Security analysis of TLS ML-DSA authentication with ML-KEM

This report in another language: 中文.

draft-ietf-tls-mldsa-06 · ProVerif+ Agent report

Documents	Use of ML-DSA in TLS 1.3 (protocol definition); ML-KEM Post-Quantum Key Agreement for TLS 1.3 (supporting); The Transport Layer Security (TLS) Protocol Version 1.3 (supporting)
Properties examined	17: 17 hold, 0 fail
Findings	None

Contents

1 Protocol and scope
2 Findings
3 Properties examined
4 Limits of this analysis
Appendix A: Reading of each property
Appendix C: Claims and prover answers
Appendix D: How the analysis was run

1 Protocol and scope

This is the TLS 1.3 handshake with ML-KEM-768 key agreement and ML-DSA-65 certificates, used in one full round trip with authentication of both endpoints. The client holds its ML-DSA-65 signing key and its certificate; the server holds its own; each validates the peer certificate against a configured trust root and stops when the chain, the peer's signature or the peer's Finished does not check out.

The client offers a fresh ML-KEM-768 encapsulation key in its ClientHello. The server answers with a ciphertext, both ends derive the shared secret that is inserted into the TLS 1.3 key schedule, and everything after ServerHello is encrypted under the handshake traffic secrets. The server then sends its certificate, a CertificateVerify over the transcript and a Finished; the client checks all three, replies with its own certificate, CertificateVerify and Finished, and each side is sure of the other's identity, of the transcript and of the keys once the peer's Finished verifies.

The parties:

- **Client**: TLS client, validates the server chain against the configured root. Holds: sk_c (ML-DSA-65), client certificate chain, CA public key.
- **Server**: TLS server, validates the client chain when it asks for it. Holds: sk_s (ML-DSA-65), server certificate chain, CA public key.

Figure 1 shows the exchange as the documents specify it.

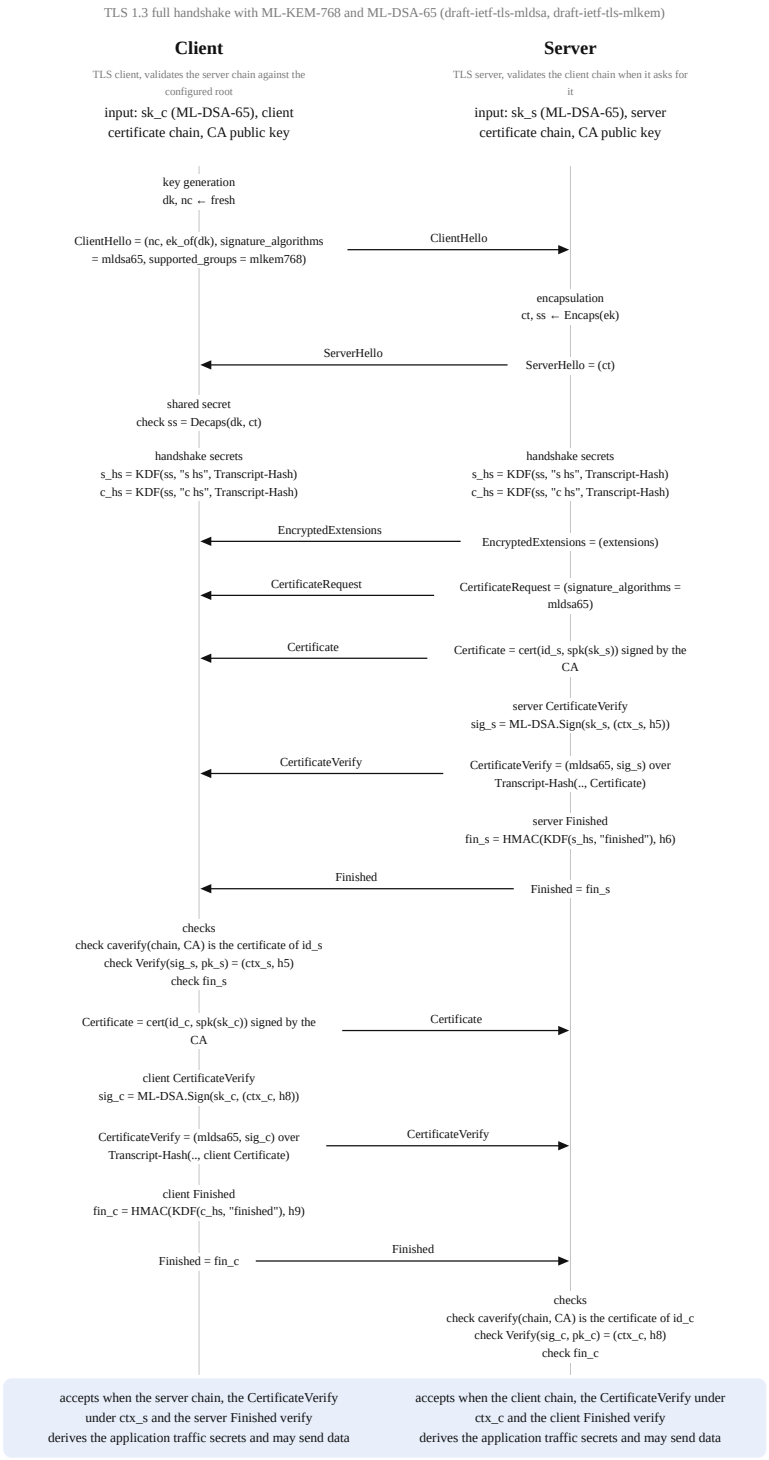


Figure 1. The protocol as specified.

The scope of the analysis is in Appendix D: what was analysed, what was left out, and the sections of the documents it is built from.

2 Findings

No finding. The analysis found nothing that the documents do not already state or set aside. The table in section 3 has the result for every property.

Within the scope analysed the handshake gives what it is meant to give: mutual authentication, agreement on the transcript and on the ML-KEM-768 secret, session uniqueness, key confirmation, secrecy of the session secrets against a network attacker, and undisclosed handshake flights. No finding: every result is something the documents already state or require. The two known limits are reproduced: a leaked signing key after the session does not expose that session, and neither a KEM-state leak nor a signing-key leak during the handshake is in the model, which are the cases where the session secret is lost by definition. The proofs rest on idealised ML-KEM-768, ML-DSA-65, hash and AEAD, on a fixed single suite with no negotiation, and on a model without the record layer.

3 Properties examined

Each row is one property put to ProVerif. The result is ProVerif's answer on the analyst's model of the protocol; value and whether the documents already state the result are the analyst's judgement, with the reasons in Appendix A.

