Common Authentication Technology Next Generation
Internet-Draft
Intended status: Standards Track
Expires: 3 December 2026

A. Bokovoy
J. Rische
Red Hat, Inc.
N. Williams
Cryptonector

1 June 2026

Post-quantum Key Encapsulation with ML-KEM in Public Key Cryptography for Initial Authentication in Kerberos (PKINIT)
draft-bokovoy-kitten-pkinit-pqc-00

Abstract

This document specifies extensions to the Kerberos PKINIT pre-authentication mechanism [RFC4556] [RFC8636] to support post-quantum key establishment using the Module-Lattice-Based Key-Encapsulation Mechanism (ML-KEM) algorithms defined in [FIPS203].

The extensions define a new kemInfo arm in PA-PK-AS-REP, a KDCKEMInfo structure signed by the KDC, HKDF-based AS reply key derivation (HKDF-SHA-512 for ML-KEM), downgrade-prevention rules, and a PAChecksum2 extension providing checksum algorithm agility in PKAuthenticator. The KEM path framework supports multiple KEM algorithms including ML-KEM, composite ML-KEM algorithms, and future KEM standards.

Discussion Venues

This note is to be removed before publishing as an RFC.

Discussion of this document takes place on the Common Authentication Technology Next Generation Working Group mailing list (kitten@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/kitten/.

Source for this draft and an issue tracker can be found at https://github.com/abbra/kitten-pkinit-pqc.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Copyright Notice

Table of Contents

1. Introduction

The Kerberos PKINIT pre-authentication mechanism [RFC4556] relies on public-key cryptography for initial authentication. The Diffie-Hellman and RSA paths it defines are vulnerable to a cryptographically relevant quantum computer. [RFC5349] adds Elliptic Curve Diffie-Hellman (ECDH) support and [RFC8636] adds algorithm agility, but neither addresses the quantum threat.

This document defines a new KEM path in PKINIT that uses Key Encapsulation Mechanism (KEM) algorithms, in particular the Module-Lattice-Based Key-Encapsulation Mechanism (ML-KEM) [FIPS203]. Rather than agreeing on a shared secret via a DH exchange, the client generates an ephemeral KEM key pair and sends the encapsulation key in a CMS-signed AuthPack ([RFC5652] Section 5). The KDC encapsulates against it, signs the ciphertext and algorithm selection in KDCKEMInfo, and returns the signed structure. Both parties derive the AS reply key using HKDF ([RFC5869]).

The design preserves the security properties of the RFC 4556 DH path (the client's ephemeral key is authenticated by the client's signing certificate; the KDC's response is authenticated by the KDC's signing certificate) while providing post-quantum forward secrecy through ML-KEM.

This document also defines PAChecksum2, an extension to PKAuthenticator that provides checksum algorithm agility, supplementing the SHA-1-only paChecksum field of RFC 4556 for new deployments.

2. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

3. Protocol Overview

The KEM path is activated when AuthPack.clientPublicValue contains an ML-KEM or composite ML-KEM OID. The exchange proceeds as follows:

1. The client generates an ephemeral KEM key pair, places the encapsulation key in AuthPack.clientPublicValue, and sends a signed AuthPack in PA-PK-AS-REQ.

2. The KDC verifies the AuthPack signature, encapsulates against the client's encapsulation key to obtain a shared secret ss and ciphertext kemct, signs them in a KDCKEMInfo structure, and returns PA-PK-AS-REP.kemInfo.

3. The client verifies the KDC signature, decapsulates to recover ss, and both parties derive the AS reply key using HKDF-SHA-512 over PkinitKEMSuppPubInfo.

No DH exchange takes place. The shared secret is established entirely through one-sided encapsulation; freshness is provided by the per-request ephemeral key pair and the echoed nonce.

4. KEM Algorithm Interface

This specification uses the following generic KEM algorithm interface:

Encap(ek) → (ss, ct): Takes an encapsulation key ek and returns a shared secret ss and ciphertext ct.

Decap(dk, ct) → ss: Takes a decapsulation key dk and ciphertext ct and returns the shared secret ss.

For ML-KEM ([FIPS203]), these correspond to: - Encap(ek) = ML-KEM.Encaps(ek) (FIPS 203 Algorithm 17) - Decap(dk, ct) = ML-KEM.Decaps(dk, ct) (FIPS 203 Algorithm 18)

The notation Encap() and Decap() is used throughout this document to describe the generic KEM path protocol. When implementing ML-KEM specifically, use the FIPS 203 algorithms. Future specifications extending this framework to other KEM algorithms MUST define their mapping to this interface.

