Security analysis of OpenPGP composite KEM encrypted message

This report in another language: 中文.

rfc9980 · ProVerif+ Agent report

Documents	Post-Quantum Cryptography in OpenPGP (protocol definition); OpenPGP (supporting)
Properties examined	12: 11 hold, 1 fail (1 attack)
Findings	1

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 encrypts one OpenPGP message to one recipient whose composite ML-KEM-768 + X25519 public key is already known and trusted. The sender encapsulates to both components of that key, combines the ML-KEM key share, the X25519 key share, the algorithm ID, the recipient's ECDH public key and a domain separator into a key encryption key with SHA3-256, and wraps the session key under it. A v6 PKESK packet carries the two ciphertexts and the wrapped key, and the message itself travels in a v2 SEIPD packet whose message key is derived from the session key and a fresh salt.

The recipient holds the two private keys. It decapsulates both components, recomputes the key encryption key with its own public key and the algorithm ID it is configured with, unwraps the session key, and stops if the key-wrap integrity check or the SEIPD authentication tag fails. Nobody is promised anything about who the other side is: any party that holds the recipient's public key can compose a message to it, and the sender learns no identity in return.

The parties:

- **Sender**: encrypts one message to a trusted public key. Holds: recipient's composite public key: R (ECDH) and ML-KEM encapsulation key; algId.
- **Recipient**: holds the two private keys of the composite scheme. Holds: r (ECDH secret key), ML-KEM decapsulation key.

Figure 1 shows the exchange as the documents specify it.

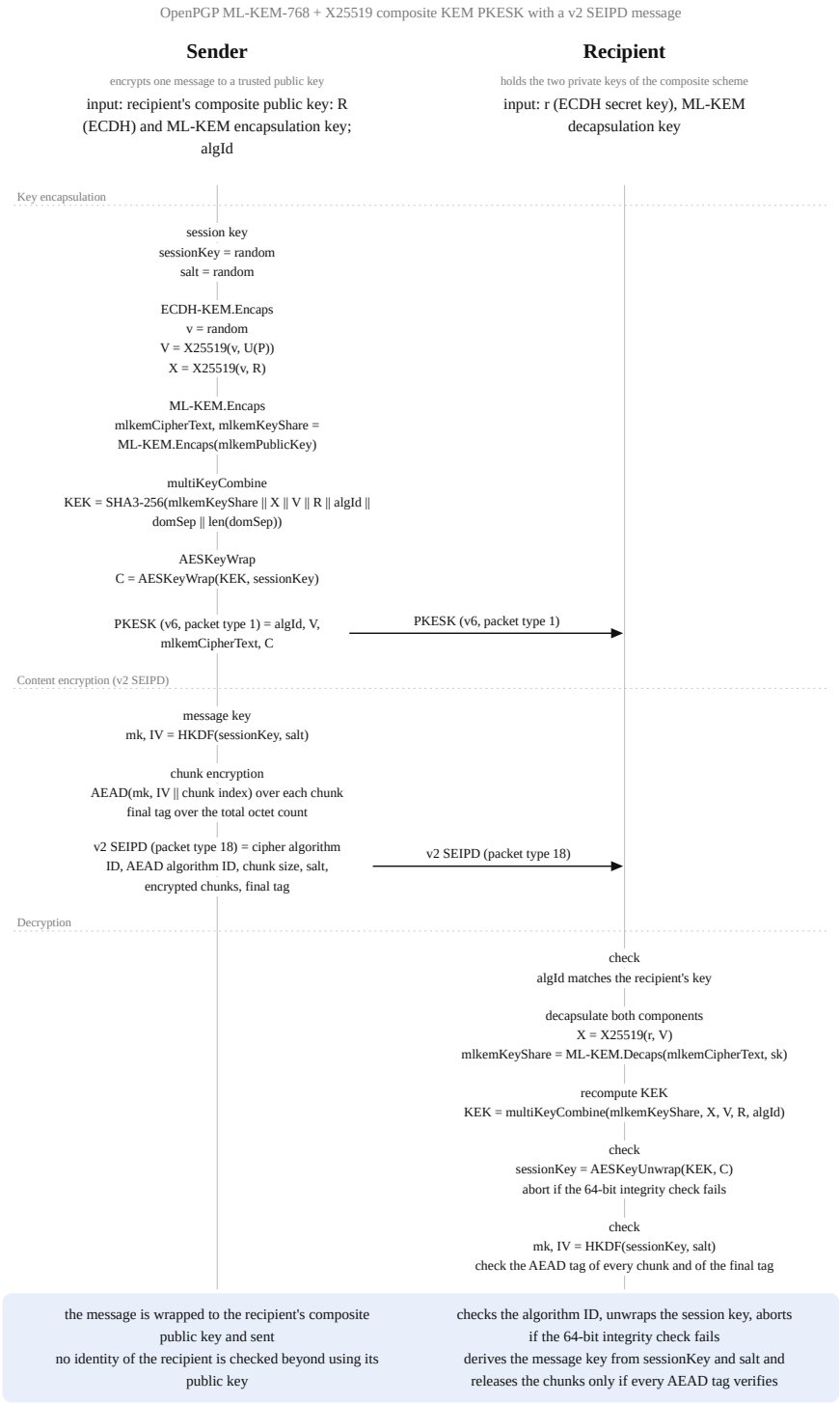


Figure 1. The protocol as specified.

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

The analysis has 1 finding: what follows is not stated in the documents. For 1 of them it is unclear whether the documents state it.

The protocol gives what it is meant to give within the modelled scope: an attacker on the network, and an attacker who breaks one component of the composite KEM, learns neither the key encryption key, nor the session key, nor the plaintext. The boundaries of that protection are exact: both private keys of the recipient at once open every recorded message, so there is no forward secrecy, and a KEK or a session key in the attacker's hands opens exactly its own message. Of the twelve targets, eleven are proved, one only for three unrolled chunks of a message, and one fails with the forward-secrecy attack. No finding is new: the documents state the combiner's IND-CCA2 argument, the AEAD integrity rule and the binding of the algorithm ID and public key that the model reproduces, and they say nothing about forward secrecy, which is inherent to encrypting to a static key. The results rest on idealised SHA3-256, AES key wrap, ML-KEM and AEAD and on Diffie-Hellman with only the commutative law, so they say nothing about the strength of those primitives.

The attack figures use the columns of Figure 1. Red is what the attacker does. Yellow marks a message that a party sends although the specified exchange does not have it at that point, and the outcome that the attack breaks.

2.1 A leaked recipient key opens every message already received

Kind	Weakness of the protocol as specified
New	Unclear: the documents state it in part, or could not be searched
Value	low
Properties	C01 (section 3)
Source text	s0064, s0084, s0085, s0097, s0098
Shown by	ProVerif attack query `c01`; Figure 2

What happens. A recipient's long-term private keys decrypt every message it has already received. With the ECDH private key the attacker recomputes the X25519 key share from the recorded ciphertext, and with the ML-KEM decapsulation key he decapsulates the recorded ML-KEM ciphertext; the key combiner then gives the key encryption key, the key wrap opens the session key, and the salt in the SEIPD packet gives the message key.

What the attacker needs. The recipient's two private keys, obtained after the message was received; the recorded PKESK and SEIPD packets, which a network attacker keeps.

What the attacker gains. The attacker reads every message the recipient ever received with that key, and every message it will receive until the recipient replaces the key. Nothing signals the loss.

