that of the Double Ratchet. For G4, the stolen state includes at most four epoch keys and a pending seed; the next freshly generated ML-KEM key pair is out of reach, and if its EK and the matching ct arrive unmodified, ss_t and hence PQ_t and all successors are unknown. In the epoch key schedule the secret that restores security, ss_t, is HKDF's *ikm* while the stolen PQ_{t-1} is the salt (the HMAC key); in $\mathsf{combine}$ the secret is the salt PQ_e under G4 but the *ikm* mk_{dr} under G3 after an epoch key was stolen. A computational treatment would therefore need HKDF to be secure when *either* input is secret: a PRF assumption on HMAC [7] for a secret salt, and extraction [31] or a dual-PRF assumption [5] for a secret *ikm* under a known or adversarial salt; $\mathsf{KDF}_{\mathsf{RK}}$ has the same structure as in the Double Ratchet. We flag this as an assumption, not as a known weakness.

G5. See Section 4.4. Because identity keys are persistent and shared between the invite path and link frames, cross-protocol interaction is possible in principle; the draft 3 model includes both uses of the same keys in parallel (Section 6.2).

G6. Header ciphertexts and nonces are pseudo-random under the header keys; link frames are fully sealed. Clear-text fields are the channel identifier, ciphertext type, envelope size bucket and, in an invite-path HELLO, ID_J, E and ct (D7). No model covers G6; it rests on inspection of the formats.

6.2 Formal models

Six current models exist (FR Files): `swell3-link.pv` (draft 3: persistent identity keys, unbounded link handshakes in both roles, RESET frames both ways, and one invite-path handshake with the same keys, all in parallel); `swell2-handshake.pv` (draft 2 invite path with the 128-byte handshake output and PQ_0); `swell2-pq-ratchet.spthy` (the epoch ratchet, epochs 0–3); `swell1-handshake.pv` and `swell1-ratchet.spthy` (draft 1 handshake and classical ratchet); and `swell2-upgrade.pv`, which models a draft 2 mechanism that draft 3 removed; its results are not repeated here. The draft 2 and draft 1 results apply to draft 3 only for components that draft 3 did not change: the handshake key schedule, the Double Ratchet and the epoch ratchet; those models use random per-relationship identity keys rather than draft 3's persistent keys. All jobs were run internally, one at a time on one machine, with a 600 or 900 s limit per job; a job that hit the limit is reported as "no result". The models were written within the project, with the same AI assistance as the specification, and no one outside the project has reviewed them.

Common abstractions (FR Limits). Perfect cryptography: X25519 is an abstract DH group without small-subgroup or all-zero issues (the identity validity check of Section 4.2 is what is meant to make this sound for identity keys); ML-KEM is an ideal KEM; HKDF and SHA-256 are random oracles with one function per label; AES-GCM is perfect authenticated encryption. Not modelled: nonce reuse, the derived header nonce (D6), key commitment [1], padding, lengths and byte

encodings; key deletion, crash and rollback rules, and constant-time behaviour (deleted keys are assumed gone); time, expiry, acknowledgement, approval and outbox rules. In addition: the handshake models of drafts 1–2 have *one* inviter and *one* joiner session, and the inviter-side authentication queries use a cut model; the classical ratchet model has four messages in strict ping-pong, no header encryption, no skipped keys or out-of-order delivery; the epoch-ratchet model hands the whole Double Ratchet to the adversary, unrolls four epochs in strict order and omits pacing, the four-epoch window, the repeat rule and out-of-order delivery. The engine rules of link frames (time windows, "only I offers", SYNC, the capability rule, daily limits) are not modelled; RESET is modelled only as "who can produce one that opens", not what it deletes. SwellSig and groups are in no model. No model covers skipped keys, padding, or the code.

Table 3 summarises the results; PV is ProVerif 2.05 [12], TM is Tamarin 1.12 [38]. A first formulation of 2-P2 ("the stolen state was made strictly after the key generation") was falsified because the set-up state that holds PQ_0 is created at the step that "generates" epoch 0; the lemma was reworded to "not before", which is the property intended, and then proved (FR 2-P2). We note that 3-7, the authentication of the joiner on the invite path in the presence of unbounded link sessions under the same persistent keys, is open.

