# Security analysis of SSH: ML-KEM/X25519 hybrid KEX and host authentication

This report in another language: 中文.

rfc10042 · ProVerif+ Agent report

| | |
|---|---|
| **Documents** | Post-Quantum/Traditional Hybrid Key Exchange with the Module-Lattice-Based Key-Encapsulation Mechanism for Use in SSH (protocol definition); The Secure Shell (SSH) Transport Layer Protocol (supporting) |
| **Properties examined** | 19: 18 hold, 1 fail (1 attack) |
| **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

The protocol is the transport-layer key exchange of SSH, using the hybrid method mlkem768x25519-sha256. A client and a server first exchange their algorithm name lists, then run one X25519 key agreement and one ML-KEM-768 encapsulation, and the server signs the resulting exchange hash with its host key. The client holds no long-term key at this layer; the server holds a host key pair, and the client is expected to know the server's public host key in advance.

At the end of the exchange both sides hold the shared secret K, the exchange hash H that serves as the session identifier, and the two directional encryption and integrity keys. The client is sure that the peer holds the server's host private key and that every value it hashed came from that server run. The server is sure of nothing about the client's identity at this layer, because user authentication is a separate protocol.

The parties:

- **Client C**: holds no long-term key at this layer; generates a fresh X25519 and ML-KEM-768 key pair per connection and knows the server's public host key in advance. Holds: V_C; the server's public host key.
- **Server S**: holds the host key pair whose public part is K_S. Holds: V_S; host key pair (host private key, K_S).

Figure 1 shows the exchange as the documents specify it.

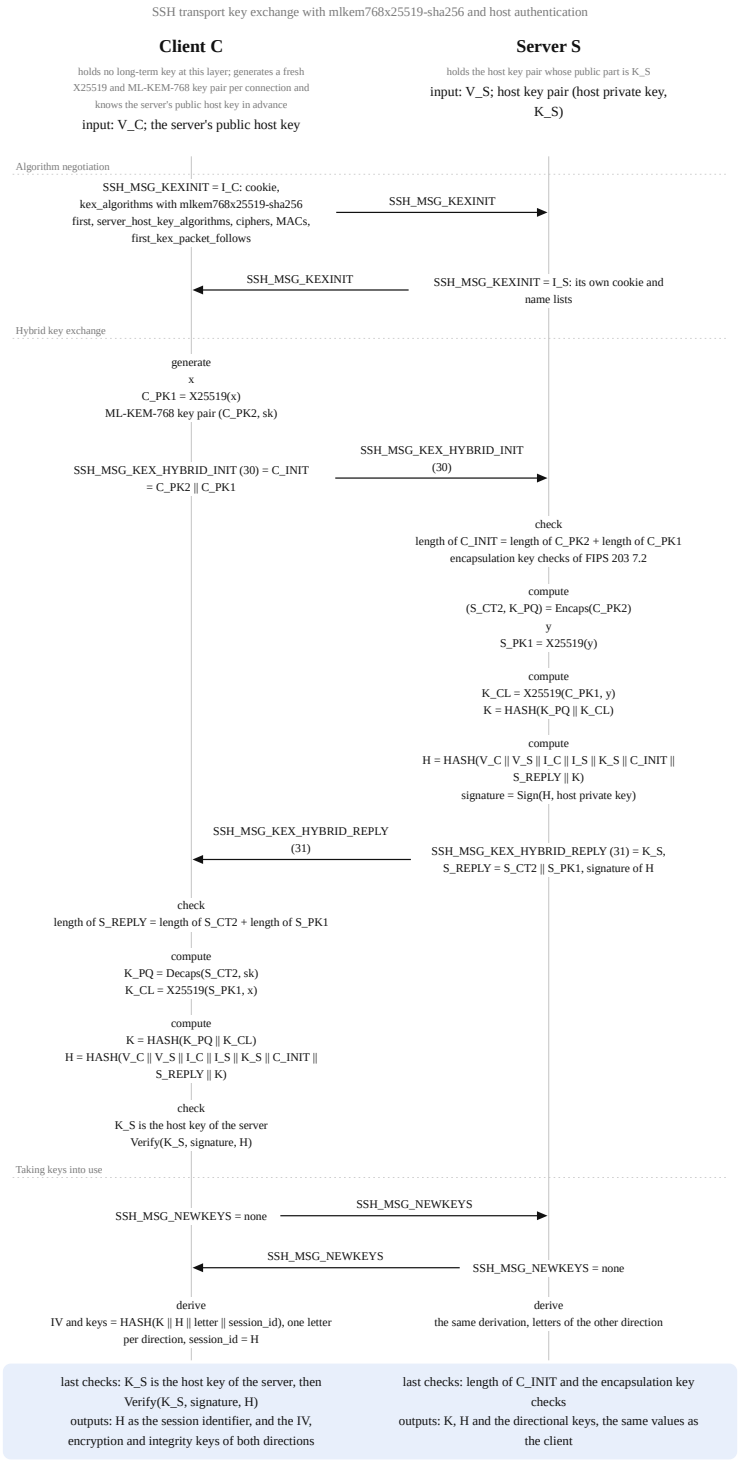



Figure 1. The protocol as specified.

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 hybrid key exchange gives what it is meant to give: the shared secret and the session keys stay out of an attacker's hands, the server authentication holds and is bound to the session, and one component falling does not reveal the key. The analysis found no weakness that the documents do not already state. It reproduced the one weakness they do state, that a client which does not compare the host key with a stored one accepts an active attacker, with the exact conditions of the attack. Every proof carries a control that fails when the protection it depends on is removed, so the verified results are not empty. The results rest on the model treating the hash, the signature and ML-KEM as ideal primitives and X25519 with only the law g^xy = g^yx; the length checks, the record layer and re-exchange are not modelled.

## 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 |
|---|---|---|---|---|---|
| K01 | **hybrid shared secret stays secret.** The attacker cannot derive the shared secret K, K_CL, K_PQ, or the session encryption and integrity keys derived from K, from the protocol interaction. | no compromise | Holds | medium | yes |
| A02 | **host key belongs to real server.** Whenever the client accepts the server's host key and signature, there exists a real server run whose intended identity matches this client. | no compromise | Holds | medium | yes |
| A03 | **agreement on exchange hash inputs.** When the client accepts, the values V_C, V_S, I_C, I_S, K_S, C_INIT, S_REPLY and K used in the server's real run match the client's local values. | no compromise | Holds | medium | yes |
| A04 | **each acceptance is a distinct run.** Each successful acceptance by the client corresponds to a different server run; the same server run cannot be accepted twice by the client. | no compromise | Holds | medium | yes |
| A05 | **no third party obtains K.** No party other than the client and the server can obtain K or the session keys derived from K. | no compromise | Holds | low | yes |
| A08 | **negotiated parameters bound into H.** The session parameters the client accepts (kex method, host key algorithm, hash) match the result negotiated in the two parties' KEXINIT messages and are bound by H. | no compromise | Holds | medium | yes |
| R05 | **accepted method meets both policies.** The accepted kex method and hash meet the minimum requirements of both parties' policies, and the negotiated result is authenticated by H and the signature. | no compromise | Holds | low | unclear |
| C01 | **forward secrecy against host key.** After the session completes and the ephemeral state is erased, even if the server's host private key is compromised, the attacker cannot obtain that session's K or session keys. | server host private key (after NEWKEYS completes and ephemeral keys are erased) | Holds | medium | yes |