ID	Property	Compromise assumed	Result	Value	In the documents
A02	**Agreement on peer identity.** When the client accepts the server Finished, there exists a server run that intends this client as its peer; when the server accepts the client Finished, there exists a client run that intends this server as its peer.	no compromise	Holds	medium	yes
A03	**Matching conversation and transcript.** Every time a party accepts the handshake, there exists a matching peer run, and both sides agree on each other's identities, on the ML-KEM-768 shared secret, on the negotiated signature algorithm and key exchange group, and on the transcript contents.	no compromise	Holds	medium	yes
A04	**Session uniqueness.** Every time a party accepts the handshake, it corresponds to a distinct peer run; the same peer run cannot be accepted twice.	no compromise	Holds	medium	yes
A06	**Key confirmation of handshake keys.** When the client accepts the server Finished, the server has indeed derived and holds the handshake traffic keys of the same session; likewise when the server accepts the client Finished.	no compromise	Holds	medium	yes
A08	**Negotiated parameters and direction binding.** The signature algorithm, key exchange group, CertificateVerify direction context string and transcript contents accepted by the acceptor all correspond to the values actually negotiated and sent by both parties.	no compromise	Holds	medium	yes
K01	**Secrecy of session secrets.** The attacker cannot derive the ML-KEM-768 shared secret, the handshake traffic keys, the application traffic keys, or the ML-DSA-65 signing private key.	no compromise	Holds	medium	yes
C01	**Forward secrecy after signing key leak.** After the session completes, a leak of the ML-DSA-65 signing private key still does not let the attacker derive that session's ML-KEM shared secret and traffic keys.	ML-DSA-65 signing private key (leaked after the session completes)	Holds	medium	yes
C02	**Key compromise impersonation resistance.** After the acceptor's own ML-DSA-65 signing private key leaks, and provided the party itself keeps executing honestly, the attacker still cannot impersonate an uncompromised peer.	the acceptor's own ML-DSA-65 signing private key (leaked before or during the handshake)	Holds	low	unclear

ID	Property	Compromise assumed	Result	Value	In the documents
C04	**Resilience to KEM state leak.** Under a leak of the ML-KEM-768 decapsulation private key or of the encapsulation randomness, check whether basic goals such as session key secrecy still hold.	ML-KEM-768 decapsulation private key sk or encapsulation randomness (leaked during the handshake)	Holds	low	unclear
C08	**Secrecy under broken signatures.** Under a broken ML-DSA signature primitive (the attacker can forge signatures), check whether the ML-KEM key exchange and Finished still protect goals such as session key secrecy.	ML-DSA signature primitive (before or during the handshake)	Holds	low	yes
R01	**Freshness of authentication evidence.** The CertificateVerify and Finished evidence accepted by the acceptor must be bound to this handshake's randoms and transcript; old evidence cannot be reused in a new session.	no compromise	Holds	medium	yes
R05	**No downgrade of suite.** The signature algorithm and key exchange group accepted by the acceptor satisfy the minimum policy and match the capabilities negotiated by both parties; the attacker cannot force a downgrade.	no compromise	Holds	low	yes
S01	**No reuse of encapsulation randomness.** Under the same real key, the encryption nonce and the ML-KEM encapsulation randomness are not reused.	no compromise	Holds	low	yes
A08.14	**Direction and algorithm binding.** The signature algorithm, direction context string and transcript contents accepted by the acceptor all correspond to the values actually negotiated and sent by both parties.	no compromise	Holds	medium	unclear
A03.15	**Finished binds actual transcript.** Every time a party accepts the handshake, there exists a matching peer run, and both sides agree on the transcript contents.	no compromise	Holds	medium	yes
U01	**Separate impact of key leaks.** Separately check the impact of an ML-DSA signing private key leak and of an ML-KEM decapsulation private key leak on session key secrecy and authentication.	ML-DSA signing private key or ML-KEM decapsulation private key (leaked during the handshake)	Holds	low	unclear
X01	**Handshake flight confidentiality.** A network attacker without the handshake traffic secrets cannot read the server's or the client's Certificate, CertificateVerify and Finished, and so cannot learn the certificate identities from them.	no compromise	Holds	medium	yes

Goals the documents state that the drafted list of properties did not have, added by the analyst (Appendix D): X01.

The questions asked, and the analyst's answers:

Question	Answer	Rests on
Q1: Peer authentication and direction/context string binding	Yes: the client accepts a CertificateVerify only if it was produced with the certified ML-DSA-65 key of the peer it validated, over the transcript it computed itself, with the server direction context string; the client direction is bound the same way, so the direction strings keep the two signatures apart even though the FIPS 204 context is empty.	A08, A08.14, A02
Q2: Finished bound to the actual transcript	Yes: the Finished a party accepts is a MAC over the transcript that this party computed, including both Hello randoms and the key share values, and an accepted session is matched to exactly one peer run, so a Finished of an older handshake does not fit a new one.	A03, A03.15, A04, R01

Question	Answer	Rests on
Q3: Separate impact of signing key leak / KEM key leak	They differ. A leak of the ML-DSA-65 signing key after the handshake leaves that session's shared secret and traffic secrets secret, because they come only from the ephemeral ML-KEM exchange. A leak of the ML-KEM state does expose the session secret by definition and does not give peer authentication; that case is not in the model and is left open for a per-compromise model.	C01, U01, K01, C04

4 Limits of this analysis

A result here is a result on a model of the protocol: it says what the specified messages and checks give or fail to give, with cryptography taken as ideal, and nothing about an implementation.

What this analysis does not cover, as the analyst states it:

- Cryptography is ideal: ML-KEM-768 and ML-DSA-65 are modelled as an ideal KEM and an unforgeable signature, so the analysis says nothing about their computational or post-quantum strength.
- No negotiation is modelled: one fixed group and one fixed signature scheme, so downgrade and HelloRetryRequest paths are outside the model.
- The record layer is not modelled, so record-layer AEAD nonces, sequence numbers and application data are outside it.
- A leak of the ML-KEM decapsulation key or of the encapsulation randomness, and a signing-key leak during the handshake, are not in the model, because such a leak lets the intruder run one endpoint and falsifies the honest-run agreement queries in the same file.
- Certificate validation is modelled as a CA signature checked against a pinned root plus an identity check; revocation, path building and certificate policy are outside the model.
- Resumption, PSK, 0-RTT and post-handshake authentication are out of scope by the task statement.

What stays open for each property, and what would settle it, is in Appendix A.

Appendix A: Reading of each property

The analyst's own reading of the prover's answers, not an answer of the prover: what the answers mean for the protocol, what the reading rests on, whether it can be used as it stands, and what it is worth.