6.3 Attacks found and how the design responded

A1 (Tamarin, 13a): second compromise within the healing window. The adversary copies I's state at step 0, learning the root key of the next step. After J sends m_1 and I answers with m_2, the adversary copies I again; I still holds the private ratchet key used for m_2. Together, the two copies give m_2's key; neither alone does. The published Double Ratchet behaves the same. Response: forward secrecy is stated with limit L1 and the deletion table lists DHs (Spec §3, Spec §12).

A2 (Tamarin, 16): active adversary after compromise. Holding the chain keys, the adversary injects its own ratchet public key, which the victim accepts; all later keys are then known to it. No ratchet prevents this; only re-comparing safety numbers or a new invite ends it. Response: stated as out of scope; the app asks users to compare numbers again after loss or restore (Spec §17, G5).

A3 (Tamarin, 18): no post-quantum healing in draft 1. One compromise of J after sending m_1 plus a later quantum break exposed m_4, because draft 1 healed only through X25519. Response: the epoch ratchet of draft 2 (Section 4.6); 2-P2 and 2-P5 prove the corresponding properties in the abstract model.

A4 (ProVerif, 6a, 7): the invite is a bearer token. Whoever knows the invite, or only the bundle's public keys, can build a valid HELLO and becomes the contact; the real J's message remains secret. Response: stated (Spec §3, Spec §6.3); safety numbers (property 10).

Table 3. Formal results (from FR; the draft 2 upgrade results and draft 1 sanity checks are omitted). P: proved in the model (no attack found). T: exists-trace lemma verified (a trace was found). A: attack found. NR: no result (time limit). "Exp." marks an attack that was expected (a control or an admitted limit).

#	Tool	Property (model)	Result
Draft 3, `swell3-link.pv`: persistent identities, unbounded link sessions + one invite session			
3-1	PV	Secrecy of I's first DATA after a link handshake, $\mathsf{RK}_0/\mathsf{PQ}_0$ probes, invite-path first message and reply; active network adversary	P
3-2	PV	Injective agreement: J accepts I's first DATA $\Rightarrow$ an I session sent it; I accepts a link HELLO $\Rightarrow$ a J session made it for this offer	P
3-3	PV	A RESET that opens was made by the contact, in that direction (no third party, no reflection)	P
3-4	PV	Both identity secrets revealed later: earlier session secrets remain secret	P
3-5	PV	All X25519 private keys known, frames recorded but not injected: secrets remain secret	P
3-6	PV	As 3-5 but adversary may inject	A (exp., D16); invite-path secrets kept
3-7	PV	Invite path with persistent keys: I's accepted HELLO came from the real J	NR (twice)
3-8	PV	Control: link AD without sender/receiver	A (exp.): reflection
Draft 2 handshake and epoch ratchet (one session, random identity keys)			
2-1	PV	Secrecy of J's first message, I's reply, RK_0, PQ_0	P
2-2/2-3	PV	J authenticates I; I authenticates J ("holder of the invite"; cut model)	P / P
2-4/2-5	PV	Hybrid: ML-KEM key leaked; every X25519 key broken (active)	P / P
2-6	PV	Control: both broken	A (exp.)
2-7	PV	Handshake forward secrecy: all kept state stolen later, incl. PQ_0	P
2-8/2-9	PV	Invite replaced but safety numbers match: secrecy; I authenticates J	P / P
2-P1	TM	Four epochs executable	T
2-P2	TM	PQ recovery: with all DR keys public, an epoch-e message leaks only if the stolen state was made no earlier than epoch e's key generation; passive	P
2-P3	TM	No compromise: every epoch secret with DR public	P
2-P4	TM	Tightness: theft of the pending epoch-2 seed exposes epoch 2	T
2-P5	TM	Draft 1's A3 case: one early theft, epoch 3 secret	P
2-P6	TM	2-P2 against an active adversary	A (exp., A6)
Draft 1 handshake (selected) and classical ratchet, `swell1-*`			
1, 2, 3	PV	Secrecy; J authenticates I; I authenticates J (one session; cut model for 3)	P
4, 5	PV	Hybrid secrecy with ML-KEM leaked / X25519 broken	P
6a	PV	Quantum adversary who also saw the bundle	A on I's reply (A4)
7	PV	Leaked invite, I authenticates J	A (exp., A4)
8	PV	Stolen id_I before handshake: J still authenticates I	P
9	PV	Replaced invite, numbers not compared	NR; attack A5 argued
10	PV	Replaced invite, numbers match: secrecy and mutual authentication	P
11	PV	Handshake forward secrecy	P
13	TM	FS and PCS, classical, passive while healing (with case (c), see A1)	P
13a	TM	Same without case (c)	A (A1)
14	TM	Plain FS: only theft after sending	P
15	TM	Control: bound of 13 is tight	A (exp.)
16	TM	13 against an active adversary	A (exp., A2)
17	TM	PQ secrecy of the conversation, no theft	P
18	TM	PQ recovery after theft (draft 1, no epoch ratchet)	A (A3)