| ID | Property | Compromise assumed | Result | Value | In the documents |
|---|---|---|---|---|---|
| C04 | **one ephemeral leak leaves K secret.** If either the X25519 ephemeral private key or the ML-KEM ephemeral private key leaks, K and the derived session keys remain secret from the attacker. | X25519 or ML-KEM ephemeral private key (during key exchange) | Holds | medium | unclear |
| C08 | **one component leak keeps K.** If either K_CL or K_PQ becomes known to the attacker, K and the derived session keys remain secret. | K_CL or K_PQ (during key exchange) | Holds | medium | unclear |
| D01 | **reply contents are authentic.** The HYBRID_REPLY contents the client accepts (K_S, S_REPLY, signature) were indeed sent by the claimed server and were not tampered with. | no compromise | Holds | low | yes |
| A07 | **accepted host key is the server's.** The principal of the host key K_S the client accepts matches the claimed server identity, and the signature is produced by the private key corresponding to that host key. | no compromise | Holds | low | yes |
| K04 | **distinct keys per direction and session.** The client-to-server and server-to-client encryption/integrity keys differ, and the keys of different sessions differ. | no compromise | Holds | low | yes |
| K01.14 | **attacker cannot derive K or keys.** The attacker cannot derive K or the derived session keys. | no compromise | Holds | low | yes |
| A03.15 | **exchange hash binds all inputs.** When the client accepts, the version, negotiated content, host key and both parties' ephemeral public keys bound in H match the server's real run. | no compromise | Holds | low | yes |
| A02.16 | **server authentication bound to session.** When the client accepts the server's host key and signature, the peer is indeed a server running with this client as its target, and the authentication is bound to this session's H. | no compromise | Holds | low | yes |
| X01 | **fresh ML-KEM encapsulation per session.** Two server encapsulations never produce the same S_CT2 unless they encapsulate the same secret K_PQ; in particular the ML-KEM ciphertext and its randomness are never reused across sessions (query g1). | no compromise | Holds | low | yes |
| X02 | **host key verification stops impersonation.** If the client does not compare K_S with an a priori known host key, an active attacker that owns a host key pair of its own completes the key exchange with the client and is accepted as the server (query g2 must fail). | no compromise | Attack holds (stated in the documents) | medium | yes |
| X03 | **exchange hash kept secret.** The exchange hash H, which is the session identifier, is not derivable by the network attacker from the exchange (query h_secret must hold). | no compromise | Holds | low | yes |

"Attack holds (stated in the documents)": the prover found the attack, and the documents themselves already state it. The property fails; this is a known result, not a finding (Appendix B).

Goals the documents state that the drafted list of properties did not have, added by the analyst (Appendix D): X01, X02, X03.

The questions asked, and the analyst's answers:

| Question | Answer | Rests on |
|---|---|---|
| Q1: shared secret and session keys stay secret | Yes: K, both component secrets and both directional session keys stay with the client and the server in every session, and disclosing one component secret or one ephemeral private key does not reveal K; only disclosing the whole ephemeral state of one session does. | K01, C04, C08 |
| Q2: exchange hash binds version, negotiated content, host key and both shares | Yes: the exchange hash the client verifies covers the two identification strings, both name lists (so the negotiated method and hash), the host key, both exchange values and the shared secret, and the model shows that every one of those fields equals the server's own value. | A03, A03.15, A08, X03 |
| Q3: server authentication bound to the session | Yes, provided the client compares the offered host key with one it knows: an accepted host key and signature then imply a server run that used this session's exchange values, so the authentication is bound to the session rather than replayable. | A02, A02.16, A07, A04, X02 |

## 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:

- The length checks on C_INIT and S_REPLY (main s0009, s0010) are modelled by typed parsing, not by lengths, so an attack that shifts the boundary between C_PK2 and C_PK1 or extends S_CT2 is outside the model.
- ML-KEM is modelled as a KEM that encapsulates a fresh secret; FIPS 203 implicit rejection (a decapsulation that returns a pseudorandom key instead of failing), the encapsulation-key and modulus checks and the exact encodings are not modelled.
- X25519 carries only the law g^xy = g^yx: small-order public keys, the all-zero shared secret and the validity checks of RFC 5656 section 4 are not modelled, and no other curve algebra is available.
- The hash, the key derivation and the signature are opaque functions; the key extension of RFC 4253 7.2 beyond the first block and the IND-CCA2 assumptions of main s0047 are cryptographic assumptions, not what was proved.
- Algorithm negotiation is fixed to mlkem768x25519-sha256 with one host key algorithm: the name-list selection of rfc4253 s0065, first_kex_packet_follows and a downgrade to another method are not modelled, and the model has no minimum policy set.
- One server host key exists and the client knows it in advance; certificate or known-hosts distribution and the update of that store on first contact are not modelled.
- Re-exchange, user authentication and the record layer (packet replay, MAC, sequence numbers, rekeying) are out of scope.
- Nothing about progress, termination, resource cost, timing, traffic analysis or side channels is asked or answered.
- The session keys are derived as kdf(K, H, letter); that the two directions and different sessions never coincide was shown, but the strength of that derivation is assumed.

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.

### K01: hybrid shared secret stays secret

The attacker cannot derive the shared secret K, K_CL, K_PQ, or the session encryption and integrity keys derived from K, from the protocol exchange.

Neither the network attacker nor any party other than the client and the server obtains K, the two component secrets or the derived session keys. Each component secret is protected by its own half of the exchange, so the session keys are only reachable for someone who breaks both an X25519 and an ML-KEM-768 exchange in the same session.

- Result: Holds; answers what is asked: yes; usable with caveats
- Value medium: This is the central secrecy guarantee of the method, and it rests on idealised primitives rather than on any simplification of the protocol itself.
- Stated in the documents: yes. Where the documents say it: main:425, s0034, s0047
- Rests on: ML-KEM is modelled as encapsulation of a fresh secret under the client's public key, without FIPS 203 implicit rejection or its input checks.; X25519 carries only the law g^xy = g^yx.; The hash and the key derivation are opaque one-way functions, so no length-extension or random-oracle reasoning is available.; The session keys are kdf(K, H, letter) rather than the full RFC 4253 7.2 key extension.; Negotiation is fixed to the one method, so no other cipher suite is in the model.; The claim is a secrecy query on a situation that must never occur, and its control gives the whole ephemeral state of a session away and does reach K.
- Next step: An equivalence-based prover with a real ML-KEM and X25519 model would settle the claim without the idealisation.

### A02: host key belongs to real server

Whenever the client accepts the server host key and signature, there exists a real running server whose intended identity matches this client.

A client that holds the server's host public key and verifies the signature accepts only when a server run produced that very reply for that very exchange. An attacker on the network cannot make the client accept a host key or a signature that no server sent; the control shows the same claim failing once the comparison of the host key is left out.

- Result: Holds; answers what is asked: yes; usable with caveats
- Value medium: Server host authentication is the guarantee the transport layer exists for; the proof covers the protocol logic but not the distribution of the host key it depends on.
- Stated in the documents: yes. Where the documents say it: rfc4251:652, s0112
- Rests on: Signatures are modelled so that the client can check them but the attacker cannot recover the signed hash from a signature.; The client's knowledge of the host public key is a given; no certificate or known-hosts infrastructure is modelled.; One host key exists, so the identity the client is assured of is a single server.
- Open: The case where the client accepts a host key it has not seen before is not part of this claim; the added target on host key verification covers it.
- Next step: Model a host key store that is written on first contact to state what the first connection guarantees.

### A03: agreement on exchange hash inputs

When the client accepts, the values V_C, V_S, I_C, I_S, K_S, C_INIT, S_REPLY and K used in the server's actual run match the client's local values.

Every field the client hashes into the exchange hash is the value of the server run it is talking to, so the signed hash binds the two identification strings, both name lists, the host key, both exchange values and the shared secret to one session. A change to any of them by the attacker makes the signature check fail.

- Result: Holds; answers what is asked: yes; usable with caveats
- Value medium: Agreement on the exchange hash inputs is what makes the signature meaningful, and the model shows it field by field.
- Stated in the documents: yes. Where the documents say it: s0040, s0041
- Rests on: The hash is an injective, opaque function, so agreement on H is taken as agreement on all its inputs.; The two identification strings and the name lists are opaque values in the model.
- Next step: None needed within the scope.

### A04: each acceptance is a distinct run

Each successful acceptance by the client corresponds to a different server run; the same server run cannot be accepted twice by the client.