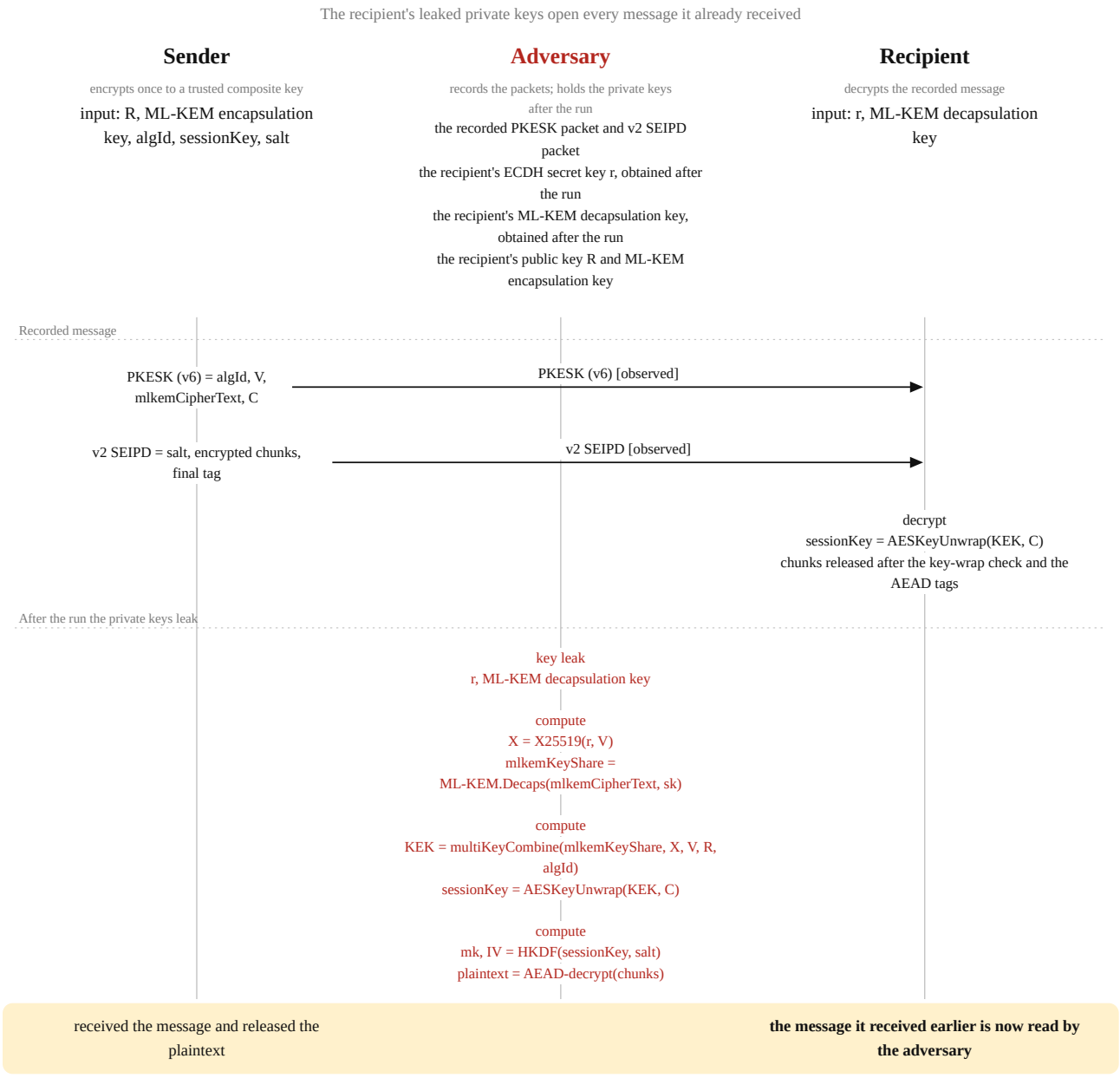


Figure 2. The recipient's leaked private keys open every message it already received. Property C01; attack query c01.

Proposed change. None. No change of the messages or of the derivations removes this: the two key shares are functions of the recipient's static ECDH key and static ML-KEM decapsulation key, so forward secrecy needs an interactive or ratcheted exchange, which is a different protocol and cannot be tested on this model.

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	**Secrecy of KEK and private keys.** The attacker cannot derive the KEK or the session key, nor the recipient's ECDH private key or ML-KEM private key.	no compromise	Holds	medium	yes

ID	Property	Compromise assumed	Result	Value	In the documents
K02	**Confidentiality of the message plaintext.** The message plaintext is visible only to the intended recipient; a network attacker cannot obtain the plaintext.	no compromise	Holds	medium	yes
A03	**Agreement on the accepted KEK.** When the recipient successfully recovers the session key, that KEK was computed by the sender over the recipient's public key, the algorithm ID and the encapsulation ciphertext of this session.	no compromise	Holds	low	no
A08	**KEK bound to algorithm and key.** The algId, ecdhPublicKey and ecdhCipherText bound to the KEK the recipient accepts match its own keys and the received PKESK; the attacker cannot redirect an encapsulation made for another algorithm or public key to this recipient.	no compromise	Holds	medium	yes
C08	**One broken component keeps the KEK.** When either the X25519 component fails (the attacker obtains ecdhKeyShare) or the ML-KEM component fails (the attacker obtains mlkemKeyShare), the KEK and the session key remain secret from the attacker.	a single component of the composite KEM (X25519 or ML-KEM) (after encryption, before decryption, or at any time)	Holds	medium	yes
C01	**No forward secrecy of past messages.** After a session completes and the recipient's long-term private keys leak, the attacker still cannot obtain the KEK, the session key or the plaintext of that earlier session.	recipient (leaked after the session completes)	Fails (1 attack)	low	unclear
S01	**AEAD nonce not reused.** Under the same message key, the AEAD nonce (the combination of IV and block index) is never reused.	no compromise	Holds	low	yes
K01.8	**Session key and plaintext secret.** The attacker cannot obtain the session key or the message plaintext.	no compromise	Holds	medium	yes
A08.9	**KEK bound to algId, key, ciphertext.** The algId, ecdhPublicKey and ecdhCipherText bound to the KEK the recipient accepts match its own keys and the received PKESK.	no compromise	Holds	medium	yes
C08.10	**Component-leak resilience with double-leak control.** When the X25519 component leaks, the ML-KEM component leaks, or both leak at once, checks whether the KEK and the session key remain secret; a single-component leak should hold, and a simultaneous leak of both components serves as the control.	the classical component and/or the PQ component of the composite KEM (after encryption, before decryption, or at any time)	Holds	medium	yes
X01	**SEIPD plaintext integrity.** The recipient never releases plaintext it did not receive from the sender under that session: if it recovers a plaintext m from the v2 SEIPD packet, the sender sealed m for the same session, so no modified or forged ciphertext is accepted as a new plaintext.	no compromise	Holds	medium	yes
X02	**Per-message KEK separation.** Two messages never share a KEK: if two encapsulations yield the same KEK they are the same encapsulation, so one message's KEK leaking says nothing about another message.	no compromise	Holds	low	no

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

The questions asked, and the analyst's answers:

Question	Answer	Rests on
Q1: plaintext and session key secrecy	The session key and the plaintext are out of reach of an attacker who has only the network, and of one who breaks only one of the two KEM components; they are lost as soon as both of the recipient's long-term private keys leak.	K01.8, K01, K02, C01
Q2: binding of the algorithm ID, recipient public key and encapsulation ciphertext to the KEK	The KEK is bound to the algorithm ID, to the recipient's ECDH public key and to the ECDH and ML-KEM ciphertexts: an encapsulation made under the other algorithm or for another party's ECDH key is never accepted, and the recipient's KEK for the sender's encapsulation is the sender's KEK.	A08.9, A08, A03
Q3: ideal-combiner conclusions when the classical and PQ components leak separately, and the double-leak control	A break of either component alone leaves the KEK and session key secret, so the combiner behaves as the documents say; with both components broken the KEK is derived at once, which is the control for the two proofs.	C08.10, C08

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:

- SHA3-256, AES key wrap, ML-KEM and the AEAD are ideal constructors and destructors: no attack exploits their structure, and nothing is stated about their computational strength.
- Diffie-Hellman carries only the law that the shared coordinate is symmetric. The group has no order, no small-order points and no contributory behaviour, so the X25519 warnings about small-order inputs and about equivalent public keys (rfc7748:810, rfc7748:820) are not tested; Tamarin+ Agent, which models the algebra, would settle them.
- Lengths, encodings and the octet layout of the PKESK and of the SEIPD packet are not modelled; the v3 PKESK check that the unwrapped key length matches the announced symmetric algorithm (s0111) is outside the modelled flow, because a v2 SEIPD packet pairs with a v6 PKESK (s0147, s0155).
- Certificates, key distribution and the recipient-identifying fields of the PKESK are out of scope, so recipient anonymity (s0200) and fingerprint checks are not analysed.
- One recipient key pair, one ECDH key and the two composite algorithm IDs are modelled; multi-recipient messages, for which the documents say the confidentiality is not post-quantum secure unless every key is (main:1133), are out of scope.
- One chunk is modelled in the main model and three in the bounded model; truncation of trailing chunks, streaming release of already-authenticated chunks, and the final length tag are not modelled.
- The strength of the random generator cannot be modelled: fresh names are assumed unguessable, so the documents' CSPRNG requirements are assumed rather than checked.
- A malicious sender is not analysed, because any party may compose a message to the public key; the sender side enters the model only as the honest encryptor.
- Side channels, timing and traffic analysis through message length cannot be asked of the prover.