5. Algorithm Identifier Encoding

The parameters field MUST be absent from all algorithm identifiers used in this specification. The source of this requirement differs by algorithm family:

(a) ML-KEM and ML-DSA identifiers ([RFC9935], [RFC9881]): the algorithm parameter set is fully encoded in the OID; no parameters field is defined. Applies to:

* ‘clientPublicValue.algorithm‘ when carrying an ML-KEM ephemeral key
* ‘KDCKEMInfo.kemAlgorithm‘

(b) HKDF identifiers ([RFC8619] Section 3): the OID id-alg-hkdf-with-sha512 has absent parameters by definition. Only SHA-512 is defined for the KEM path (Section 9.1). Applies to:

* ‘KDCKEMInfo.kdfAlgorithm‘
* ‘AuthPack.supportedKDFs‘ entries

6. New ASN.1 Types

6.1. Extended PA-PK-AS-REP

[RFC4556] uses an IMPLICIT TAGS module environment. All three arms are IMPLICIT OCTET STRING carrying DER-encoded structures; receivers identify the chosen arm by the context tag alone.

```
-- KRB5PkinitTypes DEFINITIONS IMPLICIT TAGS ::= BEGIN
PA-PK-AS-REP ::= CHOICE {
    dhSignedData    [0] IMPLICIT OCTET STRING,
        -- RFC 4556: DH/ECDH path
        --   content: KDCDHKeyInfo ({{RFC4556}} Section 3.2.3.1)
    encKeyPack      [1] IMPLICIT OCTET STRING,
        -- RFC 4556: RSA path (deprecated)
        --   content: ReplyKeyPack ({{RFC4556}} Section 3.2.3.2)
    kemInfo         [2] IMPLICIT OCTET STRING,
        -- NEW: KEM path (this specification)
        --   content: KEMRepInfo ({{sec-kemrepinfo}})
    ...
}
```

The field name dhSignedData matches RFC 4556’s actual ASN.1 module; the informal name dhInfo used in some descriptions refers to the same arm.

Tag [2] MUST be verified against the IANA Kerberos PKINIT Parameters registry before publication to confirm no other extension has claimed it.

6.2. KEMRepInfo

```
-- id-pkinit OID arc (RFC 4556):
-- id-pkinit OBJECT IDENTIFIER ::= { 1 3 6 1 5 2 3 }

id-pkinit-KEMKeyData OBJECT IDENTIFIER ::= { id-pkinit TBD-IANA }

KEMRepInfo ::= SEQUENCE {
    kemSignedData       [0] IMPLICIT OCTET STRING,
        -- CMS SignedData ({{RFC5652}} Section 5):
        --   eContentType = id-pkinit-KEMKeyData
        --   eContent     = DER(KDCKEMInfo)
        --   signerInfos  = KDC signature over KDCKEMInfo
    ...
}
```

eContent in kemSignedData MUST be present (detached signatures are prohibited).

6.3. KDCKEMInfo

```
KDCKEMInfo ::= SEQUENCE {
    kemAlgorithm    [0] AlgorithmIdentifier,
        -- KEM algorithm and parameter set used (e.g., ML-KEM-768).
        -- MUST match clientPublicValue.algorithm OID.
    kemct           [1] OCTET STRING,
        -- KEM ciphertext produced by Encap(ek) (see {{sec-kem-interface}}).
        -- Algorithm-specific sizes: see {{sec-mlkem-sizes}} for ML-KEM.
    kdfAlgorithm    [2] AlgorithmIdentifier,
        -- HKDF variant selected from supportedKDFs (fixes RFC 8636
        -- unauthenticated selection flaw).
    nonce           [3] INTEGER (0..4294967295) OPTIONAL,
        -- When present, MUST equal pkAuthenticator.nonce from the
        -- client's AS-REQ. Implementations SHOULD include this field.
        -- Future KEM variants and hybrid DH+KEM modes MAY omit it if
        -- alternative freshness mechanisms are defined by their
        -- respective specifications.
    serverNonce     [4] OCTET STRING OPTIONAL,
        -- Reserved for future hybrid DH+KEM modes. Analogous to
        -- RFC 4556 serverDHNonce. MUST be absent in pure ML-KEM.
    ...
}
```

kemAlgorithm makes KDCKEMInfo self-describing and provides signed confirmation that the KDC processed the correct algorithm (verified in Section 8.3 step 4), avoiding implicit inference from kemct length alone.

6.4. Extended AuthPack

```
AuthPack ::= SEQUENCE {
    pkAuthenticator     [0] PKAuthenticator,
    clientPublicValue   [1] SubjectPublicKeyInfo OPTIONAL,
        -- DH/ECDH path: ephemeral DH/ECDH public key (RFC 4556).
        -- KEM path:     ephemeral ML-KEM encapsulation key, encoded per
        --               RFC 9935.
        -- RSA path:     MUST be absent.
    supportedCMSTypes   [2] SEQUENCE OF AlgorithmIdentifier OPTIONAL,
        -- Used in RSA path only. It is deprecated in {{sec-rsa-deprecation}}.
    clientDHNonce       [3] DHNonce OPTIONAL,
        -- Pure KEM path (this specification): MUST be absent when
        -- clientPublicValue contains a KEM algorithm OID (see
        -- {{sec-asn1-types}}). Future hybrid DH+KEM specifications MAY define
        -- use of this field alongside KEM OIDs.
    supportedKDFs       [4] SEQUENCE OF AlgorithmIdentifier OPTIONAL,
        -- KDFAlgorithmId is AlgorithmIdentifier; no separate type is
        -- defined.
        -- KEM path: HKDF algorithm OIDs ({{sec-kdf-oids}}). Only
        --   HKDF-SHA-512 is defined for the KEM path; this field
        --   SHOULD contain id-alg-hkdf-with-sha512. If absent when a
        --   KEM OID is in clientPublicValue, HKDF-SHA-512 is assumed.
        -- DH/ECDH path: DH-KDF algorithm OIDs (RFC 8636).
    ...
}
```

