Crypto Forum
Internet-Draft
Intended status: Informational
Expires: 5 March 2026

D. Connolly
SandboxAQ
P. Schwabe
MPI-SP & Radboud University
B. E. Westerbaan
Cloudflare

1 September 2025

X-Wing: general-purpose hybrid post-quantum KEM
draft-connolly-cfrg-xwing-kem-09

Abstract

This memo defines X-Wing, a general-purpose post-quantum/traditional hybrid key encapsulation mechanism (PQ/T KEM) built on X25519 and ML-KEM-768.

About This Document

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

The latest revision of this draft can be found at https://dconnolly.github.io/draft-connolly-cfrg-xwing-kem/draft-connolly-cfrg-xwing-kem.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-connolly-cfrg-xwing-kem/.

Discussion of this document takes place on the Crypto Forum Research Group mailing list (mailto:cfrg@ietf.org), which is archived at https://mailarchive.ietf.org/arch/search/?email_list=cfrg. Subscribe at https://www.ietf.org/mailman/listinfo/cfrg/.

Source for this draft and an issue tracker can be found at https://github.com/dconnolly/draft-connolly-cfrg-xwing-kem.

Status of This Memo

This Internet-Draft will expire on 5 March 2026.

Copyright Notice

Table of Contents

1. Introduction

1.1. Motivation

There are many choices that can be made when specifying a hybrid KEM: the constituent KEMs; their security levels; the combiner; and the hash within, to name but a few. Having too many similar options are a burden to the ecosystem.

The aim of X-Wing is to provide a concrete, simple choice for post-quantum hybrid KEM, that should be suitable for the vast majority of use cases.

1.2. Design goals

By making concrete choices, we can simplify and improve many aspects of X-Wing.

* Simplicity of definition. Because all shared secrets and cipher texts are fixed length, we do not need to encode the length. Using SHA3-256, we do not need HMAC-based construction. For the concrete choice of ML-KEM-768, we do not need to mix in its ciphertext, see Section 6.

* Security analysis. Because ML-KEM-768 already assumes the Quantum Random Oracle Model (QROM), we do not need to complicate the analysis of X-Wing by considering stronger models.

* Performance. Not having to mix in the ML-KEM-768 ciphertext is a nice performance benefit. Furthermore, by using SHA3-256 in the combiner, which matches the hashing in ML-KEM-768, this hash can be computed in one go on platforms where two-way Keccak is available.

We aim for "128 bits" security (NIST PQC level 1). Although at the moment there is no peer-reviewed evidence that ML-KEM-512 does not reach this level, we would like to hedge against future cryptanalytic improvements, and feel ML-KEM-768 provides a comfortable margin.

We aim for X-Wing to be usable for most applications, including specifically HPKE [RFC9180].

1.3. Not an interactive key-agreement

Traditionally most protocols use a Diffie-Hellman (DH) style non-interactive key-agreement. In many cases, a DH key agreement can be replaced by the interactive key-agreement afforded by a KEM without change in the protocol flow. One notable example is TLS [HYBRID] [XYBERTLS]. However, not all uses of DH can be replaced in a straight-forward manner by a plain KEM.

1.4. Not an authenticated KEM

In particular, X-Wing is not, borrowing the language of [RFC9180], an _authenticated_ KEM.

1.5. Comparisons

1.5.1. With HPKE X25519Kyber768Draft00

X-Wing is most similar to HPKE’s X25519Kyber768Draft00 [XYBERHPKE]. The key differences are:

* X-Wing uses the final version of ML-KEM-768.

* X-Wing hashes the shared secrets, to be usable outside of HPKE.

* X-Wing has a simpler combiner by flattening DHKEM(X25519) into the final hash.

* X-Wing does not hash in the ML-KEM-768 ciphertext.

There is also a different KEM called X25519Kyber768Draft00 [XYBERTLS] which is used in TLS. This one should not be used outside of TLS, as it assumes the presence of the TLS transcript to ensure non malleability.

1.5.2. With generic combiner

The generic combiner of [I-D.ounsworth-cfrg-kem-combiners] can be instantiated with ML-KEM-768 and DHKEM(X25519). That achieves similar security, but:

* X-Wing is more performant, not hashing in the ML-KEM-768 ciphertext, and flattening the DHKEM construction, with the same level of security.

* X-Wing has a fixed 32 byte shared secret, instead of a variable shared secret.

* X-Wing does not accept the optional counter and fixedInfo arguments.

2. Requirements Notation

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. Conventions and Definitions

This document is consistent with all terminology defined in [I-D.driscoll-pqt-hybrid-terminology].

The following terms are used throughout this document to describe these operations:

* concat(x0, ..., xN): returns the concatenation of byte strings. concat(0x01, 0x0203, 0x040506) = 0x010203040506.

* random(n): returns a byte string of length n bytes produced by a cryptographically-secure pseudorandom number generator.

4. Cryptographic Dependencies

X-Wing relies on the following primitives:

* ML-KEM-768 post-quantum key-encapsulation mechanism (KEM) [MLKEM]:

- ML-KEM-768.KeyGen_internal(d, z): Deterministic algorithm to generate an ML-KEM-768 key pair (pk_M, sk_M) of an encapsulation key pk_M and decapsulation key sk_M. It is the derandomized version of ML-KEM-768.KeyGen. Note that ML-KEM-768.KeyGen_internal() returns the keys in reverse order of GenerateKeyPair() defined below. d and z are both 32 byte strings.

- ML-KEM-768.Encaps(pk_M): Randomized algorithm to generate (ss_M, ct_M), an ephemeral 32 byte shared key ss_M, and a fixed-length encapsulation (ciphertext) of that key ct_M for encapsulation key pk_M.

  ML-KEM-768.Encaps(pk_M) MUST perform the encapsulation key check of [MLKEM] 則7.2 and raise an error if it fails.

- ML-KEM-768.Decap(ct_M, sk_M): Deterministic algorithm using the decapsulation key sk_M to recover the shared key from ct_M.

  ML-KEM-768.Decap(ct_M, sk_M) is NOT required to perform the decapsulation key check of [MLKEM] 則7.3.

To generate deterministic test vectors, we also use

- ML-KEM-768.Encaps_internal(pk_M, m): Algorithm to generate (ss_M, ct_M), an ephemeral 32 byte shared key ss_M, and a fixed-length encapsulation (ciphertext) of that key ct_M for encapsulation key pk_M. m is a 32 byte string.

  ML-KEM-768.Encaps_internal(pk_M) MUST perform the encapsulation key check of [MLKEM] 則7.2 and raise an error if it fails.

* X25519 elliptic curve Diffie-Hellman key-exchange defined in Section 5 of [RFC7748]:

  - X25519(k, u): takes 32 byte strings k and u representing a Curve25519 scalar and the u-coordinate of a point respectively, and returns the 32 byte string representing the u-coordinate of their scalar multiplication.

  - X25519_BASE: the 32 byte string representing the standard base point of Curve25519. In hexadecimal, it is given by 0900000000 000000000000000000000000000000000000000000000000000000.

Note that 9 is the standard basepoint for X25519, cf Section 6.1 of [RFC7748].

* Symmetric cryptography.

- SHAKE256(message, outlen): The extendable-output function (XOF) with that name defined in Section 6.2 of [FIPS202]. Note that outlen counts bits.

- SHA3-256(message): The hash with that name defined in Section 6.1 of [FIPS202].

5. X-Wing Construction

5.1. Encoding and sizes

X-Wing encapsulation key, decapsulation key, ciphertexts and shared secrets are all fixed-length byte strings.

Decapsulation key (private): 32 bytes

Encapsulation key (public): 1216 bytes

Ciphertext: 1120 bytes

Shared secret: 32 bytes

5.2. Key generation

An X-Wing keypair (decapsulation key, encapsulation key) is generated as follows.

```
def expandDecapsulationKey(sk):
  expanded = SHAKE256(sk, 96*8) # expand sk to 96 bytes using SHAKE256
  (pk_M, sk_M) = ML-KEM-768.KeyGen_internal(expanded[0:32], expanded[32:64])
  sk_X = expanded[64:96]
  pk_X = X25519(sk_X, X25519_BASE)
  return (sk_M, sk_X, pk_M, pk_X)

def GenerateKeyPair():
  sk = random(32)
  (sk_M, sk_X, pk_M, pk_X) = expandDecapsulationKey(sk)
  return sk, concat(pk_M, pk_X)
```

GenerateKeyPair() returns the 32 byte secret decapsulation key sk and the 1216 byte encapsulation key pk.