One server reply is accepted at most once: because every session uses fresh ephemeral keys and the client's own exchange values enter the hash, a recorded reply does not verify in another session. Each acceptance therefore corresponds to its own server run.

- Result: Holds; answers what is asked: yes; usable with caveats
- Value medium: It is the replay claim of the key exchange, shown injectively rather than as plain correspondence.
- Stated in the documents: yes. Where the documents say it: rfc4251:857, s0090
- Rests on: Freshness comes from fresh ephemeral values; no counter or timestamp is involved.; The hash is treated as collision free.
- Next step: None needed within the scope.

### A05: no third party obtains K

No subject other than the client and the server can obtain K or the session keys derived from K.

Only the two endpoints of a session can compute K; a third party, whether another client, another server or someone on the network, stays outside. The model has no third role in the key exchange, so the attacker stands for any other party.

- Result: Holds; answers what is asked: yes; usable with caveats
- Value low: This is answered by the same query as the secrecy of K; it adds no separate content.
- Stated in the documents: yes. Where the documents say it: rfc4251:652
- Rests on: The model has one client role and one server role on the public channel; there is no third participating entity.; The same idealisations as the secrecy of K apply.
- Next step: Add a third party role if a protocol variant with an intermediary is to be checked.

### A08: negotiated parameters bound into H

The session parameters the client accepts (kex method, host key algorithm, hash) match the result negotiated in both sides' KEXINIT and are bound by H.

The two name lists the client accepted are the server's own, and they are covered by the signed exchange hash, so no peer can have run the session with different parameters. The method itself is fixed to mlkem768x25519-sha256 and SHA-256 in the model, so the claim is about the binding rather than about the choice.

- Result: Holds; answers what is asked: yes; usable with caveats
- Value medium: Binding the negotiated parameters into the signed hash is what would stop a downgrade, and it is shown for the parameters the model carries.
- Stated in the documents: yes. Where the documents say it: s0016, s0040, s0041
- Rests on: The model has a single key exchange method and a single hash, and the name lists are opaque payloads.; The choice procedure of rfc4253 s0065 is not modelled.
- Open: A model with several methods would be needed to show that a downgrade to a weaker method is detected rather than merely that the parameters are bound.
- Next step: Add the three PQ/T method names of main s0017-s0019 and the classical methods, then ask whether an accepted session can use a method one side did not list.

### R05: accepted method meets both policies

The accepted kex method and hash meet the minimum requirements of both sides' policies, and the negotiation result is authenticated by H and the signature.

What can be shown is that the parameters a session runs with are authenticated: the name lists the client accepted are the server's and are covered by the signature. Whether an accepted method meets a minimum strength policy is not answered, because the model fixes the one method and the documents set no minimum for it.

- Result: Holds; answers what is asked: partly; usable with caveats
- Value low: The half that is answered follows from the binding of the name lists; the strength-policy half is outside what this model can say.
- Stated in the documents: unclear. Source text it rests on: rfc4251:1046, s0016
- Rests on: One fixed method and one hash in the model, with no policy threshold.; Strength ordering is a policy question that a trace prover cannot weigh.
- Open: No minimum key exchange strength is defined for this method in the documents, and the model has no policy to compare against.
- Next step: Fix a policy set in the model, for instance by listing the accepted methods, and ask whether a session can be accepted with a method outside it.

### C01: forward secrecy against host key

After the session completes and the ephemeral state is erased, the attacker cannot obtain the session's K or session keys even if the server's host private key is disclosed.

A session's shared secret is not recoverable from the server's host private key, even if that key is disclosed after the session ends. The secret lives in the ephemeral X25519 exponent and ML-KEM private key, which the protocol requires to be fresh per connection.

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

- Value medium: It reproduces mechanically, under the exact adversary the documents define, the forward secrecy they attribute to SSH key exchange.
- Stated in the documents: yes. Where the documents say it: rfc4251:1021, s0045
- Rests on: The compromise is a phase after all sessions, and the ephemeral values are not disclosed.; The control shows that the same query fails once the whole ephemeral state of the session is given out, so the claim is not empty.
- Open: Erasure of the ephemeral values is assumed, not modelled.
- Next step: Model re-exchange and the erasure of the old keys to state forward secrecy across a rekey.

### C04: one ephemeral leak leaves K secret

When one of the X25519 ephemeral private key or the ML-KEM ephemeral private key is disclosed, K and the derived session keys remain secret from the attacker.

If only one ephemeral private key of a session is disclosed, the shared secret stays out of reach: the X25519 exponent alone does not give K_PQ and the ML-KEM private key alone does not give K_CL. Only the disclosure of both halves of the same session gives the secret, which the control confirms.

- Result: Holds; answers what is asked: yes; usable with caveats
- Value medium: This is the hybrid property the method is designed for, and the failure case is exhibited by the control.
- Stated in the documents: unclear. Source text it rests on: main:425, s0034, s0035
- Rests on: The ML-KEM secret is protected by the KEM abstraction, so the claim is only as strong as treating ML-KEM-768 as a secure KEM.; The two component secrets are combined by an opaque one-way hash.
- Open: The documents cite external proofs of the key combination rather than stating this per-component case.
- Next step: A computational proof of the combination, or a prover with a full ML-KEM model, would settle the strength of the claim.

### C08: one component leak keeps K

When one of K_CL or K_PQ becomes known to the attacker, K and the derived session keys remain secret.

Knowing one of the two component secrets does not give the shared secret: a known K_CL leaves the ML-KEM secret in place and a known K_PQ leaves the X25519 secret in place. So the hybrid survives the failure of one of its two halves, but not of both.

- Result: Holds; answers what is asked: yes; usable with caveats
- Value medium: It states the hybrid's fallback guarantee, which is the reason to run two exchanges instead of one.
- Stated in the documents: unclear. Source text it rests on: main:425, s0047, s0048
- Rests on: The hash combining the two secrets is treated as one way.; The classical half is modelled with the single Diffie-Hellman law, so forcing K_CL to a known value such as the all-zero point is representable but the curve checks that would prevent it are not modelled.
- Open: The documents give the goal and cite proofs, but say nothing about the single-component case in these words.
- Next step: Add the X25519 all-zero and small-order checks and ask again whether a forced classical secret still leaves the key secret.

### D01: reply contents are authentic

The HYBRID_REPLY contents the client accepts (K_S, S_REPLY, the signature) really were sent by the claimed server and have not been tampered with.

The host key, the exchange value string and the signature that the client accepts are the ones the server produced for this session, so nothing in the reply can be modified or forged in transit.

- Result: Holds; answers what is asked: yes; usable with caveats
- Value low: It is the practical form of the host authentication claim and shares its query.
- Stated in the documents: yes. Where the documents say it: s0008, s0112
- Rests on: The same idealisation of the signature and of the hash as for the host authentication claim.; The reply is modelled as one message; the record layer framing is not modelled.
- Next step: None needed within the scope.

### A07: accepted host key is the server's

The principal of the host key K_S the client accepts matches the claimed server identity, and the signature was produced by the private key corresponding to that host key.

The host key the client accepts is the key whose private key signed this session's exchange hash, and it is the key the client holds for the server. A key offered by anyone else fails either the comparison or the signature check.

- Result: Holds; answers what is asked: yes; usable with caveats
- Value low: It restates the identity side of the host authentication claim, which is answered by the same query.
- Stated in the documents: yes. Where the documents say it: rfc4251:204, s0112
- Rests on: One host key per server and a client that knows it in advance.; Signatures do not reveal the signed hash to the attacker.
- Next step: None needed within the scope.

### K04: distinct keys per direction and session

The encryption/integrity keys from client to server and from server to client differ, and keys differ between sessions.

The two directions of a session never end up with the same key, and two sessions never derive the same client key: the direction and the fresh exchange values enter the derivation, so the keys are separated per direction and per session.