6.5. PkinitKEMSuppPubInfo

```
-- Types imported from RFC 4120 (KerberosV5 module): Int32

PkinitKEMSuppPubInfo ::= SEQUENCE {
    enctype         [0] Int32,
        -- Kerberos enctype of the AS reply key.
    as-REQ          [1] OCTET STRING,
        -- DER(AS-REQ).
    kemSignedData   [2] OCTET STRING,
        -- DER(KEMRepInfo.kemSignedData): the KDC-signed KDCKEMInfo.
    ...
}
```

6.6. PAChecksum2 Extension

[RFC4556] hardwires the paChecksum field in PKAuthenticator to use SHA-1. [RFC8636] Section 3 acknowledges this limitation but does not provide a mechanism to negotiate alternative checksum algorithms, noting that for DH and ECDH paths the KDF binding (which includes the entire AS-REQ in key derivation) provides an eventual integrity check.

This specification extends PKAuthenticator with a paChecksum2 field to provide checksum algorithm agility at the request validation layer. PAChecksum2 was first defined in [MS-PKCA] § 2.2.3 (PA-PK-AS-REQ).

```
PAChecksum2 ::= SEQUENCE {
    checksum                [0] OCTET STRING,
        -- Checksum computed over KDC-REQ-BODY using the algorithm
        -- specified in algorithmIdentifier.
    algorithmIdentifier     [1] AlgorithmIdentifier
        -- Digest algorithm OID. The parameters field MUST be absent.
        -- Implementations MUST support:
        --   SHA-512: 2.16.840.1.101.3.4.2.3 (NIST CSOR, RFC 5754)
        -- Implementations MAY support:
        --   SHA-256: 2.16.840.1.101.3.4.2.1 (NIST CSOR, RFC 5754)
        --   SHA-384: 2.16.840.1.101.3.4.2.2 (NIST CSOR, RFC 5754)
}
```

The PKAuthenticator structure from [RFC4556] is extended as follows:

```
PKAuthenticator ::= SEQUENCE {
    cusec                   [0] INTEGER (0..999999),
    ctime                   [1] KerberosTime,
    nonce                   [2] INTEGER (0..4294967295),
    paChecksum              [3] OCTET STRING OPTIONAL,
        -- RFC 4556: SHA-1 checksum over KDC-REQ-BODY.
    freshnessToken          [4] OCTET STRING OPTIONAL,
        -- RFC 8070: PA_AS_FRESHNESS token from KDC.
    paChecksum2             [5] PAChecksum2 OPTIONAL,
        -- This specification: algorithm-agile checksum over
        -- KDC-REQ-BODY.
    ...
}
```

Client behavior: A client constructing a PKINIT request conforming to this specification MUST include the paChecksum2 field and SHOULD include the paChecksum field (SHA-1, per [RFC4556]). Both checksums, when present, are computed over the same KDC-REQ-BODY input.

KDC validation: A KDC conforming to this specification MUST require paChecksum2 to be present in the request. If paChecksum2 is absent, the KDC returns KDC_ERR_PA_CHECKSUM_MUST_BE_INCLUDED (error code 79, [RFC4556]).

The KDC MUST validate paChecksum2. If paChecksum is also present, the KDC MUST validate it as well. The KDC returns the following errors:

* KDC_ERR_SUMTYPE_NOSUPP (error code 15, [RFC4120]): if the digest algorithm in paChecksum2.algorithmIdentifier is not supported by the KDC.

* KRB_AP_ERR_MODIFIED (error code 41, [RFC4120]): if verification of paChecksum2 fails, or if paChecksum is present and its verification fails.

7. Mode Selection

The exchange mode is determined by the algorithm OID in clientPublicValue:

clientPublicValue	OID type	Mode
Absent	—	RSA path (encKeyPack); deprecated for new deployments
Present	DH or ECDH OID	DH/ECDH path ([RFC4556] / [RFC8636])
Present	ML-KEM or composite ML-KEM OID	KEM path (this specification)
Present	Unrecognized OID	Error (see Section 11); MUST NOT fall back to RSA path

Table 1: Mode selection by clientPublicValue OID

For the pure KEM path defined in this specification, when clientPublicValue contains a KEM OID and clientDHNonce is also present, the KDC MUST return KDC_ERR_PREAUTH_FAILED. Future hybrid DH+KEM specifications may define different nonce semantics and relax this requirement.

When present, supportedKDFs MUST contain only KDFs applicable to the path indicated by clientPublicValue.algorithm: HKDF algorithm OIDs (Section 9.1) for the KEM path, or DH-KDF algorithm OIDs per [RFC8636] for the DH/ECDH path.

8. KEM Path Operation

8.1. Client Request Construction

1. Generate a fresh ephemeral KEM key pair (ek, dk) for the chosen algorithm using a cryptographically secure pseudorandom number generator (CSPRNG). The security of the KEM path depends entirely on the unpredictability of dk (for ML-KEM CSPRNG requirements, see Section 15.2).