Here and in the balance of the document for clarity we use the M and X subscripts for ML-KEM-768 and X25519 components respectively.

5.2.1. Key derivation

For testing, it is convenient to have a deterministic version of key generation. An X-Wing implementation MAY provide the following derandomized variant of key generation.

```
def GenerateKeyPairDerand(sk):
  sk_M, sk_X, pk_M, pk_X = expandDecapsulationKey(sk)
  return sk, concat(pk_M, pk_X)
```

sk must be 32 bytes.

GenerateKeyPairDerand() returns the 32 byte secret encapsulation key sk and the 1216 byte decapsulation key pk.

Note GenerateKeyPair() is the same as GenerateKeyPairDerand(random(32)).

5.3. Combiner

Given 32 byte strings ss_M, ss_X, ct_X, pk_X, representing the ML-KEM-768 shared secret, X25519 shared secret, X25519 ciphertext (ephemeral public key) and X25519 public key respectively, the 32 byte combined shared secret is given by:

```
def Combiner(ss_M, ss_X, ct_X, pk_X):
  return SHA3-256(concat(
    ss_M,
    ss_X,
    ct_X,
    pk_X,
    XWingLabel
  ))
```

where XWingLabel is the following 6 byte ASCII string

```
XWingLabel = concat(
    "\./",
    "/^\",
)
```

In hexadecimal, XWingLabel is given by 5c2e2f2f5e5c.

5.4. Encapsulation

Given an X-Wing encapsulation key pk, encapsulation proceeds as follows.

```
def Encapsulate(pk):
  pk_M = pk[0:1184]
  pk_X = pk[1184:1216]
  ek_X = random(32)
  ct_X = X25519(ek_X, X25519_BASE)
  ss_X = X25519(ek_X, pk_X)
  (ss_M, ct_M) = ML-KEM-768.Encaps(pk_M)
  ss = Combiner(ss_M, ss_X, ct_X, pk_X)
  ct = concat(ct_M, ct_X)
  return (ss, ct)
```

pk is a 1216 byte X-Wing encapsulation key resulting from GeneratePublicKey()

Encapsulate() returns the 32 byte shared secret ss and the 1120 byte ciphertext ct.

Note that Encapsulate() may raise an error if the ML-KEM encapsulation does not pass the check of [MLKEM] 則7.2.

5.4.1. Derandomized

For testing, it is convenient to have a deterministic version of encapsulation. An X-Wing implementation MAY provide the following derandomized function.

```
def EncapsulateDerand(pk, eseed):
  pk_M = pk[0:1184]
  pk_X = pk[1184:1216]
  ek_X = eseed[32:64]
  ct_X = X25519(ek_X, X25519_BASE)
  ss_X = X25519(ek_X, pk_X)
  (ss_M, ct_M) = ML-KEM-768.EncapsDerand(pk_M, eseed[0:32])
  ss = Combiner(ss_M, ss_X, ct_X, pk_X)
  ct = concat(ct_M, ct_X)
  return (ss, ct)
```

pk is a 1216 byte X-Wing encapsulation key resulting from GeneratePublicKey() eseed MUST be 64 bytes.

EncapsulateDerand() returns the 32 byte shared secret ss and the 1120 byte ciphertext ct.

Encapsulate(pk) can be implemented as EncapsulateDerand(pk,random(64)).

5.5. Decapsulation

```
def Decapsulate(ct, sk):
  (sk_M, sk_X, pk_M, pk_X) = expandDecapsulationKey(sk)
  ct_M = ct[0:1088]
  ct_X = ct[1088:1120]
  ss_M = ML-KEM-768.Decapsulate(ct_M, sk_M)
  ss_X = X25519(sk_X, ct_X)
  return Combiner(ss_M, ss_X, ct_X, pk_X)
```

ct is the 1120 byte ciphertext resulting from Encapsulate() sk is a 32 byte X-Wing decapsulation key resulting from GenerateKeyPair()

Decapsulate() returns the 32 byte shared secret.

5.5.1. Keeping expanded decapsulation key around

For efficiency, an implementation MAY cache the result of expandDecapsulationKey. This is useful in two cases:

1. If multiple ciphertexts for the same key are decapsulated.

2. If a ciphertext is decapsulated for a key that has just been generated. This happen on the client-side for TLS.

A typical API pattern to achieve this optimization is to have an opaque decapsulation key object that hides the cached values. For instance, such an API could have the following functions.

1. GenerateKeyPair() returns an encapsulation key and an opaque object that contains the expanded decapsulation key.

2. Decapsulate(ct, esk) takes a ciphertext and an expanded decapsulation key.

3. PackDecapsulationKey(sk) takes an expanded decapsulation key, and returns the packed decapsulation key.

4. UnpackDecapsulationKey(sk) takes a packed decapsulation key, and returns the expanded decapsulation key. In the case of X-Wing this would be the same as a derandomized GenerateKeyPair().

The expanded decapsulation key could cache even more computation, such as the expanded matrix A in ML-KEM.

Any such expanded decapsulation key MUST NOT be transmitted between implementations, as this could break the security analysis of X-Wing. In particular, the MAL-BIND-K-PK and MAL-BIND-K-CT binding properties of X-Wing do not hold when transmitting the regular ML-KEM decapsulation key.

5.6. Use in HPKE

X-Wing satisfies the HPKE KEM interface as follows.

The SerializePublicKey, SerializePrivateKey, and DeserializePrivateKey are the identity functions, as X-Wing keys are fixed-length byte strings, see Section 5.1.

DeriveKeyPair() is given by

```
def DeriveKeyPair(ikm):
  # Extract 32-byte seed from variable-length ikm using SHAKE.
  sk = SHAKE256(ikm, 32*8)
  return GenerateKeyPairDerand(sk)
```

where the HPKE private key and public key are the X-Wing decapsulation key and encapsulation key respectively.

Encap() is Encapsulate() from Section 5.4, where an ML-KEM encapsulation key check failure causes an HPKE EncapError.

Decap() is Decapsulate() from Section 5.5.

X-Wing is not an authenticated KEM: it does not support AuthEncap() and AuthDecap(), see Section 1.4.

Nsecret, Nenc, Npk, and Nsk are defined in Section 7.

5.7. Use in TLS 1.3

For the client’s share, the key_exchange value contains the X-Wing encapsulation key.

For the server’s share, the key_exchange value contains the X-Wing ciphertext.

On ML-KEM encapsulation key check failure, the server MUST abort with an illegal_parameter alert.

5.8. Use in X.509 Public Key Infrastructure

We use the OID 1.3.6.1.4.1.62253.25722 to identify X-Wing keys as described below. In ASN.1 notation:

```
id-XWing OBJECT IDENTIFIER ::= { iso(1) identified-organization(3)
        dod(6) internet(1) private(4) enterprise(1) 62253 25722 }
```

5.8.1. Certificate

In a X.509 certificate, the subjectPublicKeyInfo field has the SubjectPublicKeyInfo type, which has the following ASN.1 syntax.

```
SubjectPublicKeyInfo  ::=  SEQUENCE  {
      algorithm         AlgorithmIdentifier,
      subjectPublicKey  BIT STRING
  }

AlgorithmIdentifier{ALGORITHM-TYPE, ALGORITHM-TYPE:AlgorithmSet} ::=
  SEQUENCE {
    algorithm   ALGORITHM-TYPE."&"id({AlgorithmSet}),
    parameters  ALGORITHM-TYPE.
                  "&"Params({AlgorithmSet}{@algorithm}) OPTIONAL
  }
```

An X-Wing encapsulation key MUST be encoded directly using the ASN.1 type XWingPublicKey.

```
XWingPublicKey ::= OCTET STRING
```

The X-Wing encapsulation key is mapped to a subjectPublicKey (a value of type BIT STRING) as follows: the most significant bit of the OCTET STRING value becomes the most significant bit of the BIT STRING value, and so on; the least significant bit of the OCTET STRING becomes the least significant bit of the BIT STRING.

The id-XWing identifier MUST be used as the algorithm field in the SubjectPublicKeyInfo to identify an X-Wing encapsulation key.

The contents of the parameters component MUST be absent.