- Result: Holds; answers what is asked: yes; usable with caveats
- Value low: It shows key separation, which is a prerequisite for the record layer but not a guarantee the documents argue for at length.
- Stated in the documents: yes. Where the documents say it: s0090, s0092, s0100
- Rests on: The derivation kdf(K, H, letter) is modelled as injective in its arguments.; The letters of RFC 4253 7.2 are modelled as two distinct constants.; The controls show that the opposite statements fail, so the two directions and two sessions are genuinely distinct in the model.
- Next step: None needed within the scope.

### K01.14: attacker cannot derive K or keys

The attacker cannot derive K or the derived session keys.

The shared secret and the session keys derived from it stay with the two endpoints; the attacker on the network gets neither the secret nor the keys of any session.

- Result: Holds; answers what is asked: yes; usable with caveats
- Value low: It is the user's restatement of the secrecy claim and is answered by the same queries.
- Stated in the documents: yes. Where the documents say it: s0034
- Rests on: The same idealisations as the secrecy claim: a KEM abstraction of ML-KEM, one Diffie-Hellman law, an opaque hash and derivation.
- Next step: None needed within the scope.

### A03.15: exchange hash binds all inputs

When the client accepts, the version, negotiation contents, host key and both parties' ephemeral public keys bound in H match the server's actual run.

All the fields that go into the exchange hash, including the two name lists and the two exchange values, are the server run's own values, so the hash the client verifies is bound to this session.

- Result: Holds; answers what is asked: yes; usable with caveats
- Value low: It is the user's restatement of the agreement on the hash inputs.
- Stated in the documents: yes. Where the documents say it: s0040
- Rests on: The hash is opaque and collision free, so agreement on the hash is agreement on its inputs.
- Next step: None needed within the scope.

### A02.16: server authentication bound to session

When the client accepts the server host key and signature, the peer really is a server run whose target is this client, and the authentication is bound to this session's H.

An accepted host key and signature imply that a server run took part in this very exchange, because the client's own exchange value is inside the hash the server signed. The authentication is therefore bound to the session and cannot be moved to another one.

- Result: Holds; answers what is asked: yes; usable with caveats
- Value low: It is the user's restatement of the host authentication claim in its session-bound form.
- Stated in the documents: yes. Where the documents say it: s0057, s0112
- Rests on: The client's session values are fresh per session and enter the signed hash.; The same signature idealisation as above.
- Next step: None needed within the scope.

### X01: fresh ML-KEM encapsulation per session

Two server encapsulations never produce the same S_CT2 unless they encapsulate the same secret K_PQ; in particular the ML-KEM ciphertext and its randomness are never reused across sessions (query g1).

Each session's ML-KEM ciphertext is distinct unless the same secret is encapsulated twice, so the freshness the documents require for encapsulation is present. A ciphertext repeated across two sessions would mean a reused encapsulated secret, which the model does not reach; the control shows that two sessions do encapsulate different secrets.

- Result: Holds; answers what is asked: yes; usable with caveats
- Value low: It follows directly from the freshness of the encapsulated secret and adds little beyond the secrecy claim.
- Stated in the documents: yes. Where the documents say it: main:396
- Rests on: The encapsulation is modelled with a fresh secret and an injective ciphertext constructor.; The randomness itself is not visible in the model; only the consequence for the ciphertext.
- Next step: None needed within the scope.

### X02: host key verification stops impersonation

If the client does not compare K_S with an a priori known host key, an active attacker that owns a host key pair of its own completes the key exchange with the client and is accepted as the server (query g2 must fail).

A client that skips the comparison of the offered host key with a key it knows accepts a reply from an attacker that holds a host key pair of its own. The attacker answers the client's exchange, computes the shared secret and the exchange hash itself and signs them, and the client has no other means to notice.

- Result: Attack holds (stated in the documents); answers what is asked: yes; usable with caveats
- Value medium: It shows the weakness the documents state, for the exact configuration and with the exact attacker, which is what an implementer needs in order to see that the check is load-bearing.
- Stated in the documents: yes. Where the documents say it: rfc4251:984, s0112
- Rests on: The client role without the comparison is modelled separately, exactly as rfc4253 s0112 permits.; The attacker is given a host key pair of its own, which anyone can generate.; The exchange values are modelled so that the attacker can pick its own classical exponent.
- Open: How often clients are configured this way, and what a user is shown when a key changes, is outside the model.
- Next step: Model a known-hosts store written on first contact to see what the first connection promises.

### X03: exchange hash kept secret

The exchange hash H, which is the session identifier, is not derivable by the network attacker from the exchange (query h_secret must hold).

The exchange hash, which doubles as the session identifier, is not derivable by the attacker from the exchange, because it depends on the shared secret and only appears inside a signature the attacker cannot open. Keeping it secret is therefore not load-bearing for the transport key exchange.

- Result: Holds; answers what is asked: yes; usable with caveats
- Value low: The documents only recommend keeping the hash secret, and nothing in the transport exchange stops without it.
- Stated in the documents: yes. Where the documents say it: s0121
- Rests on: The signature is modelled so that the attacker cannot recover the signed hash from it.; Everything else in the hash is public, so the claim depends only on the secrecy of K.
- Next step: None needed within the scope.

## 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 |
|---|---|---|---|---|---|---|
| K01 | proved | | k01_kcl | checked | | 5 proved, 1 reachable |
| A02 | proved | | a02 | checked | | 1 proved, 1 reachable |
| A03 | proved | | a02 | checked | | 1 proved, 1 reachable |
| A04 | proved | | a04 | checked | | 1 proved, 1 reachable |
| A05 | proved | | k01_k | checked | | 1 proved, 1 reachable |
| A08 | proved | | a02 | checked | | 1 proved, 1 reachable |
| R05 | proved | | a02 | checked | | 1 proved, 1 reachable |
| C01 | proved | | c01 | checked | | 2 proved, 1 reachable |
| C04 | proved | | c04_x | checked | | 2 proved, 2 reachable |
| C08 | proved | | c08_kcl | checked | | 2 proved, 2 reachable |
| D01 | proved | | a02 | checked | | 1 proved, 1 reachable |
| A07 | proved | | a02 | checked | | 1 proved, 1 reachable |
| K04 | proved | | k04b | checked | | 2 proved, 1 reachable |
| K01.14 | proved | | k01_k | checked | | 2 proved, 1 reachable |
| A03.15 | proved | | a02 | checked | | 1 proved, 1 reachable |
| A02.16 | proved | | a02 | checked | | 1 proved, 1 reachable |
| X01 | proved | | g1 | checked | | 1 proved, 1 reachable |
| X02 | reachable | | reach_nocheck | | g2 (holds_in_source) | 1 reachable, 1 attack |
| X03 | proved | | h_secret | checked | | 1 proved, 1 reachable |

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 |
|---|---|---|---|---|---|
| K01 | A session reaches the point where the client holds K, K_CL, K_PQ and both session keys. | | reachable | reach_secrets | |
| K01 | K_CL (the X25519 shared secret) is not derivable by the network attacker, except in a session that itself gives it away. | `network` | proved | k01_kcl | checked |
| K01 | K_PQ (the ML-KEM-768 shared secret) is not derivable by the network attacker, except in a session that itself gives it away. | `network` | proved | k01_kpq | checked |
| K01 | K = HASH(K_PQ \|\| K_CL) is not derivable by the network attacker. | `network` | proved | k01_k | checked |
| K01 | The client-to-server encryption/integrity key derived from K is not derivable by the network attacker. | `network` | proved | k01_kcs | checked |

