The Design and Formal Analysis of the Quantum Secure Tunneling Protocol

John G. Underhill

Quantum Resistant Cryptographic Solutions Corporation

Abstract. This paper gives a formal analysis of the Quantum Secure Tunneling Protocol (QSTP), a root-anchored, server-authenticated tunneling protocol for post-quantum client-server deployments. The implemented protocol uses a trusted root certificate to authenticate a *pinned* server certificate, a post-quantum signature scheme to authenticate the server's ephemeral encapsulation key, a post-quantum key encapsulation mechanism to establish a session secret, cSHAKE-256 to derive directional channel keys with transcript-bound domain separation, and the RCS authenticated encryption construction to protect application packets. By default the asymmetric primitives are instantiated at NIST security category 5 (approximately 256-bit, e.g. ML-KEM-1024 and ML-DSA-87); lower-category parameter sets are available as compile-time options and are not claimed to provide category-5 security. The analysis models the implemented one-way trust structure rather than an idealized mutual-authentication protocol. It states the assumptions required for authenticated key exchange, server identity authentication, signed ephemeral-key authenticity, transcript acknowledgement, channel confidentiality, channel integrity, strict in-connection replay rejection, and forward secrecy against later compromise of the long-term server signing key. The paper also identifies the limits of the construction, including the absence of client authentication, dependence on authenticated root distribution and on pinned certificate provisioning, the absence of an implemented revocation protocol, the reliance of in-connection replay rejection on a monotonic authenticated sequence number (with a bounded timestamp window only for stale and newly initialized packets), the fact that the cleartext exchange response provides transcript acknowledgement rather than cryptographic key confirmation, and the requirement that the active transport profile remain the RCS profile unless a compliant per-record GCM construction is specified and tested.

Keywords: QSTP · post-quantum tunneling · server authentication · authenticated encryption · cSHAKE

1 Introduction

QSTP is a post-quantum tunneling protocol designed for controlled client-server deployments in which a client authenticates a server through a root-signed certificate and then establishes an authenticated encrypted channel. The protocol is deliberately narrow. It does not implement mutual authentication, cipher-suite negotiation, public web PKI semantics, or a general-purpose transport framework. It implements a fixed root-anchored trust model, a linear key exchange, and a symmetric channel whose record headers are authenticated as associated data.

The security objective is to protect application messages between a client and a server after the client has accepted the server certificate and completed the key exchange. QSTP uses post-quantum asymmetric primitives for authentication and key establishment, SHA3-256 for certificate and transcript hashing, cSHAKE-256 for key derivation, and RCS for authenticated encryption. The implemented channel is unilateral with respect to authentication: the server is authenticated to the client, while the client is identified only by possession of the transport connection and the derived channel state.

QSTP uses a *pinned* server-identity model. The client is provisioned out of band with both the trusted root certificate and the specific server certificate it intends to contact; the server certificate is not transmitted in band during the handshake. Server identity is therefore established by two independent bindings: the client already holds the exact server certificate (pinning), and that certificate is verified to chain to the trusted root. QSTP certificates are a fixed-field custom format rather than X.509; this removes ASN.1 parsing ambiguity and makes the certificate-hash and transcript serialization deterministic by construction. This model is closer to SSH known-hosts pinning or certificate pinning than to in-band web PKI certificate presentation, and it is the structure that the formal model below reflects.

The analysis in this paper follows the implemented behavior of QSTP. It treats the source code and technical specification as the protocol definition, but abstracts from C-level details except where such details affect security. In particular, the model includes the concrete transcript inputs, fixed packet header format, little-endian serialization, strict sequence validation, timestamp freshness checks, certificate hash construction, cSHAKE domain separation, ratchet-token length requirements, and generic handling of unauthenticated pre-key error packets.

The formal model is aligned with the QSTP reference implementation: the core protocol library (`qstp.c`, `kex.c`, `client.c`, `server.c`, `root.c`, and the headers) together with the reference client, server, and root applications. The model fixes the transcript-hash chaining algebra, the session-KDF construction and literal name string, the 21-byte header field order, the packet encryption/decryption and sequence/timestamp logic, the certificate-pair transcript seed, the ratchet key schedule, the per-handshake freshness of the server KEM key pair, and the cleartext form of the exchange response. Recommended conformance vectors are listed in Section 33.

1.1 Contributions

The paper makes the following technical contributions.

1. It gives a code-aligned formal specification of the QSTP handshake, certificate validation procedure, transcript construction, key derivation process, channel state, and packet acceptance rules.
2. It defines a server-authenticated AKE model appropriate for a one-way trust protocol. The model does not assume client authentication and does not infer properties that the implemented protocol does not provide.
3. It proves server identity authentication, signed ephemeral-key authenticity, and session-

key indistinguishability under the EUF-CMA security of the signature scheme, the IND-CCA security of the KEM, and the pseudorandomness of cSHAKE when used with fixed domain separation and transcript customization.

4. It gives a channel-security argument reducing application packet confidentiality and integrity to the authenticated encryption security of RCS under unique directional key and nonce state.
5. It records the operational limits of the construction, including timestamp-window replay exposure, certificate revocation limitations, server-key compromise consequences, endpoint compromise limits, and disabled AES-GCM transport profile status.

1.2 Terminology

The word *server* denotes the QSTP application server (QAS). The word *root* denotes the root domain security server (RDS) or equivalent root signing authority. The root authority signs the server certificate or self-signs its own root certificate, depending on deployment mode. The root is not an online participant in ordinary channel establishment. The client possesses the root certificate through embedding, provisioning, or a trusted registration process before it validates the server certificate.

Content Index

2 Related Work and Standards Context

QSTP belongs to the family of authenticated key exchange and secure-channel protocols. It is structurally simpler than TLS 1.3 and WireGuard because it has no cipher-suite negotiation and no mutual-authentication branch. TLS 1.3 specifies a general Internet transport protocol with negotiated parameters, certificate-based authentication, optional client authentication, resumption, and extensive extension processing. WireGuard is a VPN protocol based on the Noise framework and has been the subject of mechanized analysis. QSTP instead targets fixed-configuration post-quantum deployments in which minimizing negotiation and constraining the state machine are explicit design goals.

The asymmetric primitive references are the NIST post-quantum standards for ML-KEM, ML-DSA, and SLH-DSA, together with Classic McEliece for code-based KEM configuration sets. The hash and KDF references are FIPS 202 for SHA-3 and SHAKE, and SP 800-185 for cSHAKE and KMAC. The packet-security analysis uses the standard AEAD notion of confidentiality and integrity with associated data. The RCS construction is a QRCS-specific authenticated encryption construction and remains the active QSTP transport profile in the implementation considered here.

3 Implemented Protocol Summary

3.1 Roles

The protocol has three logical roles.

Root authority. The root authority owns a long-term signing key and issues or verifies root certificate material. In the default mode the root certificate is self-signed, and verification checks the root signature against the root's own verification key. In external-root mode (selected by the `QSTP_EXTERNAL_SIGNED_ROOT` build configuration), the root certificate is signed by an external authority whose verification key is fetched via the certificate authority and key-identity fields or embedded in the application. The root authority signs server certificates offline or through a controlled signing workflow.

Server. The server owns a root-signed server certificate and a long-term signing key. During a QSTP handshake it generates an ephemeral KEM key pair, signs a transcript-bound hash of the ephemeral public key, decapsulates the client's KEM ciphertext, derives channel keys, and enters the established state.

Client. The client is provisioned out of band with the trusted root certificate and the specific (pinned) server certificate before any handshake. It validates the server certificate under the trusted root, verifies the server's signature on the transcript-bound hash of the ephemeral KEM public key, encapsulates the KEM secret, derives directional channel keys, and enters the established state only after the exchange response matches the locally computed final transcript value. This exchange-response check acknowledges transcript agreement; possession of the derived server-to-client channel key is confirmed only when the first authenticated record from the server is accepted. The server certificate is held locally and is not received in band, so in-band certificate substitution is not part of the threat surface.

The protocol does not authenticate the client to the server. A deployment requiring client authorization must add a separate authorization layer, certificate mechanism, token mechanism, or application-layer policy.

3.2 Primitive Inventory

QSTP uses the following primitive interfaces.

Primitive	Implemented role	Security assumption
Post-quantum KEM	ML-KEM (FIPS 203, default ML-KEM-1024) or Classic McEliece configuration sets. The server generates the ephemeral KEM key pair; the client encapsulates.	IND-CCA security of the selected KEM at the selected parameter set.
Post-quantum signature	ML-DSA (FIPS 204, default ML-DSA-87) or SLH-DSA (FIPS 205) configuration sets. The server signs transcript-bound values and the root signs server certificates.	EUF-CMA security of the selected signature scheme.
SHA3-256	Certificate hashing and transcript hashing.	Collision resistance and second-preimage resistance sufficient for transcript binding.
cSHAKE-256	Session key derivation. The session KDF uses a fixed name (function) string and the final transcript hash sch_2 as customization.	Pseudorandomness under secret input with fixed domain separation.
RCS AEAD	Application data and authenticated error packets after establishment, applied as a per-direction *online/stateful* AEAD whose cipher and authentication state advance with each record.	AEAD confidentiality and integrity under unique directional keys with non-repeating per-record state.