A5 (argued, 9): replaced invite is a man in the middle. An adversary who replaces the invite in transit runs two sessions. ProVerif did not terminate within 600 s; the attack is argued, not machine-found. Property 10 proves the converse under matching safety numbers.

A6 (Tamarin, 2-P6): active adversary on the epoch ratchet. After a compromise, the adversary can substitute its own EK or ct inside messages it can now forge, and remain in the middle; this is A2 for the epoch ratchet. In the model EK and ct travel unauthenticated and the passivity premise of 2-P2 stands in for the AEAD that protects them in the protocol.

3-6: active quantum adversary on link sessions. With all identity secrets, the adversary forges an OFFER with its own ML-KEM key and reads that session. This is expected because link frames are classical (D16). Invite-path secrets stay secret because their EK arrived out of band.

3-8: reflection without directional AD. If the associated data does not name sender and receiver, a device's own RESET, reflected back to it, opens as if sent by the contact, since both directions share LK. This is why the AD contains both serialized keys. An analogous reflection attack was found on a resume mechanism of an earlier draft (0.1) that has since been removed (FR Old models).

Findings from internal review (not from the models). Issues below the models' abstraction level were found by internal review and fixed (Spec §1): the *sealKey* exposure (B27 H1, Section 4.3), unreadable re-sent frames (M1), replayed OFFERs and held-back RESETs (L1, Section 4.7), a stuck gap above MAX_SKIP (L2) and an unauthenticated "older app" mark (L3). Open items include: the operator can stall the epoch ratchet by dropping the conspicuously sized frames that carry EK or ct (O12); after healing, a receiver still accepts messages under up to three older epoch keys, so an adversary who stole a state and stayed passive through healing can later forge messages under an old epoch (O13); and header-key trial is quadratic in the number of distinct stored header keys (O15) (Spec §16).

Relation to recent Double Ratchet attacks (our analysis, not modelled). Cheval et al. [16] report three forward-secrecy attacks: use of an absent (None) chain key; commitment of an unconfirmed initial state that holds a signed-prekey secret shared by many sessions; and two sessions with one key, from a replayed PQXDH message whose ephemeral key is mangled by a small-order point. For draft 3: (i) Only header keys that are set are tried (Spec §8.3), and every header and chain key of a new session descends from hs, so no absent key is ever used. (ii) There is no prekey shared between sessions: I's initial state holds one-use bundle secrets, deleted at the accept commit (Spec §12), and a HELLO that fails leaves no state (Spec §6.2, Spec §8.3). (iii) A mangled E^* has a different encoding with the same DH outputs, so it changes T and hs and the HELLO no longer opens; and a one-use bundle accepts one HELLO (Spec §6.2). Arguments (ii) and (iii) do not cover the reusable service bundles of Section 4.3, for which we have not analysed

replay of an identical HELLO. These are informal arguments; no model covers them.