What the host records:

- The answers of S01 rest on an operation the model simplifies (Appendix D).

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: Secrecy of KEK and private keys

The attacker cannot derive the KEK, the session key, or the recipient's ECDH private key and ML-KEM private key.

An attacker who sees every packet cannot derive the KEK, the session key or either of the recipient's private keys. The KEK needs both key shares, and each of them comes from a different component of the composite KEM. The control is the same queries asked of an attacker who has been given the two key shares, where they fail at once.

- Result: Holds; answers what is asked: yes; usable with caveats
- Value medium: This is the central confidentiality of the construction, proved without simplification other than the idealised primitives.
- Stated in the documents: yes. Where the documents say it: main:1210
- Rests on: SHA3-256, AES key wrap, ML-KEM and the AEAD are ideal constructors and destructors, so no computational weakness is captured.; Diffie-Hellman carries only the commutative law; no group order or small-order point exists in the model.; The recipient's public key is assumed authentic and is simply published; certificates are out of scope.; Lengths, encodings and the octet layout of the packets are not modelled.
- Open: The model gives the attacker the recipient's public key but not a certificate, so nothing about key authenticity is answered here.
- Next step: A model that includes the certificate and its binding signatures would settle the authenticity part.

K02: Confidentiality of the message plaintext

The message plaintext is visible only to the intended recipient; a network attacker cannot obtain the plaintext.

The plaintext stays hidden from an attacker on the network: it is encrypted under a message key derived from a session key that the attacker cannot obtain. The plaintext is readable only by a holder of both of the recipient's private keys, which the control shows by handing the attacker those keys.

- Result: Holds; answers what is asked: yes; usable with caveats
- Value medium: A central guarantee proved with the usual idealisation of the symmetric primitive.
- Stated in the documents: yes. Where the documents say it: rfc9580:434
- Rests on: The AEAD is an ideal authenticated encryption: a wrong key or nonce yields no plaintext at all.; The message key derivation from the session key and the salt is a private function.; One chunk is modelled.
- Open: Message length and the number of chunks are not modelled, so length leakage is not covered.
- Next step: A model with padding and observable lengths would be needed for traffic analysis, which the prover cannot be asked about.

A03: Agreement on the accepted KEK

When the recipient successfully unwraps the session key, that KEK was computed by the sender for the recipient's public key, the algorithm ID and this encapsulation's ciphertext.

When the recipient unwraps a session key, the KEK it holds is the sender's KEK for its own public key, the algorithm ID it is configured with and exactly the two ciphertexts it received. The injective reading fails, because the same message may be delivered to several recipient runs and because anyone may act as a sender.

- Result: Holds; answers what is asked: yes; usable with caveats
- Value low: It follows directly from both parties recomputing the same function, so it confirms the design rather than discovering anything.
- Stated in the documents: no
- Rests on: The claim is stated against the sender's own encapsulation; without sender authentication a message the attacker composes is not a violation.; The KEK is idealised as an injective function of its inputs.; The control is the injective variant, which fails: one sender's encapsulation can be accepted by two recipient runs.
- Open: Only one recipient key pair is in the model, so a transplant aimed at a second recipient cannot be expressed.
- Next step: A second recipient key pair with its own identifier would let the transplant be asked directly.

A08: KEK bound to algorithm and key

The algId, ecdhPublicKey and ecdhCipherText bound to the KEK that the recipient accepts are consistent with its own keys and the received PKESK; an attacker cannot transplant an encapsulation generated for another algorithm or public key onto this recipient.

The KEK is bound to the algorithm ID and to the recipient's ECDH public key: a recipient never accepts a KEK computed under the other composite algorithm, and never one computed for another party's ECDH key, even with the same ML-KEM key and the same ECDH ciphertext.

- Result: Holds; answers what is asked: yes; usable with caveats
- Value medium: The documents claim this binding as the reason multi-target attacks on X25519 are covered; the model reproduces it mechanically.
- Stated in the documents: yes. Where the documents say it: main:1214, main:1233
- Rests on: Collision resistance of SHA3-256 is assumed; the KEK is an ideal function of its five inputs.; The second public key is a constant with no matching private key, so the test is a transplant test and not a full second party.; The control for each claim is a reachability query: it shows that the sender under the other algorithm, and the sender to the other public key, do run, so the absence of acceptance is not vacuous. No query can show the recipient accepting the wrong KEK, because the failure is exactly that it never does.
- Open: Only the two algorithm IDs of the composite table are modelled; no other suite can be substituted.
- Next step: Nothing further is needed for the two composite schemes.

C08: One broken component keeps the KEK

When one of the X25519 component fails (the attacker obtains ecdhKeyShare) or the ML-KEM component fails (the attacker obtains mlkemKeyShare), the KEK and the session key remain secret from the attacker.

Breaking one component of the composite KEM does not reveal the KEK or the session key: an attacker who has the X25519 key share still needs the ML-KEM key share of that message, and the other way round. With both shares the KEK is derived at once, which is exactly the control.

- Result: Holds; answers what is asked: yes; usable with caveats
- Value medium: This is the property the hybrid combiner exists for; the proof holds with the caveat that the primitives are idealised.

- Stated in the documents: yes. Where the documents say it: main:1210
- Rests on: Each component is idealised as an unbreakable encapsulation except for the one share the compromise gives away.; The compromise is applied to the key share of a recorded message, which is what holding the corresponding private key gives.
- Open: The strength of ML-KEM and of X25519 themselves is not modelled; only the structure of the combination is.
- Next step: Tamarin+ Agent with the real algebra and primitive assumptions would settle the computational side.

C01: No forward secrecy of past messages

After the session completes, if the recipient's long-term private key leaks, the attacker still cannot obtain the KEK, session key or plaintext of earlier sessions.

Once both of the recipient's long-term private keys leak, every message it already received is readable: the attacker recomputes the X25519 key share from the ECDH private key, decapsulates the ML-KEM ciphertext with the decapsulation key, rebuilds the KEK and unwraps the session key. There is no forward secrecy, which follows from encrypting to a static key.

- Result: Fails (1 attack); answers what is asked: yes; usable with caveats
- Value low: It is inherent to KEM encryption against a static recipient key, and no reader of OpenPGP would expect otherwise.
- Stated in the documents: unclear. Source text it rests on: rfc9580:6277
- Rests on: The compromise gives both private keys at once; a single one is not enough, as C08 shows.; The attacker keeps the recorded packets, which a network attacker does.
- Open: What a leaked private key should still leave protected is not settled by the documents, so no stronger reading is answered.
- Next step: An interactive or ratcheted variant would be needed before forward secrecy could be asked.

S01: AEAD nonce not reused

Under the same message key, the AEAD nonce (the combination of the IV and the chunk index) is never reused.