Unless a different compile-time configuration is selected, the asymmetric primitives are instantiated at NIST security category 5 (ML-KEM-1024 and ML-DSA-87), and the symmetric primitives (RCS, SHA3-256, cSHAKE-256) operate at a 256-bit security target. Concrete-security statements in this paper refer to this default unless a specific configuration set is named (Section 25).

The AES-GCM transport profile is not analyzed as an active profile. It is reserved and disabled unless a per-record nonce construction and state-reset design are specified and validated against an independent GCM implementation. All channel-security theorems in this paper refer to the active RCS profile.

3.3 Packet Header

Each QSTP packet begins with a fixed 21-byte header:

$$\mathsf{hdr} = \mathsf{flag}[1] \parallel \mathsf{msglen}[4] \parallel \mathsf{sequence}[8] \parallel \mathsf{utctime}[8].$$

The integer fields are serialized in little-endian order. This serialized order—flag, then little-endian msglen, sequence, and utctime—is the exact byte layout produced by `qstp_packet_header_serialize` and is interoperability-critical, since the serialized header is both transcript-bound (in phash) and supplied as AEAD associated data

(Section G). The header is authenticated as associated data for encrypted packets. The maximum packet message length is `0x10000` = 65536 bytes. The active RCS profile uses a 32-byte authentication tag; therefore, an encrypted packet carrying plaintext must have $\mathsf{msglen} > 32$. Symmetric ratchet packets have an exact encrypted payload length of $32 + 32$ bytes, namely a 32-byte token and a 32-byte tag.

3.4 Live Handshake Flags

The live key-exchange path uses the following flags: `connect_request`, `connect_response`, `exchange_request`, and `exchange_response`. The client then performs a local establishment verification by comparing the returned transcript hash against its locally computed transcript hash. The flags `exstart`, `establish_request`, `establish_response`, and `transfer_request` are not part of the live key-exchange path analyzed here.

4 Certificate and Transcript Construction

4.1 Certificate Hashing

The root and server certificate hash functions process fixed-size fields in a canonical order. No variable-length string operation determines the hash boundary. For a server certificate, the hash commits to the signature verification key, issuer, root serial number, server serial number, validity interval, algorithm identifier, version, and any mode-specific external-signing fields required by the compilation configuration. The hash output length is 32 bytes.

Definition 1 (Server certificate validity)**.** A server certificate is valid for a client if the root certificate is trusted, the server certificate signature verifies under the root verification key, the recomputed server certificate hash equals the signed value, the certificate validity interval contains the current time, and the advertised algorithm and version fields match the local protocol configuration.

Certificate validation authenticates the long-term server verification key. It does not prove that the server endpoint is uncompromised, that the root key is uncompromised, or that a revoked certificate is rejected unless a revocation mechanism is added by deployment policy.

4.2 Initial Transcript Value

The implemented initial transcript value is the hash of the root and server certificate pairing, not merely a hash of the configuration string, serial number, and verification key. Concretely the implementation computes

$$\mathsf{sch}_0 = \mathsf{CertHash}(\mathsf{RootCert}, \mathsf{ServerCert}) = H\big(H(\mathsf{RootCert}) \parallel H(\mathsf{ServerCert})\big),$$

where each certificate hash commits the certificate's fixed fields in canonical order. This value binds the session to the trusted root certificate and the authenticated server certificate. The connect request carries the server certificate serial number and the protocol configuration string. The server verifies, with a constant-time comparison of the serial and a byte comparison of the configuration string, that both match its certificate and configured protocol set before proceeding. The analyzed transcript seed is the certificate-pair hash above; descriptions inconsistent with this expression are non-normative.

4.3 Handshake Transcript

Let hdr_2 denote the serialized header of the connect response. Let pk_{kem} denote the server's ephemeral KEM public key. The server computes

$$\mathsf{phash} = H(\mathsf{hdr}_2 \parallel \mathsf{sch}_0 \parallel \mathsf{pk}_{kem}),$$

where H is SHA3-256. The server signs phash, sends pk_{kem} and the signature, and updates

$$\mathsf{sch}_1 = H(\mathsf{sch}_0 \parallel \mathsf{phash}).$$

The client verifies the signature, recomputes phash, compares it in constant time, and then performs the same transcript update.

Let ct_{kem} denote the client KEM ciphertext. Both parties compute

$$\mathsf{sch}_2 = H(\mathsf{sch}_1 \parallel \mathsf{ct}_{kem}).$$

The final transcript value sch_2 is the customization input to the session KDF and is returned by the server in the exchange response. In the implementation the server copies sch_2 into the exchange-response message in cleartext (it is not encrypted under the derived channel key), and the client accepts only on a byte-equality match with its local sch_2. Because sch_2 is a hash of public transcript values, this step verifies that the received acknowledgement equals the client's local transcript value; it does not prove possession of the derived channel key or independent server liveness after the signed connect response. Key confirmation is implicit on the first authenticated application record. This is analyzed in Section 12 and Appendix E, and the protocol's security results do not depend on the stronger reading.

5 Handshake Algorithms

Algorithm 1 Client-side QSTP handshake

Require: trusted root certificate, server certificate, server address

verify the server certificate under the trusted root certificate
compute $\mathsf{sch}_0 \leftarrow \mathsf{CertHash}(\mathsf{RootCert}, \mathsf{ServerCert})$
send $\mathsf{ConnectRequest}(\mathsf{serial}, \mathsf{cfg})$
receive $\mathsf{ConnectResponse}(\mathsf{pk}_{kem}, \sigma)$
compute $\mathsf{phash} \leftarrow H(\mathsf{hdr}_2 \parallel \mathsf{sch}_0 \parallel \mathsf{pk}_{kem})$
verify $\mathsf{SIG.Verify}(\mathsf{vk}_S, \mathsf{phash}, \sigma) = 1$
update $\mathsf{sch}_1 \leftarrow H(\mathsf{sch}_0 \parallel \mathsf{phash})$
compute $(\mathsf{ct}_{kem}, \mathsf{sec}) \leftarrow \mathsf{KEM.Encaps}(\mathsf{pk}_{kem})$
update $\mathsf{sch}_2 \leftarrow H(\mathsf{sch}_1 \parallel \mathsf{ct}_{kem})$
derive directional keys using $\mathsf{KDF}(\mathsf{sec}, \mathsf{sch}_2)$
send $\mathsf{ExchangeRequest}(\mathsf{ct}_{kem})$
receive $\mathsf{ExchangeResponse}(\mathsf{sch}_2')$
if byte comparison $\mathsf{sch}_2' = \mathsf{sch}_2$ fails **then**
    erase provisional state and reject
else
    set channel state to established
end if

Algorithm 2 Server-side QSTP handshake

Require: server certificate and server signing key

receive $\mathsf{ConnectRequest}(\mathsf{serial}, \mathsf{cfg})$
verify requested serial and configuration string
compute $\mathsf{sch}_0 \leftarrow \mathsf{CertHash}(\mathsf{RootCert}, \mathsf{ServerCert})$
generate $(\mathsf{pk}_{kem}, \mathsf{sk}_{kem}) \leftarrow \mathsf{KEM.Gen}()$
compute $\mathsf{phash} \leftarrow H(\mathsf{hdr}_2 \parallel \mathsf{sch}_0 \parallel \mathsf{pk}_{kem})$
compute $\sigma \leftarrow \mathsf{SIG.Sign}(\mathsf{sk}_S, \mathsf{phash})$
update $\mathsf{sch}_1 \leftarrow H(\mathsf{sch}_0 \parallel \mathsf{phash})$
send $\mathsf{ConnectResponse}(\mathsf{pk}_{kem}, \sigma)$
receive $\mathsf{ExchangeRequest}(\mathsf{ct}_{kem})$
compute $\mathsf{sec} \leftarrow \mathsf{KEM.Decaps}(\mathsf{sk}_{kem}, \mathsf{ct}_{kem})$
erase sk_{kem}
update $\mathsf{sch}_2 \leftarrow H(\mathsf{sch}_1 \parallel \mathsf{ct}_{kem})$
derive directional keys using $\mathsf{KDF}(\mathsf{sec}, \mathsf{sch}_2)$
send $\mathsf{ExchangeResponse}(\mathsf{sch}_2)$
set channel state to established

6 Key Derivation and Directional Channel State

The session KDF uses cSHAKE-256 with a fixed name string and a transcript customization string:

$$\mathsf{prnd} \leftarrow \mathsf{cSHAKE256}(N = \texttt{QSTP-1.4-SESSION},\ S = \mathsf{sch}_2,\ X = \mathsf{sec}).$$

The output is partitioned into directional key and nonce material:

$$\mathsf{prnd} \mapsto (k_0, n_0, k_1, n_1).$$

The client assigns (k_0, n_0) to its transmit cipher and (k_1, n_1) to its receive cipher. The server assigns (k_0, n_0) to its receive cipher and (k_1, n_1) to its transmit cipher. Thus the client transmit state equals the server receive state, and the server transmit state equals the client receive state. The transcript value sch_2 is public, but it binds all key derivation to the certificate pair, connect-response header, signed ephemeral KEM public key, and KEM ciphertext. The shared secret sec is the confidential KDF input.

