Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: 16 November 2026

G. Berra, Ed.
C. F. Myers, Ed.
R. Shane, Ed.
Freedom of the Press Foundation

15 May 2026

Authenticated Modes for HPKE
draft-ms-hpke-auth-modes-01

Abstract

The standards-track [I-D.ietf-hpke-hpke] supersedes the informational [RFC9180], omitting its authenticated modes mode_auth and mode_auth_psk. This document restores mode_auth_psk mode as a strict extension and illustrates how the restored mode can be used with a post-quantum "shared secret" as the PSK in order to achieve hybrid PQ/T confidentiality while transitioning to quantum-resistant encryption, without deprecating the classical implicit authentication on which many applications still rely.

This extension requires only the addition of AuthEncap()/AuthDecap() to the DHKEM, the definition of four setup functions, and a change in VerifyPSKInputs(). The extension does not alter the externally observable behavior of the existing HPKE modes standardized in [I-D.ietf-hpke-hpke]. Although AuthEncap()/AuthDecap() reintroduce the functionality of mode_auth, the mode itself is not restored, since it cannot provide quantum resistance.

This document discusses the transitional nature of the AuthPSK construction and its security properties as a transitional step towards quantum resistance. In particular, this document illustrates how the restored mode_auth_psk can be used to provide ready-made KEM combiner-like functionality ([I-D.ounsworth-cfrg-kem-combiners]) without requiring downstream API users to manage their own encryption context.

About This Document

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

Status information for this document may be found at https://datatracker.ietf.org/doc/draft-ms-hpke-auth-modes/.

Source for this draft and an issue tracker can be found at https://github.com/cfm/draft-ms-hpke-auth-modes.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

[I-D.ietf-hpke-hpke] is the standards-track successor to the informational [RFC9180] and omits the authenticated modes mode_auth and mode_auth_psk to simplify the standard. However, some applications make use of the type of implicit sender authentication provided by these modes.

The normative portion of this document is small. It restores mode_auth_psk as a strict extension to [I-D.ietf-hpke-hpke]. This requires AuthEncap()/AuthDecap() from the DHKEM construction in [RFC9180], and a modified VerifyPSKInputs(). The externally observable behavior of existing HPKE modes is unchanged. AuthEncap() computes a static-static DH value DH(skS, pkR) alongside the ephemeral-static value and mixes both into the key derivation, binding the output to the sender's key pair.

Although the functionality of mode_auth is re-introduced, the mode identifer itself, which relies solely on DHKEM, is not restored, since it cannot provide quantum-resistant encryption. mode_auth is instead treated as a building block to mode_auth_psk, as seen in Section 3.7.

Section 3.3 describes how the restored mode_auth_psk can be used to achieve hybrid PQ/T confidentiality as defined in Section 5 of [RFC9794] during the transition to quantum-resistant encryption. Section 3.8 discusses the transitional nature of this construction and reviews the guidance available to application developers looking to begin the transition to quantum-resistant encryption with existing libraries and APIs.

To motivate the extension, Section 3.7.2 discusses how the extension can be used as a type of black-box KEM combiner ([I-D.ounsworth-cfrg-kem-combiners]), similar to the construction proposed in [Alwen2023], to allow application developers to begin the transition to quantum-resistant encryption via usable libraries and APIs.

2. Conventions and Definitions

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.

Terms from [I-D.ietf-hpke-hpke] are used without redefinition; particular reference is made to the *DHKEM* construction Section 4.1 of [RFC9180]. The following additional term is used herein:

* *AKEM:* a KEM whose encapsulation additionally takes the sender’s static private key and implicitly binds the shared secret to it.

3. Authenticated Pre-Shared Key Mode Extension to [I-D.ietf-hpke-hpke]

This section specifies the additions to [I-D.ietf-hpke-hpke] required to restore mode_auth_psk. These extensions are defined so that the externally observable behavior of the existing HPKE modes is unchanged, although this document modifies the VerifyPSKInputs() procedure in [I-D.ietf-hpke-hpke].

3.1. Mode Identifiers

The reserved entry for value 0x03 in Table 1 of [I-D.ietf-hpke-hpke] is replaced with the following mode identifier, as originally specified in Table 1 of [RFC9180]:

mode_auth_psk = 0x03

The value 0x02 remains reserved.