2. Encode ek as SubjectPublicKeyInfo and place it in AuthPack.clientPublicValue. parameters MUST be absent. For ML-KEM, encoding follows [RFC9935].

3. Set supportedKDFs to { id-alg-hkdf-with-sha512 }. If omitted, HKDF-SHA-512 is assumed.

4. Wrap AuthPack as the eContent of a CMS SignedData ([RFC5652] Section 5) per [RFC4556] Section 3.2.2 and sign with the client's signing certificate. For full quantum resistance, the client SHOULD use an ML-DSA certificate ([RFC9881]); traditional ECDSA and RSA certificates are permitted during the transition period.

The ephemeral encapsulation key ek is authenticated by the client's signing certificate via AuthPack.SignedData ([RFC5652] Section 5.2), following the [RFC4556] DH path model. No separate encapsulation certificate is required.

8.2. KDC Response Construction

1. Detect the KEM algorithm OID in clientPublicValue.algorithm.

2. Check whether the algorithm is supported and meets the KDC's security policy (per [RFC4556] Section 3.2.2):

   a. If the algorithm OID is not recognized or not implemented, return KDC_ERR_EPHEMERAL_KEY_PARAMS_NOT_ACCEPTED (error code 65) with TD-EPHEMERAL-KEY-PARAMETERS-DATA listing supported algorithms; stop.

   b. If the algorithm does not satisfy the KDC's security policy, return KDC_ERR_EPHEMERAL_KEY_PARAMS_NOT_ACCEPTED (error code 65) with TD-EPHEMERAL-KEY-PARAMETERS-DATA listing acceptable algorithms; stop.

3. Select a KDF from supportedKDFs that is approved for the KEM algorithm (see Section 9.1). If supportedKDFs is absent, default to id-alg-hkdf-with-sha512. If no approved KDF can be selected, return KDC_ERR_NO_ACCEPTABLE_KDF (error code 100, [RFC8636]); stop.

4. Call Encap(ek) → (ss, kemct) using the selected algorithm (see Section 4). Exactly one encapsulation MUST be performed per exchange; the resulting (ss, kemct) pair MUST be used in all subsequent steps (for ML-KEM specifics, see Section 15.3).

5. Build KDCKEMInfo:

   * kemAlgorithm = OID from clientPublicValue.algorithm (echoed back)

   * kemct = the ciphertext from step 4

   * kdfAlgorithm = selected HKDF OID

   * nonce = pkAuthenticator.nonce from the client's request (SHOULD be included; see Section 6.3)

6. Sign KDCKEMInfo using CMS SignedData ([RFC5652] Section 5) with ML-DSA ([RFC9882]) RECOMMENDED. Place in kemSignedData. eContent MUST be present. Step 7 MUST follow step 6 because PkinitKEMSuppPubInfo.kemSignedData is set to the DER encoding of KEMRepInfo.kemSignedData produced in this step.

7. Derive the AS reply key from ss per Section 9. The KDC uses it to encrypt the AS-REP enc-part.

8. Return PA-PK-AS-REP.kemInfo containing KEMRepInfo.

8.3. Client Response Processing

The client MUST perform the following steps in order. On any abort the client MUST erase dk before returning.

1. *Verify KDC signature* over kemSignedData. Abort if invalid.

2. *Verify serverNonce is absent*: KDCKEMInfo.serverNonce MUST NOT be present in pure ML-KEM exchanges defined by this specification. Abort if present.

3. *Extract and verify nonce*: If KDCKEMInfo.nonce is present, it MUST equal pkAuthenticator.nonce. Abort if not. If absent, implementations MUST verify freshness through alternative means (e.g., timestamp in PKAuthenticator); future KEM specifications MUST define which mechanism applies when nonce is omitted.

4. *Verify echoed algorithm*: KDCKEMInfo.kemAlgorithm MUST exactly match the algorithm OID in the client's own clientPublicValue.algorithm. Abort if they differ. This confirms the KDC did not substitute a different algorithm.

5. *Validate kemct length*: the byte length of KDCKEMInfo.kemct MUST match the fixed ciphertext size for KDCKEMInfo.kemAlgorithm (see Section 15.1 for ML-KEM sizes). Abort if not. KEM algorithms MUST NOT be called on incorrectly-sized ciphertexts.

6. *Decapsulate*: ss = Decap(dk, KDCKEMInfo.kemct) using the algorithm in KDCKEMInfo.kemAlgorithm. Erase dk immediately after this call completes, before any further processing.

7. *Derive reply key* from ss per Section 9. Use this key to decrypt the AS-REP enc-part.

8. *Confirm dk erasure*. The ephemeral decapsulation key MUST have been erased in step 6 and MUST NOT be retained.

Steps 15 MUST complete before step 6. Decapsulation MUST NOT be called on an unauthenticated ciphertext.

8.4. KDC Certificate Validation

The KDC certificate validation rules of [RFC4556] Section 3.2.3 apply unchanged to kemSignedData. The client uses the same trust anchors and validation procedure as for the DH path.