The fixed name string separates the session KDF from other Keccak uses in the protocol. The customization string separates sessions by transcript. The security argument assumes that the KDF invocation used for session keys is not reused with the same name string for unrelated purposes.

Per-record state and nonce uniqueness. Each directional cipher is initialized once from its derived key and nonce (k_i, n_i) and is thereafter used as an *online (stateful) AEAD*: the keystream and authentication state advance internally with every sealed or opened record, so no two records in a direction are processed under the same internal cipher/MAC state even though the initialization nonce is fixed. Record uniqueness within a direction is therefore a consequence of state advancement, not of a per-record nonce. Uniqueness *across* connections follows from the freshness of the per-handshake server KEM key pair and KEM ciphertext, which make sec and sch_2—hence (k_0, n_0, k_1, n_1)—fresh for each session except with negligible probability (Assumption 6). This per-record running state is the property that the disabled AES-GCM profile lacks: GCM is not self-chaining across records, so it would require an explicit unique per-record IV and per-record re-initialization, which is why the GCM profile is reserved pending such a design.

7 Encrypted Channel

7.1 Packet Encryption

For a plaintext message m, the sender increments its transmit sequence number, constructs a header hdr, serializes it in little-endian field order, and passes hdr to RCS as associated data. It then encrypts m and appends the authentication tag:

$$(c, \mathsf{tag}) \leftarrow \mathsf{RCS.Seal}(k, n, \mathsf{aad} = \mathsf{hdr}, m).$$

The message length field records the ciphertext and tag length. The timestamp records the local UTC time at packet creation.

7.2 Packet Decryption

The receiver first checks that the packet sequence is exactly one greater than the receive sequence counter. It next checks the timestamp against the configured freshness threshold. It then serializes the header as associated data, verifies the RCS tag, and decrypts only if authentication succeeds. The receive sequence counter advances only after successful authentication. A failed tag, stale timestamp, unexpected flag, short encrypted message, or out-of-sequence packet causes rejection.

Definition 2 (Strict monotonic acceptance)**.** A QSTP encrypted packet is sequence-valid for a connection state s if and only if its sequence number is $s.\mathsf{rxseq} + 1$. No replay window accepts older packets and no out-of-order delivery is accepted by the base protocol.

Strict monotonic acceptance simplifies replay reasoning. The pre-authentication sequence and timestamp checks on the cleartext header act as denial-of-service prefilters; the binding security guarantee is that the sequence number is part of the AEAD associated data, so an accepted record carries an *authenticated* sequence equal to $\mathsf{rxseq} + 1$, and a replayed or reordered record (whose authenticated sequence is not the expected value) cannot pass AEAD verification without a forgery. Strict acceptance does not tolerate packet reordering and therefore places ordering responsibility on the transport or application layer.

7.3 Error Packets

After establishment, error packets are encrypted and authenticated like other channel messages. Before establishment, a received error-condition flag is treated as a generic connection failure. The receiver does not derive protocol control flow from an unauthenticated payload byte in the pre-key phase. This limits pre-key attacker influence to denial of service through connection interruption, which cannot be fully eliminated from an unauthenticated network handshake.

8 Symmetric Ratchet

QSTP seeds a persistent ratchet state rtcs during the base handshake, copied from the internal cSHAKE state after the directional keys have been extracted and the state has been permuted, so that the stored ratchet seed is independent of the active channel keys. The ratchet operation is not automatic. It is invoked through a dedicated authenticated packet flag and carries an encrypted 32-byte ratchet token; the encrypted ratchet record has exact length $32 + 32 = 64$ bytes (token plus tag), is bound to its header as associated data, and is rejected unless its length and sequence are exact. The ratchet is implemented in the protocol library and exposed through `qstp_send_symmetric_ratchet_request`;

the reference console applications do not invoke it, so it is an available but not-default channel-maintenance feature.

A ratchet step derives new directional key material from the received 32-byte token τ_t and the current ratchet state r_t via a domain-separated cSHAKE-512 invocation, $\mathsf{prnd} \leftarrow \mathsf{cSHAKE512}(N = \text{protocol-set string}, S = r_t, X = \tau_t)$, re-initializes both directional ciphers from prnd, then permutes and stores the next ratchet state r_{t+1} and erases r_t and the prior keys. The session KDF and the ratchet KDF are domain-separated by distinct cSHAKE name strings (the session name `QSTP-1.4-SESSION` versus the protocol-set string) and distinct rates (256 versus 512). The intended property is *forward secrecy across epochs* (sometimes called backward protection): compromise of the ratchet state at epoch t does not reveal channel keys of epochs $t' < t$, assuming the state update is one-way and earlier states were erased. We use "forward secrecy across epochs" for this property to avoid the well-known forward/backward terminology collision. The ratchet does *not* provide post-compromise security in the resident-adversary model: because tokens traverse the (possibly compromised) channel, an adversary who holds the epoch-t channel keys reads τ_{t+1} and can recompute epoch-t+1 keys. Healing therefore requires that some ratchet token be injected with entropy the adversary does not observe at the time of compromise.

9 Formal Model

9.1 Parties and Sessions

There is a set of clients $\mathcal{C}$, servers $\mathcal{S}$, and a root authority R. Each server $S \in \mathcal{S}$ has a long-term signing key pair $(\mathsf{sk}_S, \mathsf{vk}_S)$ and a server certificate signed under the root. A client accepts vk_S only after validating that certificate under the trusted root certificate.

A session is identified by a role, peer identity, local state, transcript values $\mathsf{sch}_0, \mathsf{sch}_1, \mathsf{sch}_2$, packet sequence counters, and derived channel keys. A client session and server session are partners if they use the same server certificate, the same ephemeral KEM public key, the same KEM ciphertext, the same final transcript hash, and matching directional key assignment.

9.2 Adversary

The adversary controls the network. It may deliver, delay, reorder, drop, inject, and modify packets. It may initiate sessions with honest parties. It may reveal completed session keys subject to freshness restrictions. It may corrupt long-term server signing keys after target sessions complete in forward-secrecy experiments. It may not obtain the trusted root private signing key in the base model, because compromise of that key invalidates the server-authentication assumption. It may not read protected endpoint memory except through explicit corruption queries.

9.3 Freshness

A client session is fresh for AKE security if the adversary has not revealed the session key of that session or its matching partner, has not corrupted the server signing key before the signed connect response for the target session, and has not obtained the ephemeral KEM secret key for the target session. A completed session is post-erasure fresh for forward-secrecy analysis if the server ephemeral KEM secret key and intermediate shared secret buffers have been erased before the server signing key is corrupted.

10 Security Definitions

Definition 3 (Explicit server authentication)**.** QSTP provides server identity and signed ephemeral-key authentication if, whenever an honest client accepts a signed connect response for server S, except with negligible probability that response was generated under S's certified signing key and binds the accepted ephemeral KEM public key to the certificate-pair transcript seed and serialized response header. The cleartext exchange response provides transcript acknowledgement; it does not by itself prove that the responder possesses the derived channel key. Key possession is confirmed when an authenticated server-to-client record is accepted under the derived receive state.

Definition 4 (Session-key indistinguishability)**.** QSTP provides session-key indistinguishability if no probabilistic polynomial-time adversary can distinguish the derived keys of a fresh accepted client session from uniformly random strings of the same length, except with negligible advantage.

Definition 5 (Channel confidentiality and integrity)**.** The established channel provides confidentiality if encrypted messages reveal no information beyond length and public header metadata. It provides integrity if no adversary can cause an honest receiver to accept a new packet not generated by its partner under the same directional channel state, except with negligible probability.

11 Assumptions

Assumption 1 (Certificate authenticity)**.** *The trusted root certificate available to the client is authentic. The root signing key is not compromised before the server certificate is issued, and the server certificate validation function correctly rejects invalid signatures, modified fields, expired certificates, algorithm mismatches, and version mismatches.*

Assumption 2 (KEM security)**.** *The selected KEM is IND-CCA secure at the selected parameter set. For ML-KEM this assumption is tied to the module-learning-with-errors family of assumptions. For Classic McEliece configuration sets it is tied to the code-based decoding assumptions underlying that scheme.*

Assumption 3 (Signature security)**.** *The selected server and root signature schemes are EUF-CMA secure at the selected parameter set. The implementation exposes no signing oracle that allows an adversary to obtain signatures on malicious QSTP transcript values under the server signing key outside authorized protocol execution.*

Assumption 4 (KDF security)**.** *The session KDF, cSHAKE-256 evaluated with the fixed session name string* `QSTP-1.4-SESSION`*, customization* sch_2*, and secret input* sec*, is a pseudorandom function of its secret input. (The literal name string is a fixed domain label and is independent of the runtime protocol-version macro, which is 1 in the analyzed build.) Random-oracle modeling of SHA3/cSHAKE is invoked only for the transcript-collision bounds, not for the session-key derivation step, which requires only PRF security because* sec *is secret and high-entropy. Domain separation is required: the session name string must not be reused for unrelated protocol outputs (transcript hashing, certificate hashing, RCS keying, or ratchet derivation); the implementation separates the session KDF (name* `QSTP-1.4-SESSION`*, rate 256) from the ratchet KDF (name = protocol-set string, rate 512).*

Assumption 5 (AEAD security)**.** *The RCS channel construction is a secure online (stateful) AEAD providing confidentiality and integrity per direction. Each directional instance is keyed once with a unique directional key and initialization nonce and advances its internal cipher and authentication state per record, so that distinct records in a direction*

are never processed under the same internal state. The implementation does not key two distinct plaintext streams with the same directional key and initialization nonce.