3.2. Exclusion of 0x02 mode_auth from Mode Identifiers

Although restoring the AuthEncap() and AuthDecap() functions would also allow for restoring mode_auth, this mode cannot offer quantum-resistant encryption, and its identifier is therefore not reintroduced.

3.3. DHKEM Extension: AuthEncap() and AuthDecap()

The following two functions are added to the DHKEM, extending it to an AKEM (DHAKEM). They are reproduced verbatim from Section 4.1 of [RFC9180]. All helper functions (GenerateKeyPair, DH, SerializePublicKey, DeserializePublicKey, ExtractAndExpand) are as defined in [I-D.ietf-hpke-hpke].

```
def AuthEncap(pkR, skS):
  skE, pkE = GenerateKeyPair()
  dh = concat(DH(skE, pkR), DH(skS, pkR))
  enc = SerializePublicKey(pkE)

  pkRm = SerializePublicKey(pkR)
  pkSm = SerializePublicKey(pk(skS))

  kem_context = concat(enc, pkRm, pkSm)
  shared_secret = ExtractAndExpand(dh, kem_context)

  return shared_secret, enc

def AuthDecap(enc, skR, pkS):
  pkE = DeserializePublicKey(enc)
  dh = concat(DH(skR, pkE), DH(skR, pkS))

  pkRm = SerializePublicKey(pk(skR))
  pkSm = SerializePublicKey(pkS)
  kem_context = concat(enc, pkRm, pkSm)

  shared_secret = ExtractAndExpand(dh, kem_context)
  return shared_secret
```

Note that the AuthEncap() and AuthDecap() functions are vulnerable to key-compromise impersonation (KCI): the assurance that the shared secret was generated by the holder of the private key skS does not hold if the recipient private key skR is compromised. See Section 4 for further discussion; see Section 3.5 for integration of a pre-shared key.

3.4. VerifyPSKInputs

The VerifyPSKInputs() function defined in Section 5.1 of [RFC9180] and [I-D.ietf-hpke-hpke] is extended to handle the reintroduced mode_auth_psk. The updated function replaces the original:

```
def VerifyPSKInputs(mode, psk, psk_id):
  got_psk = (psk != default_psk)
  got_psk_id = (psk_id != default_psk_id)
  if got_psk != got_psk_id:
    raise Exception("Inconsistent PSK inputs")

  if got_psk and mode in [mode_base]:
    raise Exception("PSK input provided when not needed")
  if (not got_psk) and mode in [mode_psk, mode_auth_psk]:
    raise Exception("Missing required PSK input")
```

The only change from the original is that mode_psk is replaced by [mode_psk, mode_auth_psk] in the final guard.

3.5. Setup Functions

The following four setup functions are reproduced verbatim from Sections 5.1.3 and 5.1.4 of [RFC9180]. KeyScheduleS()/KeyScheduleR() and AuthEncap()/AuthDecap() are as defined in [I-D.ietf-hpke-hpke] and Section 3.3 respectively.

```
def SetupAuthPSKS(pkR, info, psk, psk_id, skS):
  shared_secret, enc = AuthEncap(pkR, skS)
  return enc, KeyScheduleS(mode_auth_psk, shared_secret, info,
                           psk, psk_id)

def SetupAuthPSKR(enc, skR, info, psk, psk_id, pkS):
  shared_secret = AuthDecap(enc, skR, pkS)
  return KeyScheduleR(mode_auth_psk, shared_secret, info,
                      psk, psk_id)
```

3.6. Input Validation and Error Handling

In addition to the validation requirements in Section 7.1.4 of [I-D.ietf-hpke-hpke], the recipient MUST validate the sender’s static public key pkS before use in AuthDecap(), applying the same validation rules as for other public key inputs. Validation failure MUST yield a ValidationError.

3.7. DHAKEM with PSK

This section illustrates how the authenticated mode defined in Section 3 can be used. Any AEAD identifier from [I-D.ietf-hpke-hpke] may be used; the resulting context supports Seal(), Open(), and Export(). Key generation follows GenerateKeyPair() from [I-D.ietf-hpke-hpke].

mode_auth_psk may be used via its setup function: the sender calls SetupAuthPSKS() and the receiver calls SetupAuthPSKR(), with a pre-shared key psk and a PSK identifier psk_id, as defined in Section 3.5. As in [RFC9180], both parties are assumped to have been provisioned with the PSK value psk and another byte string psk_id.