7 Implementation and Disclosure

Libraries. The Swift library `SwellProtocol` targets iOS 26 and macOS 26 and uses CryptoKit for X25519, HKDF-SHA-256, SHA-256, AES-GCM and ML-KEM-768 (`MLKEM768`); the minimum platform was raised to iOS 26 because there is no classical-only session (Spec §4.1). The Kotlin library `swellprotocol` runs on the JVM (Java 17, used by the Android app) and uses the lightweight API of Bouncy Castle 1.79 for X25519 and ML-KEM-768 [29]. Both are released under Apache-2.0 without application code. SwellSig's strict point checks are hand-written in both; the final RFC 8032 equation is delegated to CryptoKit and Bouncy Castle respectively (Spec §5.2).

Shared vectors. Ten JSON files fix the bytes of every layer (Spec §13): ML-KEM-768 (seeds to `EK`, recorded `ct` and `ss`, implicit rejection, an `EK` failing the FIPS 203 modulus check); identities and invalid keys; SwellSig, including the strict edge cases; every KDF, tagged hash, binding and safety number; a handshake and two-epoch conversation with out-of-order, duplicate, large, best-effort and tampered messages; link frames with reflection, wrong-channel and stale cases; invites, group states, reports and backups. Because CryptoKit cannot fix encapsulation randomness, each encapsulation is recorded and replayed, and the decapsulating side checks it with its own ML-KEM; each party's random draws are recorded in order so that an implementation can replace its random source and reproduce every byte (Spec §A.12). The vectors were generated by the Swift code; the published repository states that both implementations pass them [29]. The Kotlin code was written from the specification, the vectors and the Swift code (`CLEAN-ROOM.md`), so the two are not independent in the N-version sense: the vectors catch divergence, not a misreading the two share. NIST ACVP ML-KEM-768 vectors have not yet been added (O10).

State handling follows Section 4.5. The store is SQLCipher with `secure_delete`; neither it nor Swift's memory model guarantees physical erasure, and buffers are wiped only where the code holds the sole copy (Spec §12).

Testing status. A headless driver runs the Swift engine against the production mailbox; a live Swift-to-Swift run (first contact, messages, files, a group of three, epoch changes, restarts) was completed. Live cross-implementation runs, fuzzing beyond unit level, crash tests at every transaction boundary and on-device size reports are still to do (Spec §13, O10).

Clean-room record. The specification and both implementations were written only from public specifications and papers (the Double Ratchet, X3DH and PQXDH documents, Signal's public descriptions of SPQR and the ML-KEM

Braid, and the NIST and RFC standards cited above), the project's own documents and code, and the public APIs of CryptoKit and Bouncy Castle. No source code of libsignal, Signal's applications, the SPQR or Braid reference code, or any AGPL or GPL implementation was read, nor the XEdDSA document; `docs/CLEAN-ROOM.md` records the sources of each piece of work. That record also states a limit: because the work was AI-assisted, it cannot be ruled out that the tools' training data included public implementations; the visible differences from Signal's designs (deviations D1–D18) are listed so that reviewers can check.

Disclosure. The text of this paper, much of the specification and implementation, and the formal models and their analysis were produced with AI assistance under human direction and review. The author designed and directed the work, reviewed it, and takes responsibility for its content.