5.8.2. Private key

Below we replicate part of the definition of OneAsymmetricKey from [RFC5958].

```
OneAsymmetricKey ::= SEQUENCE {
  version                  Version,
  privateKeyAlgorithm      SEQUENCE {
  algorithm                PUBLIC-KEY.&id({PublicKeySet}),
  parameters               PUBLIC-KEY.&Params({PublicKeySet}
                             {@privateKeyAlgorithm.algorithm})
                                OPTIONAL}
  privateKey               OCTET STRING (CONTAINING
                             PUBLIC-KEY.&PrivateKey({PublicKeySet}
                               {@privateKeyAlgorithm.algorithm})),
  attributes           [0] Attributes OPTIONAL,
  ...,
  [[2: publicKey       [1] BIT STRING (CONTAINING
                             PUBLIC-KEY.&Params({PublicKeySet}
                               {@privateKeyAlgorithm.algorithm})
                               OPTIONAL,
  ...
}
```

When storing an X-Wing decapsulation key in a OneAsymmetricKey, the privateKey OCTET STRING contains the raw octet string encoding the X-Wing decapsulation key.

The id-XWing identifier MUST be used as the algorithm field in the OneAsymmetricKey to identify an X-Wing decapsulation key.

The publicKey component MUST be absent.

6. Security Considerations

Informally, X-Wing is secure if SHA3 is secure, and either X25519 is secure, or ML-KEM-768 is secure.

More precisely, if SHA3-256, SHA3-512, and SHAKE-256 may be modelled as a random oracle, then the IND-CCA security of X-Wing is bounded by the IND-CCA security of ML-KEM-768, and the gap-CDH security of Curve25519, see [PROOF].

The security of X-Wing relies crucially on the specifics of the Fujisaki-Okamoto transformation used in ML-KEM-768: the X-Wing combiner cannot be assumed to be secure, when used with different KEMs. In particular it is not known to be safe to leave out the post-quantum ciphertext from the combiner in the general case.

6.1. Binding properties

Some protocols rely on further properties of the KEM. X-Wing satisfies the binding properties MAL-BIND-K-PK and MAL-BIND-K-CT (TODO: reference to proof). This implies [KSMW] X-Wing also satisfies

* MAL-BIND-K,CT-PK
* MAL-BIND-K,PK-CT
* LEAK-BIND-K-PK
* LEAK-BIND-K-CT
* LEAK-BIND-K,CT-PK
* LEAK-BIND-K,PK-CT
* HON-BIND-K-PK
* HON-BIND-K-CT
* HON-BIND-K,CT-PK
* HON-BIND-K,PK-CT

In contrast, ML-KEM on its own does not achieve MAL-BIND-K-PK, MAL-BIND-K-CT, nor MAL-BIND-K,PK-CT. [SCHMIEG]

7. IANA Considerations

This document requests/registers a new entry to the "HPKE KEM Identifiers" registry.

Value: 25722 = 25519 + 203 = 0x647a (please)

KEM: X-Wing

Nsecret: 32

Nenc: 1120

Npk: 1216

Nsk: 32

Auth: no

Reference: This document

Furthermore, this document requests/registers a new entry to the TLS Named Group (or Supported Group) registry, according to the procedures in Section 6 of [TLSIANA].

Value: 25722 = 25519 + 203 = 0x647a (please)

Description: X-Wing

DTLS-OK: Y

Recommended: N

Reference: This document

Comment: PQ/T hybrid of X25519 and ML-KEM-768

Finally, for the ASN.1 module in {asn1}, IANA is requested to assign an object identifier (OID) for the module identifier (TBD) with a Description of "id-mod-XWing-kem-2024".

8. References

8.1. Normative References

[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>.

[RFC5958] Turner, S., "Asymmetric Key Packages", RFC 5958, DOI 10.17487/RFC5958, August 2010, <https://www.rfc-editor.org/rfc/rfc5958>.

[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>.

[X680] ITU-T, "Information Technology -- Abstract Syntax Notation One (ASN.1): Specification of basic notation", ITU-T Recommendation X.680, ISO/IEC 8824-1:2021, February 2021, <https://www.itu.int/rec/T-REC-X.680>.

8.2. Informative References

[FIPS202] National Institute of Standards and Technology, "FIPS 202: SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions", n.d., <https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf>.

[HYBRID] Stebila, D., Fluhrer, S., and S. Gueron, "Hybrid key exchange in TLS 1.3", Work in Progress, Internet-Draft, draft-stebila-tls-hybrid-design-03, 12 February 2020, <https://datatracker.ietf.org/doc/html/draft-stebila-tls-hybrid-design-03>.

[I-D.driscoll-pqt-hybrid-terminology]
D, F., "Terminology for Post-Quantum Traditional Hybrid Schemes", Work in Progress, Internet-Draft, draft-driscoll-pqt-hybrid-terminology-02, 7 March 2023, <https://datatracker.ietf.org/doc/html/draft-driscoll-pqt-hybrid-terminology-02>.

[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>.

[KSMW] Kraemer, J., Struck, P., and M. Weishaupl, "Binding Security of Implicitly-Rejecting KEMs and Application to BIKE and HQC", n.d., <https://eprint.iacr.org/2024/1233>.

[MLKEM] National Institute of Standards and Technology, "FIPS 203: Module-Lattice-Based Key-Encapsulation Mechanism Standard", n.d., <https://nvlpubs.nist.gov/nistpubs/fips/nist.fips.203.pdf>.

[PROOF] Barbosa, M., Connolly, D., Duarte, J., Kaiser, A., Schwabe, P., Varner, K., and B. E. Westerbraan, "X-Wing: The Hybrid KEM You’ve Been Looking For", n.d., <https://eprint.iacr.org/2024/039>.

[RFC7748] Langley, A., Hamburg, M., and S. Turner, "Elliptic Curves for Security", RFC 7748, DOI 10.17487/RFC7748, January 2016, <https://www.rfc-editor.org/rfc/rfc7748>.

[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>.

[SCHMIEG] Schmieg, S., "Unbindable Kemmy Schmidt: ML-KEM is neither MAL-BIND-K-CT nor MAL-BIND-K-PK", n.d., <https://eprint.iacr.org/2024/523>.

[TLSIANA] Salowey, J. A. and S. Turner, "IANA Registry Updates for TLS and DTLS", Work in Progress, Internet-Draft, draft-ietf-tls-rfc8447bis-15, 21 July 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-tls-rfc8447bis-15>.

[XYBERHPKE] Westerbaan, B. and C. A. Wood, "X25519Kyber768Draft00 hybrid post-quantum KEM for HPKE", Work in Progress, Internet-Draft, draft-westerbaan-cfrg-hpke-xyber768d00-03, 14 May 2024, <https://datatracker.ietf.org/doc/html/draft-westerbaan-cfrg-hpke-xyber768d00-03>.

[XYBERTLS] Westerbaan, B. and D. Stebila, "X25519Kyber768Draft00 hybrid post-quantum key agreement", Work in Progress, Internet-Draft, draft-tls-westerbaan-xyber768d00-03, 24 September 2023, <https://datatracker.ietf.org/doc/html/draft-tls-westerbaan-xyber768d00-03>.

Appendix A. Implementations

* Platform
  - Apple CryptoKit (https://developer.apple.com/documentation/cryptokit/xwingmlkem768x25519)
* C
  - Google’s BoringSSL (https://github.com/google/boringssl/blob/main/include/openssl/xwing.h)
* Go
  - Cloudflare’s CIRCL (https://github.com/cloudflare/circl/pull/471)
  - Filippo (https://github.com/FiloSottile/mlkem768)
* Rust
  - xwing-kem.rs (https://github.com/rugo/xwing-kem.rs)

Note: implements the older -00 version of this memo at the time of writing.

- RustCrypto x-wing (https://github.com/RustCrypto/KEMs/tree/master/x-wing)

- Orion (https://github.com/orion-rs/orion)

Appendix B. Machine-readable specification

For the convenience of implementors, we provide a reference specification in Python. This is a specification; not production ready code: it should not be deployed as-is, as it leaks the private key by its runtime.