A02: Agreement on peer identity

When the client accepts the server Finished, a server run exists whose intended peer is that client; when the server accepts the client Finished, a client run exists whose intended peer is that server.

The client accepts a handshake only when a server run exists that holds the same ML-KEM shared secret and computed the same transcript, and the server accepts only when a client run does. The peer identity is fixed by the certificate that validates against the trust root and by the key that verifies CertificateVerify.

- Result: Holds; answers what is asked: yes; usable with caveats
- Value medium: A proof of a central guarantee, with caveats on certificate validation and on the idealised signature.
- Stated in the documents: yes. Where the documents say it: rfc9846:520
- Rests on: Certificates are a CA signature checked against a pinned root plus an identity check; revocation and path building are not modelled.; Cryptography is ideal and the suite is fixed.

A03: Matching conversation and transcript

Whenever one party accepts the handshake, a matching peer run exists, and both sides agree on each other's identity, the ML-KEM-768 shared secret, the negotiated signature algorithm and key exchange group, and the transcript contents.

Both parties end with the same transcript hash and the same ML-KEM shared secret, and the transcript covers every handshake message including the certificate chains and both CertificateVerify messages.

- Result: Holds; answers what is asked: yes; usable with caveats
- Value medium: A proof of handshake integrity as the documents intend it, over an idealised hash and signature.
- Stated in the documents: yes. Where the documents say it: rfc9846:521
- Rests on: The transcript is a running hash over the messages, with the fields the drafts define.; Cryptography is ideal.

A04: Session uniqueness

Each time a party accepts the handshake, it corresponds to a distinct peer run; the same peer run cannot be accepted twice.

One client run is accepted by at most one server acceptance and one server run by at most one client acceptance, so a recorded handshake cannot be credited twice.

- Result: Holds; answers what is asked: yes; usable with caveats
- Value medium: An injective agreement proof resting only on the freshness of the Hello values.
- Stated in the documents: yes. Where the documents say it: rfc9846:2190

- Rests on: Freshness comes from the fresh client key share and nonce and the fresh server encapsulation randomness; no counter is needed.

A06: Key confirmation of handshake keys

When the client accepts the server Finished, the server really did derive and hold the handshake traffic keys for the same session; the same holds when the server accepts the client Finished.

A client acceptance proves the server derived and used the same handshake traffic secret, and a server acceptance proves the same of the client, because the Finished is checkable only under that secret.

- Result: Holds; answers what is asked: yes; usable with caveats
- Value medium: Key confirmation is one of the three things RFC 9846 gives the Authentication phase; the proof shows it follows from the transcript-bound MAC.
- Stated in the documents: yes. Where the documents say it: rfc9846:521
- Rests on: Finished is modelled as a MAC under a finished_key derived from the handshake secret.

A08: Negotiated parameters and direction binding

The signature algorithm, key exchange group, CertificateVerify direction context string and transcript contents that the acceptor accepts all correspond to the values the two sides actually negotiated and sent.

The CertificateVerify a party accepts is bound to the peer's certified key, to the transcript that carries the ML-KEM share and ciphertext, and to the direction context string, so neither another transcript nor the other direction's signature is accepted.

- Result: Holds; answers what is asked: yes; usable with caveats
- Value medium: Shows that transcript and direction binding survive the empty FIPS 204 context, which is where the ML-DSA draft could have lost separation.
- Stated in the documents: yes. Where the documents say it: rfc9846:540
- Rests on: The key exchange group and the signature scheme are fixed, so the parameter part of the claim holds by construction and negotiation is not modelled.

A08.14: Direction and algorithm binding

The signature algorithm, direction context string and transcript contents that the acceptor accepts all correspond to the values the two sides actually negotiated and sent.

The two direction context strings do the separating work, as the draft requires: a signature produced for the client direction is never accepted by a server and vice versa, and each is bound to the transcript the receiver holds.

- Result: Holds; answers what is asked: yes; usable with caveats
- Value medium: The draft requires the empty FIPS 204 context but does not spell out that direction separation then rests entirely on the RFC 9846 string.
- Stated in the documents: unclear
- Rests on: Only the RFC 9846 context strings separate the directions; the empty FIPS 204 context is taken from the draft.; An implementation that omitted the context string would fail here.

A03.15: Finished binds actual transcript

Whenever one party accepts the handshake, a matching peer run exists, and both sides agree on the transcript contents.

The Finished a party accepts is a MAC over the transcript that this party computed itself, so a Finished produced for another transcript, another key share or another session does not verify.

- Result: Holds; answers what is asked: yes; usable with caveats
- Value medium: Confirms the binding RFC 9846 relies on for handshake integrity.
- Stated in the documents: yes. Where the documents say it: rfc9846:521
- Rests on: Finished is a MAC; the transcript is a running hash.

K01: Secrecy of session secrets

The attacker cannot derive the ML-KEM-768 shared secret, the handshake traffic keys, the application traffic keys, or the ML-DSA-65 signing private keys.

A network attacker learns neither the ML-KEM shared secret of a completed session nor the handshake and application traffic secrets derived from it, and it never learns either ML-DSA private key. The encrypted flight and the handshake keys rest on exactly this.

- Result: Holds; answers what is asked: yes; usable with caveats
- Value medium: A proof of the secrecy the whole construction rests on, against idealised primitives.
- Stated in the documents: yes. Where the documents say it: rfc9846:657
- Rests on: ML-KEM is ideal: the shared secret is recoverable only with the decapsulation key.; Secrecy is stated for the client's completed sessions; a session the intruder runs as a client is its own.

C01: Forward secrecy after signing key leak

If the ML-DSA-65 signing private keys leak after the session has completed, the attacker still cannot derive that session's ML-KEM shared secret and traffic keys.

Once the handshake has completed, handing the attacker both ML-DSA signing keys does not reveal that session's shared secret or traffic secrets: they come only from the ephemeral ML-KEM exchange, which is independent of the signing keys.

- Result: Holds; answers what is asked: yes; usable with caveats
- Value medium: Gives the forward-secrecy argument for the ML-DSA keys mechanically; the documents state the goal but not this form of it.
- Stated in the documents: yes. Where the documents say it: rfc9846:657
- Rests on: The leak is modelled at the end of the run, not before or during the handshake.; The KEM exchange is ideal and no KEM state is leaked.

C02: Key compromise impersonation resistance

After the acceptor's own ML-DSA-65 signing private key leaks, and provided the party itself keeps executing honestly, the attacker still cannot impersonate a peer whose key has not leaked.

Holding one endpoint's own signing key does not help to authenticate as the other endpoint: what the client accepts still needs a server Finished over the same transcript and a CertificateVerify by the server's certified key, and symmetrically at the server.