Within one message the nonce written into each chunk never repeats under the message key: the message key and the IV come from a fresh salt, and the chunk index advances with every chunk. The bounded model shows this over three chunks and shows that it fails as soon as the index is not advanced.

- Result: Holds; answers what is asked: partly; usable with caveats
- Value low: It confirms a construction the documents prescribe and points at the one thing it depends on, the advancing index.
- Stated in the documents: yes. Where the documents say it: rfc9580:6733
- Rests on: Three chunks, not an arbitrary number; the counter is unrolled.; The salt is assumed fresh, which the model represents by a fresh name; the quality of the generator is not modelled.; The control on this model only shows that a chunk is sealed at all; the control that makes the property fail, a sender that does not advance the index, is the one in the bounded model.; The wrap of the 64-bit index is recorded as an unconfirmed candidate in the state review.
- Open: The counter is unrolled to three chunks, so nothing is said about very long messages beyond the wrap observation.
- Next step: A counting argument outside the prover would settle whether a 64-bit index can be exhausted.

K01.8: Session key and plaintext secret

The attacker cannot obtain the session key and the message plaintext.

Neither the session key nor the plaintext is available to an attacker who holds only the network. Both are lost once the recipient's two private keys leak, which is the boundary of the claim and what the controls show.

- Result: Holds; answers what is asked: yes; usable with caveats
- Value medium: It answers the user's question about confidentiality directly and with the standard caveats.
- Stated in the documents: yes. Where the documents say it: rfc9580:434, main:1210
- Rests on: The primitives are ideal functions and destructors.; The recipient's long-term key is assumed to stay with the recipient during the session.
- Open: The same limits as K01 apply: primitives idealised, no lengths, no certificates.
- Next step: Nothing further for this question at this level of abstraction.

A08.9: KEK bound to algId, key, ciphertext

The algId, ecdhPublicKey and ecdhCipherText bound to the KEK that the recipient accepts match its own key and the PKESK it received.

The accepted KEK is bound to the algorithm ID, the recipient's ECDH public key and the two ciphertexts it received: the recipient's KEK for the sender's encapsulation is the sender's KEK, and no encapsulation made under the other algorithm or for another ECDH key is accepted.

- Result: Holds; answers what is asked: yes; usable with caveats
- Value medium: It answers the user's binding question with a proof rather than an argument.
- Stated in the documents: yes. Where the documents say it: main:1214, main:1233
- Rests on: The KEK is an ideal, injective function of its five inputs.; The other algorithm ID is exercised with the same key material, which is a sharper test than a full second scheme.; The controls are reachability controls showing that the other sender runs, plus the injective variant of the KEK agreement, which fails.
- Open: Only two algorithm IDs and one recipient key pair are present, so binding to a third suite or to a subkey is not answered.
- Next step: Adding a third algorithm ID and a subkey would extend the binding test.

C08.10: Component-leak resilience with double-leak control

Check whether the KEK and the session key remain secret when the X25519 component leaks, the ML-KEM component leaks, or both leak at once; a single-component leak should hold, and the double-component leak serves as the control.

A break of the X25519 component alone leaves the KEK and the session key secret because the ML-KEM share is still needed, and a break of the ML-KEM component alone leaves them secret because the ECDH share is still needed. With both broken the KEK falls at once, which is the control that shows the property can fail.

- Result: Holds; answers what is asked: yes; usable with caveats
- Value medium: It answers the user's third question, including the double-leak control they asked for.
- Stated in the documents: yes. Where the documents say it: main:1210
- Rests on: Each component is idealised apart from the leaked share.; The control is a falsified query on the same model, not a separate protocol.

- Open: Only one component is ever broken at a time in the proof; the double break is the control, not a separate target.
- Next step: Nothing further at this level of abstraction.

X01: SEIPD plaintext integrity

The recipient never releases plaintext it did not receive from the sender under that session: if it recovers a plaintext m from the v2 SEIPD packet, the sender sealed m for the same session, so no modified or forged ciphertext is accepted as a new plaintext.

The recipient releases no plaintext that the sender did not seal under that session: a modified ciphertext, a wrong key or a wrong nonce yields no plaintext at all. Anyone may encrypt to the recipient's public key, which is not a break of integrity but a consequence of there being no sender authentication.

- Result: Holds; answers what is asked: yes; usable with caveats
- Value medium: Malformed or modified ciphertext being rejected is the property RFC 9580 13.7 requires, and the model reproduces it.
- Stated in the documents: yes. Where the documents say it: rfc9580:6516, rfc9580:6520
- Rests on: The AEAD is an ideal authenticated encryption with no forgeries.; The claim is stated against the sender's own encapsulation, because a message the attacker composes is a legitimate new message.; One chunk, no final length tag and no streaming.; The control is a reachability query showing that the recipient does accept a sealed plaintext, not a query in which it accepts a wrong one.
- Open: Truncation of the last chunks and the streaming release rules are not modelled.; A compromised private key lets the attacker compose messages of its own, which the claim does not forbid.
- Next step: A multi-chunk model with the final length tag would cover truncation.

X02: Per-message KEK separation

Two messages never share a KEK: if two encapsulations yield the same KEK they are the same encapsulation, so one message's KEK leaking says nothing about another message.

Two messages never share a KEK: if two encapsulations produce the same KEK, they are the same encapsulation. One message's KEK leaking therefore says nothing about another message.

- Result: Holds; answers what is asked: yes; usable with caveats
- Value low: It follows from fresh randomness per message; it is worth stating because it is what limits the damage of a leaked KEK.
- Stated in the documents: no. Source text it rests on: main:1219
- Rests on: Fresh key shares per message are represented by fresh names.; Only two messages are compared; the argument does not count sessions.; The control only shows that a KEK is produced at all; no variant of this model makes two messages share one.
- Open: The separation is shown for KEKs, not for the message keys of a session-key-reuse reply.
- Next step: A model of the session-key-reuse feature would extend it to message keys under one session key.

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_kek	checked		4 proved
K02	proved		k02	checked		1 proved
A03	proved		a03	checked		1 proved
A08	proved		a08_alg	checked		2 proved
C08	proved		c08_pq	checked		2 proved
C01	none				c01 (holds_in_source)	1 attack
S01	proved		s01	checked		1 proved, 1 bounded
K01.8	proved		k01_sk	checked		2 proved
A08.9	proved		a08_alg	checked		2 proved
C08.10	proved		c08_pq	checked		2 proved
X01	proved		x01	checked		1 proved
X02	proved		x02	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
K01	The attacker on the network cannot derive the KEK of a message from its PKESK and the recipient's public key.	`network`	proved	k01_kek	checked
K01	The attacker on the network cannot derive the session key that the sender wrapped under the KEK.	`network`	proved	k01_sk	checked
K01	The attacker on the network cannot derive the recipient's ECDH private key.	`network`	proved	k01_priv	checked
K01	The attacker on the network cannot derive the recipient's ML-KEM decapsulation key.	`network`	proved	k01_mlk	checked
K02	The attacker on the network cannot obtain the plaintext of the message from the PKESK and the SEIPD packet.	`network`	proved	k02	checked
A03	The KEK the recipient accepts for the sender's encapsulation (same ecdhPublicKey, ecdhCipherText and mlkemCipherText) is the KEK the sender computed.	`network`	proved	a03	checked
A08	No KEK computed under the other algorithm ID (ML-KEM-1024 + X448, algId 36) is accepted by this recipient.	`network`	proved	a08_alg	checked