9. AS Reply Key Derivation

9.1. HKDF OIDs

The KEM path uses [RFC8636]'s KDF negotiation mechanism via supportedKDFs. For ML-KEM and composite ML-KEM, this specification approves only HKDF-SHA-512:

```
id-alg-hkdf-with-sha512 OBJECT IDENTIFIER ::=
    { 1 2 840 113549 1 9 16 3 30 }
```

OID	Hash	Conformance for ML-KEM
id-alg-hkdf-with-sha512	SHA-512	MUST implement

Table 2: Approved KDFs for ML-KEM

When clientPublicValue.algorithm contains an ML-KEM or composite ML-KEM OID, the KDC selects a KDF from supportedKDFs that appears in the approved list above. If supportedKDFs is absent or contains no approved KDF, the KDC defaults to id-alg-hkdf-with-sha512.

9.2. Derivation

```
reply_key_material = HKDF-SHA-512(
    IKM  = ss,
        -- ML-KEM.Decaps output (see {{sec-mlkem-sizes}} for ML-KEM sizes)
    salt = <not provided>,
        -- defaults to HashLen zero bytes (RFC 5869 Section 2.2);
        -- ss is uniformly random so extraction is unnecessary
    info = DER(PkinitKEMSuppPubInfo),
    L    = <random-to-key input length for enctype>
)
reply_key = random-to-key(reply_key_material)
    -- per RFC 3961 Section 3
```

L is the random-to-key input string length for the Kerberos enctype in PkinitKEMSuppPubInfo.enctype, as defined in the enctype's specification:

Enctype	L
aes128-cts-hmac-sha256-128 ([RFC8009])	16 bytes
aes256-cts-hmac-sha384-192 ([RFC8009])	32 bytes

Table 3: random-to-key input lengths by enctype

For these enctypes random-to-key is the identity function. Other enctypes use the key-generation seedlength from their [RFC3961] crypto profile.

PkinitKEMSuppPubInfo.kemSignedData is set to DER(KEMRepInfo.kemSignedData): the KDC-signed KDCKEMInfo only, not the full PA-PK-AS-REP. This avoids a circular dependency: the full response cannot be included in the context used to derive the key that protects it.

The value reply_key produced by this derivation IS the AS reply key used to encrypt the Kerberos AS-REP enc-part. Both the KDC and the client independently derive the same value from ss using the KDF algorithm and context recorded in PkinitKEMSuppPubInfo. The reply key is never transmitted; both parties arrive at it through independent computation.

10. Error Handling

10.1. Proactive Advertisement

A KDC SHOULD include TD-EPHEMERAL-KEY-PARAMETERS-DATA in KDC_ERR_PREAUTH_REQUIRED to allow the client to select an acceptable algorithm on its first attempt. This avoids a retry round trip, which is particularly valuable for post-quantum deployments where both ML-DSA signatures and ML-KEM encapsulation keys are significantly larger than their traditional counterparts, making the overhead of a failed attempt much higher.

```
-- TD-EPHEMERAL-KEY-PARAMETERS (formerly TD-DH-PARAMETERS) reuses the
-- existing IANA integer from RFC 4556 Section 3.2.2. The ASN.1 encoding
-- is unchanged (SEQUENCE OF AlgorithmIdentifier); RFC 5349 extended the
-- scope to include ECDH. This specification further extends it to include
-- ML-KEM and composite ML-KEM parameter sets.

TD-EPHEMERAL-KEY-PARAMETERS-DATA ::= SEQUENCE OF AlgorithmIdentifier
    -- DH, ECDH, ML-KEM, and composite ML-KEM algorithms the KDC supports,
    -- in decreasing preference order (RFC 4556 Section 3.2.2).
```

10.2. Ephemeral Key Parameter Errors

When the algorithm and parameter set in clientPublicValue.algorithm does not satisfy the KDC's security policy, the KDC returns:

```
KDC_ERR_EPHEMERAL_KEY_PARAMS_NOT_ACCEPTED    65
```

This error code is a renamed and expanded version of KDC_ERR_DH_KEY_PARAMETERS_NOT_ACCEPTED from [RFC4556], which already covers DH ([RFC4556]) and ECDH ([RFC5349]). The error code number (65) is reused; this specification extends the scope to also cover ML-KEM and composite ML-KEM.

This error is returned when:

* The algorithm OID is not recognized or not implemented by the KDC

* The algorithm does not meet the KDC's security requirements

The KDC SHOULD include TD-EPHEMERAL-KEY-PARAMETERS-DATA (as defined in Section 10.1) in the error response. After receiving this error, the client follows [RFC4556] Section 3.2.2 retry behavior, selecting a different parameter set from TD-EPHEMERAL-KEY-PARAMETERS-DATA that satisfies the client's security policy. If no mutually acceptable parameter set exists, the exchange MUST be terminated.

10.3. KEM Path Errors

When clientPublicValue contains a KEM OID, the KDC MUST NOT return DH digest negotiation errors (KDC_ERR_DIGEST_IN_SIGNED_DATA_NOT_ACCEPTED, KDC_ERR_DIGEST_IN_CERT_NOT_ACCEPTED). The only applicable error for parameter negotiation failure on the KEM path is KDC_ERR_EPHEMERAL_KEY_PARAMS_NOT_ACCEPTED (Section 10.2).