Assumption 6 (Ephemeral freshness)**.** *For each handshake the server generates a fresh ephemeral KEM key pair* $(\mathsf{pk}_{kem}, \mathsf{sk}_{kem})$ *that is not reused across connections, and the client encapsulation produces a fresh KEM ciphertext and shared secret. Consequently* sec *and the final transcript* sch_2*, and hence the derived directional keys and nonces, are fresh per session except with negligible probability. The server erases* sk_{kem} *and the intermediate shared secret immediately after channel-key derivation.*

12 Authentication Analysis

The server-authentication argument rests on three bindings. First, because the server certificate is pinned (held locally by the client and not transmitted in band), the verification key the client uses is fixed before the handshake and is not subject to in-band substitution. Second, the server certificate binds that verification key to the trusted root. Third, the signed connect response binds the ephemeral KEM public key to the current transcript and serialized packet header. A client accepts the KEM public key only after verifying the signature on phash and comparing the recovered value against a locally recomputed phash. A substituted public key changes phash and requires a new valid server signature under the pinned, root-chained verification key.

Lemma 1 (Connect-response authenticity)**.** *If an honest client accepts a connect response containing* pk_{kem}*, then either the response was generated by a party holding the certified server signing key, or an adversary has produced an existential forgery against the selected signature scheme.*

Proof. The client computes $\mathsf{phash} = H(\mathsf{hdr}_2 \parallel \mathsf{sch}_0 \parallel \mathsf{pk}_{kem})$ and verifies the signature under the certified server verification key. The value includes the serialized header, the certificate-pair transcript seed, and the ephemeral KEM public key. If no honest signer produced a signature on this exact value, an accepting client yields a valid signature on a previously unsigned message. This is an EUF-CMA forgery, except for the negligible probability of finding a distinct input with the same SHA3-256 hash under the certificate and transcript binding conditions. □

Theorem 1 (Server identity and ephemeral-key authentication)**.** *Assuming certificate authenticity, collision resistance of SHA3-256 for the transcript inputs, and EUF-CMA security of the selected signature scheme, QSTP authenticates the server identity and the server's signed ephemeral KEM public key for accepted client sessions. Handshake-time acceptance provides transcript acknowledgement; cryptographic confirmation that the peer possesses the derived server-to-client channel key is obtained only when an authenticated server-to-client channel record is accepted.*

Proof. Consider an honest client session that accepts. It must have verified the pinned server certificate under the trusted root, verified the connect-response signature, computed the same phash, encapsulated to the accepted pk_{kem}, and matched the exchange response against its local sch_2. By the previous lemma, the accepted pk_{kem} is authenticated by a valid server signature unless a signature forgery or hash collision occurred. The exchange response equals the client's sch_2, which commits the accepted public key and the KEM ciphertext. In the implementation this response is a cleartext echo of sch_2 (Appendix E); since sch_2 is a function of public transcript values, the match verifies only that the returned acknowledgement equals the client's local transcript value; it does not prove possession of the derived channel key or independent server liveness after the signed connect response. Therefore the authentication result established at handshake acceptance is server identity

plus signed ephemeral-key authenticity, together with transcript acknowledgement. If the client subsequently accepts an authenticated server-to-client record under the derived receive state, then the sender possessed the corresponding derived channel state except with the probability of an AEAD forgery. The remaining alternatives are root-certificate failure, signature forgery, hash collision, KEM break, AEAD forgery for the first accepted server record, or endpoint compromise, each outside the applicable theorem assumptions. □

13 Session-Key Security

Theorem 2 (Session-key indistinguishability)**.** *Let $\mathcal{A}$ be a probabilistic polynomial-time adversary against a fresh QSTP client session, running in an environment with at most n_s sessions. Under the EUF-CMA security of the signature scheme, the IND-CCA security of the selected KEM, and the pseudorandomness of the domain-separated cSHAKE-256 KDF, the AKE advantage is bounded by*

$$\mathsf{Adv}^{AKE}_{\mathsf{QSTP}}(\mathcal{A}) \le n_s\Big(\mathsf{Adv}^{EUF\text{-}CMA}_{\mathsf{SIG}}(\mathcal{B}_0) + \mathsf{Adv}^{IND\text{-}CCA}_{\mathsf{KEM}}(\mathcal{B}_1) + \mathsf{Adv}^{PRF}_{\mathsf{KDF}}(\mathcal{B}_2)\Big) + \epsilon_{coll},$$

where the factor n_s accounts for guessing the target session in the reduction and ϵ_{coll} bounds transcript-hash collisions, negligible for SHA3-256 under the modeled input sizes.

Proof. The proof uses a standard sequence of games over a guessed target session (the guess succeeds with probability at least $1/n_s$, giving the stated factor). In Game 1 the experiment aborts if the target client accepts a connect response whose phash was not signed by the honest server holding the pinned, root-chained signing key; the gap to Game 0 is bounded by $\mathsf{Adv}^{EUF\text{-}CMA}_{\mathsf{SIG}}(\mathcal{B}_0)$ plus the collision term ϵ_{coll}, and ensures that in all later games the accepted pk_{kem} is the honest server's ephemeral key. In Game 2 the KEM shared secret of the target session is replaced by a uniformly random value; any distinguisher yields an IND-CCA adversary $\mathcal{B}_1$ that embeds the challenge public key and ciphertext as the target ephemeral key and exchange ciphertext, and answers all other (non-target) server decapsulations using its IND-CCA decapsulation oracle, which is why IND-CCA rather than IND-CPA is required against an active adversary that may submit chosen ciphertexts to decapsulating servers.

In Game 3 the output of the KDF for the target session is replaced by uniformly random directional key and nonce strings. The KDF input is the now-random shared secret with public customization sch_2 under the fixed session name string; any distinguisher gives a PRF distinguisher $\mathcal{B}_2$ against the session-KDF domain. In Game 3 the test key is uniformly random and independent of the adversary's view, so the test bit is information-theoretically hidden except for the residual transcript-collision probability. Summing the game gaps and the guessing factor gives the stated bound. □

14 Channel Security

Theorem 3 (QSTP channel confidentiality and integrity)**.** *Assume that the session keys are indistinguishable from random and that RCS is an AEAD-secure authenticated encryption scheme with associated data. Then QSTP encrypted packets provide confidentiality for plaintext messages and integrity for packet headers and ciphertexts under the active RCS transport profile.*

Proof. The sender uses a directional key and initialization nonce derived for exactly one direction, applied as an online AEAD whose internal state advances per record (Assumption 6 gives per-session freshness of these keys, and the AEAD assumption gives non-repeating per-record state). The serialized header is supplied as associated data.

Therefore any modification to the flag, message length, sequence number, or timestamp changes the associated data and invalidates the tag. Under AEAD integrity, an adversary cannot produce a new packet accepted by the receiver except with negligible probability. Under AEAD confidentiality, ciphertexts reveal no plaintext information beyond length and public header metadata. Strict sequence validation prevents replay of previously accepted packets because the authenticated sequence in an accepted record must equal $\mathsf{rxseq} + 1$, and rxseq advances only after successful authentication, so a replayed record carries a stale authenticated sequence and is rejected. □

The theorem is deliberately restricted to the RCS profile. A GCM profile requires a proof that each record uses a unique IV under the same key and that the GCM authentication state is reinitialized per record. Since the active implementation disables the AES-GCM transport profile pending such a design, no GCM channel theorem is stated.

15 Replay, Freshness, and Ordering

QSTP combines strict sequence checking with a timestamp threshold. The sequence rule rejects replays and out-of-order packets after any successful packet. The timestamp check rejects packets outside the configured time window. These mechanisms address different attacks. The sequence value prevents reuse within a connection, while the timestamp limits acceptance of stale packets in cases where connection state is newly initialized or adversarial network delay is significant.

The timestamp threshold is not equivalent to a nonce challenge. It requires bounded clock disagreement and admits the usual denial-of-service behavior associated with clock manipulation. In the analyzed build the threshold is a symmetric ±60 second window (`QSTP_PACKET_TIME_THRESHOLD`). A system with unreliable clocks should introduce an additional authenticated challenge, monotonic local epoch, or application-level freshness control if the timestamp window is insufficient.