8 Limitations and Open Problems

1. **No independent audit**; internal review only. One is planned (Spec §14).
2. **Evidence is symbolic and partial.** The models are simplified, written within the project and not reviewed from outside. There is no computational proof, in particular of the combiner (D3) and of the HKDF assumptions of Section 6.1. The drafts 1–2 handshake models are single-session and 3-7 has no result. Skipped keys, out-of-order delivery, header encryption in the ratchet model, pacing, padding, the derived header nonce, SwellSig, groups and the attack classes of Section 6.3 are unmodelled.
3. **Admitted protocol limits**: L1, L2 and non-physical deletion (Section 3); no healing against an active adversary (A2, A6), and forgery under old epoch keys after healing (O13); classical identity authentication, and link frames without forward secrecy (D14, D16, 3-6); no deniability claim (group states are signed).
4. **SwellSig**: one scalar for DH and signing without a joint-security analysis; arithmetic not constant time (O17).
5. **Metadata**: timing, sizes, volume, channel identifiers, the joiner's identity key in invite-path HELLOs (D7, O2), epoch changes by size (O12), the client IP address unless Tor mode is on, and identities linkable across contacts (D14).
6. **Engineering**: groups cost one pairwise message per member and their consistency is unanalysed; ML-KEM patent status unchecked (O9); ACVP vectors, fuzzing, crash tests and live cross-implementation runs pending (O10); calls (DTLS-SRTP) are not post-quantum and are out of scope (O16).
7. **Server statement**: no proof that the stated facts or the running code are as claimed; no key rotation; witness not yet operating (Section 5).

9 Related Work

Signal protocols and their analyses. X3DH [37] and the Double Ratchet [42] were analysed computationally by Cohn-Gordon et al. [17]; Alwen, Coretti

and Dodis [2] gave modular security notions and proofs, and Bienstock et al. [10] a more complete analysis of the Double Ratchet. Cheval, Jacomme and Richards [16] gave an automated symbolic analysis in ProVerif that covers out-of-order delivery and found forward-secrecy attacks on the specification and on an implementation (Section 6.3); our ratchet model, with four messages in strict ping-pong, is far weaker. Ratcheting has also been studied as a primitive [8,27,43]; see [47] for an earlier systematisation.

Post-quantum handshakes. PQXDH [33] was analysed symbolically by Bhargavan et al. [9] and computationally by Fiedler and Günther [23]. Post-quantum X3DH-style handshakes were proposed by Brendel et al. [14] and Hashimoto et al. [26]. Hybrid KEM combiners are treated in [24,11]. Swell's handshake differs from PQXDH in having no prekey directory or prekey signatures (D1) and in binding the transcript to the invite.

Post-quantum ratchets. Dodis et al. [21] proposed the Triple Ratchet, which runs a KEM-based ratchet beside the Double Ratchet and transports large KEM material in erasure-coded chunks; Auerbach et al. [4] compared bandwidth-constrained post-quantum designs. Signal's specification [42] now includes SPQR and the Triple Ratchet, with the ML-KEM Braid [45] as a sparse continuous key agreement. There the post-quantum ratchet has its own root key and chains, and a KDF combines one message key from each ratchet. Swell's combine has the same final shape, but one epoch key serves every message of an epoch in both directions, whole EK and ct travel in single messages, and epoch changes alternate between the parties (D10, D13); we have not analysed this choice in the model of [4]. Apple's PQ3 protocol for iMessage also rekeys with a post-quantum KEM during a conversation; Stebila [46] gave a computational analysis and Linker, Sasse and Basin [36] a Tamarin analysis that, unlike ours, covers unbounded ratchet loops.

Groups. MLS [6] provides tree-based group key agreement, analysed for example in [3]. Swell instead fans group frames out over pairwise sessions, as some of the designs studied by Rösler, Mainka and Schwenk [44] do; we have not evaluated their group-consistency attacks against Swell.

Signatures and logs. Divergent EdDSA verification behaviour is discussed in [15,13]; joint use of a key for encryption and signing in [25,19]. Our statement log is a minimal hash chain in the spirit of Certificate Transparency [35] and key transparency [39], without their Merkle-tree proofs.

10 Conclusion and Invitation to Cryptanalysis

Symbolic models of simplified components of the Swell Protocol found no attack on the stated properties (one authentication query, 3-7, gave no result), found the expected attacks on the admitted limits, and led to two changes: post-quantum healing (A3) and a corrected forward-secrecy claim (A1). Open are

computational proofs, multi-session models of the invite path, selective dropping (O12) and stale epochs (O13), the shared-scalar signature, replay against reusable bundles, and an independent audit.