- Result: Holds; answers what is asked: partly; usable with caveats
- Value low: The interesting timing of the leak is not modelled, so the answer is weaker than the target asks.
- Stated in the documents: unclear
- Rests on: The leaked key is only revealed to the attacker after the handshake; a leak before or during it is not in the model, because it lets the intruder run that endpoint and falsifies the honest-run answers in the same file.

C04: Resilience to KEM state leak

Under the condition that the ML-KEM-768 decapsulation private key or the encapsulation randomness leaks, checks whether basic goals such as session key secrecy still hold.

Peer authentication does not rest on the KEM state: the certificate and the certified signature are what bind the peer, so an attacker holding a session's KEM secret still cannot produce the peer's CertificateVerify. That the session secret itself follows from the decapsulation key is by construction of ML-KEM and is not shown by a query.

- Result: Holds; answers what is asked: partly; usable with caveats
- Value low: One of the two claims is answered; the secrecy claim is definitional and left open.
- Stated in the documents: unclear
- Rests on: The KEM-state leak itself is not in the model; only the authentication side is answered.; Session key secrecy under a KEM-state leak is lost by definition of ML-KEM.

C08: Secrecy under broken signatures

Under the condition that the ML-DSA signature primitive is broken (the attacker can forge signatures), checks whether the ML-KEM key exchange and Finished still protect goals such as session key secrecy.

The documents say signature-based TLS depends on signatures secure against chosen-message attacks, and the model shows the other side of it: an attacker that obtains the signing keys only after the handshake learns nothing about that session's keys. A forgery during the handshake is not modelled; with one, the attacker runs its own KEM exchange with the client and knows the session secret.

- Result: Holds; answers what is asked: partly; usable with caveats
- Value low: Only the after-the-fact half is shown mechanically.
- Stated in the documents: yes. Where the documents say it: main:181
- Rests on: The broken primitive is modelled as revealed signing keys after the run, not as a signing oracle during it.

R01: Freshness of authentication evidence

The CertificateVerify and Finished evidence accepted by the receiver must be bound to the random values and transcript of this handshake; old evidence must not be reusable for a new session.

Evidence is fresh: an accepted run is matched to exactly one peer run, and the peer's CertificateVerify and Finished are bound to a transcript that carries fresh Hello values, so evidence of an earlier handshake does not fit a new one.

- Result: Holds; answers what is asked: yes; usable with caveats

- Value medium: Freshness of the authentication evidence is the property that fails first in most TLS-like designs; here it holds.
- Stated in the documents: yes. Where the documents say it: rfc9846:521
- Rests on: Freshness rests on the fresh ML-KEM key share and nonce, not on a counter; the model has no state to replay.

R05: No downgrade of suite

The signature algorithm and key exchange group accepted by the receiver satisfy the minimum policy and are consistent with the capabilities negotiated by both parties; an attacker cannot force a downgrade.

The accepted handshake uses ML-DSA-65 with the fixed direction context and a transcript that carries the ML-KEM-768 share, and the model has no weaker suite to be forced to.

- Result: Holds; answers what is asked: partly; usable with caveats
- Value low: A statement that holds by construction of the model, not by an analysis of the negotiation.
- Stated in the documents: yes. Where the documents say it: rfc9846:540
- Rests on: Negotiation is not modelled: one group and one scheme are fixed, so downgrade is excluded by construction rather than proved.

S01: No reuse of encapsulation randomness

Under the same real key, the encryption nonce and the ML-KEM encapsulation randomness are not reused.

Each session uses a fresh client key share and fresh encapsulation randomness, so no ciphertext or shared secret is reused across sessions. The AEAD nonce side of the target belongs to the record layer, which is not modelled.

- Result: Holds; answers what is asked: partly; usable with caveats
- Value low: Half of the target is outside the model; the rest follows from freshness of the Hello values.
- Stated in the documents: yes. Where the documents say it: draft-ietf-tls-mlkem:220
- Rests on: Freshness is a new value per session in the model; the record-layer nonce sequence is not modelled at all.

U01: Separate impact of key leaks

Examine separately the impact of an ML-DSA signing private key leak and an ML-KEM decapsulation private key leak on session key secrecy and authentication.

Leaking the ML-DSA signing key after the session leaves the session keys secret, which is the KEM's contribution; leaking the ML-KEM decapsulation key instead exposes the session key by definition but gives no peer authentication. Only the first is shown by a query.

- Result: Holds; answers what is asked: partly; usable with caveats
- Value low: The user asked for the two leaks separately; one of them has no secrecy query.
- Stated in the documents: unclear
- Rests on: Only the signing-key leak is modelled; the KEM-state leak is compared by argument, not by a query.

X01: Handshake flight confidentiality

A network attacker without the handshake traffic secrets cannot read the server's or the client's Certificate, CertificateVerify and Finished, and so cannot learn the certificate identities from them.

Everything after ServerHello is encrypted under the handshake traffic secrets, and a network attacker that does not hold them cannot read the server's or the client's certificate, CertificateVerify and Finished.

- Result: Holds; answers what is asked: yes; usable with caveats
- Value medium: A goal the documents state for the Authentication phase, proved for the modelled flight.
- Stated in the documents: yes. Where the documents say it: rfc9846:512
- Rests on: AEAD is ideal and no traffic key leaks; the record layer itself is not modelled.

Appendix C: Claims and prover answers

The prover's record for each target: the level reached, the query that reached it, whether a control was checked, and the attacks with the analyst's verdict on each.

target	level	bound	lemma	control	attacks	claims
A02	proved		client_auth	checked		2 proved
A03	proved		client_auth	checked		2 proved
A04	proved		uniq	checked		2 proved
A06	proved		client_auth	checked		2 proved
A08	proved		cv_server	checked		2 proved
K01	proved		secret_ss	checked		5 proved
C01	proved		secret_fs	checked		1 proved
C02	proved		client_auth	checked		2 proved
C04	proved		cv_server	checked		1 proved
C08	proved		secret_fs	checked		1 proved
R01	proved		uniq	checked		2 proved
R05	proved		cv_server	checked		1 proved
S01	proved		uniq	checked		1 proved
A08.14	proved		cv_server	checked		2 proved
A03.15	proved		client_auth	checked		2 proved
U01	proved		secret_fs	checked		2 proved
X01	proved		secret_hs	checked		1 proved