| target | claim | attacker | answer | query | control |
|---|---|---|---|---|---|
| K01 | The server-to-client encryption/integrity key derived from K is not derivable by the network attacker. | `network` | proved | k01_ksc | checked |
| A02 | A client accepts a host key and signature (end of the host authentication step). | | reachable | reach_accept | |
| A02 | Whenever the client accepts a host key and signature, a server run sent exactly that K_S and that exchange hash H for the same session values. | `network` | proved | a02 | checked |
| A03 | A client accepts, having hashed V_C, V_S, I_C, I_S, K_S, C_INIT, S_REPLY and K. | | reachable | reach_accept | |
| A03 | Every field the client hashes into H (I_C, I_S, K_S, C_INIT, S_REPLY and K) equals the value of the server run that produced the reply. | `network` | proved | a02 | checked |
| A04 | A client accepts a reply (the situation whose freshness is asked about). | | reachable | reach_accept | |
| A04 | Each client acceptance of a host key and signature corresponds to a different server run: one server run's reply is never accepted twice. | `network` | proved | a04 | checked |
| A05 | A session reaches the point where K and its components exist. | | reachable | reach_secrets | |
| A05 | No party other than the client and the server of a session obtains K: the attacker standing for any third party cannot derive it. | `network` | proved | k01_k | checked |
| A08 | A client accepts a reply, so the KEXINIT payloads were hashed into H. | | reachable | reach_accept | |
| A08 | The accepted I_C and I_S, which carry the name lists and hence the single method mlkem768x25519-sha256, are the ones the server used and are bound into the signed exchange hash. | `network` | proved | a02 | checked |
| R05 | A client accepts a reply. | | reachable | reach_accept | |
| R05 | The negotiated parameters a party ends up using are authenticated: the I_C and I_S the client accepted are the server's own, so no peer can have used a different method for this session. | `network` | proved | a02 | checked |
| C01 | A session completes and the server host private key is afterwards revealed (the situation the forward-secrecy claim speaks about). | | reachable | reach_c01 | |
| C01 | After the session, the revealed host private key does not give K of that session, unless the session's whole ephemeral state (both the X25519 exponent and the ML-KEM private key) is revealed. | `host_key_after` | proved | c01 | checked |
| C01 | After the session, the revealed host private key does not give K of that session. | `host_key_after` | proved | c01 | checked |
| C04 | A session completes and only its X25519 ephemeral private key is revealed. | | reachable | reach_c04x | |
| C04 | A session completes and only its ML-KEM-768 encapsulation private key is revealed. | | reachable | reach_c04sk2 | |

| target | claim | attacker | answer | query | control |
|---|---|---|---|---|---|
| C04 | When only the X25519 ephemeral private key of the session is revealed, K stays secret; the ML-KEM component still protects it. | `eph_x25519_known` | proved | c04_x | checked |
| C04 | When only the ML-KEM-768 encapsulation private key of the session is revealed, K stays secret; the X25519 component still protects it. | `mlkem_sk_known` | proved | c04_sk2 | checked |
| C08 | A session completes and its K_CL is given to the attacker. | | reachable | reach_c08kcl | |
| C08 | A session completes and its K_PQ is given to the attacker. | | reachable | reach_c08kpq | |
| C08 | When K_CL is known to the attacker, K stays secret because the ML-KEM component still protects it. | `kcl_known` | proved | c08_kcl | checked |
| C08 | When K_PQ is known to the attacker, K stays secret because the X25519 component still protects it. | `kpq_known` | proved | c08_kpq | checked |
| D01 | A client accepts the reply carrying K_S, S_REPLY and the signature. | | reachable | reach_accept | |
| D01 | The K_S, S_REPLY and signature the client accepts are exactly the ones a server run sent for this session, so nothing in the reply was changed or forged. | `network` | proved | a02 | checked |
| A07 | A client accepts a host key and signature. | | reachable | reach_accept | |
| A07 | The accepted K_S is the key whose private key produced the signature over H, and it is the host key the client holds for the server it talks to. | `network` | proved | a02 | checked |
| K04 | A session completes with both session keys derived. | | reachable | reach_secrets | |
| K04 | Two sessions never share a client-to-server session key: the same key implies the same session. | `network` | proved | k04b | checked |
| K04 | The client-to-server and server-to-client keys of one session are never equal. | `network` | proved | k04a | checked |
| K01.14 | A session reaches the point where K and the derived keys exist. | | reachable | reach_secrets | |
| K01.14 | The attacker cannot derive K = HASH(K_PQ \|\| K_CL). | `network` | proved | k01_k | checked |
| K01.14 | The attacker cannot derive the session encryption/integrity keys derived from K. | `network` | proved | k01_kcs | checked |
| A03.15 | A client accepts after hashing its own copy of the fields. | | reachable | reach_accept | |
| A03.15 | The identification strings, KEXINIT payloads, host key, C_INIT, S_REPLY and K that the client hashed are the server run's own values. | `network` | proved | a02 | checked |
| A02.16 | A client accepts a host key and signature. | | reachable | reach_accept | |
| A02.16 | When the client accepts the host key and signature, a server run took part in this very exchange (same H), so the authentication is bound to this session. | `network` | proved | a02 | checked |
| X01 | A server encapsulation S_CT2 happens (a session reaches the encapsulate step). | | reachable | reach_encap | |

| target | claim | attacker | answer | query | control |
|---|---|---|---|---|---|
| X01 | No two server encapsulations produce the same S_CT2 unless they encapsulate the same secret K_PQ; the ML-KEM ciphertext and its randomness are never reused across sessions. | `network` | proved | g1 | checked |
| X02 | A client that does not compare K_S with an a priori known host key accepts any host key; a session reaches that client's accept event. | `network` | reachable | reach_nocheck | |
| X02 | Without the comparison of K_S with a known host key, an active attacker with a host key pair of its own is accepted by the client as the server: the accepted reply was never sent by a server run. | `network` | attack (holds_in_source) | g2 (the control of a02) | |
| X03 | A client accepts a reply after checking the host key and the signature, so a value of H exists. | | reachable | reach_accept | |
| X03 | The exchange hash H (the session identifier) is not derivable by the network attacker during the exchange. | `network` | proved | h_secret | checked |

# Appendix D: How the analysis was run

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

ProVerif+ Agent session session-20261114T183559, 2026-11-14. Status: finished. Model: deepseek-v4-1-flash-261017. Steps: 48, prover 37 s, wall 19 min, cost 0.141 USD.

ProVerif: 144 proofs run, 141 decided, 3 without an answer, 32 s, 70 answers recorded.

## What was analysed

| | |
|---|---|
| **Protocol** | SSH: ML-KEM/X25519 hybrid KEX with host authentication |
| **Version** | rfc10042 |
| **Analysed** | Fixed mlkem768x25519-sha256; KEXINIT, hybrid key exchange, host key check and NEWKEYS. User authentication and rekeying are extended separately. |
| **Not analysed** | The full transport/host key semantics come from the SSH baseline and cannot be given by the extension draft alone; server authentication is not treated as client user authentication. Only the properties under the explicit model and attacker conditions are proved; missing checks, a fully secure channel or a fully honest NF cannot be taken as a substitute for the conclusions to be proved. |
| **Source text** | 131 numbered passages (30345 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 | 2 | PQ/T Hybrid Key Exchange |
| main | 2.1 | PQ/T Hybrid Key Exchange Method Abstraction |
| main | 2.2 | PQ/T Hybrid Key Exchange Message Numbers |
| main | 2.3 | PQ/T Hybrid Key Exchange Method Names |
| main | 2.3.1 | mlkem768nistp256-sha256 |
| main | 2.3.2 | mlkem1024nistp384-sha384 |
| main | 2.3.3 | mlkem768x25519-sha256 |
| main | 2.4 | Shared Secret K |
| main | 2.5 | Key Derivation |
| main | 5 | Security Considerations |
| rfc4253 | 7 | Key Exchange |
| rfc4253 | 7.1 | Algorithm Negotiation |
| rfc4253 | 7.2 | Output from Key Exchange |
| rfc4253 | 7.3 | Taking Keys Into Use |
| rfc4253 | 8 | Diffie-Hellman Key Exchange |
| rfc4253 | 8.1 | diffie-hellman-group1-sha1 |
| rfc4253 | 8.2 | diffie-hellman-group14-sha1 |

| Document | Section | Title |
|---|---|---|
| rfc4253 | 9 | Key Re-Exchange |

The purpose of the protocol, as the source step located it in the documents: fix mlkem768x25519-sha256; KEXINIT, hybrid key exchange, host key check and NEWKEYS. User authentication and re-keying are extended separately.