16 Forward Secrecy and Compromise Analysis

QSTP provides forward secrecy against later compromise of the long-term server *signing* key, provided the target session has completed and the ephemeral KEM secret key and shared-secret buffers have been erased. This is a meaningful but qualified property: the signing key only authenticates the handshake and never decrypts the KEM ciphertext, so confidentiality of completed sessions does not depend on it. Because the only long-term asymmetric secret in QSTP is the signing key—the KEM key pair is ephemeral and per handshake (Assumption 6)—there is no long-term KEM secret whose compromise could retroactively expose sessions. After erasure, recovering a completed session's keys requires recovering the KEM shared secret from the public ciphertext or breaking the KDF.

Theorem 4 (Post-erasure server-key forward secrecy)**.** *Let an honest QSTP session complete, and suppose the server has erased the ephemeral KEM secret key and intermediate shared secret. If the adversary later compromises the server long-term signing key but does not obtain endpoint memory containing the completed session keys, then prior session keys remain indistinguishable from random under the KEM and KDF assumptions.*

Proof. The compromised signing key allows the adversary to forge future server signatures and impersonate the server in future sessions. It does not reveal the completed session's ephemeral KEM secret key. The session key was derived from sec and sch_2. The value sch_2 is public or reconstructible from the transcript; the only confidential input is sec. Without the erased KEM secret key, computing sec from ct_{kem} breaks the selected KEM.

Replacing sec by random and then replacing the KDF output by random gives the same game sequence as in the session-key theorem. □

The theorem does not cover compromise of the root signing key before certificate issuance, compromise of the server signing key before the target handshake, compromise of endpoint memory while session keys remain live, or compromise of the ephemeral KEM secret key before erasure.

17 Ratchet Security

Theorem 5 (Forward secrecy across ratchet epochs)**.** *Assume the ratchet state update is one-way, ratchet tokens are fresh and authenticated, and old ratchet states and channel keys are erased. Compromise of ratchet state at epoch t does not reveal channel keys for epochs $t' < t$, except with the advantage of inverting the ratchet update or breaking the KDF.*

Proof. The keys for epoch t' are functions of the ratchet state and token history before epoch t. If each update maps the prior state and token through a one-way KDF and old states are erased, then later state does not provide an efficient method for reconstructing prior states. An adversary distinguishing earlier epoch keys from random from later state can be converted into an adversary that either inverts one of the ratchet update steps or distinguishes the KDF output from random. □

The ratchet does not provide full post-compromise recovery if the adversary observes all future ratchet tokens and remains resident in endpoint memory. Recovery requires an update containing entropy unknown to the adversary at the time of compromise and correct erasure of compromised state.

18 Malformed Message and Memory-Safety Boundaries

The protocol security model assumes that malformed packets are rejected before they affect secret state. The implementation enforces fixed handshake message lengths, fixed header size, a maximum message length of $0x10000$, encrypted-message lower bounds greater than the RCS tag size, exact ratchet-token encrypted length, and generic handling of pre-key error packets. Runtime validation is security-relevant because assertions alone do not protect release builds.

A parser that accepts a length smaller than the authentication tag size before subtracting the tag length can underflow. A ratchet parser that decrypts an arbitrary plaintext length into a fixed 32-byte token buffer can overflow. The code-aligned requirements therefore are: encrypted payloads must have $\mathsf{msglen} > \mathsf{taglen}$, ratchet requests must have $\mathsf{msglen} = 64$ under RCS, and allocation resizing must preserve the original pointer on failure.

19 Denial-of-Service Considerations

QSTP cannot prevent an on-path adversary from dropping traffic or interrupting a pre-key handshake. The meaningful goal is to prevent unauthenticated data from causing memory corruption, secret disclosure, or state confusion. The implementation treats pre-key error-condition payloads as generic connection failures rather than authenticated protocol commands. The server still performs post-quantum operations during handshakes and therefore remains exposed to handshake flooding unless the deployment adds rate limits, address throttling, listener backlog controls, or admission policy.

The 64 KB packet message bound limits per-packet allocation pressure compared with large-message designs. However, allocation failure must be handled without leaks and without continuing with partially initialized state. Implementations should run sanitizer and dynamic-analysis tests for short encrypted messages, oversized ratchet packets, allocation failure, invalid sequence numbers, invalid timestamps, and repeated handshake failure.

20 Security Limitations

QSTP's claims are scoped by the following limitations.

1. The protocol authenticates the server to the client. It does not authenticate the client to the server.
2. Root certificate authenticity is an external provisioning assumption. If a client is provisioned with a malicious root certificate, QSTP cannot distinguish that condition.
3. Certificate revocation is not part of the analyzed base protocol. Expiration is checked, but revocation requires deployment policy or an additional protocol mechanism.
4. The active channel proof applies to the RCS profile. AES-GCM is reserved unless a per-record standards-conformant nonce and state-reset design is implemented and tested.
5. The timestamp freshness rule assumes bounded clock drift. It does not replace strict sequence validation and does not protect against endpoint clock compromise.
6. Endpoint compromise defeats confidentiality for live session state. Post-erasure forward secrecy applies only after ephemeral secrets have been erased and only against later compromise of the long-term signing key, not against memory disclosure of live channel keys.
7. The default build uses NIST category 5 (≈256-bit) parameters (ML-KEM-1024, ML-DSA-87). Lower compile-time parameter sets (category 1 or 3, or SLH-DSA/McEliece variants) MAY be selected and MUST NOT be described as providing category-5/256-bit security. A profile must name the configuration it claims.
8. The exchange response carries sch_2 in cleartext, so client acceptance provides transcript acknowledgement, not cryptographic key confirmation or independent proof that the server processed the KEM ciphertext. Key possession is confirmed on the first authenticated server-to-client channel record. The phash and certificate-serial comparisons use a constant-time primitive, whereas the sch_2 and configuration-string comparisons use ordinary byte equality; this is acceptable only because those values are public, and the constant-time property should not be claimed for them.
9. The symmetric ratchet provides forward secrecy across epochs but not post-compromise security: because ratchet tokens traverse the channel, a resident adversary holding current channel keys learns subsequent tokens. The ratchet is also not exercised by the reference applications.
10. Downgrade resistance is structural: QSTP performs no cipher-suite negotiation, and the cryptographic algorithm and version are bound into sch_0 through the certificate hash. The protocol-configuration string in the connect request is an equality-checked routing field and is not itself transcript-bound; it must not be treated as a negotiated security parameter.
11. The reference server is a single-session console application. Concurrency, connection admission, and multi-tenant isolation are deployment concerns and are outside the analyzed construction.

21 Conformance-Oriented Test Requirements

A QSTP implementation should include deterministic and negative tests for the following properties.

1. Certificate hash determinism for root and server certificates.
2. Server certificate verification for valid, expired, wrong-root, tampered, wrong-algorithm, and wrong-version certificates.
3. Transcript determinism for sch_0, phash, sch_1, and sch_2.
4. KDF determinism using name string `QSTP-1.4-SESSION` and customization value sch_2.
5. Directional key assignment equality between client transmit and server receive, and server transmit and client receive.
6. Rejection of encrypted messages with $\mathsf{msglen} \leq \mathsf{taglen}$.
7. Rejection of ratchet packets whose encrypted payload length is not exactly 64 bytes under RCS.
8. Rejection of replayed, skipped, duplicated, and stale packets without receive-sequence advancement.
9. Authenticated error-packet processing after establishment and generic pre-key error handling before establishment.
10. Sanitizer and dynamic-analysis tests for allocation failure, packet truncation, malformed headers, and connection teardown.

22 Comparison with TLS 1.3 and WireGuard

QSTP is narrower than TLS 1.3. TLS 1.3 is a general Internet security protocol with broad negotiation, public extension points, several authentication modes, and mechanisms for resumption and early data. QSTP avoids those features to reduce its state surface and to make a fixed post-quantum posture easier to analyze. QSTP also differs from WireGuard, which is based on a Noise handshake pattern and provides a VPN-oriented peer model. QSTP's one-way root-anchored certificate model is closer to a constrained server-authenticated channel than to a peer VPN construction.

The tradeoff is explicit. QSTP gains a compact state machine and straightforward transcript binding. It loses general-purpose interoperability, negotiated agility, client authentication, and standardized deployment semantics. It is most appropriate where those properties are deliberate deployment constraints rather than missing requirements.

23 Detailed State Machine

The implementation realizes a deterministic, linear state machine. The model below is normative for the analysis because each accepting transition consumes a single expected flag, a fixed message length, an expected sequence number, and a timestamp within the configured tolerance. A transition outside the table is invalid and leads to connection failure or teardown.

State	Expected input	Output or action	Security effect

Initial client state	Server certificate object	Validate certificate under root certificate	Establishes authenticated server verification key before any network handshake message is trusted.
Client sent connect request	`connect_response`	Verify signature on phash, advance sch_1, encapsulate KEM secret	Binds ephemeral KEM public key to certificate-pair transcript and serialized header.
Server received connect request	`connect_request`	Verify serial, configuration, certificate validity; generate ephemeral KEM key pair	Prevents cross-configuration and wrong-certificate handshakes.
Server sent connect response	`exchange_request`	Decapsulate KEM ciphertext, erase ephemeral secret key, derive keys, send sch_2	Commits client ciphertext into final transcript and creates the channel state.
Client sent exchange request	`exchange_response`	Compare returned sch_2 to local sch_2 by byte equality	Provides transcript acknowledgement before client acceptance.
Established	`encrypted_message`	Validate sequence and time, authenticate header as AAD, decrypt on tag success	Protects channel confidentiality and integrity.
Established	`symmetric_ratchet_request`	Require exact encrypted token length, authenticate, derive new keys	Updates channel keys only after an authenticated ratchet message.
Any pre-key state	`error_condition`	Treat as generic connection failure	Prevents unauthenticated payload bytes from selecting protocol error semantics.