This mode SHOULD be used with a quantum-resistant PSK value as described below in order to offer hybrid PQ/T confidentiality properties.

3.7.1. PQ/T Hybrid Construction

```
def HybridSetupS(pkR, skS, info, psk, psk_id):
  enc_dh, ctx = SetupAuthPSKS(pkR, info, psk, psk_id, skS)
  return enc_dh, ctx

def HybridSetupR(enc, skR, pkS, info, psk, psk):
  return SetupAuthPSKR(enc_dh, skR, info, psk, psk_id, pkS)
```

The suite_id in the HPKE key schedule reflects only the ciphersuite (KEM_ID, KDF_ID, AEAD_ID), where KEM_ID is a DHAKEM. Generation and distribution of a quantum-safe PSK value is left to the application.

Hybrid confidentiality. KeyScheduleS()/KeyScheduleR() delegate to CombineSecrets, for which [I-D.ietf-hpke-hpke] defines two variants. In CombineSecrets_TwoStage(), the combination is secret = LabeledExtract(dhkem_shared_secret, "secret", psk), equivalent to HKDF-Extract(salt = dhkem_shared_secret, IKM = psk) [RFC5869]. In CombineSecrets_OneStage(), dhkem_shared_secret and psk are length-prefixed and concatenated before a single LabeledDerive() call. In both cases, dhkem_shared_secret and psk enter the combination as independent inputs. The intended design property is that secret remains pseudorandom as long as at least one of the two inputs is--- meaning an adversary would need to attack both the classical DH-based component and the PQ-KEM to recover secret, as seen in [I-D.ounsworth-cfrg-kem-combiners] and subsequently, [I-D.draft-connolly-cfrg-xwing-kem-10].

Whether this property holds formally for a specific CombineSecrets variant depends on that variant’s security analysis, which is outside the scope of this document.

Authentication. This mode retains the implicit sender authentication properties of DHAKEM described in [RFC9180]. A quantum adversary with access to the PSK can also forge authenticated messages.

PSK freshness. The PSK psk MUST satisfy the entropy requirement in Section 9.5 of [I-D.ietf-hpke-hpke] (32 bytes of uniform randomness).

3.7.2. AKEM/PQ KEM "Combiner" (Informative)

Using mode_auth_psk with the shared secret from a PQ-KEM as the provided "psk" value allows application developers to provide hybrid PQ/T security properties using ready-made libraries and APIs. Although this psk value is not a true pre-shared key, this document adopts the pskAPKE terminology from [Alwen2023].

The result is a KEM combiner-style construction [I-D.ounsworth-cfrg-kem-combiners] that provides hybrid PQ/T confidentiality and classical authentication, without requiring developers to manage their own encryption context---a frequent source of developer error and a motivating factor for the API design choices of [RFC9180].

In addition to the keypair specified in Section 3.5, the receiver holds an additional post-quantum keypair, (skR_pq, pkR_pq). Prior to setting up an HPKE encryption context (either via Setup or via a single-shot API), the sender uses receiver's PQ public key pkR_pq to generate a shared secret and its encapsulation (ss_pq, enc_pq), and uses ss_pq as the psk and a static identifier as the PSK identifier. The ciphertext encapsulation of ss_pq, enc_pq, is included in info to bind it to the key schedule.

An example construction is provided below, with reference to the following terms:

* PQKEM: a post-quantum KEM, e.g., ML-KEM [FIPS203] or an algorithm from [I-D.ietf-hpke-pq].

* PQKEM.Encap(pkR_pq): PQ-KEM encapsulation; returns (enc_pq, ss_pq).

* PQKEM.Decap(enc_pq, skR_pq): PQ-KEM decapsulation; returns ss_pq.