## The properties against the documents

Before a query was written, the analyst set the drafted list of properties against what the documents themselves say the protocol is for. The name, what a property is about and what the documents say are the analyst's reading; the places named exist in the documents.

| ID | Name | About | What the documents say | Where |
|---|---|---|---|---|
| K01 | hybrid shared secret stays secret | K = HASH(K_PQ \|\| K_CL), the component secrets K_CL and K_PQ, and the encryption/integrity keys derived from K, kept from the network attacker and known only to the client and the server | main s0034-s0035 define K as the hash of the two component secrets, s0045 requires a fresh ephemeral key pair per connection, rfc4253 s0092-s0100 derive the session keys from K and H | s0034, s0035, s0045, s0092 |
| A02 | host key belongs to real server | the client's assurance that a peer which proves possession of the host private key for K_S is a server run that took part in this exchange | rfc4253 s0107-s0112: the key exchange uses explicit server authentication, the client verifies that K_S really is the host key of S and verifies the signature on H | s0057, s0108, s0112 |
| A03 | agreement on exchange hash inputs | the values V_C, V_S, I_C, I_S, K_S, C_INIT, S_REPLY and K that the client hashes into H are the values the server used | main s0040-s0041 give the exact concatenation that H covers; rfc4253 s0112 has the client recompute H over its own copies and check the signature | s0040, s0041, s0112 |
| A04 | each acceptance is a distinct run | one server run's S_REPLY and signature cannot be accepted by a client twice, and two accepted sessions cannot share their session values | rfc4253 s0090 the exchange hash of the first key exchange is a unique session identifier; rfc4251:857 warns that two sessions with the same session id can be replayed against each other; main s0045 requires fresh ephemeral keys per connection | s0045, s0090 |
| A05 | no third party obtains K | K and the keys derived from it, kept from any party other than the client and the server of that session | rfc4251:652 the transport protocol provides a confidential channel; main s0034-s0035; rfc4253 s0108 the shared secret cannot be determined by either party alone | s0034, s0035, s0108 |
| A08 | negotiated parameters bound into H | the key exchange method and host key algorithm carried in I_C and I_S, and thus the choice of mlkem768x25519-sha256 and SHA-256, bound into the signed exchange hash | main s0016 and s0019 fix the method name lists, s0032 fixes SHA-256 for the method, s0040-s0041 put I_C and I_S into H; rfc4253 s0061 defines the KEXINIT payload | s0016, s0019, s0032, s0040, s0041 |
| R05 | accepted method meets both policies | the key exchange method and hash that a party ends up using, and the strength ordering the peers ask for | main s0016 lists the allowed method names; rfc4251:1046 recommends ordering the name lists by cryptographic strength, strongest first; rfc4253 s0065 fixes how the method is chosen | s0016 |
| C01 | forward secrecy against host key | K and the session keys of a finished session, kept from an attacker who obtains the server's long-term host private key after the session | rfc4251:1021-1031 defines perfect forward secrecy and states that SSH key exchanges give it, that a session is secure even if long-term keying material is later revealed; main s0045 requires ephemeral key pairs per connection | s0034, s0045 |

| ID | Name | About | What the documents say | Where |
|---|---|---|---|---|
| C04 | one ephemeral leak leaves K secret | K and the session keys, kept from an attacker who learns only the X25519 ephemeral private key, or only the ML-KEM-768 encapsulation private key, of one session | main s0034-s0035 combine both component secrets into K, s0047 states the hybrid construction is provable with some assumptions; rfc4251:1037 notes that revealing both DH private parameters reveals the session key | s0034, s0035, s0047 |
| C08 | one component leak keeps K | K and the session keys, kept from an attacker who learns only K_CL, or only K_PQ, of one session | main s0034-s0035 hash both component secrets into K, s0047 discusses the IND-CPA/IND-CCA2 proofs of the combination, s0048 the choice of X25519 | s0034, s0035, s0047 |
| D01 | reply contents are authentic | K_S, S_REPLY and the signature that the client accepts, as sent by the server and unmodified | main s0008 defines the reply fields, rfc4253 s0112 and s0122 have the signature computed over H, which covers K_S and S_REPLY | s0008, s0040, s0112, s0122 |
| A07 | accepted host key is the server's | the subject of K_S and the private key that produced the signature, as the client is to check them | rfc4251:204-207 the host key is used to verify that the client is really talking to the correct server, which needs a priori knowledge of the key; rfc4253 s0112 | s0041, s0112 |
| K04 | distinct keys per direction and session | the client-to-server and server-to-client encryption/integrity keys, and the keys of different sessions | rfc4253 s0090-s0100 give a session identifier unique per connection and separate letters for each direction's keys; rfc4251:673 says each session key is bound to the session by session-specific random data | s0090, s0092, s0100 |
| K01.14 | attacker cannot derive K or keys | K and the keys derived from it, kept from the network attacker | user question Q1; main s0034-s0035 and s0039 | s0034, s0035, s0039 |
| A03.15 | exchange hash binds all inputs | the fields of H the client ends up hashing, as against the server's own values | user question Q2; main s0040-s0041 | s0040, s0041 |
| A02.16 | server authentication bound to session | the client's assurance about the server identity, tied to this session's H | user question Q3; rfc4253 s0057-s0058 explicit server authentication, s0112 | s0057, s0112 |
| X01 | fresh ML-KEM encapsulation per session | the server's ML-KEM ciphertext S_CT2 and the secret K_PQ it encapsulates | main s0045: implementations MUST NOT reuse randomness in the generation of ML-KEM ciphertexts, and MUST generate an ephemeral ML-KEM key pair per connection | main:396 |
| X02 | host key verification stops impersonation | the client's authentication of the server, and the host key association the client relies on | rfc4253 s0112: the client is also allowed to accept the host key without verification, however doing so renders the protocol insecure against active attacks; rfc4251:984: use of the protocol without a reliable association between a host and its host keys is inherently insecure | rfc4251:984, s0112 |
| X03 | exchange hash kept secret | H, the session identifier of rfc4253 s0090 | rfc4253 s0121: the exchange hash SHOULD be kept secret; rfc4253 s0090 makes it the session identifier | s0090, s0121 |

Added from the documents by the analyst: X01, X02, X03.

## The security statements of the documents, paragraph by paragraph

The host listed the 98 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: 20. Not taken: 78. 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 | main:392-394 | No goal stated | a reference: it delegates the ECDH part to the security considerations of RFC5656 and RFC8731, which are listed separately below (P81-P92, P95-P98) |
| P02 | main:396-406 | K01, X01, `network` | |
| P03 | main:408-423 | The prover cannot be asked | the fixed-length encodings are required against side channels and timing attacks, which a trace-based prover cannot weigh |
| P04 | main:425-438 | K01, C08 | |
| P05 | main:440-449 | The prover cannot be asked | comparison of curve implementations and of adoption, i.e. the strength and quality of primitives, not a protocol property |
| P06 | rfc4251:648-650 | No goal stated | the introduction to the gathered security considerations, no goal of its own |
| P07 | rfc4251:652-655 | K01, A02, K04, `network` | |
| P08 | rfc4251:657-662 | Outside the scope | client user authentication, which the task leaves to a separate extension |
| P09 | rfc4251:664-669 | Outside the scope | the connection protocol (multiplexing, shells, forwarding), outside the transport key exchange |
| P10 | rfc4251:673-684 | K04, X01 | |
| P11 | rfc4251:690-694 | The prover cannot be asked | the amount of entropy available to an implementation, which no trace property can express |
| P12 | rfc4251:698-701 | Outside the scope | filtering of terminal control characters when displaying text, outside the key exchange |
| P13 | rfc4251:707-721 | Outside the scope | choice and relative strength of bulk ciphers, i.e. the record layer, which the task does not model |
| P14 | rfc4251:723-726 | Outside the scope | the debug-only none cipher, record layer |
| P15 | rfc4251:728-736 | Outside the scope | CBC mode weaknesses of the record layer encryption |
| P16 to P31 | rfc4251:743-818 | Outside the scope | the CBC/IV and SSH_MSG_IGNORE discussion, all of it record-layer encryption detail |
| P32 to P34 | rfc4251:822-835 | Outside the scope | record-layer data integrity, the none MAC and the sequence-number wrap of the MAC |
| P35 to P38 | rfc4251:839-887 | A04, K04, `network` | |
| P39 to P40 | rfc4251:891-903 | A02, X02, `network` | |
| P41 | rfc4251:905-932 | A02, A07, X02, `network` | |
| P42 to P43 | rfc4251:934-962 | A02, X02, `network` | |
| P44 | rfc4251:964-982 | Outside the scope | the third case: manipulation of packets after the session is established, i.e. MAC security in the record layer |
| P45 | rfc4251:984-989 | A02, X02, `network` | |
| P46 to P48 | rfc4251:993-1009 | The prover cannot be asked | denial of service and the cost of connection set-up, which a trace property cannot measure |