B.1. xwing.py

```
# WARNING This is a specification of X-Wing; not a production-ready
# implementation. It is slow and does not run in constant time.

# Requires the CryptoDome for SHAKE, and pytest for testing. To install, run
#
#   pip install pycryptodome pytest

import binascii
import hashlib

import mlkem
import x25519

XWingLabel = br"""
                \./
                /^\
              """.replace(b'\n', b'').replace(b' ', b'')

assert len(XWingLabel) == 6
assert binascii.hexlify(XWingLabel) == b'5c2e2f2f5e5c'

def expandDecapsulationKey(seed):
    expanded = hashlib.shake_256(seed).digest(length=96)
    pkM, skM = mlkem.KeyGen(expanded[0:64], mlkem.params768)
    skX = expanded[64:96]
    pkX = x25519.X(skX, x25519.BASE)
    return skM, skX, pkM, pkX

def GenerateKeyPairDerand(seed):
    assert len(seed) == 32
    skM, skX, pkM, pkX = expandDecapsulationKey(seed)
```

```
    return seed, pkM + pkX

def Combiner(ssM, ssX, ctX, pkX):
    return hashlib.sha3_256(
        ssM +
        ssX +
        ctX +
        pkX +
        XWingLabel
    ).digest()

def EncapsulateDerand(pk, eseed):
    assert len(eseed) == 64
    assert len(pk) == 1216
    pkM = pk[0:1184]
    pkX = pk[1184:1216]
    ekX = eseed[32:64]
    ctX = x25519.X(ekX, x25519.BASE)
    ssX = x25519.X(ekX, pkX)
    ctM, ssM = mlkem.Enc(pkM, eseed[0:32], mlkem.params768)
    ss = Combiner(ssM, ssX, ctX, pkX)
    return ss, ctM + ctX

def Decapsulate(ct, sk):
    assert len(ct) == 1120
    assert len(sk) == 32
    ctM = ct[0:1088]
    ctX = ct[1088:1120]
    skM, skX, pkM, pkX = expandDecapsulationKey(sk)
    ssM = mlkem.Dec(skM, ctM, mlkem.params768)
    ssX = x25519.X(skX, ctX)
    return Combiner(ssM, ssX, ctX, pkX)
```

B.2. x25519.py

```
# WARNING This is a specification of X25519; not a production-ready
# implementation. It is slow and does not run in constant time.

p = 2**255 - 19
a24 = 121665

BASE = b'\x09' + b'\x00'*31

def decode(bs):
    return sum(bs[i] << 8*i for i in range(32)) % p

def decodeScalar(k):
    bs = list(k)
```

```
        bs[0] &= 248
        bs[31] &= 127
        bs[31] |= 64
        return decode(bs)

    # See rfc7748 §5.
    def X(k, u):
        assert len(k) == 32
        assert len(u) == 32

        k = decodeScalar(k)
        u = decode(u)
        x1, x2, x3, z2, z3, swap = u, 1, u, 0, 1, 0

        for t in range(255, -1, -1):
            kt = (k >> t) & 1
            swap ^= kt
            if swap == 1:
                x3, x2 = x2, x3
                z3, z2 = z2, z3
            swap = kt

            A = x2 + z2
            AA = (A*A) % p
            B = x2 - z2
            BB = (B*B) % p
            E = AA - BB
            C = x3 + z3
            D = x3 - z3
            DA = (D*A) % p
            CB = (C*B) % p
            x3 = DA + CB
            x3 = (x3 * x3) % p
            z3 = DA - CB
            z3 = (x1 * z3 * z3) % p
            x2 = (AA * BB) % p
            z2 = (E * (AA + (a24 * E) % p)) % p

        if swap == 1:
            x3, x2 = x2, x3
            z2, z3 = z3, z2

        ret = (x2 * pow(z2, p-2, p)) % p
        return bytes((ret >> 8*i) & 255 for i in range(32))
```