* Nenc_pq: the fixed ciphertext length of the chosen PQ-KEM, in bytes.

```
def HybridSetupS(pkR, pkR_pq, skS, info):
  enc_pq, ss_pq = PQKEM.Encap(pkR_pq)
  enc_dh, ctx = SetupAuthPSKS(pkR, concat(info, enc_pq), ss_pq, enc_pq, skS)
  return concat(enc_dh, enc_pq), ctx

def HybridSetupR(enc, skR, skR_pq, pkR_pq, pkS, info):
  enc_dh, enc_pq = enc[:Nenc], enc[Nenc:]
  ss_pq = PQKEM.Decap(enc_pq, skR_pq)
  return SetupAuthPSKR(enc_dh, skR, concat(info, enc_pq), ss_pq, enc_pq, pkS)
```

Classical (implicit) sender authentication: This construction provides only classical sender authentication is provided. In contrast to Section 3.7.1, this "psk" value provides no sender authentication, as it is constructed rather than pre-shared. This limitation is assessed in Section 3.8.

The info parameter will exceed 64 bytes. Implementors must ensure their choice of algorithms and underlying implementation can support parameters of this length.

The shared secret ss_pq must satisfy the entropy requirement in Section 9.5 of [I-D.ietf-hpke-hpke].

Implementations should verify len(enc) == Nenc + Nenc_pq and reject encapsulations of any other length. A fresh (ss_pq, enc_pq) pair should be generated for each encapsulation; reuse of a prior enc_pq is prohibited. The suite_id in the HPKE key schedule reflects only the ciphersuite (AKEM_ID, KDF_ID, AEAD_ID); the PQ-KEM algorithm identity should be conveyed via application-layer framing when multiple PQ-KEM algorithms are supported.

Note that [Alwen2023] describes a related hybrid construction in which a PQ _AKEM_ (rather than an unauthenticated KEM) is used to generate the PSK, which would additionally provide post-quantum sender authentication; that stronger construction is outside the scope of this document.

3.8. Motivation (Informative)

Application developers are in a bind: though they may be aware of advice to implement quantum-resistant encryption on an accelerated timeline, they may encounter rapidly-evolving guidance on best practices, a lack of direct parity with classical constructions, and a relative paucity of stable libraries and APIs [PQCodePkgs].

Developers looking to offer hybrid PQ/T encryption in their own applications, for reasons described for example in Section 2.1 of [RFC9370], can look to early-stage implementations of [I-D.draft-connolly-cfrg-xwing-kem-10], or may refer to the now-expired [I-D.ounsworth-cfrg-kem-combiners], but will need to manage their own combiner implementation.

On the other hand, production-ready quantum-resistant authentication is still maturing. Standardized schemes offer non-repudiable, signature-based authentication, which is not a direct replacement for the type of DH-based implicit authentication described in the authenticated modes of [RFC9180] or this document.

Although the strategy of hybrid encryption with classical authentication is not as straightforward to communicate as unauthenticated hybrid encryption that forgoes implict authentication entirely, protocols such as Noise IK, as used in [Wireguard2020], indicate that this strategy has real-world uses until quantum-resistant authentication methods become more available.

Allowing application developers to deploy quantum-resistant encryption as a transitional measure without deprecating the classical authentication properties of their application provides a path towards quantum readiness, similar in concept to Section 3 of [RFC8773] and Section 5.1 of [RFC9257], which also discuss the introduction of quantum-resistent PSK as a transitional measure.

4. Security Considerations

The sender-authentication and key-compromise impersonation (KCI) properties of mode_auth_psk are as described in Sections 9.1 and 9.1.1 of [RFC9180], which apply without change to the functions defined in Section 3. Security properties specific to the hybrid PQ/T construction are discussed informatively in Section 3.7.1.

The formal security of the DHKEM authenticated modes under the Gap-DH assumption is established in [Alwen2021]. The security of mode_auth_psk---termed AuthPSK in [Alwen2023]---is analyzed there as the pskAPKE scheme.

5. IANA Considerations

This document requests no IANA actions; all identifiers are drawn from registries defined in [I-D.ietf-hpke-hpke].

6. References

6.1. Normative References

[I-D.ietf-hpke-hpke]
Barnes, R., Bhargavan, K., Lipp, B., and C. A. Wood, "Hybrid Public Key Encryption", Work in Progress, Internet-Draft, draft-ietf-hpke-hpke-03, 2 March 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-hpke-hpke-03>.