| Paragraphs | Where | Taken up as | Why not |
|---|---|---|---|
| P49 | rfc4251:1013-1017 | The prover cannot be asked | covert channels through padding and ignore messages, not expressible as a trace property |
| P50 | rfc4251:1021-1035 | C01, `host_key_after` | |
| P51 | rfc4251:1037-1042 | C04, `eph_x25519_known`, `mlkem_sk_known` | |
| P52 | rfc4251:1046-1051 | R05 | |
| P53 | rfc4251:1063-1071 | The prover cannot be asked | traffic analysis and message lengths, outside what a symbolic prover can be asked |
| P54 to P56 | rfc4251:1075-1090 | Outside the scope | the authentication protocol: client user authentication, its methods and its local policy |
| P57 | rfc4251:1094-1100 | Outside the scope | which authentication methods to disable over a weak transport, a user-authentication rule |
| P58 | rfc4251:1102-1108 | A02, X02, `network` | |
| P59 | rfc4251:1114-1118 | Outside the scope | credentials divulged by an authentication protocol run against a man in the middle |
| P60 | rfc4251:1122-1139 | Outside the scope | debug messages during user authentication |
| P61 to P66 | rfc4251:1143-1182 | Outside the scope | local security policy and authorization of the authenticated user, outside the transport key exchange |
| P67 to P68 | rfc4251:1186-1194 | Outside the scope | password authentication |
| P69 | rfc4251:1198-1201 | Outside the scope | host-based user authentication |
| P70 to P71 | rfc4251:1219-1232 | Outside the scope | end-point security assumptions of the connection protocol |
| P72 to P74 | rfc4251:1236-1251 | Outside the scope | proxy forwarding |
| P75 to P78 | rfc4251:1255-1287 | Outside the scope | X11 forwarding |
| P79 | rfc4253:1428-1431 | K01, A02, K04, `network` | |
| P80 | rfc4253:1433-1434 | No goal stated | a reference to the SSH architecture document for the full security considerations |
| P81 | rfc5656:708-712 | No goal stated | a reference to the SSH-wide and elliptic-curve security considerations, reviewed here under rfc4251 and rfc4253 |
| P82 to P91 | rfc5656:714-795 | The prover cannot be asked | choice and strength of curves and hashes, i.e. the strength of primitives |
| P92 | rfc5656:797-803 | C04, `eph_x25519_known`, `mlkem_sk_known` | |
| P93 to P94 | rfc6234:6499-6506 | No goal stated | the SHA document asserts nothing about the security of its functions for a particular use |
| P95 | rfc8731:165-166 | No goal stated | a reference to the inherited security considerations |

| Paragraphs | Where | Taken up as | Why not |
|---|---|---|---|
| P96 to P98 | rfc8731:168-190 | The prover cannot be asked | security levels of Curve25519 and Curve448 and the mpint side channel, i.e. primitive strength and side channels |

## The catalogue of properties, entry by entry

The properties were drafted from a catalogue of security and privacy properties. The draft left out 29 of its entries, and the analyst went through each of them: examined here as a property, or left out with the reason. Examined: 5. Left out: 24. That the documents do not speak of a property is not a reason the analyst may give.

| Entry | Property of the catalogue | Examined as | Why not |
|---|---|---|---|
| K02 | Payload and end-to-end confidentiality | Outside the scope | no application payload is carried or protected in the transport key exchange analysed here |
| K03 | Strong confidentiality / indistinguishability of secrets | The prover cannot be asked | K is a computed value, not a free private name, so ProVerif cannot ask an indistinguishability question about it; IND-CCA2 of the key combination is a cryptographic assumption (main s0047), not a trace property |
| A01 | Peer participation (aliveness) | A02 | |
| A06 | Explicit key confirmation | Not applicable | the SSH transport key exchange has no key-confirmation message; rfc4253 s0105 explains that NEWKEYS exists so a failure can still be reported |
| R01 | Authentication / challenge freshness | A04 | |
| R02 | Message-level replay resistance | A04 | |
| R03 | Business operation at most once | Outside the scope | no business operation is carried out by the key exchange |
| R04 | Message acceptance order / window safety | Outside the scope | acceptance windows and ordering are record-layer properties, outside the task |
| C02 | Impersonation resistance under key compromise (KCI) | Not applicable | the transport layer has no client long-term credential; the only long-term key is the server host key, and its leak is C01 |
| C03 | Known-key security / leak isolation | K04 | |
| C05 | Post-compromise security recovery (PCS) | Outside the scope | re-keying and post-compromise recovery are the separate re-exchange extension the task leaves out |
| C06 | Enhanced authentication after compromise of registration credentials | Outside the scope | user authentication and registration records are out of scope |
| C07 | Resistance to offline dictionary attacks | Not applicable | the key exchange uses no password or other low-entropy secret |
| Z01 | Correctness of authorization enforcement | Outside the scope | the transport key exchange grants no authorization |
| Z02 | Delegated privileges do not expand | Outside the scope | no delegation in the transport key exchange |
| Z03 | Authorization bound to identity / session / operation | Outside the scope | no token or authorization decision in the transport key exchange |
| Z04 | Revocation and expiry take effect | Outside the scope | no revocation or expiry in the transport key exchange |

| Entry | Property of the catalogue | Examined as | Why not |
|---|---|---|---|
| Z05 | Secure isolation across tenants / across services | Outside the scope | no tenants or services in the transport key exchange |
| S01 | Encryption nonce uniqueness | X01 | |
| S02 | Retransmission / key re-installation safety | Outside the scope | retransmission and key re-installation are record-layer and re-keying matters, left out of the task |
| S03 | Rollback resistance and state monotonicity | Outside the scope | rollback of installed keys belongs to key re-exchange, which the task leaves out |
| S04 | Consistency of the security context across phases | Outside the scope | cross-phase context consistency involves user authentication and re-keying, the separate extension |
| S05 | Termination and invalidation of old state | Outside the scope | connection termination and old-state invalidation are outside the key exchange |
| P01 | Identity anonymity | Not applicable | the server host identity is a public, stable name by design and no party is anonymous in this exchange |
| P02 | Session unlinkability | Not applicable | sessions are deliberately linked to the server's long-term host key, which is the point of host authentication |
| P03 | Hiding the existence of users / credentials | Outside the scope | hiding whether a user or credential exists belongs to user authentication |
| V01 | Conditional progress / error recovery | The prover cannot be asked | whether a run can always complete or recover is a progress property a trace prover cannot be asked |
| V02 | No unexpected deadlock | The prover cannot be asked | absence of deadlock is a branching-time property |
| V03 | Bounded resource abuse / amplification resistance | The prover cannot be asked | resource cost and amplification cannot be measured in this model |

## The parties and the protected values, one by one

The host listed the 23 parties and protected values that the drafted properties name, and the analyst said of each which attacker controls the party or holds the value, or why no such attacker was set. With an attacker: 5. Without: 18. The list follows from the properties; what was made of each entry is the analyst's judgement.