We invite cryptanalysis of the specification, the models and the code. The specification, deviations table (Spec §15), open items (Spec §16), formal models, test vectors and both libraries are at `https://github.com/swellchat/swell-protocol` [29]; reporting instructions are at `https://getswell.org/security` [28].

References

1. Albertini, A., Duong, T., Gueron, S., Kölbl, S., Luykx, A., Schmieg, S.: How to abuse and fix authenticated encryption without key commitment. In: 31st USENIX Security Symposium (USENIX Security 22). pp. 3291–3308 (2022)
2. Alwen, J., Coretti, S., Dodis, Y.: The Double Ratchet: Security notions, proofs, and modularization for the Signal protocol. In: Advances in Cryptology – EUROCRYPT 2019. pp. 129–158. LNCS (2019). doi:10.1007/978-3-030-17653-2_5
3. Alwen, J., Coretti, S., Dodis, Y., Tselekounis, Y.: Security analysis and improvements for the IETF MLS standard for group messaging. In: Advances in Cryptology – CRYPTO 2020. pp. 248–277. LNCS (2020). doi:10.1007/978-3-030-56784-2_9
4. Auerbach, B., Dodis, Y., Jost, D., Katsumata, S., Schmidt, R.: How to compare bandwidth constrained two-party secure messaging protocols: A quest for a more efficient and secure post-quantum protocol. In: 34th USENIX Security Symposium (USENIX Security 25). pp. 6717–6736 (2025)
5. Backendal, M., Bellare, M., Günther, F., Scarlata, M.: When messages are keys: Is HMAC a dual-PRF? In: Advances in Cryptology – CRYPTO 2023. pp. 661–693. LNCS (2023). doi:10.1007/978-3-031-38548-3_22
6. Barnes, R., Beurdouche, B., Robert, R., Millican, J., Omara, E., Cohn-Gordon, K.: The Messaging Layer Security (MLS) protocol. RFC 9420 (Jul 2023). doi:10.17487/RFC9420
7. Bellare, M.: New proofs for NMAC and HMAC: Security without collision-resistance. In: Advances in Cryptology – CRYPTO 2006. pp. 602–619. LNCS (2006). doi:10.1007/11818175_36
8. Bellare, M., Singh, A.C., Jaeger, J., Nyayapati, M., Stepanovs, I.: Ratcheted encryption and key exchange: The security of messaging. In: Advances in Cryptology – CRYPTO 2017. pp. 619–650. LNCS (2017). doi:10.1007/978-3-319-63697-9_21
9. Bhargavan, K., Jacomme, C., Kiefer, F., Schmidt, R.: Formal verification of the PQXDH post-quantum key agreement protocol for end-to-end secure messaging. In: 33rd USENIX Security Symposium (USENIX Security 24). pp. 469–486 (2024)
10. Bienstock, A., Fairoze, J., Garg, S., Mukherjee, P., Raghuraman, S.: A more complete analysis of the Signal Double Ratchet algorithm. In: Advances in Cryptology – CRYPTO 2022. pp. 784–813. LNCS (2022). doi:10.1007/978-3-031-15802-5_27
11. Bindel, N., Brendel, J., Fischlin, M., Goncalves, B., Stebila, D.: Hybrid key encapsulation mechanisms and authenticated key exchange. In: Post-Quantum Cryptography (PQCrypto 2019). pp. 206–226. LNCS (2019). doi:10.1007/978-3-030-25510-7_12
12. Blanchet, B.: Modeling and verifying security protocols with the applied pi calculus and ProVerif. Foundations and Trends in Privacy and Security **1**(1–2), 1–135 (2016). doi:10.1561/3300000004