Claim by claim, in the analyst's wording of each claim; the answer to each is the prover's:

target	claim	attacker	answer	query	control
A02	The client accepts the server Finished only if a server run exists with the same ML-KEM shared secret and the same transcript hash.	`network`	proved	client_auth	checked
A02	The server accepts the client Finished only if a client run exists with the same ML-KEM shared secret and the same transcript hash, and the client's certificate is the one that was validated.	`network`	proved	server_auth	checked
A03	Agreement on the transcript (all messages through the peer Finished) and on the ML-KEM shared secret, client side.	`network`	proved	client_auth	checked
A03	Agreement on the transcript (all messages through the client Finished) and on the ML-KEM shared secret, server side.	`network`	proved	server_auth	checked
A04	One client run is matched by at most one server acceptance (injective agreement on the server side).	`network`	proved	uniq	checked
A04	One server run is matched by at most one client acceptance (injective agreement on the client side).	`network`	proved	uniq_c	checked

target	claim	attacker	answer	query	control
A06	When the client accepts, the server has derived and used the same handshake traffic secret (its Finished is under it).	`network`	proved	client_auth	checked
A06	When the server accepts, the client has derived and used the same handshake traffic secret.	`network`	proved	server_auth	checked
A08	The CertificateVerify the client accepts is the one the peer signed over the client's own transcript with the server direction context string and the peer's certified key.	`network`	proved	cv_server	checked
A08	The CertificateVerify the server accepts is the one the peer signed over the server's own transcript with the client direction context string and the peer's certified key.	`network`	proved	cv_client	checked
K01	The ML-KEM-768 shared secret of a session the client accepts is not derivable by a network attacker.	`network`	proved	secret_ss	checked
K01	The handshake traffic secrets derived from that session are not derivable by a network attacker.	`network`	proved	secret_hs	checked
K01	The application traffic secret derived from that session is not derivable by a network attacker.	`network`	proved	secret_ap	checked
K01	The server's ML-DSA-65 private signing key is not derivable during the handshake.	`network`	proved	key_s	checked
K01	The client's ML-DSA-65 private signing key is not derivable during the handshake.	`network`	proved	key_c	checked
C01	After the ML-DSA signing keys are given to the attacker at the end of the run, the ML-KEM shared secret and the traffic secrets of the completed session stay unknown.	`sig_key_after_session`	proved	secret_fs	checked
C02	Holding the client's own signing key does not let the attacker make the client accept a handshake in which the server did not run.	`own_sig_key`	proved	client_auth	checked
C02	Holding the server's own signing key does not let the attacker make the server accept a handshake in which the client did not run.	`own_sig_key`	proved	server_auth	checked
C04	Peer authentication does not rest on the KEM state: a party holding a session's KEM key still cannot produce the peer's CertificateVerify, so the accepted handshake is still bound to the peer's certified signature.	`kem_dk`	proved	cv_server	checked
C08	An attacker that obtains the ML-DSA signing keys only after the handshake (the modelled form of a broken signature primitive for past runs) does not learn the session secret of that run.	`sig_broken`	proved	secret_fs	checked
R01	The client accepts only evidence bound to the current transcript, so evidence of an old handshake cannot be replayed into a new acceptance at the client.	`network`	proved	client_auth	checked
R01	Authentication evidence is fresh: one client run is accepted at most once, so an old CertificateVerify/Finished cannot be reused for a second acceptance.	`network`	proved	uniq	checked
R05	The accepted CertificateVerify is bound to the fixed ML-DSA-65 direction context and to a transcript that carries the ML-KEM-768 key share and ciphertext, and the model has no other suite to fall back to.	`network`	proved	cv_server	checked

target	claim	attacker	answer	query	control
S01	Each accepted session carries its own fresh key share, ciphertext and shared secret, and one run is accepted at most once, so encapsulation material is not reused across sessions.	`network`	proved	uniq	checked
A08.14	The direction context string separates the two CertificateVerify directions: a client-side signature is never accepted in the server direction.	`network`	proved	cv_server	checked
A08.14	The direction context string separates the two CertificateVerify directions: a server-side signature is never accepted in the client direction.	`network`	proved	cv_client	checked
A03.15	The Finished the client accepts is the MAC under the key schedule of the transcript the client itself computed, so the server's Finished is bound to the actual transcript.	`network`	proved	client_auth	checked
A03.15	The Finished the server accepts is the MAC under the key schedule of the transcript the server itself computed.	`network`	proved	server_auth	checked
U01	Leaking the ML-DSA signing key after the session leaves the session key and the traffic secrets secret; only leaking the KEM state would expose them.	`sig_key_after_session`	proved	secret_fs	checked
U01	Holding the ML-KEM decapsulation key of a session does not give peer authentication: the accepted handshake is still bound to the peer's certified ML-DSA signature over the transcript.	`kem_dk`	proved	cv_server	checked
X01	The handshake traffic secrets are not known to the network attacker, so the encrypted server and client flights cannot be read.	`network`	proved	secret_hs	checked

Appendix D: How the analysis was run

An LLM analyst builds the model of the protocol from the documents and puts each property to ProVerif; every result in section 3 is an answer of the prover.

ProVerif+ Agent session session-20261114T220938, 2026-11-14. Status: finished. Model: deepseek-v4-1-flash-261017. Steps: 50, prover 132 s, wall 15 min, cost 0.128 USD.

ProVerif: 83 proofs run, 83 decided, 0 without an answer, 82 s, 52 answers recorded.

What was analysed

Protocol	TLS ML-DSA authentication with ML-KEM
Version	draft-ietf-tls-mldsa-06
Analysed	Fixed ML-DSA-65 and ML-KEM-768, one full 1-RTT mutual certificate authentication; the trust anchors and validated public keys are the boundary, certificate validation is explicitly recorded, and handshake encryption and the full transcript must be preserved; HRR/PSK/0-RTT are excluded.
Not analysed	The symbolic model does not prove the computational security or the quantum resistance of the algorithms; attack traces hold only under explicit assumptions and are not directly equivalent to specification vulnerabilities. Every property must have a reachability/positive control.
Source text	543 numbered passages (138081 bytes) from the sections listed below. A reference such as s0001 in this report is one of these passages

The sections of the documents the analysis is built from:

Document	Section	Title
main	3	ML-DSA SignatureScheme Values
main	3.1	Certificate Chain
main	3.2	Handshake Signature
main	4	Security Considerations
draft-ietf-tls-mlkem	3	Key encapsulation mechanisms
draft-ietf-tls-mlkem	4	Construction
draft-ietf-tls-mlkem	4.1	Negotiation
draft-ietf-tls-mlkem	4.2	Transmitting encapsulation keys and ciphertexts
draft-ietf-tls-mlkem	4.3	Shared secret calculation
rfc9846	4	Handshake Protocol
rfc9846	4.1	The Transcript Hash
rfc9846	4.2	Key Exchange Messages
rfc9846	4.2.1	Cryptographic Negotiation
rfc9846	4.2.2	Client Hello
rfc9846	4.2.3	Server Hello
rfc9846	4.2.4	Hello Retry Request
rfc9846	4.3	Extensions