B.3. mlkem.py

```
# WARNING This is a specification of Kyber; not a production ready
# implementation. It is slow and does not run in constant time.

# Requires the CryptoDome for SHAKE. To install, run
#
#   pip install pycryptodome pytest
from Crypto.Hash import SHAKE128, SHAKE256

import io
import hashlib
import functools
import collections

from math import floor

q = 3329
nBits = 8
zeta = 17
eta2 = 2

n = 2**nBits
inv2 = (q+1)//2 # inverse of 2

params = collections.namedtuple('params', ('k', 'du', 'dv', 'eta1'))

params512  = params(k = 2, du = 10, dv = 4, eta1 = 3)
params768  = params(k = 3, du = 10, dv = 4, eta1 = 2)
params1024 = params(k = 4, du = 11, dv = 5, eta1 = 2)

def smod(x):
    r = x % q
    if r > (q-1)//2:
        r -= q
    return r

# Rounds to nearest integer with ties going up
def Round(x):
    return int(floor(x + 0.5))

def Compress(x, d):
    return Round((2**d / q) * x) % (2**d)

def Decompress(y, d):
    assert 0 <= y and y <= 2**d
    return Round((q / 2**d) * y)

def BitsToWords(bs, w):
    assert len(bs) % w == 0
```

```
    return [sum(bs[i+j] * 2**j for j in range(w))
            for i in range(0, len(bs), w)]

def WordsToBits(bs, w):
    return sum([[(b >> i) % 2 for i in range(w)] for b in bs], [])

def Encode(a, w):
    return bytes(BitsToWords(WordsToBits(a, w), 8))

def Decode(a, w):
    return BitsToWords(WordsToBits(a, 8), w)

def brv(x):
    """ Reverses a 7-bit number """
    return int(''.join(reversed(bin(x)[2:].zfill(nBits-1))), 2)

class Poly:
    def __init__(self, cs=None):
        self.cs = (0,)*n if cs is None else tuple(cs)
        assert len(self.cs) == n

    def __add__(self, other):
        return Poly((a+b) % q for a,b in zip(self.cs, other.cs))

    def __neg__(self):
        return Poly(q-a for a in self.cs)
    def __sub__(self, other):
        return self + -other

    def __str__(self):
        return f"Poly({self.cs}"

    def __eq__(self, other):
        return self.cs == other.cs

    def NTT(self):
        cs = list(self.cs)
        layer = n // 2
        zi = 0
        while layer >= 2:
            for offset in range(0, n-layer, 2*layer):
                zi += 1
                z = pow(zeta, brv(zi), q)

                for j in range(offset, offset+layer):
                    t = (z * cs[j + layer]) % q
                    cs[j + layer] = (cs[j] - t) % q
                    cs[j] = (cs[j] + t) % q
```

```
            layer //= 2
        return Poly(cs)

    def RefNTT(self):
        # Slower, but simpler, version of the NTT.
        cs = [0]*n
        for i in range(0, n, 2):
            for j in range(n // 2):
                z = pow(zeta, (2*brv(i//2)+1)*j, q)
                cs[i] = (cs[i] + self.cs[2*j] * z) % q
                cs[i+1] = (cs[i+1] + self.cs[2*j+1] * z) % q
        return Poly(cs)

    def InvNTT(self):
        cs = list(self.cs)
        layer = 2
        zi = n//2
        while layer < n:
            for offset in range(0, n-layer, 2*layer):
                zi -= 1
                z = pow(zeta, brv(zi), q)

                for j in range(offset, offset+layer):
                    t = (cs[j+layer] - cs[j]) % q
                    cs[j] = (inv2*(cs[j] + cs[j+layer])) % q
                    cs[j+layer] = (inv2 * z * t) % q
            layer *= 2
        return Poly(cs)

    def MulNTT(self, other):
        """ Computes self o other, the multiplication of self and other
            in the NTT domain. """
        cs = [None]*n
        for i in range(0, n, 2):
            a1 = self.cs[i]
            a2 = self.cs[i+1]
            b1 = other.cs[i]
            b2 = other.cs[i+1]
            z = pow(zeta, 2*brv(i//2)+1, q)
            cs[i] = (a1 * b1 + z * a2 * b2) % q
            cs[i+1] = (a2 * b1 + a1 * b2) % q
        return Poly(cs)

    def Compress(self, d):
        return Poly(Compress(c, d) for c in self.cs)

    def Decompress(self, d):
        return Poly(Decompress(c, d) for c in self.cs)
```

```
    def Encode(self, d):
        return Encode(self.cs, d)

def sampleUniform(stream):
    cs = []
    while True:
        b = stream.read(3)
        d1 = b[0] + 256*(b[1] % 16)
        d2 = (b[1] >> 4) + 16*b[2]
        assert d1 + 2**12 * d2 == b[0] + 2**8 * b[1] + 2**16*b[2]
        for d in [d1, d2]:
            if d >= q:
                continue
            cs.append(d)
            if len(cs) == n:
                return Poly(cs)

def CBD(a, eta):
    assert len(a) == 64*eta
    b = WordsToBits(a, 8)
    cs = []
    for i in range(n):
        cs.append((sum(b[:eta]) - sum(b[eta:2*eta])) % q)
        b = b[2*eta:]
    return Poly(cs)

def XOF(seed, j, i):
    h = SHAKE128.new()
    h.update(seed + bytes([j, i]))
    return h

def PRF1(seed, nonce):
    assert len(seed) == 32
    h = SHAKE256.new()
    h.update(seed + bytes([nonce]))
    return h

def PRF2(seed, msg):
    assert len(seed) == 32
    h = SHAKE256.new()
    h.update(seed + msg)
    return h.read(32)

def G(seed):
    h = hashlib.sha3_512(seed).digest()
    return h[:32], h[32:]

def H(msg): return hashlib.sha3_256(msg).digest()
```

```
class Vec:
    def __init__(self, ps):
        self.ps = tuple(ps)

    def NTT(self):
        return Vec(p.NTT() for p in self.ps)

    def InvNTT(self):
        return Vec(p.InvNTT() for p in self.ps)

    def DotNTT(self, other):
        """ Computes the dot product <self, other> in NTT domain. """
        return sum((a.MulNTT(b) for a, b in zip(self.ps, other.ps)),
                   Poly())

    def __add__(self, other):
        return Vec(a+b for a,b in zip(self.ps, other.ps))

    def Compress(self, d):
        return Vec(p.Compress(d) for p in self.ps)

    def Decompress(self, d):
        return Vec(p.Decompress(d) for p in self.ps)

    def Encode(self, d):
        return Encode(sum((p.cs for p in self.ps), ()), d)

    def __eq__(self, other):
        return self.ps == other.ps

def EncodeVec(vec, w):
    return Encode(sum([p.cs for p in vec.ps], ()), w)
def DecodeVec(bs, k, w):
    cs = Decode(bs, w)
    return Vec(Poly(cs[n*i:n*(i+1)]) for i in range(k))
def DecodePoly(bs, w):
    return Poly(Decode(bs, w))

class Matrix:
    def __init__(self, cs):
        """ Samples the matrix uniformly from seed rho """
        self.cs = tuple(tuple(row) for row in cs)

    def MulNTT(self, vec):
        """ Computes matrix multiplication A*vec in the NTT domain. """
        return Vec(Vec(row).DotNTT(vec) for row in self.cs)

    def T(self):
```

```
         """ Returns transpose of matrix """
         k = len(self.cs)
         return Matrix((self.cs[j][i] for j in range(k))
                       for i in range(k))

 def sampleMatrix(rho, k):
     return Matrix([[sampleUniform(XOF(rho, j, i))
             for j in range(k)] for i in range(k)])

 def sampleNoise(sigma, eta, offset, k):
     return Vec(CBD(PRF1(sigma, i+offset).read(64*eta), eta)
                for i in range(k))

 def constantTimeSelectOnEquality(a, b, ifEq, ifNeq):
     # WARNING! In production code this must be done in a
     # data-independent constant-time manner, which this implementation
     # is not. In fact, many more lines of code in this
     # file are not constant-time.
     return ifEq if a == b else ifNeq

 def InnerKeyGen(seed, params):
     assert len(seed) == 32
     rho, sigma = G(seed + bytes([params.k]))
     A = sampleMatrix(rho, params.k)
     s = sampleNoise(sigma, params.eta1, 0, params.k)
     e = sampleNoise(sigma, params.eta1, params.k, params.k)
     sHat = s.NTT()
     eHat = e.NTT()
     tHat = A.MulNTT(sHat) + eHat
     pk = EncodeVec(tHat, 12) + rho
     sk = EncodeVec(sHat, 12)
     return (pk, sk)

 def InnerEnc(pk, msg, seed, params):
     assert len(msg) == 32
     tHat = DecodeVec(pk[:-32], params.k, 12)
     if EncodeVec(tHat, 12) != pk[:-32]:
         raise Exception("ML-KEM public key not normalized")
     rho = pk[-32:]
     A = sampleMatrix(rho, params.k)
     r = sampleNoise(seed, params.eta1, 0, params.k)
     e1 = sampleNoise(seed, eta2, params.k, params.k)
     e2 = sampleNoise(seed, eta2, 2*params.k, 1).ps[0]
     rHat = r.NTT()
     u = A.T().MulNTT(rHat).InvNTT() + e1
     m = Poly(Decode(msg, 1)).Decompress(1)
     v = tHat.DotNTT(rHat).InvNTT() + e2 + m
     c1 = u.Compress(params.du).Encode(params.du)
```

```
     c2 = v.Compress(params.dv).Encode(params.dv)
     return c1 + c2

 def InnerDec(sk, ct, params):
     split = params.du * params.k * n // 8
     c1, c2 = ct[:split], ct[split:]
     u = DecodeVec(c1, params.k, params.du).Decompress(params.du)
     v = DecodePoly(c2, params.dv).Decompress(params.dv)
     sHat = DecodeVec(sk, params.k, 12)
     return (v - sHat.DotNTT(u.NTT()).InvNTT()).Compress(1).Encode(1)

 def KeyGen(seed, params):
     assert len(seed) == 64
     z = seed[32:]
     pk, sk2 = InnerKeyGen(seed[:32], params)
     h = H(pk)
     return (pk, sk2 + pk + h + z)

 def Enc(pk, seed, params):
     assert len(seed) == 32

     K, r = G(seed + H(pk))
     ct = InnerEnc(pk, seed, r, params)
     return (ct, K)

 def Dec(sk, ct, params):
     sk2 = sk[:12 * params.k * n//8]
     pk = sk[12 * params.k * n//8 : 24 * params.k * n//8 + 32]
     h = sk[24 * params.k * n//8 + 32 : 24 * params.k * n//8 + 64]
     z = sk[24 * params.k * n//8 + 64 : 24 * params.k * n//8 + 96]
     m2 = InnerDec(sk, ct, params)
     K2, r2 = G(m2 + h)
     ct2 = InnerEnc(pk, m2, r2, params)
     return constantTimeSelectOnEquality(
         ct2, ct,
         K2,                 # if ct == ct2
         PRF2(z, ct),        # if ct != ct2
     )
```