This table makes clear that QSTP does not use a transcript-free early data state. The client initializes provisional cipher state before final acceptance, but the established flag is not set until the transcript-acknowledgement check succeeds. Therefore application data is not processed by a valid client session before the final comparison of sch_2.

24 Message Encoding Requirements

24.1 Handshake Encodings

The handshake message lengths are fixed by the selected parameter set. The connect request contains the server certificate serial number and the protocol configuration string. The connect response contains the server ephemeral KEM public key, a transcript-bound 32-byte hash, and a post-quantum signature over that hash. The exchange request contains the KEM ciphertext. The exchange response contains the 32-byte final transcript hash.

Message	Critical fields	Validation rule
Connect request	$\mathsf{serial}, \mathsf{cfg}$	Server verifies serial equality, protocol configuration equality, certificate validity interval, and expected sequence and time.
Connect response	$\mathsf{pk}_{kem}, \mathsf{phash}, \sigma$	Client recomputes $\mathsf{phash} = H(\mathsf{hdr}_2 \parallel \mathsf{sch}_0 \parallel \mathsf{pk}_{kem})$, verifies σ, and compares hash output in constant time.
Exchange request	ct_{kem}	Server decapsulates using the ephemeral secret key associated with the signed public key and commits ct_{kem} into sch_2.

Exchange response	sch_2	Client accepts only if the returned hash equals its locally computed final transcript hash.
Encrypted message	header, ciphertext, tag	Receiver validates sequence and time before AEAD verification; plaintext is released only after tag success.
Ratchet request	encrypted token, tag	Receiver requires token length 32 and encrypted message length $32 + \mathsf{taglen}$.

24.2 Canonical Serialization

The formal model requires canonical serialization because the transcript commits to exact byte strings. The header byte string used in phash and in associated data is not an abstract structure; it is the exact serialized form. Integer fields are little-endian. A conforming implementation must not hash native C structures, padding bytes, host-endian values, or uninitialized memory. Two interoperable implementations must produce the same serialized header bytes for the same abstract packet.

25 Configuration Sets and Concrete Security Level

The QSTP header defines multiple compile-time configuration sets. Each combines a signature scheme, a KEM, the RCS profile, and a SHAKE/cSHAKE rate. The security level of the resulting protocol is the minimum of the selected signature security, selected KEM security, KDF output strength, and channel AEAD strength, subject to implementation security. The default configuration is the category 5 lattice set (ML-DSA-87 with ML-KEM-1024); the entries below are ordered from this default downward and across alternative families.

Configuration family	Signature	KEM	Category
ML-DSA / ML-KEM (default)	ML-DSA-87 (Dilithium 5)	ML-KEM-1024	5 (default; ≈256-bit).
ML-DSA / ML-KEM, mid	ML-DSA-65 (Dilithium 3)	ML-KEM-768	3.
ML-DSA / ML-KEM, low	ML-DSA-44 (Dilithium 2)	ML-KEM-512	1; not a category-5 claim.
ML-DSA / McEliece	ML-DSA family	Classic McEliece family	Code-based KEM alternative with larger public keys.
SLH-DSA / McEliece	SLH-DSA (SPHINCS+)	Classic McEliece family	Hash-based signature with code-based KEM.

A document or implementation profile shall identify the selected configuration set. A generic QSTP claim should not state that all configurations provide the same concrete post-quantum security level. Parameter-set names and implementation macros determine the actual concrete posture.

26 Expanded Game Sequence for AKE Security

This section expands the proof of session-key indistinguishability to make explicit where each primitive assumption is used.

Game 0: real execution. The adversary interacts with all honest clients and servers. The challenge client validates the root and server certificates, receives a signed connect response, encapsulates to the accepted KEM public key, sends the KEM ciphertext, receives sch_2, compares it locally, and accepts. The test oracle returns either the real channel keys or random keys of the same length.

Game 1: certificate and signature failure abort. The game aborts if the challenge client accepts a connect response for which no honest server generated a valid signature over phash. The difference between Game 0 and Game 1 is bounded by the advantage of a signature forger, plus the probability of a collision in the hash input used to define phash. The reduction embeds the certified verification key as the signature challenge key and uses any accepted unsigned phash as the forgery.

Game 2: replace KEM shared secret. The game replaces the KEM shared secret in the challenge session with a random value. The reduction to IND-CCA embeds the KEM public key and challenge ciphertext as the ephemeral public key and exchange request ciphertext. The adversary's ability to distinguish whether the resulting channel keys come from the real or random secret is bounded by the KEM challenge advantage.

Game 3: replace KDF output. The game replaces the cSHAKE output block derived from the random secret and sch_2 with a random string. Because the name string is fixed to `QSTP-1.4-SESSION`, the session derivation domain is separated from transcript hashing, certificate hashing, RCS authentication, and ratchet derivation. If an adversary distinguishes Game 2 from Game 3, it distinguishes the KDF output from random under a secret input.

Game 4: ideal channel keys. The challenge keys are now uniformly random and independent of the adversary's view. The test bit is information-theoretically hidden. The only remaining failure events are collisions in transcript hashing or violations of the freshness restrictions. Therefore the adversary's advantage is bounded by the sum of the previous transition bounds.

27 Authentication Failure Cases

The authentication theorem requires rejecting several concrete failure cases. This subsection makes the failure behavior explicit.

1. If the client cannot verify the server certificate under the root certificate, it rejects before the network key exchange is trusted.
2. If the server certificate has expired or the algorithm/version fields do not match the configured QSTP instance, it rejects before key derivation.
3. If the connect response signature does not verify under the server certificate verification key, the client rejects.
4. If the recovered or verified phash differs from the locally recomputed phash, the client rejects.

5. If the returned exchange-response transcript hash differs from the locally computed sch_2, the client rejects and erases provisional state.

6. If an encrypted application packet fails associated-data authentication, plaintext is not released and the receive sequence counter does not advance.

These cases exclude unknown-key-share and simple transcript-substitution attacks under the stated assumptions. A substituted certificate changes sch_0, a substituted ephemeral public key changes phash, and a substituted KEM ciphertext changes sch_2.

28 Metadata and Traffic Analysis

QSTP encrypts packet payloads but does not hide all metadata. The header flag, message length, sequence number, and timestamp are visible unless the deployment encapsulates QSTP inside an outer protection layer. The header fields are integrity protected as associated data, not encrypted. An observer may therefore learn packet timing, packet sizes, directionality, connection duration, and approximate application message cadence. QSTP should not be described as an anonymity protocol.

Traffic analysis resistance must be provided by padding, batching, cover traffic, outer tunnels, or application-layer traffic shaping. These mechanisms are outside the base protocol analyzed here. The base security claims are confidentiality and integrity of payload contents, authenticated header integrity, strict replay rejection, and server authentication.

29 Root Authority and Revocation Analysis

The root authority is a central authentication dependency. If the root signing key is compromised before server certificate issuance, an adversary can create certificates accepted by clients provisioned with that root. If a valid server certificate must be withdrawn before expiration, the base protocol requires an external revocation mechanism because it validates certificate signatures and expiration intervals but does not define an online revocation query.

The following deployment controls are compatible with the protocol model: short certificate validity periods, root epochs, offline root signing, out-of-band certificate pinning, signed revocation lists, and application policy that rejects serial numbers known to be withdrawn. These controls are outside the base AKE proof but materially affect operational security.

30 Implementation-Security Guidance

A conforming implementation must preserve the cryptographic invariants used in the proof. The following requirements are derived directly from the proof dependencies.

1. The KEM secret key generated by the server for a handshake must be erased immediately after decapsulation.

2. The shared secret and KDF output buffer must be erased after channel key initialization.

3. Packet parsing must validate message lengths before subtracting tag lengths or allocating plaintext buffers.

4. Ratchet-token packets must be length checked before decryption into a fixed-size token buffer.

5. Associated data must be the serialized 21-byte header, not a native structure representation.
6. The receive sequence number must advance only after authenticated decryption succeeds.
7. Pre-key error packets must not be interpreted as authenticated protocol commands.
8. The AES-GCM transport profile must remain disabled unless it is replaced by a per-record construction with unique IVs and independent validation.