Document	Section	Title
rfc9846	4.3.1	Supported Versions
rfc9846	4.3.2	Cookie
rfc9846	4.3.3	Signature Algorithms
rfc9846	4.3.4	Certificate Authorities
rfc9846	4.3.5	OID Filters
rfc9846	4.3.6	Post-Handshake Certificate-Based Client Authentication
rfc9846	4.3.7	Supported Groups
rfc9846	4.3.8	Key Share
rfc9846	4.3.8.1	Diffie-Hellman Parameters
rfc9846	4.3.8.2	ECDHE Parameters
rfc9846	4.3.9	Pre-Shared Key Exchange Modes
rfc9846	4.3.10	Early Data Indication
rfc9846	4.3.11	Pre-Shared Key Extension
rfc9846	4.3.11.1	Ticket Age
rfc9846	4.3.11.2	PSK Binder
rfc9846	4.3.11.3	Processing Order
rfc9846	4.4	Server Parameters
rfc9846	4.4.1	Encrypted Extensions
rfc9846	4.4.2	Certificate Request
rfc9846	4.5	Authentication Messages
rfc9846	4.5.1	Certificate
rfc9846	4.5.1.1	OCSP Status and SCT Extensions
rfc9846	4.5.1.2	Certificate Selection
rfc9846	4.5.1.3	Receiving a Certificate Message
rfc9846	4.5.2	Certificate Verify
rfc9846	4.5.3	Finished
rfc9846	4.6	End of Early Data
rfc9846	4.7	Post-Handshake Messages
rfc9846	4.7.1	New Session Ticket Message
rfc9846	4.7.2	Post-Handshake Authentication
rfc9846	4.7.3	Key and Initialization Vector Update
rfc9846	7	Cryptographic Computations
rfc9846	7.1	Key Schedule
rfc9846	7.2	Updating Traffic Secrets
rfc9846	7.3	Traffic Key Calculation
rfc9846	7.4	Asymmetric Shared Secret Calculation

Document	Section	Title
rfc9846	7.4.1	Finite Field Diffie-Hellman
rfc9846	7.4.2	Elliptic Curve Diffie-Hellman
rfc9846	7.5	Exporters

The purpose of the protocol, as the source step located it in the documents: fix ML-DSA-65 and ML-KEM-768, one full 1-RTT mutually authenticated handshake with certificate-based authentication on both sides; trusted roots and pre-validated public keys form the boundary, certificate validation is recorded explicitly, handshake encryption and the complete transcript must be preserved; HRR/PSK/0-RTT are excluded.

The properties against the documents

Before writing a query, the analyst checked the drafted list of properties against what the documents themselves say the protocol is for. The name, what a property concerns and what the documents say are the analyst's reading; the places cited do exist in the documents.

ID	Name	About	What the documents say	Where
A02	Agreement on peer identity	The client is assured that the server it accepts is the party named in the certificate it validated against the trusted root, and the server is assured the same of the client.	The Authentication phase authenticates the server and, optionally, the client (s0346, s0403); the ML-DSA CertificateVerify proves possession of the certificate key (s0009).	rfc9846:520, s0346, s0403, s0009, s0429
A03	Matching conversation and transcript	Both parties agree on the handshake transcript, the peer identities, the ML-KEM-768 shared secret and the negotiated parameters.	CertificateVerify covers Transcript-Hash(Handshake Context, Certificate) (s0407); Finished provides handshake integrity and key confirmation (s0429, s0430, rfc9846:521).	rfc9846:521, s0354, s0355, s0407, s0439, s0485
A04	Session uniqueness	One run of a peer is accepted at most once; no two accepts belong to the same peer run.	Key shares and ML-KEM ciphertexts must not be reused across sessions (rfc9846:2190, s0044); Finished is bound to the transcript (s0429).	rfc9846:2190, s0429, s0430, s0485
A06	Key confirmation of handshake keys	When the client accepts, the server has derived and used the same handshake traffic secrets, and conversely.	The Authentication phase provides key confirmation and handshake integrity; Finished is computed from the Base Key of the handshake traffic secret (s0429, s0434, rfc9846:521).	rfc9846:521, s0429, s0430, s0435, s0439
A08	Negotiated parameters and direction binding	The signature scheme, the key exchange group, the direction context string of CertificateVerify and the transcript the receiver checks are those the parties negotiated and sent.	ML-DSA is used with empty FIPS 204 ctx (s0009) and the RFC 9846 context strings (s0409-s0414); the server share MUST be in a group the client offered (rfc9846:540).	rfc9846:540, s0001, s0009, s0016, s0025, s0485
K01	Secrecy of session secrets	The ML-KEM shared secret, the handshake and application traffic secrets, and the ML-DSA signing keys are kept from a network attacker.	The ML-KEM shared secret is inserted into the key schedule (s0049, s0017-s0020) and the leaf secrets derive from it (s0505); signature keys never leave the endpoint.	rfc9846:657, s0017, s0019, s0020, s0049, s0505