11. Downgrade Prevention

When a client uses a post-quantum certificate (e.g., ML-DSA per [RFC9881], composite ML-DSA per [I-D.ietf-lamps-pq-composite-sigs], or future quantum-resistant signature algorithms) and sends a post-quantum KEM encapsulation key (ML-KEM or composite ML-KEM) in clientPublicValue, the client MUST NOT fall back to traditional key-establishment algorithms (DH, ECDH, RSA). This ensures quantum-resistant authentication and key establishment are paired. The client MAY retry with a different post-quantum KEM algorithm from Section 10.3. If no post-quantum KEM is available, the client MUST fail the authentication attempt.

Clients using traditional certificates (RSA, ECDSA) MAY fall back from post-quantum KEM to traditional key establishment (DH, ECDH) for backward compatibility with non-upgraded KDCs. The security considerations regarding unauthenticated error messages in [RFC4556] Section 5 apply.

12. Algorithm Requirements

12.1. KEM Algorithms

12.1.1. Pure ML-KEM Algorithms

Algorithm	OID	NIST Category	Conformance
ML-KEM-512	2.16.840.1.101.3.4.4.1	1	MAY
ML-KEM-768	2.16.840.1.101.3.4.4.2	3	MUST

			implement
ML-KEM-1024	2.16.840.1.101.3.4.4.3	5	SHOULD

Table 4: Pure ML-KEM algorithm requirements

12.1.2. Composite ML-KEM Algorithms

Algorithm	OID	NIST Category	Conformance
id-MLKEM768-ECDH-P256-SHA3-256	1.3.6.1.5.5.7.6.59	3	SHOULD
id-MLKEM768-X25519-SHA3-256	1.3.6.1.5.5.7.6.58	3	MAY
id-MLKEM1024-ECDH-P384-SHA3-256	1.3.6.1.5.5.7.6.63	5	SHOULD

Table 5: Composite ML-KEM algorithm requirements

Composite algorithms are defined in [I-D.ietf-lamps-pq-composite-kem].

12.2. Client Algorithm Selection

As with [RFC4556] DH path algorithm selection, the client selects which KEM algorithm to use based on local policy. Algorithm selection is implementation-defined.

If the client receives proactive advertisement (Section 10.1) and supports none of the advertised algorithms, it MUST fail the authentication attempt rather than trying an unadvertised algorithm.

12.3. KDC Security Policy

As with [RFC4556] Section 3.2.2, the KDC enforces a security policy for ephemeral key algorithms. The specific policy is implementation-defined.

For ML-KEM, implementations MAY use NIST security categories as a basis for policy decisions:

Category	Algorithm	Post-quantum bit security
1	ML-KEM-512	128 bits
3	ML-KEM-768	192 bits
5	ML-KEM-1024	256 bits

Table 6: NIST security categories for ML-KEM

NIST recommends ML-KEM-768 (Category 3) as the default parameter set, as it provides a large security margin at a reasonable performance cost. Composite parameter sets combining ML-KEM with traditional algorithms provide the security category of their ML-KEM component for post-quantum resistance.

13. RSA Path Deprecation

The encKeyPack [1] path is quantum-vulnerable. New deployments SHOULD NOT use encKeyPack. Existing deployments MAY continue using it for traditional compatibility during migration.

14. Message Size Considerations

An AuthPack with an ephemeral ML-KEM-768 encapsulation key (1184 bytes, see Section 15.1) signed with ML-DSA-65 (3309-bytes signature [FIPS204]) will exceed UDP datagram limits. TCP transport ([RFC5021]) is REQUIRED for ML-KEM PKINIT. All Kerberos infrastructure (KDCs, clients, firewalls) MUST support TCP Kerberos before enabling ML-KEM PKINIT.

15. ML-KEM-Specific Considerations

This section captures behavior specific to ML-KEM ([FIPS203]). When this specification is extended to other KEM algorithms, per-algorithm sections following this structure SHOULD be added. The core protocol defined in Section 4 through Section 10 is intentionally algorithm-agnostic; ML-KEM details are isolated here following the model of [RFC3961].

15.1. Key and Ciphertext Sizes

All sizes are fixed by [FIPS203]; no variability is permitted.

Algorithm	Encapsulation key	Ciphertext	Shared secret
ML-KEM-512	800 bytes	768 bytes	32 bytes
ML-KEM-768	1184 bytes	1088 bytes	32 bytes
ML-KEM-1024	1568 bytes	1568 bytes	32 bytes

Table 7: ML-KEM fixed key and ciphertext sizes

The client MUST validate KDCKEMInfo.kemct length against these values before calling Decapsulate (Section 8.3 step 5). [FIPS203] does not define behavior for ML-KEM.Decaps on incorrectly-sized input.

15.2. CSPRNG Requirement

Both ML-KEM key generation and encapsulation MUST use a cryptographically secure pseudorandom number generator (CSPRNG) satisfying the requirements of [FIPS203] Section 3.3. The security of the KEM path depends on:

* Client-side: the unpredictability of the ephemeral decapsulation key dk during key generation.

* KDC-side: the unpredictability of the randomness used during ML-KEM.Encaps(ek), which produces the shared secret ss and ciphertext kemct.

A weak or predictable RNG on either side compromises the AS reply key.

15.3. Encapsulation and Decapsulation

KDC: (ss, kemct) = ML-KEM.Encaps(ek) per Section 8.2 step 4.