13. Brendel, J., Cremers, C., Jackson, D., Zhao, M.: The provable security of Ed25519: Theory and practice. In: 2021 IEEE Symposium on Security and Privacy (SP). pp. 1659–1676 (2021). doi:10.1109/SP40001.2021.00042
14. Brendel, J., Fischlin, M., Günther, F., Janson, C., Stebila, D.: Towards post-quantum security for Signal's X3DH handshake. In: Selected Areas in Cryptography (SAC 2020). pp. 404–430. LNCS (2021). doi:10.1007/978-3-030-81652-0_16
15. Chalkias, K., Garillot, F., Nikolaenko, V.: Taming the many EdDSAs. In: Security Standardisation Research (SSR 2020). pp. 67–90. LNCS (2020). doi:10.1007/978-3-030-64357-7_4
16. Cheval, V., Jacomme, C., Richards, J.: Automated formal analysis of Signal's Double Ratchet: Attacks, fixes and security proofs. In: 2026 IEEE Symposium on Security and Privacy (SP). pp. 1522–1538 (2026). doi:10.1109/SP63933.2026.00120
17. Cohn-Gordon, K., Cremers, C., Dowling, B., Garratt, L., Stebila, D.: A formal security analysis of the Signal messaging protocol. In: 2017 IEEE European Symposium on Security and Privacy (EuroS&P). pp. 451–466 (2017). doi:10.1109/EuroSP.2017.27
18. Cohn-Gordon, K., Cremers, C., Garratt, L.: On post-compromise security. In: IEEE 29th Computer Security Foundations Symposium (CSF 2016). pp. 164–178 (2016). doi:10.1109/CSF.2016.19
19. Degabriele, J.P., Lehmann, A., Paterson, K.G., Smart, N.P., Strefler, M.: On the joint security of encryption and signature in EMV. In: Topics in Cryptology – CT-RSA 2012. pp. 116–135. LNCS (2012). doi:10.1007/978-3-642-27954-6_8
20. Dingledine, R., Mathewson, N., Syverson, P.: Tor: The second-generation onion router. In: 13th USENIX Security Symposium (2004)
21. Dodis, Y., Jost, D., Katsumata, S., Prest, T., Schmidt, R.: Triple ratchet: A bandwidth efficient hybrid-secure Signal protocol. In: Advances in Cryptology – EUROCRYPT 2025. pp. 302–331. LNCS (2025). doi:10.1007/978-3-031-91101-9_11, revised version: Cryptology ePrint Archive, Paper 2025/078
22. Dworkin, M.: Recommendation for block cipher modes of operation: Galois/Counter Mode (GCM) and GMAC. Special Publication 800-38D, NIST (Nov 2007). doi:10.6028/NIST.SP.800-38D
23. Fiedler, R., Günther, F.: Security analysis of Signal's PQXDH handshake. In: Public-Key Cryptography – PKC 2025. pp. 137–169. LNCS (2025). doi:10.1007/978-3-031-91823-0_5
24. Giacon, F., Heuer, F., Poettering, B.: KEM combiners. In: Public-Key Cryptography – PKC 2018. pp. 190–218. LNCS (2018). doi:10.1007/978-3-319-76578-5_7
25. Haber, S., Pinkas, B.: Securely combining public-key cryptosystems. In: Proceedings of the 8th ACM Conference on Computer and Communications Security (CCS 2001). pp. 215–224 (2001). doi:10.1145/501983.502013
26. Hashimoto, K., Katsumata, S., Kwiatkowski, K., Prest, T.: An efficient and generic construction for Signal's handshake (X3DH): Post-quantum, state leakage secure, and deniable. In: Public-Key Cryptography – PKC 2021. pp. 410–440. LNCS (2021). doi:10.1007/978-3-030-75248-4_15
27. Jaeger, J., Stepanovs, I.: Optimal channel security against fine-grained state compromise: The safety of messaging. In: Advances in Cryptology – CRYPTO 2018. pp. 33–62. LNCS (2018). doi:10.1007/978-3-319-96884-1_2
28. Jiji Labs: Swell security page. https://getswell.org/security (2026)
29. Jiji Labs: SwellProtocol: specification, formal models, test vectors and Swift/Kotlin library. https://github.com/swellchat/swell-protocol (2026), Apache-2.0. Specification draft 3 in docs/SWELL-PROTOCOL-SPEC.md