31 Proof-Carrying Traceability Matrix

Protocol mechanism	Formal property supported	Required implementation invariant
Root-signed server certificate	Server identity binding	Certificate hash must cover fixed fields and signature verification must precede handshake acceptance.
Signed phash	Ephemeral KEM public-key authenticity	phash must include serialized header, sch_0, and pk_{kem}.
KEM ciphertext commit	Transcript binding and KDF customization	ct_{kem} must be included in sch_2 before KDF invocation.
Domain-separated cSHAKE	Session-key indistinguishability	Name string must be fixed and unique to session derivation; customization must be sch_2.
Directional key assignment	Channel separation	Client transmit must equal server receive; server transmit must equal client receive; directions must not share nonce state.
Header as AAD	Header integrity	Serialized header must be supplied to AEAD before encryption and verification.
Strict sequence check	Replay rejection	Receiver must accept only $rxseq + 1$ and must not advance on failed authentication.
Exact ratchet length	Memory safety and state correctness	Ratchet payload must decrypt to exactly one 32-byte token.
Secret erasure	Forward secrecy after long-term key compromise	Ephemeral KEM secret and shared secret buffers must be cleared after use.

32 Recommended Verification Artifacts

A formal analysis gains practical value only when it is paired with reproducible verification artifacts. The following artifacts are recommended for QSTP.

1. A known-answer vector for certificate hashing that includes the root certificate hash, server certificate hash, and root/server certificate-pair hash.
2. A handshake transcript vector that records the serialized connect-response header, phash, sch_1, KEM ciphertext, and sch_2.
3. A KDF vector with $N = \texttt{QSTP-1.4-SESSION}$, $S = \mathsf{sch}_2$, input sec, and the derived key/nonce slices.
4. A directional-channel vector proving that client transmit output is accepted by server receive state and server transmit output is accepted by client receive state.
5. Negative vectors for wrong root, expired certificate, altered signature, altered public KEM key, altered ciphertext, altered header, altered tag, replayed sequence, out-of-order sequence, short encrypted message, and oversized ratchet request.
6. Sanitizer and leak-check runs for failure paths, including allocation failure and early teardown.

33 Evidence and Verification Scope

This section states the implementation-alignment scope for the formal model. The following protocol facts define the analyzed reference behavior: the 21-byte header and its little-endian $\mathsf{flag} \parallel \mathsf{msglen} \parallel \mathsf{sequence} \parallel \mathsf{utctime}$ serialization; the maximum message size 0x10000 and 32-byte tag; the certificate-pair seed $\mathsf{sch}_0 = H(H(\mathsf{RootCert}) \parallel H(\mathsf{ServerCert}))$; the transcript chain $\mathsf{phash} = H(\mathsf{hdr}_2 \parallel \mathsf{sch}_0 \parallel \mathsf{pk}_{kem})$, $\mathsf{sch}_1 = H(\mathsf{sch}_0 \parallel \mathsf{phash})$, $\mathsf{sch}_2 = H(\mathsf{sch}_1 \parallel \mathsf{ct}_{kem})$; the session KDF $\mathsf{cSHAKE256}(N = \texttt{QSTP-1.4-SESSION}, S = \mathsf{sch}_2, X = \mathsf{sec})$; the directional key/nonce partition with client-transmit equal to server-receive; per-handshake server KEM key generation with secret-key and shared-secret erasure after decapsulation; the encrypt/decrypt path with the serialized header as associated data, strict $\mathsf{rxseq} + 1$ sequencing, a ± 60 second timestamp window, the $\mathsf{msglen} > \mathsf{taglen}$ check, and receive-sequence advancement only after authentication; the configuration-string equality check; the self-signed and external-root verification modes; the cSHAKE-512 ratchet schedule on a 32-byte token in a 64-byte authenticated record; and connection teardown with state erasure.

The exchange response transmits sch_2 in cleartext, so client acceptance reflects transcript acknowledgement rather than key confirmation (Appendix E). The sch_2 and configuration-string comparisons use ordinary byte equality rather than the constant-time primitive used for phash and the serial; this is acceptable because those operands are public transcript or routing values. Independent conformance artifacts should include the known-answer vectors of Section 32: deterministic certificate-hash, transcript, KDF, and directional-channel vectors, and the negative-test corpus.

QSTP is a server-authenticated, root-anchored, post-quantum tunneling protocol with a linear handshake and an authenticated encrypted transport. Under authentic root provisioning, secure server certificates, EUF-CMA-secure signatures, IND-CCA-secure KEM encapsulation, domain-separated cSHAKE-256 key derivation, and AEAD security of the RCS channel, the protocol provides server identity authentication, signed ephemeral-key authenticity, transcript acknowledgement, session-key indistinguishability, channel confidentiality, channel integrity, and strict replay rejection for accepted packets. Key possession by the server is confirmed when the client accepts an authenticated server-to-client channel record.

The construction does not provide mutual authentication, does not solve certificate revocation by itself, and does not protect live endpoint memory from compromise. Its forward-secrecy claim is limited to post-erasure compromise of long-term server signing keys.

The active channel-security theorem applies to the RCS transport profile. A standards-conformant AES-GCM profile would require a separate per-record nonce and state-reset specification with independent test vectors before it could be analyzed as an active QSTP transport profile.

A Implementation Function Mapping

This appendix maps the formal components used in the proof to implementation modules. The mapping is included so that the security claims can be traced to concrete protocol operations.

Formal object	Implementation area	Security relevance
Root certificate hash	`qstp_root_certificate_hash`	Defines the deterministic commitment to root certificate fields.
Server certificate hash	`qstp_server_certificate_hash`	Defines the deterministic commitment to server certificate fields.
Root/server pairing hash	`qstp_server_root_certificate_hash`	Implements sch_0, the initial transcript value.
Server certificate verification	`qstp_server_root_certificate_verify`	Establishes the authenticated server verification key.
Connect request	`kex_client_connect_request`	Sends certificate serial number and protocol set string.
Connect response	`kex_server_connect_response`	Generates ephemeral KEM key pair, computes phash, signs phash, and advances sch_1.
Exchange request	`kex_client_exchange_request`	Verifies signature, encapsulates KEM secret, commits ciphertext, and derives client channel keys.
Exchange response	`kex_server_exchange_response`	Decapsulates KEM ciphertext, erases KEM secret key, derives server channel keys, and returns sch_2.
Establish verify	`kex_client_establish_verify`	Compares returned sch_2 by byte equality and sets established state.
Packet encryption	`qstp_encrypt_packet`	Serializes header, uses header as AAD, encrypts payload, and advances transmit state.
Packet decryption	`qstp_decrypt_packet`	Checks sequence and time, verifies AEAD tag, releases plaintext only on success, and advances receive state.
Header serialization	`qstp_packet_header_serialize`	Provides canonical byte encoding for associated data and transcript-bound header fields.
Header validation	`qstp_header_validate`	Rejects wrong flag, wrong size, wrong sequence, invalid time, and unauthenticated pre-key error semantics.
Connection disposal	`qstp_connection_state_dispose`	Clears cipher state, ratchet state, socket data, and counters.

B Oracle Model

For completeness, this appendix gives the oracle syntax used by the security games.

$\mathsf{Send}(P, i, m)$**.** Delivers message m to session i of party P. The oracle returns the next protocol message or $\perp$ if the message is rejected.

$\mathsf{Reveal}(P, i)$**.** Returns the established channel keys of session i, if the session has accepted. A revealed session is not fresh.

$\mathsf{CorruptServer}(S)$**.** Returns the long-term signing key of server S. If issued before the target connect response, the target session is not fresh for server-authenticated AKE.

$\mathsf{CorruptRoot}()$**.** Returns the root signing key. This query is not permitted in the base authentication theorem because it defeats the root-authenticity assumption.

$\mathsf{CorruptEphemeral}(S, j)$**.** Returns the ephemeral KEM secret key for server session j, if it still exists. If issued for the target session before erasure, the target session is not fresh.

$\mathsf{Test}(P, i)$**.** For a fresh accepted session, returns either the real channel keys or random keys of the same length depending on a hidden bit.

Partnering is defined by equality of sch_0, phash, sch_1, ct_{kem}, sch_2, and complementary directional key assignment. This definition avoids relying on network address metadata, which an adversary may spoof or relay.

C Failure Mode Matrix

Failure condition	Required response	Security rationale
Unknown certificate serial	Abort key exchange and report unrecognized key	Prevents use of unintended server credentials.
Configuration mismatch	Abort before KEM key generation or before accepting peer state	Prevents downgrade or cross-profile mixing.
Expired certificate	Abort before accepting long-term verification key	Prevents stale credential use.
Invalid certificate signature	Abort certificate validation	Maintains root-authenticated server identity.
Invalid connect-response signature	Abort before encapsulation-derived acceptance	Prevents unauthenticated ephemeral KEM keys.
phash mismatch	Abort before KEM ciphertext is accepted as authenticated transcript continuation	Prevents header or public-key substitution.
KEM decapsulation failure	Abort and erase ephemeral state	Prevents use of undefined or adversary-controlled shared secret.
sch_2 mismatch	Abort client session and erase provisional channel state	Prevents transcript-acknowledgement failure.
Packet sequence mismatch	Reject packet and do not advance receive sequence	Prevents replay and out-of-order acceptance.
Timestamp invalid	Reject packet before decryption	Limits stale packet acceptance.
AEAD tag failure	Reject packet and release no plaintext	Enforces ciphertext integrity.
Short encrypted message	Reject before subtracting tag length	Prevents integer underflow and invalid allocation.
Wrong ratchet length	Reject before fixed-buffer token decryption	Prevents stack or buffer overwrite.
Allocation failure	Tear down or return explicit memory error without losing ownership of existing buffer	Prevents leak-amplified denial of service.
Pre-key error-condition packet	Treat as generic connection failure	Avoids unauthenticated protocol semantics.