Client: ss = ML-KEM.Decaps(dk, kemct) per Section 8.3 step 6, followed by immediate dk erasure.

The shared secret ss is 32 bytes for all three ML-KEM variants.

16. Security Considerations

16.1. Quantum Resistance

The KEM path achieves post-quantum confidentiality only when both the KEM algorithm and the KDC signing algorithm are quantum-resistant. Using ML-KEM with a traditional ECDSA or RSA signing certificate provides PQC key establishment but not PQC authentication; an adversary with a quantum computer could impersonate the KDC by forging its traditional signature. Deployers seeking full quantum resistance MUST use ML-DSA ([RFC9881]) or a composite ML-DSA variant ([I-D.ietf-lamps-pq-composite-sigs]) for KDC signing.

16.2. Ephemeral Decapsulation Key Hygiene

The ephemeral decapsulation key dk MUST be erased as soon as decapsulation completes (Section 8.3 step 6). Failure to erase dk negates forward secrecy: an attacker who later recovers dk can recompute ss and derive the AS reply key for any recorded exchange that used the corresponding ek.

16.3. Authenticated KDF Inputs

[RFC8636] leaves the KDF algorithm unprotected: a man-in-the-middle could modify the kdfAlgorithm field before the client processes the response. This specification places kdfAlgorithm inside the KDC-signed KDCKEMInfo, making it authenticated. Clients MUST NOT derive a reply key using an algorithm not present in the signed structure.

16.4. Unauthenticated Error Messages

The security considerations in [RFC4556] Section 5 regarding unauthenticated KRB-ERROR messages apply to TD-EPHEMERAL-KEY-PARAMETERS-DATA. Clients MUST enforce their configured security policy regardless of advertised algorithms.

16.5. Algorithm Downgrade Prevention

The downgrade prevention rules in Section 11 are mandatory and unconditional. A client that allows a fallback from the KEM path to the DH/RSA path on receiving a traditional response exposes the session to an active attacker who can exploit a traditional-path vulnerability.

16.6. paChecksum2 and Replay Prevention

paChecksum2 binds the KDC-REQ-BODY to the authenticator using a quantum-safe digest. Implementations MUST NOT accept requests in which paChecksum2 is absent when operating in KEM mode, as defined in Section 6.6.

16.7. Nonce Generation

The nonce in PKAuthenticator provides replay protection. Implementations MUST generate nonces from a Deterministic Random Bit Generator (DRBG) approved by [SP800-90A]. Counter-based or predictable nonces compromise replay protection.

17. IANA Considerations

17.1. New Kerberos Error Code

IANA is requested to assign a new Kerberos Message Error Code in the "Kerberos Message Error Codes" sub-registry of the "Kerberos Parameters" registry:

Value	Old Name	New Name	Reference
65	KDC_ERR_DH_KEY_PARAMETERS_NOT_ACCEPTED	KDC_ERR_EPHEMERAL_KEY_PARAMS_NOT_ACCEPTED	[RFC4556], This document

Table 8: Renamed Kerberos error code

17.2. New PKINIT OID

IANA is requested to assign a new object identifier under the PKINIT OID arc (id-pkinit, 1.3.6.1.5.2.3) in the "SMI Security for PKIX Module Identifier" registry:

Decimal	Description	Reference
TBD	id-pkinit-KEMKeyData	This document

Table 9: New PKINIT OID assignment

17.3. Update to Kerberos Pre-Authentication Data Types Registry

IANA is requested to update the description of the existing entry for TD-DH-PARAMETERS in the "Kerberos Pre-Authentication Data Types" sub-registry of the "Kerberos Parameters" registry:

Old description: "Typed data for KDC_ERR_KEY_TOO_WEAK; contains a list of acceptable Diffie-Hellman algorithm identifiers."

New name and description: TD-EPHEMERAL-KEY-PARAMETERS, "Typed data for KDC_ERR_EPHEMERAL_KEY_PARAMS_NOT_ACCEPTED and KDC_ERR_KEY_TOO_WEAK; contains a list of acceptable ephemeral key-establishment algorithm identifiers, including DH, ECDH, ML-KEM, and composite ML-KEM algorithms."

The integer value and ASN.1 encoding (SEQUENCE OF AlgorithmIdentifier) are unchanged. No new integer allocation is required.

18. References

18.1. Normative References

[FIPS203] National Institute of Standards and Technology (NIST), "Module-Lattice-Based Key-Encapsulation Mechanism Standard", FIPS PUB 203, August 2024, <https://doi.org/10.6028/NIST.FIPS.203>.

[FIPS204] National Institute of Standards and Technology (NIST), "Module-Lattice-Based Digital Signature Standard", FIPS PUB 204, August 2024, <https://doi.org/10.6028/NIST.FIPS.204>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/rfc/rfc2119>.

[RFC3961] Raeburn, K., "Encryption and Checksum Specifications for Kerberos 5", RFC 3961, DOI 10.17487/RFC3961, February 2005, <https://www.rfc-editor.org/rfc/rfc3961>.