30. Josefsson, S., Liusvaara, I.: Edwards-Curve Digital Signature Algorithm (EdDSA). RFC 8032 (Jan 2017). doi:10.17487/RFC8032
31. Krawczyk, H.: Cryptographic extraction and key derivation: The HKDF scheme. In: Advances in Cryptology – CRYPTO 2010. pp. 631–648. LNCS (2010). doi:10.1007/978-3-642-14623-7_34
32. Krawczyk, H., Eronen, P.: HMAC-based extract-and-expand key derivation function (HKDF). RFC 5869 (May 2010). doi:10.17487/RFC5869
33. Kret, E., Schmidt, R.: The PQXDH key agreement protocol. https://signal.org/docs/specifications/pqxdh/ (2023), revision 3, 2023-05-24, last updated 2024-01-23
34. Langley, A., Hamburg, M., Turner, S.: Elliptic curves for security. RFC 7748 (Jan 2016). doi:10.17487/RFC7748
35. Laurie, B., Langley, A., Kasper, E.: Certificate transparency. RFC 6962 (Jun 2013). doi:10.17487/RFC6962
36. Linker, F., Sasse, R., Basin, D.: A formal analysis of Apple's iMessage PQ3 protocol. In: 34th USENIX Security Symposium (USENIX Security 25). pp. 5015–5034 (2025)
37. Marlinspike, M., Perrin, T.: The X3DH key agreement protocol. https://signal.org/docs/specifications/x3dh/ (2016), revision 1, 2016-11-04; T. Perrin, ed.
38. Meier, S., Schmidt, B., Cremers, C., Basin, D.: The TAMARIN prover for the symbolic analysis of security protocols. In: Computer Aided Verification (CAV 2013). pp. 696–701. LNCS (2013). doi:10.1007/978-3-642-39799-8_48
39. Melara, M.S., Blankstein, A., Bonneau, J., Felten, E.W., Freedman, M.J.: CONIKS: Bringing key transparency to end users. In: 24th USENIX Security Symposium (USENIX Security 15). pp. 383–398 (2015)
40. National Institute of Standards and Technology: Secure hash standard (SHS). Federal Information Processing Standards Publication FIPS 180-4, NIST (Aug 2015). doi:10.6028/NIST.FIPS.180-4
41. National Institute of Standards and Technology: Module-lattice-based key-encapsulation mechanism standard. Federal Information Processing Standards Publication FIPS 203, NIST (Aug 2024). doi:10.6028/NIST.FIPS.203
42. Perrin, T., Marlinspike, M., Schmidt, R.: The Double Ratchet algorithm. https://signal.org/docs/specifications/doubleratchet/ (2025), revision 4, 2025-11-04; T. Perrin, ed.; §§5–6: SPQR and the Triple Ratchet
43. Poettering, B., Rösler, P.: Towards bidirectional ratcheted key exchange. In: Advances in Cryptology – CRYPTO 2018. pp. 3–32. LNCS (2018). doi:10.1007/978-3-319-96884-1_1
44. Rösler, P., Mainka, C., Schwenk, J.: More is less: On the end-to-end security of group chats in Signal, WhatsApp, and Threema. In: 2018 IEEE European Symposium on Security and Privacy (EuroS&P). pp. 415–429 (2018). doi:10.1109/EuroSP.2018.00036
45. Schmidt, R.: The ML-KEM Braid protocol. https://signal.org/docs/specifications/mlkembraid/ (2025), revision 1, 2025-02-21, last updated 2025-09-26
46. Stebila, D.: Security analysis of the iMessage PQ3 protocol. Cryptology ePrint Archive, Paper 2024/357 (2024), https://eprint.iacr.org/2024/357
47. Unger, N., Dechand, S., Bonneau, J., Fahl, S., Perl, H., Goldberg, I., Smith, M.: SoK: Secure messaging. In: 2015 IEEE Symposium on Security and Privacy (SP). pp. 232–249 (2015). doi:10.1109/SP.2015.22