ID	Name	About	What the documents say	Where
C01	Forward secrecy after signing key leak	After a completed session, a party's ML-DSA-65 long-term signing key leaks; the ML-KEM shared secret and traffic secrets of that session are still kept from the attacker.	Forward secrecy rests on the ephemeral asymmetric secret being the only source of the session secret (s0498, s0505); the KEM key pair is per handshake (s0018).	rfc9846:657, s0017, s0049, s0498, s0505
C02	Key compromise impersonation resistance	The receiving party's own ML-DSA-65 signing key leaks; the attacker still cannot impersonate the peer whose key did not leak.	CertificateVerify proves possession of the certificate private key over the transcript (s0403, s0423); one direction's key does not give the other's.	s0403, s0423, s0001
C04	Resilience to KEM state leak	The ML-KEM-768 decapsulation key or the encapsulation randomness of a session leaks; session key secrecy and peer authentication are examined separately.	Decapsulation with sk recovers the randomness and the shared secret (s0018, s0020, s0047); ML-KEM-768 is the fixed group (s0023, s0033).	draft-ietf-tls-mlkem:237, s0018, s0020, s0047, s0023
C08	Secrecy under broken signatures	The ML-DSA-65 signature primitive is broken and signatures can be forged; the ML-KEM exchange and Finished are examined for what they still protect.	Signature-based TLS depends on the signature scheme being secure against chosen message attacks (s0014, main:181).	main:181, s0013, s0014, s0015
R01	Freshness of authentication evidence	CertificateVerify and Finished of an old handshake cannot be reused to make a party accept a new handshake.	Both are computed over the transcript hash that includes the Hello randoms and the key shares (s0407, s0439, s0485).	rfc9846:521, s0354, s0355, s0407, s0439
R05	No downgrade of suite	The accepted handshake uses ML-KEM-768 as key exchange group and ML-DSA-65 for CertificateVerify, and the attacker cannot force anything weaker.	The drafts fix the groups and signature schemes and require the server share to be in a group the client offered (s0001, s0016, s0025, rfc9846:540).	rfc9846:540, s0001, s0016, s0025
S01	No reuse of encapsulation randomness	Under one real key, ML-KEM ciphertexts and their encapsulation randomness are never reused.	Implementations MUST NOT reuse randomness in the generation of ML-KEM ciphertexts, so ciphertexts MUST NOT be reused (s0044); key shares must not be reused across connections (rfc9846:2190).	draft-ietf-tls-mlkem:220, s0044, s0016, s0025
A08.14	Direction and algorithm binding	The client and server check a CertificateVerify whose direction context string and algorithm identify the sending side, over the transcript they hold.	The context string is part of the signed content (s0410, s0414) and ML-DSA ctx is empty, so direction separation rests on it (s0009).	s0009, s0414, s0485
A03.15	Finished binds actual transcript	A Finished that a party accepts is the MAC over the transcript that this party itself computed, including both Hello randoms and key shares.	Finished is a MAC with a finished_key derived from the handshake secret over Transcript-Hash (s0429, s0434, s0439, s0485).	s0355, s0439, s0485
U01	Separate impact of key leaks	The effect on session key secrecy and on authentication of leaking the ML-DSA-65 signing key and of leaking the ML-KEM-768 decapsulation key, examined separately.	The signing key authenticates the transcript (s0403); the KEM keys produce the shared secret (s0018, s0020, s0047).	s0014, s0018, s0020, s0047

ID	Name	About	What the documents say	Where
X01	Handshake flight confidentiality	The server and client certificates and the handshake messages after ServerHello are kept from a network attacker.	Everything after the Key Exchange phase is encrypted; the authentication messages are protected with keys derived from [sender]_handshake_traffic_secret (s0347, rfc9846:512).	rfc9846:512, s0347, s0348, s0505

Added from the documents by the analyst: X01.

The security statements of the documents, paragraph by paragraph

The host listed the 41 paragraphs of the sections in which the documents say what the protocol protects and against what, and the analyst said of each what it made of it: a property, an attacker, or why neither. Taken up: 9. Not taken: 32. Which paragraphs are listed follows from the layout of the standard; where each went is the analyst's judgement.

Paragraphs	Where	Taken up as	Why not
P01	draft-ietf-tls-mlkem:253-266	K01, A03	
P02	draft-ietf-tls-mlkem:268-273	A03, R01, A04	
P03	draft-ietf-tls-mlkem:275-278	The prover cannot be asked	Side-channel resistance of an implementation.
P04	draft-ietf-tls-mlkem:280-284	Outside the scope	Deployment guidance on standalone versus hybrid ML-KEM.
P05	draft-ietf-tls-mlkem:288-288	No goal stated	Page footer.
P06	draft-ietf-tls-mlkem:292-292	No goal stated	Draft header.
P07	draft-ietf-tls-mlkem:295-302	S01, C04, `kem_dk`	
P08	main:181-185	C08, A02, A08, `sig_broken`	
P09	main:187-192	The prover cannot be asked	Side channels of the ML-DSA implementation.
P10	rfc5280:5226-5231	No goal stated	States that certificates and CRLs are public.
P11	rfc5280:5233-5236	Outside the scope	Security factors outside X.509, such as CA practice.
P12	rfc5280:5238-5240	Outside the scope	CA and RA identity vetting, outside the task boundary.
P13	rfc5280:5246-5248	Outside the scope	Reviewing a CA certification practice statement.
P14	rfc5280:5250-5259	Outside the scope	Key-usage separation in certificates.
P15	rfc5280:5261-5274	C02, `own_sig_key`	
P16	rfc5280:5276-5280	Outside the scope	Backup of CA signing keys.
P17	rfc5280:5282-5302	Outside the scope	Revocation freshness; revocation is outside the trust boundary.
P18	rfc5280:5304-5311	Outside the scope	Distribution of trust anchors, the boundary of this task.
P19	rfc5280:5313-5318	Outside the scope	Maintenance of trusted CA information after a CA failure.
P20	rfc5280:5320-5326	A02	
P21	rfc5280:5328-5334	The prover cannot be asked	Strength of the signature algorithm and hash.

Paragraphs	Where	Taken up as	Why not
P22	rfc5280:5336-5339	Outside the scope	Name comparison rules.
P23	rfc5280:5347-5349	Outside the scope	Relaxed name comparison requirements.
P24	rfc5280:5351-5356	Outside the scope	Encoding of distinguished names.
P25	rfc5280:5358-5365	Outside the scope	Name constraints encoding.
P26	rfc5280:5367-5369	Outside the scope	Limits of nameConstraints.
P27	rfc5280:5371-5381	Outside the scope	Issuer name collisions.
P28	rfc5280:5383-5391	Outside the scope	emailAddress case sensitivity.
P29	rfc5280:5393-5397	Outside the scope	Retrieval of CRL and AIA content.
P30	rfc5280:5399-5406	Outside the scope	Circular dependencies through CRL distribution points.
P31	rfc5280:5408-5416	Outside the scope	URIs in certificate extensions.
P32	rfc5280:5418-5428	Outside the scope	Self-issued certificates and CA key update.
P33	rfc5280:5430-5437	Outside the scope	TeletexString encoding of names.
P34	rfc5280:5439-5449	Outside the scope	Visual similarity of names.
P35	rfc9846:4676-4677	A02, A03, A06, A08, R01, K01	
P36	rfc9881:573-574	Outside the scope	Points at the X.509 considerations, handled by the trust-anchor boundary.
P37	rfc9881:576-579	A02, C08, `sig_broken`	
P38	rfc9881:581-590	S01	
P39	rfc9881:592-604	The prover cannot be asked	Side-channel leakage of the signing implementation.
P40	rfc9881:606-614	The prover cannot be asked	Choice between deterministic and randomised signing.
P41	rfc9881:616-627	Outside the scope	Non-repudiation properties of the signature scheme.

The parties and the protected values, one by one

The host listed the 18 parties and protected values that the drafted properties name. For each one, the analyst said which attacker controls that party or holds that value, or why no such attacker was set. With an attacker: 4. Without: 14. The list is derived from the properties; what was made of each entry is the analyst's judgement.