D Non-Goals

The following properties are not claimed by QSTP and are not implied by the proofs.

1. **Client authentication.** The base protocol does not prove the identity of the client to the server. Application-layer authentication may be layered on the encrypted channel.
2. **Anonymity.** Packet headers, timing, lengths, and endpoint metadata remain observable to the network.
3. **Endpoint compromise resistance.** If an adversary reads live process memory containing channel keys, confidentiality of live and future packets is outside the base theorem until rekeying with uncompromised entropy occurs.
4. **Root compromise tolerance.** A compromised root signing key can authorize malicious server certificates.
5. **Revocation enforcement.** Certificate expiration is checked, but online revocation is not built into the base exchange.
6. **Transport reordering tolerance.** The strict sequence rule rejects out-of-order packets. This is a security simplification and a transport constraint.
7. **GCM profile security.** No security theorem is stated for the disabled AES-GCM transport profile.

E Transcript Acknowledgement and the Scope of Key Confirmation

The server returns sch_2 to the client in the exchange response, and the client enters the established state only if the returned value equals its locally computed sch_2. In the reference implementation the server writes sch_2 into the response message in cleartext, and the client compares it with a byte-equality check; the response is not an encrypted "Finished" message and is not authenticated under the derived channel key. Because $\mathsf{sch}_2 = H(H(\mathsf{sch}_0 \parallel \mathsf{phash}) \parallel \mathsf{ct}_{kem})$ is a function of values that are all public to a network observer (the certificate-pair seed, the signed connect-response header, the ephemeral public key, and the KEM ciphertext), a matching response body proves only that the received acknowledgement equals the client's local transcript value. It does *not* demonstrate that the responder holds the derived channel key, and it does not independently prove that the server processed the KEM ciphertext after the signed connect response. Possession of the channel key is confirmed implicitly and unilaterally when the first application (or ratchet) record authenticates under the derived directional state. The security theorems of this paper rely only on this weaker transcript-acknowledgement property together with the signature, KEM, and channel-integrity bindings; a deployment that requires cryptographic key confirmation at handshake time should transmit the response (or an additional "Finished" value) as an AEAD record under the derived key.

Lemma 2 (Transcript acknowledgement)**.** *If an honest client accepts the exchange response, then the value returned by the server equals the client's local* $\mathsf{sch}_2 = H(H(\mathsf{sch}_0 \parallel \mathsf{phash}) \parallel \mathsf{ct}_{kem})$*, except with negligible probability of hash collision or implementation fault.*

Proof. The client computes sch_2 locally after verifying phash and encapsulating the KEM ciphertext. The exchange response body has fixed length equal to the certificate hash size. The client compares that body to its own sch_2 using byte equality. Acceptance occurs only on equality. Since SHA3-256 is deterministic and the serialized input bytes are fixed, equality proves that the response body matches the client's transcript hash with respect

to the values included in sch_2. A distinct transcript producing the same sch_2 would be a hash collision for the transcript input relation. □

F Extended AEAD Associated-Data Argument

Let a packet header be $\mathsf{hdr} = (f, \ell, q, t)$, where f is the flag, ℓ is the message length, q is the sequence number, and t is the timestamp. QSTP serializes hdr to a 21-byte string and supplies that string as associated data to RCS. For any ciphertext c and tag τ, the receiver accepts only if

$$\mathsf{RCS.Open}(k, n, \mathsf{aad} = \mathsf{hdr}, c, \tau) \neq \bot.$$

If an adversary changes f, ℓ, q, or t without recomputing a valid tag, the open operation returns $\bot$ under AEAD integrity. If the adversary replays a previously valid tuple (hdr, c, τ), the sequence rule rejects it after the original acceptance because the receiver expects $q + 1$. If the adversary delays a first delivery until outside the freshness threshold, the timestamp rule rejects it. Therefore replay resistance rests on a conjunction of AEAD integrity, monotonic sequence state, and bounded timestamp acceptance.

G Interoperability Constraints

Interoperability between independent QSTP implementations requires agreement on the following byte-level facts.

1. All packet headers are 21 bytes.
2. The message-length, sequence, and UTC-time fields are little-endian.
3. Certificate hashes process fields in the order and length defined by the technical specification.
4. The session KDF name string is the exact byte string `QSTP-1.4-SESSION`.
5. The KDF customization value is the 32-byte sch_2 transcript hash.
6. The KDF output slicing order is client-to-server key, client-to-server nonce, server-to-client key, server-to-client nonce.
7. The RCS tag length under the active profile is 32 bytes.
8. Ratchet token plaintext length is 32 bytes.

Any divergence in these values can produce silent channel incompatibility or, worse, transcript values that differ across implementations while appearing syntactically valid.

H Publication and Review Checklist

A reviewer should be able to reproduce the security analysis by checking the following facts against the implementation and test vectors.

1. Server certificate verification is performed before network key exchange acceptance.
2. Connect-response signatures cover the exact serialized response header, initial transcript value, and ephemeral KEM public key.
3. The final transcript commits the KEM ciphertext before key derivation.
4. The KDF has explicit domain separation and transcript customization.

5. Both directions use independent key and nonce state.
6. Application data is rejected until the established state is reached.
7. Decryption authenticates the serialized header as associated data.
8. Receive sequence advances only after authenticated decryption.
9. Malformed short messages and malformed ratchet packets are rejected before memory-unsafe operations.
10. The AES-GCM profile is not active unless accompanied by a complete standards-conformant construction and test corpus.

References

[1] National Institute of Standards and Technology. *FIPS 202: SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions*. U.S. Department of Commerce, 2015. https://doi.org/10.6028/NIST.FIPS.202

[2] J. Kelsey, S. Chang, and R. Perlner. *NIST SP 800-185: SHA-3 Derived Functions: cSHAKE, KMAC, TupleHash, and ParallelHash*. National Institute of Standards and Technology, 2016. https://doi.org/10.6028/NIST.SP.800-185

[3] National Institute of Standards and Technology. *FIPS 203: Module-Lattice-Based Key-Encapsulation Mechanism Standard*. U.S. Department of Commerce, 2024. https://doi.org/10.6028/NIST.FIPS.203

[4] National Institute of Standards and Technology. *FIPS 204: Module-Lattice-Based Digital Signature Standard*. U.S. Department of Commerce, 2024. https://doi.org/10.6028/NIST.FIPS.204

[5] National Institute of Standards and Technology. *FIPS 205: Stateless Hash-Based Digital Signature Standard*. U.S. Department of Commerce, 2024. https://doi.org/10.6028/NIST.FIPS.205

[6] D. J. Bernstein, T. Chou, T. Lange, R. Niederhagen, P. Schwabe, and others. *Classic McEliece: Conservative Code-Based Cryptography*. NIST Post-Quantum Cryptography Project submission material. https://classic.mceliece.org/nist.html

[7] M. Dworkin. *NIST SP 800-38D: Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC*. National Institute of Standards and Technology, 2007. https://doi.org/10.6028/NIST.SP.800-38D

[8] E. Rescorla. *RFC 8446: The Transport Layer Security (TLS) Protocol Version 1.3*. Internet Engineering Task Force, 2018. https://www.rfc-editor.org/rfc/rfc8446

[9] T. Perrin. *The Noise Protocol Framework*. Revision 34, 2018. https://noiseprotocol.org/noise.html

[10] J. A. Donenfeld. *WireGuard: Next Generation Kernel Network Tunnel*. Network and Distributed System Security Symposium, 2017. https://www.wireguard.com/papers/wireguard.pdf

[11] B. Lipp, B. Blanchet, and K. Bhargavan. *A Mechanised Cryptographic Proof of the WireGuard Virtual Private Network Protocol*. IEEE European Symposium on Security and Privacy, 2019. https://inria.hal.science/hal-02100345

[12] M. Bellare and P. Rogaway. *Entity Authentication and Key Distribution*. Advances in Cryptology - CRYPTO 1993. https://cseweb.ucsd.edu/~mihir/papers/eakd.pdf

[13] R. Canetti and H. Krawczyk. *Analysis of Key-Exchange Protocols and Their Use for Building Secure Channels*. Advances in Cryptology - EUROCRYPT 2001. https://iacr.org/archive/eurocrypt2001/20450189.pdf

[14] P. Rogaway. *Authenticated-Encryption with Associated-Data*. ACM Conference on Computer and Communications Security, 2002. https://web.cs.ucdavis.edu/~rogaway/papers/ad.pdf

[15] QRCS Corporation. *Quantum Secure Tunneling Protocol Technical Specification*. Quantum Resistant Cryptographic Solutions Corporation, 2026. https://qrcscorp.ca/technology/qstp.html

[16] QRCS Corporation. *QSTP Source Code Repository*. https://github.com/QRCS-CORP/QSTP