target	claim	attacker	answer	query	control
A08	No KEK computed for another party's ECDH public key is accepted by this recipient, even with the same ML-KEM key and the same ciphertexts.	`network`	proved	a08_key	checked
C08	With the X25519 component broken (the attacker holds the ECDH key share of the recorded message) the KEK and session key stay secret: they leak only if the ML-KEM key share of that message was also revealed.	`ecdh_component_broken`	proved	c08_pq	checked
C08	With the ML-KEM component broken (the attacker holds the ML-KEM key share of the recorded message) the KEK and session key stay secret: they leak only if the ECDH key share of that message was also revealed.	`mlkem_component_broken`	proved	c08_cl	checked
C01	The session key and plaintext of a message already received stay hidden from an attacker who obtains the recipient's long-term private keys after the run. This fails: the two key shares are functions of the recipient's static keys, so the attacker recomputes X = X25519(r, V) and mlkemKeyShare = ML-KEM.Decaps(mlkemCipherText, sk), then the KEK, unwraps C and derives the message key.	`ltk_leak_after_run`	attack (holds_in_source)	c01	
S01	One message key never meets the same AEAD nonce twice: across the three unrolled chunks of one message the nonce (IV || chunk index) repeats only when the chunk index is not advanced.	`network`	bounded (the changing value unrolled to 3 steps)	s01_chunks	checked
S01	One message key is never used with the same AEAD nonce twice: two v2 SEIPD encryptions that share a message key and a nonce are the same encryption.	`network`	proved	s01	checked
K01.8	The attacker on the network cannot obtain the session key of the message.	`network`	proved	k01_sk	checked
K01.8	The attacker on the network cannot obtain the message plaintext.	`network`	proved	k02	checked
A08.9	The algorithm ID in the PKESK is bound into the KEK: only an encapsulation made with the recipient's own algorithm ID yields the accepted KEK.	`network`	proved	a08_alg	checked
A08.9	The accepted KEK is bound to the recipient's own ECDH public key and to the received ecdhCipherText and mlkemCipherText: it is the sender's KEK for exactly those values.	`network`	proved	a03	checked
C08.10	A break of the X25519 component alone does not yield the KEK: the attacker also needs the ML-KEM key share of that message.	`ecdh_component_broken`	proved	c08_pq	checked
C08.10	A break of the ML-KEM component alone does not yield the KEK: the attacker also needs the ECDH key share of that message. With both components broken the KEK is derived, which is the control that the property can fail.	`mlkem_component_broken`	proved	c08_cl	checked

target	claim	attacker	answer	query	control
X01	The recipient releases only plaintext the sender sealed for that session: no modified or forged SEIPD ciphertext is accepted as a new plaintext.	`network`	proved	x01	checked
X02	Two messages never share a KEK: two encapsulations with the same KEK are the same encapsulation.	`network`	proved	x02	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-20261114T212745, 2026-11-14. Status: finished. Model: deepseek-v4-1-flash-261017. Steps: 61, prover 6 s, wall 14 min, cost 0.134 USD.

ProVerif: 106 proofs run, 105 decided, 1 without an answer, 4 s, 30 answers recorded.

What was analysed

Protocol	OpenPGP composite KEM encrypted message
Version	rfc9980
Analysed	Encryption/decryption of a single message to a single recipient using the PKESK and v2 SEIPD of the fixed ML-KEM-768+X25519 composite KEM; the trusted recipient public key is fixed in advance. This is not an interactive handshake, and no sender authentication is claimed on the basis of encryption; signatures, key distribution, compression and multiple recipients are excluded.
Not analysed	The symbolic model does not prove the computational security of the algorithms or their resistance to quantum attack; attack traces hold only under explicit assumptions and are not directly equivalent to flaws in the specification. Every property must have a reachability/positive control.
Source text	251 numbered passages (40193 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	Algorithm Combinations
main	3.1	Composite KEMs
main	3.2	Composite Signatures
main	3.3	Multiple Signatures
main	3.4	ECC Requirements
main	3.5	Key Version Binding
main	4	Composite KEM Schemes
main	4.1	Building Blocks
main	4.1.1	ECDH KEM
main	4.1.1.1	X25519-KEM
main	4.1.1.2	X448-KEM
main	4.1.2	ML-KEM
main	4.2	Composite Encryption Schemes with ML-KEM
main	4.2.1	Key Combiner
main	4.2.2	Key Generation Procedure
main	4.2.3	Encryption Procedure

Document	Section	Title
main	4.2.4	Decryption Procedure
main	4.3	Packet Specifications
main	4.3.1	Public Key Encrypted Session Key Packets (Packet Type ID 1)
main	4.3.2	Key Material Packets
main	4.3.2.1	Public Key Packets (Packet Type IDs 6 and 14)
main	4.3.2.2	Secret Key Packets (Packet Type IDs 5 and 7)
main	7	Notes on Algorithms
main	7.1	Symmetric Algorithms for SEIPD Packets
main	7.2	Hash Algorithms for Key Binding Signatures
rfc9580	5.1	Public Key Encrypted Session Key Packet (Type ID 1)
rfc9580	5.1.1	Version 3 Public Key Encrypted Session Key Packet Format
rfc9580	5.1.2	Version 6 Public Key Encrypted Session Key Packet Format
rfc9580	5.1.3	Algorithm-Specific Fields for RSA Encryption
rfc9580	5.1.4	Algorithm-Specific Fields for Elgamal Encryption
rfc9580	5.1.5	Algorithm-Specific Fields for ECDH Encryption
rfc9580	5.1.6	Algorithm-Specific Fields for X25519 Encryption
rfc9580	5.1.7	Algorithm-Specific Fields for X448 Encryption
rfc9580	5.1.8	Notes on PKESK
rfc9580	5.13	Symmetrically Encrypted and Integrity Protected Data Packet (Type
rfc9580	5.13.1	Version 1 Symmetrically Encrypted and Integrity Protected Data
rfc9580	5.13.2	Version 2 Symmetrically Encrypted and Integrity Protected Data
rfc9580	5.13.3	EAX Mode
rfc9580	5.13.4	OCB Mode
rfc9580	5.13.5	GCM Mode

The purpose of the protocol as the source step located it in the documents: Encryption/decryption of a single message to a single recipient using the PKESK and v2 SEIPD of the fixed ML-KEM-768+X25519 composite KEM; the trusted recipient public key is fixed in advance. This is not an interactive handshake, and no sender authentication is claimed on the basis of encryption; signatures, key distribution, compression and multiple recipients are excluded.

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	Secrecy of KEK and private keys	The sender's KEK, the session key, and the recipient's ECDH and ML-KEM private keys, all kept from the attacker on the network.	Section 4.2 has both key shares fed into a key combiner to produce the single shared secret that wraps the session key (s0066, s0067, s0072), and Section 9.2 argues the combiner is IND-CCA2-secure, i.e. the KEK is not recoverable; the ECDH and ML-KEM private keys are assumed held by the recipient.	main:1210, s0003, s0066, s0067, s0072, s0087
K02	Confidentiality of the message plaintext	The plaintext of the single message, kept from the attacker on the network, who may hold every packet.	The session key wraps the content encryption; v2 SEIPD encrypts the data with an AEAD construction under a message key derived from the session key and the salt (s0204, s0231, s0233).	rfc9580:434, s0004, s0067, s0202, s0204, s0231
A03	Agreement on the accepted KEK	The recipient: when it unwraps a session key, the KEK it accepted was computed by the sender for the recipient's own public key, that algorithm ID and those encapsulation ciphertexts.	The decryption procedure recomputes the combiner from the received ciphertexts and the recipient's own public key (s0093, s0096, s0099) and aborts if the unwrap integrity check fails (s0100); the documents state the construction but not this goal as such.	s0080, s0082, s0086, s0093, s0099, s0100
A08	KEK bound to algorithm and key	The KEK accepted for one recipient public key and algorithm ID: no encapsulation made for another algorithm, another ECDH public key or another ECDH ciphertext yields it.	Section 9.2.1 states that the algorithm ID binds the derived KEK to the chosen algorithm (main:1233) and that including the ECC public key in the combiner accounts for multi-target attacks (main:1214); RFC 7748 warns that a public key used without being bound into the key derivation can lead to vulnerabilities (rfc7748:820).	main:1214, main:1233, rfc7748:820, s0066, s0072, s0093
C08	One broken component keeps the KEK	The KEK and session key of a message, kept from an attacker who breaks one of the two components of the composite KEM (the X25519 component or the ML-KEM component) but not the other.	Section 9.2: the combiner is IND-CCA2-secure if ML-KEM is IND-CCA2-secure or the Strong Diffie-Hellman problem holds, so the failure of one component must not be enough (main:1210).	main:1210, s0003, s0070, s0072
C01	No forward secrecy of past messages	The session key and plaintext of a message already received, kept from an attacker who obtains the recipient's long-term private keys after the run.	The documents say nothing about forward secrecy; they only assume the private key portion is controlled and secured by its party (rfc9580:6277). The target asks the counterfactual for that key, and the construction uses the recipient's static ECDH key and static ML-KEM decapsulation key as the only inputs to the two key shares (s0064, s0084, s0085, s0097, s0098).	rfc9580:6277, s0032, s0064, s0084, s0085, s0097, s0098
S01	AEAD nonce not reused	The AEAD nonce of the v2 SEIPD packet, never used twice under one message key.	The 32-octet salt MUST be securely generated (s0230); the message key and the IV are both derived by HKDF from the session key and the salt, and the nonce is that IV with the chunk index appended (s0233, s0242); RFC 9580 lists salts and nonces as the publicly visible random fields of OpenPGP (rfc9580:6733).	rfc9580:6733, s0230, s0233, s0238, s0242
K01.8	Session key and plaintext secret	The session key and the message plaintext of one message, kept from the attacker on the network.	The session key is generated as specified in RFC 9580 and wrapped under the KEK (s0067); the recipient outputs it only after the key-wrap integrity check (s0100, s0101).	main:1210, rfc9580:434, s0067, s0100, s0202