ID	Party or value	Kind	Attacker	Why none
S01	client	party	`own_sig_key`	
S02	server	party	`own_sig_key`, `kem_dk`	
S03	peer identity	value	Nothing gained	Peer identities are in public certificates, not secret values.
S04	ML-KEM shared secret	value	Everything lost by definition	Holding the shared secret is holding the session key; no target asks for secrecy against that attacker.
S05	negotiated parameters	value	Nothing gained	Negotiated parameters are public.
S06	transcript	value	Nothing gained	The transcript is a hash of public handshake messages.
S07	session uniqueness	value	Nothing gained	A property of a run, not a value a party holds.

ID	Party or value	Kind	Attacker	Why none
S08	handshake traffic keys	value	Everything lost by definition	Holding the handshake traffic keys gives the confidentiality goal by definition.
S09	directional context string	value	Nothing gained	The context strings are fixed public constants.
S10	application traffic key	value	Everything lost by definition	Holding an application traffic key gives the application data it protects by definition; no target asks otherwise.
S11	ML-DSA signing private key	value	`own_sig_key`, `sig_key_after_session`	
S12	traffic key	value	Everything lost by definition	Holding a traffic key gives the protected records by definition.
S13	peer authentication	value	Nothing gained	A property, not a value a party holds.
S14	session key	value	Everything lost by definition	Session key secrecy is lost by definition once the key is held.
S15	authentication	value	Nothing gained	A property, not a value a party holds.
S16	authentication evidence	value	Nothing gained	Authentication evidence is the public transcript and its signatures; replay from it is what R01 asks against the network attacker.
S17	nonce	value	Nothing gained	The Hello random is sent in clear.
S18	encapsulation randomness	value	`kem_dk`	

The properties under each attacker

The attackers are the analyst's, set before a query was written; each answer was recorded under one of them. A cell is the prover's answer for the property under that attacker; "No answer" is an attacker set for the property under which the prover left a claim undecided or nothing was recorded; an empty cell is an attacker the analyst did not set for the property.

Attacker	Has	Basis	Where
`network`	the network only: it reads, drops, changes and sends messages, and holds no secret of a party		
`sig_key_after_session`	the ML-DSA-65 private signing key of a party, obtained after the handshake completed	Target C01; the session secret comes only from the ephemeral ML-KEM exchange (s0498, s0505).	s0498, s0505
`own_sig_key`	the receiving party's own ML-DSA-65 private signing key, from before or during the handshake	Target C02; CertificateVerify proves possession of the certificate key (s0403).	s0403, s0423
`kem_dk`	the ML-KEM-768 decapsulation key, or the encapsulation randomness, of the session under attack, during the handshake	Target C04; the holder of sk recovers the encapsulation randomness (s0047) and the shared secret (s0020).	draft-ietf-tls-mlkem:237, s0018, s0020, s0047
`sig_broken`	the ability to forge ML-DSA-65 signatures without the private key	Target C08; signature-based TLS depends on secure signatures (s0014).	main:181, s0014, s0015

ID	`network`	`sig_key_after_session`	`own_sig_key`	`kem_dk`	`sig_broken`
A02	Holds				
A03	Holds				

ID	network	sig_key_after_session	own_sig_key	kem_dk	sig_broken
A04	Holds				
A06	Holds				
A08	Holds				
K01	Holds				
C01		Holds			
C02			Holds		
C04				Holds	
C08					Holds
R01	Holds				
R05	Holds				
S01	Holds				
A08.14	Holds				
A03.15	Holds				
U01		Holds		Holds	
X01	Holds				

Source and targets

Source: 543 units, 138081 bytes, frozen in source-delivery-01.json. Targets: 14 approved (A02, A03, A04, A06, A08, K01, C01, C02, C04, C08, R01, R05, S01, U01); the questions asked: 3. The list of targets was approved as drafted, beforehand, with --approve-targets.

Decisions the host took by its documented default (task `decisions: auto`), for you to check:

- approve_package: approved source package protocol-e898ccdf382fe76e7716c5724d5e

This step as it reports itself: 1 requests (1 failed), 0.000 USD, 66016 tokens in and 5516 out, 37 s.

Notes on the prover and the model

What the analyst named for targets:

step	targets	features	reason
43	A02, A03, A04, A06, A08, A08.14, A03.15, K01, C01, C02, C04, C08, R01, R05, S01, U01, X01	plain_primitives, other	Every answer rests on idealised cryptography: ML-KEM-768 as a KEM that hides its randomness and shared secret from anyone without the decapsulation key, ML-DSA-65 as an unforgeable signature, hash/HKDF/HMAC/AEAD as one-way and non-malleable, and the certificate chain as a CA signature checked against the pinned root. No algebraic law (no XOR, no Diffie-Hellman exponentiation) is used, and the record layer is not modelled, so the AEAD nonce sequence that target S01 also speaks of is outside the model; S01 is answered only for the fresh encapsulation randomness of each session.

The analyst's closing note

Model: model.pv has the full 1-RTT TLS 1.3 handshake with ML-KEM-768 (client share = Encaps pk, server share = ciphertext, ss into the key schedule), ML-DSA-65 certificates on both sides validated against a pinned CA root, every message after ServerHello AEAD-encrypted, and a running transcript hash carried in the events together with

the session secret. A phase-1 reveal of both signing keys models the after-session key leak. No finding: every result is either stated in the documents or required by them.

Established (ProVerif, all recorded): the normal run and both accept events are reachable; mutual agreement on the peer run, on transcript and on the ML-KEM secret (client_auth, server_auth), injective session uniqueness, key confirmation of the handshake secrets, CertificateVerify bound to the certified key, the transcript and the direction context string in both directions (with the FIPS 204 context empty, direction separation rests on the RFC 9846 strings), Finished bound to the actual transcript, secrecy of the shared secret, handshake and application traffic secrets and of both signing keys, forward secrecy of a completed session against a later signing-key leak, resistance to key-compromise impersonation in the direction a leaked own key does not touch, freshness of the evidence, and confidentiality of both encrypted flights. Each proof has a falsified control (ctl_key, ctl_auth, ctl_cv, ctl_uniq).

Not answered, and why: no target rests on a value that changes across steps in the modelled flow, so no bounded.pv was written; C04 and the KEM half of U01 have no secrecy query, and C02 and C08 have no during-handshake leak, because any such leak lets the intruder run one endpoint and falsifies the honest-run agreement answers in the same file (a leaky-client role was tried and did exactly that). R05 is excluded by construction (one fixed suite, no negotiation) and S01's record-layer nonce is outside the model. Those four targets are recorded partly; a per-compromise model, or Tamarin+ Agent for the record layer, would settle them.