| ID | Party or value | Kind | Attacker | Why none |
|---|---|---|---|---|
| S01 | Client C | party | Nothing gained | a client controlled by the attacker holds only the ephemeral secrets of the sessions it runs itself; the transport layer makes no promise to the server that a malicious client could break, and no other session's secret follows |
| S02 | Server S | party | `host_key_after` | |
| S03 | K_CL | value | `kcl_known` | |
| S04 | K_PQ | value | `kpq_known` | |
| S05 | K = HASH | value | Everything lost by definition | an attacker holding K holds exactly what the secrecy properties of K forbid, and no target asks anything weaker under that |
| S06 | Session encryption key and integrity key | value | Everything lost by definition | the session keys are functions of K; holding them is holding what their secrecy properties forbid |

| ID | Party or value | Kind | Attacker | Why none |
|---|---|---|---|---|
| S07 | Server host key K_S | value | Nothing gained | the host public key is sent on the wire in the reply |
| S08 | Host signature | value | Nothing gained | the signature is sent on the wire in the reply |
| S09 | The component fields of the exchange hash H | value | Nothing gained | every field of H except K is sent in the clear on the wire |
| S10 | K | value | Everything lost by definition | same as S05 |
| S11 | Ephemeral key pair | value | `eph_x25519_known`, `mlkem_sk_known` | |
| S12 | Session identifier | value | Everything lost by definition | holding the session identifier is exactly holding what the one goal about H being secret forbids; no other goal rests on H being unknown |
| S13 | Session key | value | Everything lost by definition | same as S06 |
| S14 | I_C | value | Nothing gained | I_C is sent on the wire |
| S15 | I_S | value | Nothing gained | I_S is sent on the wire |
| S16 | Negotiated method names | value | Nothing gained | the negotiated method names are public and appear in the KEXINIT payloads |
| S17 | H | value | Everything lost by definition | same as S12 |
| S18 | K_S | value | Nothing gained | K_S is the public host key, sent on the wire |
| S19 | S_REPLY | value | Nothing gained | S_REPLY is sent on the wire |
| S20 | Signature | value | Nothing gained | the signature is sent on the wire |
| S21 | Host private key | value | `host_key_after` | |
| S22 | Per-direction session keys | value | Everything lost by definition | same as S06 |
| S23 | The component fields of H | value | Nothing gained | same as S09 |

## 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 | | |
| `host_key_after` | the server's long-term host private key, obtained after the session's NEWKEYS exchange has completed | rfc4251:1021-1031 defines forward secrecy as the compromise of a long-term private key after a session not compromising earlier sessions, and states that SSH key exchanges have this property | rfc4251:1021 |

| Attacker | Has | Basis | Where |
|---|---|---|---|
| `eph_x25519_known` | the X25519 ephemeral private key of one side of one session, during or after that session | rfc4251:1037 says that if the DH private parameters for the client and server are revealed then the session key is revealed; the single-component case is what the hybrid construction is meant to survive (main s0034-s0035) | rfc4251:1037 |
| `mlkem_sk_known` | the ML-KEM-768 private key belonging to the client's C_PK2 in one session | main s0031: the K_PQ secret is decapsulated with the client's post-quantum KEM private key; a leak of that key is the PQ analogue of rfc4251:1037 | s0031 |
| `kcl_known` | the classical component shared secret K_CL of one session, for instance because the X25519 sharing is broken or forced to a known value | main s0034-s0035 combine K_PQ and K_CL; s0047 states the combination is provable only with assumptions, and the hybrid is meant to keep K secret when one component falls | main:425 |
| `kpq_known` | the post-quantum component shared secret K_PQ of one session | main s0034-s0035, s0047, s0048: the classical component is meant to protect K if the post-quantum one falls | main:425 |

| ID | `network` | `host_key_after` | `eph_x25519_known` | `mlkem_sk_known` | `kcl_known` | `kpq_known` |
|---|---|---|---|---|---|---|
| K01 | Holds | | | | | |
| A02 | Holds | | | | | |
| A03 | Holds | | | | | |
| A04 | Holds | | | | | |
| A05 | Holds | | | | | |
| A08 | Holds | | | | | |
| R05 | Holds | | | | | |
| C01 | | Holds | | | | |
| C04 | | | Holds | Holds | | |
| C08 | | | | | Holds | Holds |
| D01 | Holds | | | | | |
| A07 | Holds | | | | | |
| K04 | Holds | | | | | |
| K01.14 | Holds | | | | | |
| A03.15 | Holds | | | | | |
| A02.16 | Holds | | | | | |
| X01 | Holds | | | | | |
| X02 | Fails | | | | | |
| X03 | Holds | | | | | |

## Source and targets

Source: 131 units, 30345 bytes, frozen in source-delivery-01.json. Targets: 13 approved (K01, A02, A03, A04, A05, A08, R05, C01, C04, C08, D01, A07, K04); 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 the source package protocol-157741929283d26b083cb3d1ba49

This step as it reports itself: 1 requests (1 failed), 0.000 USD, 25956 tokens in and 5174 out, 33 s.

## Notes on the prover and the model

What the analyst named for targets:

| step | targets | features | reason |
|---|---|---|---|
| 36 | K01, A02, A03, A04, A05, A08, R05, C01, C04, C08, D01, A07, K04, K01.14, A03.15, A02.16, X01, X02, X03 | plain_primitives, many_sessions | Every answer rests on plain primitives only: HASH and the key derivation as opaque one-way functions, ML-KEM as encapsulation of a fresh secret, the host signature, and X25519 as a Diffie-Hellman key agreement with the single law g^xy=g^yx. No exclusive-or, no other algebraic law and no value that is installed and replaced within a session is involved (there is no counter and no sequence number in the transport key exchange; re-exchange is out of scope), so the answers do not rest on changing_state. The queries cover any number of client and server sessions in one run, so the properties are about unbounded sessions. |

## The analyst's closing note

Model: SSH transport key exchange with the hybrid method mlkem768x25519-sha256 (RFC 10042 / RFC 4253), plain run, host key authentication, in ProVerif. All 19 targets (16 drafted, 3 added from the documents) have an answer under the attacker each names, and all 48 recorded answers hold on the final text.

Established (level proved, with a control that fails when the protection is removed): - Secrecy for the network attacker: K = HASH(K_PQ || K_CL), both component secrets, both directional session keys (K01, K01.14, A05), and the exchange hash H (X03). - Host authentication: an accepted K_S and signature imply a server run with the same exchange hash and the same session values (A02, A02.16, A03, A03.15, A07, D01, A08), and each acceptance is a distinct server run (A04, injective). - Per-direction and per-session key separation (K04) and fresh ML-KEM encapsulation (X01). - Under a compromise: the host private key revealed after the session does not give K (C01, forward secrecy); only one ephemeral private key leaking (C04 with eph_x25519_known and mlkem_sk_known) or only one component shared secret known (C08 with kcl_known and kpq_known) leaves K secret; the whole ephemeral state of one session does reveal it (control ctl_full).

Attack: X02, the client that does not compare the offered host key with a stored one accepts an attacker holding a host key pair of its own (S_PK1 = g, its own encapsulation, its own signature). This is the weakness the documents state (rfc4253 s0112; rfc4251:984), so it is recorded as a known result, not a finding. No finding: nothing failed that the documents do not already state. R05 is answered only for the binding of the negotiated parameters; its strength-policy half is not expressible because the model fixes the one method and no minimum is defined.

What limits the results: the length checks of C_INIT and S_REPLY are modelled by typed parsing; ML-KEM is a KEM abstraction without FIPS 203 implicit rejection; X25519 carries only g^xy = g^yx; the hash, KDF and signature are ideal; negotiation, known-hosts distribution, the record layer and re-exchange are not modelled. The answers rest on plain primitives only, no target rests on a value that changes, so no bounded model or state review was called for. A prover with a full ML-KEM and X25519 model (Tamarin+ Agent) would settle the computational steps that are assumed here.