Appendix C. Test vectors # TODO: replace with test vectors that re-use ML-KEM, X25519 values

```
seed     7f9c2ba4e88f827d616045507605853ed73b8093f6efbc88eb1a6eacfa66ef26
sk     7f9c2ba4e88f827d616045507605853ed73b8093f6efbc88eb1a6eacfa66ef26
pk
  e2236b35a8c24b39b10aa1323a96a919a2ced88400633a7b07131713fc14b2b5b19cfc3d
  a5fa1a92c49f25513e0fd30d6b1611c9ab9635d7086727a4b7d21d34244e66969cf15b3b
  2a785329f61b096b277ea037383479a6b556de7231fe4b7fa9c9ac24c0699a0018a52534
```

```
   01bacfa905ca816573e56a2d2e067e9b7287533ba13a937dedb31fa44baced4076992361
   0034ae31e619a170245199b3c5c39864859fe1b4c9717a07c30495bdfb98a0a002ccf56c
   1286cef5041dede3c44cf16bf562c7448518026b3d8b9940680abd38a1575fd27b58da06
   3bfac32c39c30869374c05c1aeb1898b6b303cc68be455346ee0af699636224a148ca2ae
   a10463111c709f69b69c70ce8538746698c4c60a9aef0030c7924ceec42a5d36816f545e
   ae13293460b3acb37ea0e13d70e4aa78686da398a8397c08eaf96882113fe4f7bad4da40
   b0501e1c753efe73053c87014e8661c33099afe8bede414a5b1aa27d8392b3e131e9a70c
   1055878240cad0f40d5fe3cdf85236ead97e2a97448363b2808caafd516cd25052c5c362
   543c2517e4acd0e60ec07163009b6425fc32277acee71c24bab53ed9f29e74c66a0a3564
   955998d76b96a9a8b50d1635a4d7a67eb42df5644d330457293a8042f53cc7a69288f17e
   d55827e82b28e82665a86a14fbd96645eca8172c044f83bc0d8c0b4c8626985631ca87af
   829068f1358963cb333664ca482763ba3b3bb208577f9ba6ac62c25f76592743b64be519
   317714cb4102cb7b2f9a25b2b4f0615de31decd9ca55026d6da0b65111b16fe52feed8a4
   87e144462a6dba93728f500b6ffc49e515569ef25fed17aff52050736825352586 0f58be
   3be61c964604a6ac814e6935596402a520a4670b3d284318866593d15a4bb01c35e3e587
   ee0c67d2880d6f2407fb7a70712b838deb96c5d7bf2b44bcf6038ccbe33fbcf51a54a584
   fe90083c91c7a6d43d4fb15f48c60c2fd66e0a8aad4ad64e5c42bb8877c0ebec2b5e387c
   8a988fdc23beb9e16c8757781e0a1499c61e138c21f216c29d076979871caa6942bafc09
   0544bee99b54b16cb9a9a364d6246d9f42cce53c66b59c45c8f9ae9299a75d15180c3c95
   2151a91b7a10772429dc4cbae6fcc622fa8018c63439f890630b9928db6bb7f9438ae406
   5ed34d73d486f3f52f90f0807dc88dfdd8c728e954f1ac35c06c000ce41a0582580e3bb5
   7b672972890ac5e7988e7850657116f1b57d0809aaedec0bede1ae148148311c6f7e3173
   46e5189fb8cd635b986f8c0bdd27641c584b778b3a911a80be1c9692ab8e1bbb12839573
   cce19df183b45835bbb55052f9fc66a1678ef2a36dea78411e6c8d60501b4e60592d1369
   8a943b509185db912e2ea10be06171236b327c71716094c964a68b03377f513a05bcd99c
   1f346583bb052977a10a12adfc758034e5617da4c1276585e5774e1f3b9978b09d0e9c44
   d3bc86151c43aad185712717340223ac381d21150a04294e97bb13bbda21b5a182b6da96
   9e19a7fd072737fa8e880a53c2428e3d049b7d2197405296ddb361912a7bcf4827ced611
   d0c7a7da104dde4322095339f64a61d5bb108ff0bf4d780cae509fb22c256914193ff734
   9042581237d522828824ee3bdfd07fb03f1f942d2ea179fe722f06cc03de5b69859edb06
   eff389b27dce59844570216223593d4ba32d9abac8cd049040ef6534
eseed
   3cb1eea988004b93103cfb0aeefd2a686e01fa4a58e8a3639ca8a1e3f9ae57e235b8cc87
   3c23dc62b8d260169afa2f75ab916a58d974918835d25e6a435085b2
ct
   b83aa828d4d62b9a83ceffe1d3d3bb1ef31264643c070c5798927e41fb07914a273f8f96
   e7826cd5375a283d7da885304c5de0516a0f0654243dc5b97f8bfeb831f68251219aabdd
   723bc6512041acbaef8af44265524942b902e68ffd23221cda70b1b55d776a92d1143ea3
   a0c475f63ee6890157c7116dae3f62bf72f60acd2bb8cc31ce2ba0de364f52b8ed38c79d
   719715963a5dd3842d8e8b43ab704e4759b5327bf027c63c8fa857c4908d5a8a7b88ac7f
   2be394d93c3706ddd4e698cc6ce370101f4d0213254238b4a2e8821b6e414a1cf20f6c12
   44b699046f5a01caa0a1a55516300b40d2048c77cc73afba79afeea9d2c0118bdf2adb88
   70dc328c5516cc45b1a2058141039e2c90a110a9e16b318dfb53bd49a126d6b73f215787
   517b8917cc01cabd107d06859854ee8b4f9861c226d3764c87339ab16c3667d2f49384e5
   5456dd40414b70a6af841585f4c90c68725d57704ee8ee7ce6e2f9be582dbee985e038ff
   c346ebfb4e22158b6c84374a9ab4a44e1f91de5aac5197f89bc5e5442f51f9a5937b102b
   a3beaebf6e1c58380a4a5fedce4a4e5026f88f528f59ffd2db41752b3a3d90efabe46389
   9b7d40870c530c8841e8712b733668ed033adbfafb2d49d37a44d4064e5863eb0af0a08d
```

```
   47b3cc888373bc05f7a33b841bc2587c57eb69554e8a3767b7506917b6b70498727f16ea
   c1a36ec8d8cfaf751549f2277db277e8a55a9a5106b23a0206b4721fa9b3048552c5bd5b
   594d6e247f38c18c591aea7f56249c72ce7b117afcc3a8621582f9cf71787e183dee0936
   7976e98409ad9217a497df888042384d7707a6b78f5f7fb8409e3b535175373461b77600
   2d799cbad62860be70573ecbe13b246e0da7e93a52168e0fb6a9756b895ef7f0147a0dc8
   1bfa644b088a9228160c0f9acf1379a2941cd28c06ebc80e44e17aa2f8177010afd78a97
   ce0868d1629ebb294c5151812c583daeb88685220f4da9118112e07041fcc24d5564a99f
   dbde28869fe0722387d7a9a4d16e1cc8555917e09944aa5ebaaaec2cf62693afad42a3f5
   18fce67d273cc6c9fb5472b380e8573ec7de06a3ba2fd5f931d725b493026cb0acbd3fe6
   2d00e4c790d965d7a03a3c0b4222ba8c2a9a16e2ac658f572ae0e746eafc4feba023576f
   08942278a041fb82a70a595d5bacbf297ce2029898a71e5c3b0d1c6228b485b1ade509b3
   5fbca7eca97b2132e7cb6bc465375146b7dceac969308ac0c2ac89e7863eb8943015b243
   14cafb9c7c0e85fe543d56658c213632599efabfc1ec49dd8c88547bb2cc40c9d38cbd30
   99b4547840560531d0188cd1e9c23a0ebee0a03d5577d66b1d2bcb4baaf21cc7fef1e038
   06ca96299df0dfbc56e1b2b43e4fc20c37f834c4af62127e7dae86c3c25a2f696ac8b589
   dec71d595bfbe94b5ed4bc07d800b330796fda89edb77be0294136139354eb8cd3759157
   8f9c600dd9be8ec6219fdd507adf3397ed4d68707b8d13b24ce4cd8fb22851bfe9d63240
   7f31ed6f7cb1600de56f17576740ce2a32fc5145030145cfb97e63e0e41d354274a079d3
   e6fb2e15
ss     d2df0522128f09dd8e2c92b1e905c793d8f57a54c3da25861f10bf4ca613e384

seed     badfd6dfaac359a5efbb7bcc4b59d538df9a04302e10c8bc1cbf1a0b3a5120ea
sk     badfd6dfaac359a5efbb7bcc4b59d538df9a04302e10c8bc1cbf1a0b3a5120ea
pk
   0333285fa253661508c9fb444852caa4061636cb060e69943b431400134ae1fbc0228724
   7cb38068bbb89e6714af10a3fcda6613acc4b5e4b0d6eb960c302a0253b1f507b596f088
   4d351da89b01c35543214c8e542390b2bc497967961ef10286879c34316e6483b644fc27
   e8019d73024ba1d1cc83650bb068a5431b33d1221b3d122dc1239010a55cb13782140893
   f30aca7c09380255a0c621602ffbb6a9db064c1406d12723ab3bbe2950a21fe521b160b3
   0b16724cc359754b4c88342651333ea9412d5137791cf75558ebc5c54c520dd6c622a059
   f6b332ccebb9f24103e59a297cd69e4a48a3bfe53a5958559e840db5c023f66c10ce2308
   1c2c8261d744799ba078285cfa71ac51f44708d0a6212c3993340724b3ac38f63e82a889
   a4fc581f6b8353cc6233ac8f5394b6cca292f892360570a3031c90c4da3f02a895677390
   e60c24684a405f69ccf1a7b95312a47c844a4f9c2c4a37696dc10072a87bf41a2717d45b
   2a99ce09a4898d5a3f6b67085f9a626646bcf369982d483972b9cd7d244c4f49970f766a
   22507925eca7df99a491d80c27723e84c7b49b633a46b46785a16a41e02c538251622117
   364615d9c2cdaa1687a860c18bfc9ce8690efb2a524cb97cdfd1a4ea661fa7d08817998a
   f838679b07c9db8455e2167a67c14d6a347522e89e8971270bec858364b1c1023b82c483
   cf8a8b76f040fe41c24dec2d49f6376170660605b80383391c4abad1136d874a77ef73b4
   40758b6e7059add20873192e6e372e069c22c5425188e5c240cb3a6e29197ad17e87ec41
   a813af68531f262a6db25bbdb8a15d2ed9c9f35b9f2063890bd26ef09426f225aa1e6008
   d31600a29bcdf3b10d0bc72788d35e25f4976b3ca6ac7cbf0b442ae399b225d9714d0638
   a864bda7018d3b7c793bd2ace6ac68f4284d10977cc029cf203c5698f15a06b162d6c8b4
   fd40c6af40824f9c6101bb94e9327869ab7efd835dfc805367160d6c8571e3643ac70cba
   d5b96a1ad99352793f5af71705f95126cb4787392e94d808491a2245064ba5a7a30c0663
   01392a6c315336e10dbc9c2177c7af382765b6c88eeab51588d01d6a95747f3652dc5b5c
   401a23863c7a0343737c737c99287a40a90896d4594730b552b910d23244684206f0eb84
   2fb9aa316ab182282a75fb72b6806cea4774b822169c386a58773c3edc8229d85905abb8
```

```
   7ac228f0f7a2ce9a497bb5325e17a6a82777a997c036c3b862d29c14682ad325a9600872
   f3913029a1588648ba590a7157809ff740b5138380015c40e9fb90f0311107946f28e596
   2e21666ad65092a3a60480cd16e61ff7fb5b44b70cf12201878428ef8067fceb1e1dcb49
   d66c773d312c7e53238cb620e126187009472d41036b702032411dc96cb750631df9d994
   52e495deb4300df660c8d35f32b424e98c7ed14b12d8ab11a289ac63c50a24d52925950e
   49ba6bf4c2c38953c92d60b6cd034e575c711ac41bfa66951f62b9392828d7b45aed377a
   c69c35f1c6b80f388f34e0bb9ce8167eb2bc630382825c396a407e905108081b444ac8a0
   7c2507376a750d18248ee0a81c4318d9a38fc44c3b41e8681f87c34138442659512c4127