ID	Name	About	What the documents say	Where
A08.9	KEK bound to algId, key, ciphertext	The KEK accepted for one recipient public key and algorithm ID, and the ECDH ciphertext it is bound to.	Section 9.2.1 states the algorithm ID binds the KEK to the algorithm and the ECC public key is an input to the combiner (main:1214, main:1233).	main:1214, main:1233, s0066, s0072, s0093
C08.10	Component-leak resilience with double-leak control	The KEK and session key, kept from an attacker who breaks only the X25519 component or only the ML-KEM component, with both components broken as the control.	Section 9.2: IND-CCA2 security of the combiner holds if ML-KEM is IND-CCA2-secure or Strong DH holds, i.e. one broken component is survivable (main:1210).	main:1210, s0003, s0070, s0072
X01	SEIPD plaintext integrity	The recipient and the plaintext of the v2 SEIPD packet; message integrity, not sender authentication.	RFC 9580 13.7 requires that a failed AEAD tag means the implementation MUST NOT parse nor release decrypted data, because releasing before confirming integrity can leak it (rfc9580:6516, rfc9580:6520), and lists a v2 SEIPD packet whose chunk tag fails as malleable ciphertext (rfc9580:6548); the JKS02 decryption-oracle attack is recalled as a form of ciphertext malleability (rfc9580:6330).	rfc9580:6516, rfc9580:6520, rfc9580:6548, rfc9580:6330, s0204, s0231, s0238
X02	Per-message KEK separation	The KEK of a single message, kept separate from the KEK of every other message.	Section 9.2.1 gives domSep and its length octet so that keying material is used for one purpose only (main:1219), and both key shares are freshly generated per message (s0084, s0085).	main:1219, s0072, s0084, s0085

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

The security statements of the documents, paragraph by paragraph

The host listed the 110 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: 17. Not taken: 93. 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
P001 to P002	main:1163-1187	Outside the scope	Signature stripping and SUF-CMA of composite signatures; signatures are excluded by the task.
P003	main:1191-1204	Outside the scope	Cross-protocol reuse of signatures; signatures are excluded by the task.
P004	main:1208-1212	C08, C08.10	
P005	main:1214-1215	A08	
P006	main:1219-1231	X02	
P007	main:1233-1237	A08, A08.9	
P008 to P009	main:1241-1257	Outside the scope	Hedged ML-DSA and SLH-DSA, and digest sizes for signatures.
P010 to P011	main:1261-1272	The prover cannot be asked	Which symmetric algorithm to prefer is the strength of a primitive, not a trace property.
P012	main:1276-1281	K01, `ecdh_component_broken`, `mlkem_component_broken`	

Paragraphs	Where	Taken up as	Why not
P013	main:1285-1287	The prover cannot be asked	The quality of the random generator cannot be modelled; the model treats fresh names as unguessable.
P014 to P016	rfc7748:783-818	The prover cannot be asked	The concrete security level of curve25519/curve448, and small-order or non-contributory inputs to X25519: the symbolic model has no group order and no cofactor, so it cannot represent them.
P017	rfc7748:820-825	A08, A08.9	
P018	rfc7748:827-837	The prover cannot be asked	Observable differences between curve implementations (twist points, non-canonical values) are implementation behaviour.
P019	rfc9580:6268-6274	No goal stated	Advice to watch the literature for broken algorithms.
P020	rfc9580:6276-6278	K01	
P021	rfc9580:6280-6282	The prover cannot be asked	Weakness of MD5 and SHA-1 is the strength of a hash primitive.
P022	rfc9580:6284-6289	Outside the scope	Separating encryption keys from signing keys; signatures are excluded by the task.
P023	rfc9580:6291-6295	Outside the scope	DSA signature hash substitution.
P024	rfc9580:6297-6304	The prover cannot be asked	The weakest element of a message being strong enough is a statement about primitive strength.
P025 to P027	rfc9580:6306-6322	Outside the scope	Hash collisions and substituted hash algorithms in signatures, and preferred signing algorithms.
P028	rfc9580:6324-6328	The prover cannot be asked	How well analysed a symmetric algorithm is cannot be asked of the prover.
P029	rfc9580:6330-6339	X01	
P030 to P032	rfc9580:6343-6361	Outside the scope	SHA-1 collision detection in signatures.
P033 to P035	rfc9580:6365-6390	Outside the scope	Salted signatures and their advantages.
P036	rfc9580:6394-6404	The prover cannot be asked	Elliptic-curve side channels are an implementation property.
P037 to P040	rfc9580:6408-6431	Outside the scope	The v1 SEIPD/SED quick check and v4 SKESK passphrases; the modelled flow is a v6 PKESK with a v2 SEIPD packet, where s0147 and s0155 exclude that check.
P041	rfc9580:6433-6435	X01	
P042 to P044	rfc9580:6439-6461	Outside the scope	PKCS#1 padding errors in RSA- and ElGamal-encrypted PKESK; those algorithms are outside the composite-KEM scope.
P045 to P051	rfc9580:6465-6498	No goal stated	Fingerprint usability for humans.
P052 to P054	rfc9580:6502-6521	X01	

Paragraphs	Where	Taken up as	Why not
P055	rfc9580:6523-6526	Outside the scope	Releasing cleartext from historic non-AEAD packets; the modelled flow is AEAD only.
P056	rfc9580:6528-6533	Outside the scope	Streaming release of fully authenticated chunks before a truncated tail; the model has one chunk and no streaming.
P057	rfc9580:6535-6536	X01	
P058 to P060	rfc9580:6538-6545	Outside the scope	SED packets, compressed-data failures and v1 SEIPD MDC failures; none of these is in the modelled flow.
P061	rfc9580:6547-6549	X01	
P062 to P065	rfc9580:6551-6560	Outside the scope	Integrity of encrypted secret-key material at rest.
P066	rfc9580:6562-6565	No goal stated	Advice on which container format to select.
P067	rfc9580:6567-6567	Outside the scope	The deprecated SED packet.
P068 to P069	rfc9580:6569-6577	No goal stated	Recommendation to encrypt with a v2 SEIPD packet when the recipients support it; it states a choice of format, not a property of this protocol.
P070	rfc9580:6579-6580	Outside the scope	Falling back to v1 SEIPD when one of several recipients does not support v2; several recipients are excluded by the task.
P071	rfc9580:6582-6590	Outside the scope	AEAD for passphrase-protected secret-key material, and corruption of CFB-encrypted private keys.
P072 to P073	rfc9580:6592-6595	No goal stated	Migration advice to implementers and users.
P074	rfc9580:6599-6609	S01	
P075 to P083	rfc9580:6611-6677	Outside the scope	The session-key-reuse reply feature: subsets of recipients, adding recipients, drafts, and deployment guidance. The task models one message to one recipient with a fresh session key.
P084 to P093	rfc9580:6681-6717	Outside the scope	Escrowed revocation signatures.
P094 to P095	rfc9580:6721-6731	The prover cannot be asked	The requirement for a CSPRNG and the levels of visibility of random data; the model treats every fresh name as unguessable.
P096	rfc9580:6733-6734	S01	
P097	rfc9580:6736-6737	K01, K01.8	
P098 to P100	rfc9580:6739-6750	The prover cannot be asked	What a broken CSPRNG allows, and separate generators per visibility level.
P101 to P104	rfc9580:6754-6773	The prover cannot be asked	Traffic analysis through message length; the prover has no notion of length or of the equality of two ciphertext sizes.
P105	rfc9580:6777-6782	Outside the scope	Surreptitious forwarding of signatures.