[I-D.ounsworth-cfrg-kem-combiners]
Ounsworth, M., Wussler, A., and S. Kousidis, "Combiner function for hybrid key encapsulation mechanisms (Hybrid KEMs)", Work in Progress, Internet-Draft, draft-ounsworth-cfrg-kem-combiners-05, 31 January 2024, <https://datatracker.ietf.org/doc/html/draft-ounsworth-cfrg-kem-combiners-05>.

[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>.

[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>.

6.2. Informative References

[Alwen2021] Alwen, J., Blanchet, B., Hauck, E., Kiltz, E., Lipp, B., and D. Riepel, "HPKE: Hybrid Public Key Encryption", Advances in Cryptology -- EUROCRYPT 2021, LNCS 12696, pp. 396-427, DOI 10.1007/978-3-030-77886-6_14, 2021, <https://doi.org/10.1007/978-3-030-77886-6_14>.

[Alwen2023] Alwen, J., Janneck, J., Kiltz, E., and B. Lipp, "The Pre-Shared Key Modes of HPKE", Advances in Cryptology -- ASIACRYPT 2023, 2023, <https://eprint.iacr.org/2023/1480>.

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

[I-D.draft-connolly-cfrg-xwing-kem-10] Connolly, D., Schwabe, P., and B. Westerbaan, "X-Wing: general-purpose hybrid post-quantum KEM", Work in Progress, Internet-Draft, draft-connolly-cfrg-xwing-kem-10, 2 March 2026, <https://datatracker.ietf.org/doc/html/draft-connolly-cfrg-xwing-kem-10>.

[I-D.draft-ietf-tls-hybrid-design-16] Stebila, D., Fluhrer, S., and S. Gueron, "Hybrid key exchange in TLS 1.3", Work in Progress, Internet-Draft, draft-ietf-tls-hybrid-design-16, 7 September 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-tls-hybrid-design-16>.

[I-D.ietf-hpke-pq] Barnes, R. and D. Connolly, "Post-Quantum and Post-Quantum/Traditional Hybrid Algorithms for HPKE", Work in Progress, Internet-Draft, draft-ietf-hpke-pq-04, 2 March 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-hpke-pq-04>.

[PQCodePkgs] Post Quantum Cryptography Alliance, "PQ Code Package Repositories (accessed 2026-04-30)", 2026, <https://github.com/orgs/pq-code-package/repositories>.

[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>.

[RFC8773] Housley, R., "TLS 1.3 Extension for Certificate-Based Authentication with an External Pre-Shared Key", RFC 8773, DOI 10.17487/RFC8773, March 2020, <https://www.rfc-editor.org/rfc/rfc8773>.

[RFC9180] Barnes, R., Bhargavan, K., Lipp, B., and C. Wood, "Hybrid Public Key Encryption", RFC 9180, DOI 10.17487/RFC9180, February 2022, <https://www.rfc-editor.org/rfc/rfc9180>.

[RFC9257] Housley, R., Hoyland, J., Sethi, M., and C. A. Wood, "Guidance for External Pre-Shared Key (PSK) Usage in TLS", RFC 9257, DOI 10.17487/RFC9257, July 2022, <https://www.rfc-editor.org/rfc/rfc9257>.

[RFC9370] Tjhai, CJ., Tomlinson, M., Bartlett, G., Fluhrer, S., Van Geest, D., Garcia-Morchon, O., and V. Smyslov, "Multiple Key Exchanges in the Internet Key Exchange Protocol Version 2 (IKEv2)", RFC 9370, DOI 10.17487/RFC9370, May 2023, <https://www.rfc-editor.org/rfc/rfc9370>.

[RFC9794] Driscoll, F., Parsons, M., and B. Hale, "Terminology for Post-Quantum Traditional Hybrid Schemes", RFC 9794, DOI 10.17487/RFC9794, June 2025, <https://www.rfc-editor.org/rfc/rfc9794>.

[Wireguard2020] Donenfeld, J. A., "WireGuard: Next Generation Kernel Network Tunnel", 2020, <https://www.wireguard.com/papers/wireguard.pdf>.

Acknowledgments

_TK_

Authors' Addresses

Giulio Berra (editor)
Freedom of the Press Foundation
Email: giulio@freedom.press

Cory Francis Myers (editor)
Freedom of the Press Foundation

Email: cfm@acm.org

Rowen Shane (editor)
Freedom of the Press Foundation
Email: ro@freedom.press