   6e1cc8fc4eb66e12727bcb5a9e0e405cdea21538d6ea885ab169050e6b91e1b69f7ed34b
   cbb48fd4c562a576549f85b528c953926d96ea8a160b8843f1c89c62
eseed
   17cda7cfad765f5623474d368ccca8af0007cd9f5e4c849f167a580b14aabdefaee7eef4
   7cb0fca9767be1fda69419dfb927e9df07348b196691abaeb580b32d
ct
   c93beb22326705699bbc3d1d0aa6339be7a405debe61a7c337e1a91453c097a6f77c1306
   39d1aaeb193175f1a987aa1fd789a63c9cd487ebd6965f5d8389c8d7c8cfacbba4b44d2f
   be0ae84de9e96fb11215d9b76acd51887b752329c1a3e0468ccc49392c1e0f1aad61a73c
   10831e60a9798cb2e7ec07596b5803db3e243ecbb94166feade0c9197378700f8eb65a43
   502bbac4605992e2de2b906ab30ba401d7e1ff3c98f42cfc4b30b974d3316f331461ac05
   f43e0db7b41d3da702a4f567b6ee7295199c7be92f6b4a47e7307d34278e03c872fb4864
   7c446a64a3937dccd7c6d8de4d34b9dea45a0b065ef15b9e94d1b6df6dca7174d9bc9d14
   c6225e3a78a58785c3fe4e2fe6a0706f3365389e4258fbb61ecf1a1957715982b3f18444
   24e03acd83da7eee50573f6cd3ff396841e9a00ad679da92274129da277833d0524674fe
   ea09a98d25b888616f338412d8e65e151e65736c8c6fb448c9260fa20e7b2712148bcd3a
   0853865f50c1fc9e4f201aee3757120e034fd509d954b7a749ff776561382c4cb64cebcb
   b6aa82d04cd5c2b40395ecaf231bde8334ecfd955d09efa8c6e7935b1cb0298fb8b6740b
   e4593360eed5f129d59d98822a6cea37c57674e919e84d6b90f695fca58e7d29092bd70f
   7c97c6dfb021b9f87216a6271d8b144a364d03b6bf084f972dc59800b14a2c008bbd0992
   b5b82801020978f2bdddb3ca3367d876cffb3548dab695a29882cae2eb5ba7c847c3c71b
   d0150fa9c33aac8e6240e0c269b8e295ddb7b77e9c17bd310be65e28c0802136d086777b
   e5652d6f1ac879d3263e9c712d1af736eac048fe848a577d6afaea1428dc71db8c430edd
   7b584ae6e6aeaf7257aff0fd8fe25c30840e30ccfa1d95118ef0f6657367e9070f3d97a2
   e9a7bae19957bd707b00e31b6b0ebb9d7df4bd22e44c060830a194b5b8288353255b5295
   4ff5905ab2b126d9aa049e44599368c27d6cb033eae5182c2e1504ee4e3745f51488997b
   8f958f0209064f6f44a7e4de5226d5594d1ad9b42ac59a2d100a2f190df873a2e141552f
   33c923b4c927e8747c6f830c441a8bd3c5b371f6b3ab8103ebcfb18543aefc1beb6f776b
   bfd5344779f4aa23daaf395f69ec31dc046b491f0e5cc9c651dfc306bd8f2105be7bc7a4
   f4e21957f87278c771528a8740a92e2daefa76a3525f1fae17ec4362a2700988001d8600
   11d6ca3a95f79a0205bcf634cef373a8ea273ff0f4250eb8617d0fb92102a6aa09cf0c3e
   e2cad1ad96438c8e4dfd6ee0fcc85833c3103dd6c1600cd305bc2df4cda89b55ca237a3f
   9c3f82390074ff30825fc750130ebaf13d0cf7556d2c52a98a4bad39ca5d44aaadeaef77
   5c695e64d06e966acfcd552a14e2df6c63ae541f0fa88fc48263089685704506a21a0385
   6ce65d4f06d54f3157eeabd62491cb4ac7bf029e79f9fbd4c77e2a3588790c710e611da8
   b2040c76a61507a8020758dcc30894ad018fef98e401cc54106e20d94bd544a8f0e1fd05
   00342d123f618aa8c91bdf6e0e03200693c9651e469aee6f91c98bea4127ae66312f4ae3
   ea155b67
ss     f2e86241c64d60f6649fbc6c5b7d17180b780a3f34355e64a85749949c45f150
```

```
seed     ef58538b8d23f87732ea63b02b4fa0f4873360e2841928cd60dd4cee8cc0d4c9
sk     ef58538b8d23f87732ea63b02b4fa0f4873360e2841928cd60dd4cee8cc0d4c9
pk
  36244278824f77c621c660892c1c3886a9560caa52a97c461fd3958a598e749bbc8c7798
  ac8870bac7318ac2b863000ca3b0bdcbbc1ccfcb1a30875df9a76976763247083e646ccb
  2499a4e4f0c9f4125378ba3da1999538b86f99f2328332c177d1192b849413e655101289
  73f679d23253850bb6c347ba7ca81b5e6ac4c574565c731740b3cd8c9756caac39fba7ac
  422acc60c6c1a645b94e3b6d21485ebad9c4fe5bb4ea0853670c5246652bff65ce8381cb
  473c40c1a0cd06b54dcec11872b351397c0eaf995bebdb6573000cbe2496600ba76c8cb0
  23ec260f0571e3ec12a9c82d9db3c57b3a99e8701f78db4fabc1cc58b1bae02745073a81
  fc8045439ba3b885581a283a1ba64e103610aabb4ddfe9959e7241011b2638b56ba6a982
  ef610c514a57212555db9a98fb6bcf0e91660ec15dfa66a67408596e9ccb97489a09a073
  ffd1a0a7ebbe71aa5ff793cb91964160703b4b6c9c5390842c2c905d4a9f88111fed5787
  4ba9b03cf611e70486edf539767c7485189d5f1b08e32a274dc24a39c918fd2a4dfa946a
  8c897486f2c974031b2804aabc81749db430b85311372a3b8478868200b40e043f7bf4a1
  c3a08b0771b431e342ee277410bca034a0c77086c8f702b3aed2b4108bbd3af471633373
  a1ac74b128b148d1b9412aa66948cac6dc6614681fda02ca86675d2a756003c49c50f06e
  13c63ce4bc9f321c860b202ee931834930011f485c9af86b9f642f0c353ad305c66996b9
  a136b753973929495f0d8048db75529edcb4935904797ac66605490f66329c3bb36b8573
  a3e00f817b3082162ff106674d11b261baae0506cde7e69fdce93c6c7b59b9d4c759758a
  cf287c2e4c4bfab5170a9236daf21bdb6005e92464ee8863f845cf37978ef19969264a51
  6fe992c93b5f7ae7cb6718ac69257d630379e4aac6029cb906f98d91c92d118c36a6d161
  15d4c8f16066078badd161a65ba51e0252bc358c67cd2c4beab2537e42956e08a39cfccf
  0cd875b5499ee952c83a162c68084f6d35cf92f71ec66baec74ab87e2243160b64df54af
  b5a07f78ec0f5c5759e5a4322bca2643425748a1a97c62108510c44fd9089c5a7c14e57b
  1b77532800013027cff91922d7c935b4202bb507aa47598a6a5a030117210d4c49c17470
  0550ad6f82ad40e965598b86bc575448eb19d70380d465c1f870824c026d74a2522a799b
  7b122d06c83aa64c0974635897261433914fdfb14106c230425a83dc8467ad8234f086c7
  2a47418be9cfb582b1dcfa3d9aa45299b79fff265356d8286a1ca2f3c2184b2a70d15289
  e5b202d03b64c735a867b1154c55533ff61d6c296277011848143bc85a4b823040ae025a
  29293ab77747d85310078682e0ba0ac236548d905a79494324574d417c7a3457bd5fb525
  3c4876679034ae844d0d05010fec722db5621e3a67a2d58e2ff33b432269169b51f9dcc0
  95b8406dc1864cf0aeb6a2132661a38d641877594b3c51892b9364d25c63d637140a2018
  d10931b0daa5a2f2a405017688c991e586b522f94b1132bc7e87a63246475816c8be9c62
  b731691ab912eb656ce2619225663364701a014b7d0337212caa2ecc731f34438289e0ca
  4590a276802d980056b5d0d316cae2ecfea6d86696a9f161aa90ad47eaad8cadd31ae3cb
  c1c013747dfee80fb35b5299f555dcc2b787ea4f6f16ffdf66952461
eseed
  22a96188d032675c8ac850933c7aff1533b94c834adbb69c6115bad4692d8619f90b0cdf
  8a7b9c264029ac185b70b83f2801f2f4b3f70c593ea3aeeb613a7f1b
ct
  0d2e38cbf17a2e2e4e0c87a94ca1e7701ae1552e02509b3b00f9c82c39e3fd435b05b912
  75f47abc9f1021429a26a346598cd6cd9efdc8adc1dbc35036d0290bf89733c835309202
  232f9bf652ea82f3d49280d6e8a3bd3135fb883445ab5b074d949c5350c7c7d6ac59905b
  dbfce6639da8a9d4b390ecc1dd05522d2956f2d37a05593996e5cb3fd8d5a9eb52417732
  e1ebf545588713b4760227115aab7ada178dadbca583b26cfedba2888a0c95b950bf07f7
  50d7aa8103798aa3470a042c0105c6a037de2f9ebc396021b2ba2c16aba696fbac3454dc
  8e053b8fa55edd45215eeb57a1eab9106fb426b375a9b9e5c3419efc7610977e72640f9f
```

```
  d1b2ec337de33c35e5a7581b2aae4d8ee86d2e0ebf82a1350714de50d2d788687878a196
  44ae4e3175e8d59dc90171b3badeff65aeaf600e5e5483a3595fdeb40cbafcbd040c29a2
  f6900533ae999d24f54dfcef748c30313ca447cdddfa57ad78eaa890e90f3f7bf8d11696
  8a5713cc75fd0408f36364fa265c5617039304eaeac4cbee6fc49b9fe2276768cdbec2d7
  3a507b543cc028dc1b154b7c2b0412254c466a94a8d6ea3a47e1743469bd45c08f54cf96
  5884be3696e961741ede16e3b1bc4feb93faaef31d911dc0cb3fa90bcda991959a9d2cbc
  817a5564c5c01177a59e9577589ea344d60cf5b0aa39f31863febd54603ca87ad2363c76
  6642a3f52557bcd9e4c05a87665842ba336b83156a677030f0bad531a8387a1486a599ca
  a748fcea7bdc1eb63f3cdb97173551ab7c1c36b69acbbdb2ff7a1e7bc70439632ddc67b9
  7f3da1f59b3c1588515957cb8a2f86ab635ce0a78b7cdf24eac3445e8fc8b79ba04da9e9
  03f49a7d912c197a84b4cfabc779b97d24788419bcf58035db99717edb9fd1c1df8c4005
  f700eabba528ddfcbaeda6dd30754f795948a34c9319ab653524b19931c7900c4167988a
  f52292fe902e746b524d20ceffb4339e8f5535f41cf35f0f8ea8b4a7b949c5d2381116b1
  46e9b913a83a3fa1c65ff9468c835fe4114554a6c66a80e1c9a6bb064b380be3c95e5595
  ec979bf1c85aa938938e3f10e72b0c87811969e8ab0d83de0b0604c4016ac3a015e19514
  089271bdc6ebf2ec56fab6018e44de749b4c36cc235e370da8466dbdc253542a2d704eb3
  316fd70d5d238cb7eaaf05966d973f62c7ef43b9a806f4ed213ac8099ea15d61a9024441
  60883f6bf441a3e1469945c9b79489ea18390f1ebc83caca10bdb8f2429877b52bd44c94
  a228ef91c392ef5398c5c83982701318ccedab92f7a279c4fddebaa7fe5e986c48b7d813
  5b3fe4cd15be2004ce73ff86b1e55f8ecd6ba5b8114315f8e716ef3ab0a64564a4644651
  166ebd68b1f783e2e443dbccadfe189368647629f1a12215840b7f1d026de2f665c2eb02
  3ff51a6df160912811ee03444ae4227fb941dc9ec4f31b445006fd384de5e60e0a5061b5
  0cb1202f863090fc05eb814e2d42a03586c0b56f533847ac7b8184ce9690bc8dece32a88
  ca934f541d4cc520fa64de6b6e1c3c8e03db5971a445992227c825590688d203523f5271
  61137334
ss     953f7f4e8c5b5049bdc771d1dffada0dd961477d1a2ae0988baa7ea6898d893f
```