Paragraphs	Where	Taken up as	Why not
P106 to P108	rfc9580:6786-6808	Outside the scope	Hashed and unhashed signature subpackets.
P109 to P110	rfc9580:6812-6820	Outside the scope	Malicious compressed data; compression is excluded by the task.

The catalogue of properties, entry by entry

The properties were drafted from a catalogue of security and privacy properties. The draft left out 35 of its entries, and the analyst went through each of them: examined here as a property, or left out with the reason. Examined: 7. Left out: 28. 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
K03	Strong secrecy / indistinguishability of the secret value	K01, K02	
K04	Separation of key usage, direction and session	X02	
D01	Message integrity and authenticity of data origin	X01	
A01	Peer aliveness	Not applicable	One-way message encryption: the recipient decrypts a stored packet and has no notion of the peer having run a protocol.
A02	Weak authentication of agreement	Not applicable	The task states that encryption does not claim sender authentication, so no agreement on the peer's identity can be asked.
A04	Injectivity agreement	Not applicable	The same ciphertext may legitimately be decrypted again; the documents claim no replay protection and no once-only acceptance.
A05	Implicit key authentication	K01	
A06	Explicit key confirmation	Not applicable	No key confirmation: the sender does not wait for an answer.
A07	Identity–credential–key binding	Outside the scope	Certificates, key distribution and the recipient-identifying fields of the PKESK are excluded by the task; the recipient public key is fixed and trusted.
R01	Authentication / challenge freshness	Not applicable	No challenge, timestamp, ticket or freshness check.
R02	Message-level replay resistance	Not applicable	No message-level replay protection is claimed; re-decrypting a stored message is not a protocol violation.
R03	At most once per business operation	Not applicable	No billable or resource-consuming operation with at-most-once semantics.
R04	Message acceptance order / window safety	Not applicable	A single message with no sequence number; the chunk index of the AEAD nonce is covered by S01.
R05	Anti-downgrade / security policy preservation	A08, A08.9	
C02	Resistance to key-compromise impersonation (KCI)	Not applicable	One-way encryption has no peer authentication, so key-compromise impersonation does not apply.
C03	Known-key security / compromise isolation	X02	
C04	Resilience to ephemeral secret / partial state compromise	C08, C08.10	
C05	Post-compromise security (PCS)	Not applicable	No ratchet or rekeying mechanism, so there is no recovery after a compromise.

Entry	Property of the catalogue	Examined as	Why not
C06	Augmented authentication after compromise of the enrolment credential	Not applicable	Not a PAKE; there is no registration record.
C07	Resistance to offline dictionary attacks	Not applicable	No password or other low-entropy secret is involved.
Z01	Correct enforcement of authorization	Not applicable	No authorization decision on a protected operation.
Z02	No expansion of delegated authority	Not applicable	No delegation or proxy mechanism.
Z03	Binding of authorization to identity / session / operation	Not applicable	No authorization token or session binding.
Z04	Effect of revocation and expiry	Not applicable	No revocation or expiry inside the modelled flow.
Z05	Security isolation across tenants / services	Not applicable	No tenants, services or security domains.
S02	Safety of retransmission / key reinstallation	Not applicable	No rekeying or retransmission state machine.
S03	Resistance to rollback and state monotonicity	Not applicable	No version or counter that can be rolled back.
S04	Consistency of the security context across phases	Not applicable	No multi-phase lifecycle.
S05	Termination and invalidation of old state	Not applicable	No session teardown or old state that could be revived.
P01	Identity anonymity	Outside the scope	The anonymous-recipient PKESK of s0200 is an optional mode of the recipient-identifying fields, which the task leaves out; the modelled flow carries no recipient identifier at all.
P02	Session unlinkability	Not applicable	No ephemeral identity or per-session pseudonym in the flow.
P03	Hiding of user / credential existence	Not applicable	No user or credential lookup.
V01	Conditional progress / error recovery	The prover cannot be asked	Progress and error recovery cannot be asked of the prover.
V02	No unintended deadlock	The prover cannot be asked	Deadlock freedom is not a trace property the prover can be asked.
V03	Limited resource abuse / resistance to amplification	The prover cannot be asked	Resource consumption and amplification cannot be counted beyond events.

The parties and the protected values, one by one

The host listed the 21 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: 6. Without: 15. 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	Sender	party	Nothing gained	The sender holds only the plaintext it chose and a session key it generated; no target says anything about the sender's honesty, and the recipient is promised nothing about who sent the message.
S02	Recipient	party	`ltk_leak_after_run`	
S03	Attacker	party	Nothing gained	This is the attacker of the network itself, which is already present; it holds nothing else.

ID	Party or value	Kind	Attacker	Why none
S04	KEK	value	Everything lost by definition	Whoever holds the KEK unwraps the session key of that message by definition; no property of this protocol is still askable for that message under that attacker.
S05	Session key sessionKey	value	Everything lost by definition	The holder of the session key derives the message key and decrypts the message.
S06	Recipient ECDH private key	value	`ecdh_component_broken`	
S07	Recipient ML-KEM private key	value	`mlkem_component_broken`	
S08	Message plaintext	value	Everything lost by definition	Holding the plaintext is the loss of plaintext confidentiality itself.
S09	ecdhPublicKey	value	Nothing gained	The recipient's ECDH public key is handed to the sender before the message; it is public.
S10	algId	value	Nothing gained	The algorithm ID travels in the clear in the PKESK.
S11	ecdhCipherText	value	Nothing gained	The ECDH ciphertext is on the wire.
S12	mlkemCipherText	value	Nothing gained	The ML-KEM ciphertext is on the wire.
S13	Session key	value	Everything lost by definition	The same value as S05: the holder of the session key decrypts the message.
S14	ecdhKeyShare	value	`ecdh_component_broken`	
S15	mlkemKeyShare	value	`mlkem_component_broken`	
S16	Historical session key	value	Everything lost by definition	Holding a past session key is the loss of that message's confidentiality.
S17	Historical plaintext	value	Everything lost by definition	Holding a past plaintext is the loss of plaintext confidentiality itself.
S18	Recipient long-term private key	value	`ltk_leak_after_run`	
S19	Message key	value	Everything lost by definition	The holder of the message key decrypts that message.
S20	AEAD nonce	value	Nothing gained	The nonce is derived from the salt shipped in the clear in the SEIPD packet and the chunk index.
S21	Salt	value	Nothing gained	The 32-octet salt is a field of the v2 SEIPD packet and is public.