[RFC4120] Neuman, C., Yu, T., Hartman, S., and K. Raeburn, "The Kerberos Network Authentication Service (V5)", RFC 4120, DOI 10.17487/RFC4120, July 2005, <https://www.rfc-editor.org/rfc/rfc4120>.

[RFC4556] Zhu, L. and B. Tung, "Public Key Cryptography for Initial Authentication in Kerberos (PKINIT)", RFC 4556, DOI 10.17487/RFC4556, June 2006, <https://www.rfc-editor.org/rfc/rfc4556>.

[RFC5021] Josefsson, S., "Extended Kerberos Version 5 Key Distribution Center (KDC) Exchanges over TCP", RFC 5021, DOI 10.17487/RFC5021, August 2007, <https://www.rfc-editor.org/rfc/rfc5021>.

[RFC5349] Zhu, L., Jaganathan, K., and K. Lauter, "Elliptic Curve Cryptography (ECC) Support for Public Key Cryptography for Initial Authentication in Kerberos (PKINIT)", RFC 5349, DOI 10.17487/RFC5349, September 2008, <https://www.rfc-editor.org/rfc/rfc5349>.

[RFC5652] Housley, R., "Cryptographic Message Syntax (CMS)", STD 70, RFC 5652, DOI 10.17487/RFC5652, September 2009, <https://www.rfc-editor.org/rfc/rfc5652>.

[RFC5869] Krawczyk, H. and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)", RFC 5869, DOI 10.17487/RFC5869, May 2010, <https://www.rfc-editor.org/rfc/rfc5869>.

[RFC8009] Jenkins, M., Peck, M., and K. Burgin, "AES Encryption with HMAC-SHA2 for Kerberos 5", RFC 8009, DOI 10.17487/RFC8009, October 2016, <https://www.rfc-editor.org/rfc/rfc8009>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/rfc/rfc8174>.

[RFC8619] Housley, R., "Algorithm Identifiers for the HMAC-based Extract-and-Expand Key Derivation Function (HKDF)", RFC 8619, DOI 10.17487/RFC8619, June 2019, <https://www.rfc-editor.org/rfc/rfc8619>.

[RFC8636] Hornquist Astrand, L., Zhu, L., Cullen, M., and G. Hudson, "Public Key Cryptography for Initial Authentication in Kerberos (PKINIT) Algorithm Agility", RFC 8636, DOI 10.17487/RFC8636, July 2019, <https://www.rfc-editor.org/rfc/rfc8636>.

[RFC9881] Massimo, J., Kampanakis, P., Turner, S., and B. E. Westerbaan, "Internet X.509 Public Key Infrastructure -- Algorithm Identifiers for the Module-Lattice-Based Digital Signature Algorithm (ML-DSA)", RFC 9881, DOI 10.17487/RFC9881, October 2025, <https://www.rfc-editor.org/rfc/rfc9881>.

[RFC9935] Turner, S., Kampanakis, P., Massimo, J., and B. E. Westerbaan, "Internet X.509 Public Key Infrastructure - Algorithm Identifiers for the Module-Lattice-Based Key-Encapsulation Mechanism (ML-KEM)", RFC 9935, DOI 10.17487/RFC9935, March 2026, <https://www.rfc-editor.org/rfc/rfc9935>.

[SP800-90A]
National Institute of Standards and Technology (NIST), "Recommendation for Random Number Generation Using Deterministic Random Bit Generators", NIST SP 800-90A Rev. 1, June 2015, <https://doi.org/10.6028/NIST.SP.800-90Ar1>.

18.2. Informative References

[I-D.ietf-lamps-pq-composite-kem]
Ounsworth, M., Gray, J., Pala, M., Klauner, J., and S. Fluhrer, "Composite ML-KEM for use in X.509 Public Key Infrastructure", Work in Progress, Internet-Draft, draft-ietf-lamps-pq-composite-kem-14, 27 March 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-lamps-pq-composite-kem-14>.

[I-D.ietf-lamps-pq-composite-sigs]
Ounsworth, M., Gray, J., Pala, M., Klauner, J., and S. Fluhrer, "Composite Module-Lattice-Based Digital Signature Algorithm (ML-DSA) for use in X.509 Public Key Infrastructure", Work in Progress, Internet-Draft, draft-ietf-lamps-pq-composite-sigs-19, 21 April 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-lamps-pq-composite-sigs-19>.

[MS-PKCA] Microsoft Corporation, "[MS-PKCA]: Public Key Cryptography for Initial Authentication (PKINIT) in Kerberos Protocol", 20 September 2023, <https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-pkca/>.

[RFC9882] Salter, B., Raine, A., and D. Van Geest, "Use of the ML-DSA Signature Algorithm in the Cryptographic Message Syntax (CMS)", RFC 9882, DOI 10.17487/RFC9882, October 2025, <https://www.rfc-editor.org/rfc/rfc9882>.

Acknowledgements

The authors thank the IETF KITTEN and LAMPS working groups for discussion of post-quantum PKINIT approaches, and the NIST team for [FIPS203] and [FIPS204].

Authors’ Addresses

Alexander Bokovoy
Red Hat, Inc.
Email: abokovoy@redhat.com

Julien Rische
Red Hat, Inc.
Email: jrische@redhat.com

Nico Williams
Cryptonector
Email: nico@cryptonector.com