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		

Attacker	Has	Basis	Where
`ecdh_component_broken`	The ECDH key share of a recorded message, equivalently the recipient's X25519 private key r: with r and the recorded ecdhCipherText V the attacker computes X = X25519(r, V). The ML-KEM component stays intact.	Section 9.2 asks the combiner to stay IND-CCA2-secure when only one of its two inputs is protected (main:1210); the target C08 and the exposure of the recipient's ECDH private key (S06) and of ecdhKeyShare (S14) are answered under it.	main:1210, s0003, s0072, s0037
`mlkem_component_broken`	The ML-KEM key share of a recorded message, equivalently the ML-KEM decapsulation key: with it the attacker runs ML-KEM.Decaps on the recorded mlkemCipherText. The X25519 component stays intact.	Same requirement that one failing component be survivable (main:1210); the exposure of the recipient's ML-KEM private key (S07) and of mlkemKeyShare (S15) is read as this attacker.	main:1210, s0003, s0053, s0098
`ltk_leak_after_run`	Both private keys of the recipient, the ECDH secret key r and the ML-KEM decapsulation key, obtained after the message was received and decrypted.	The documents assume the private key portion is controlled and secured by its party (rfc9580:6277) and say nothing about a later compromise; the target C01 asks that counterfactual, and the exposure of the recipient's long-term private key (S18) and of the recipient party (S02) is read as this attacker.	rfc9580:6277, s0094, s0097, s0098

ID	`network`	`ecdh_component_broken`	`mlkem_component_broken`	`ltk_leak_after_run`
K01	Holds			
K02	Holds			
A03	Holds			
A08	Holds			
C08		Holds	Holds	
C01				Fails
S01	Holds			
K01.8	Holds			
A08.9	Holds			
C08.10		Holds	Holds	
X01	Holds			
X02	Holds			

Source and targets

Source: 251 units, 40193 bytes, frozen in source-delivery-01.json. Targets: 7 approved (K01, K02, A03, A08, C08, C01, S01); 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-3afdbf21efd53284a5be01418a83

This step as it reports itself: 1 requests (1 failed), 0.000 USD, 34295 tokens in and 4078 out, 27 s.

Notes on the prover and the model

- S01: answered in ProVerif, and the analyst names changing_state for this target

A ProVerif model that simplifies algebra (xor, dh_algebra, other_algebra) gives the attacker less than the real operation: a proof there is a proof against a weaker attacker, an attack there is an attack. A ProVerif model that leaves out a check on a value that changes (changing_state) lets parties accept more: a proof there holds, an attack there may be nothing but the missing check. Tamarin+ Agent models these as they are.

State review by the analyst

This part is the analyst's own reading of the source, not an answer of the prover. "yes" under shown by ProVerif means that ProVerif found a trace with these steps in the unrolled model, so the steps are possible; that they break a target is said only by an attack in the table of answers, and what else they cost is the analyst's reading. "nothing found" says which patterns were looked at and proves nothing.

value	kept by	changes when	checked when	refs
the 64-bit AEAD chunk index of a v2 SEIPD message	The sender (the encrypting implementation), for the duration of one message.	Advanced by one for every chunk of the same v2 SEIPD message; the first chunk has index zero.	The recipient rebuilds the nonce from the chunk's position in the stream (IV from HKDF, rightmost 64 bits the index) and the AEAD tag verifies it; the final tag covers the total number of plaintext octets.	s0242, s0238, s0236
the 32-octet SEIPD salt	The sender, per message; the recipient reads it from the packet.	Freshly generated for every message; a reply that reuses the session key uses a different salt to obtain a new message key.	Not compared by anyone; it is the HKDF salt that derives the message key and the IV, so a different salt gives a different message key.	s0230, s0233, s0234

pattern	result	target	shown by ProVerif	steps
replay	nothing found			A chunk replayed at the position of an earlier chunk gets the nonce of its position, not of its original index, so its tag fails; the bounded model accepts only the chunk the sender sealed at index 0.
repeat_one_time_action	nothing found			The chunk index is a per-message local counter; no key is installed and no counter can be set back by a repeated message. With three chunks the bounded model shows three distinct nonces.
desynchronisation	nothing found			The index is not shared state: the recipient derives it from the position of the chunk in the stream, so no two parties can hold different values of it.
update_before_check	nothing found			The sender advances the index after sealing the chunk and verifies nothing, so no value changes before a check.
no_rollback_on_failure	nothing found			An aborted encryption leaves no accepted message: the recipient accepts no chunk whose tag fails, and the salt is fresh for the next message. Streaming emission of chunks before a later failure is outside the model.
race_between_sessions	nothing found			The salt and the session key are fresh per message, and X02 shows that two messages never share a KEK, so two concurrent encryptions share neither a message key nor an index.
distinguishable_failure	nothing found			The model has no failure answer: a tag failure produces no output at all, so there is no distinguishable reply for an attacker to compare.

pattern	result	target	shown by ProVerif	steps
wrap_or_window	candidate		no: not tried with ProVerif	The sender starts the chunk index at zero and advances it by one per chunk; s0242 fixes it as a 64-bit big-endian value and states no rule for what happens when it runs out. / A message with more than 2^64 chunks brings the index back to a value already used under the same message key, because the message key and IV are derived once per message from the session key and the salt (s0233). / Two chunks then share a nonce under one message key; the bounded model shows exactly this failure when the index is not advanced.
other	nothing found			Dropping trailing chunks is detected by the final tag over the total octet count, and the source permits releasing already-authenticated chunks with an error (rfc9580:6531); my model has one chunk and no final tag, so it says nothing about truncation.

What the analyst named for targets:

step	targets	features	reason
52	K01, K02, A03, A08, C08, C01, K01.8, A08.9, C08.10, X01, X02	plain_primitives	These answers rest only on hashing (SHA3-256 as the combiner), an AEAD-style symmetric encryption with its key-unwrap, a KEM decapsulation and Diffie-Hellman used as a plain key agreement (the single law g^xy = g^yx, no other algebraic law). The model therefore gives the attacker no more and no less than the standard symbolic treatment of those primitives; the computational strength of SHA3-256, ML-KEM and AES-KeyWrap is not modelled.
52	S01	changing_state	The AEAD nonce is the HKDF-derived IV followed by a 64-bit chunk index that the sender starts at zero and advances by one per chunk (s0242); whether the nonce repeats under one message key therefore depends on that changing counter. model.pv has a single chunk, so the answer for this target is completed by bounded.pv, where the sender is unrolled to three chunks and the index is compared and advanced as the source states.

The analyst's closing note

Model of RFC 9980's ML-KEM-768 + X25519 composite KEM PKESK (v6) with a v2 SEIPD message, single recipient, single message: 12 targets, 11 proved, 1 attack, all with controls (model.pv, plus bounded.pv for the one target that rests on the changing chunk index).

Established: the KEK, the session key, the plaintext and the recipient's two private keys stay out of reach of the network attacker (K01, K02, K01.8); the recipient's KEK is the sender's KEK for its own public key, algorithm ID and the received ciphertexts, and neither an encapsulation under the other composite algorithm ID nor one made for another party's ECDH key is ever accepted (A03, A08, A08.9); breaking one component of the composite KEM gives nothing, both broken gives the KEK (C08, C08.10); the recipient releases no plaintext that was not sealed for that session (X01); two messages never share a KEK (X02); within one message the AEAD nonce never repeats under a message key, in the three-chunk unrolling and only while the chunk index is advanced (S01, bounded, control s01_bug).

Fails: C01, forward secrecy. With both long-term private keys the attacker recomputes X = X25519(r, V), decapsulates the ML-KEM ciphertext, rebuilds the KEK, unwraps the session key and derives the message key, so every recorded message is readable. This is inherent to encrypting to a static key; the documents say nothing about forward secrecy, so it is reported as a low-value finding with unclear novelty rather than as news.

Blocks and limits: SHA3-256, AES-KeyWrap, ML-KEM and the AEAD are idealised; Diffie-Hellman carries only g^xy = g^yx, so the RFC 7748 warnings about small-order points and equivalent public keys cannot be tested here (Tamarin+ Agent would settle them). Lengths, the v3 PKESK symmetric-algorithm length check, certificates,

recipient anonymity, multi-recipient messages, truncation and streaming are outside the model. The state review found nothing exploitable in the chunk index; its 64-bit wrap is recorded as an unconfirmed candidate needing 2^70 octets.