Appendix D. Example of use in X.509

The following are the encodings of the X-Wing keypair with private key 0001⋯1f.

-----BEGIN PRIVATE KEY-----
MDQCAQAwDQYLKwYBBAGD5i2ByHoEIAABAgMEBQYHCAkKCwwNDg8QERITFBUWFxgZ
GhscHR4f
-----END PRIVATE KEY-----

```
-----BEGIN PUBLIC KEY-----
MIIE1DANBgsrBgEEAYPmLYHIegOCBMEAb1QJigoOZBFGYUtpYLpg2GA9YvRH+atJ
m0e9aQbMQLBh2GNKPoiQbyhJWOdEHKbHJcu5cJW3ZxpGK2aByeZYC7yNYLFJ+mAm
EEOvu6UvIFpgKDhIUVlq3zcavqmNM0c4PSu2c0OPZ4NhK/hwFPe5Gol0AmU0XfZ5
NARz0cTBdohuXim48Fi7fHNTFmhs/1w764wmHLAJcKacGvzFS5TLhuHOY7pjbjlc
pFEB4hx70EwxPqGa8kFB79KtREFqJbpPZZEO99iAnDCT8EqvAOPNluNcSqPIAsGK
1vOdpLS42YyL15Atg6B7pFOWZ0pgJDyrk+gP2bHId3N2qcwNb6EV4mOTgLnGvnhI
vRNYjGRwOgU10ZoPgWM6l2oKEFtm7ihdD9JV6CwDMZJfQ4O278dh72CZI1oLmHJj
WKqdAbi4llGfkhR0u3wUuyIlK1wvENQSRsmyPnZEhJNn9UGhX2O8koo5u3vHPwe2
ZcSWu2VYyPRUiacuxLrNNOnFlMM4cbcj8DSV6ItDkasm5DBD3rYRezkZ5FxMGxar
KOR93XI2Y4VHZhkvwYBspwq7eGy9swky5oyKNwvPsHmDoBLDJmuT76YmV/S4ODdM
sLuV4OwGVBsHZdmc8VO8a5YTXKeApVs2R3ieMZFeRig8+ce7boRT+2aCEFFB8dwN
ANhe7XA7bGyWH3nIRSdrQkiUnAZ4LlE+spkbldlgQuOMvto1JEmytQhOvaUiamIG
QAeJEwowlkSYSLYp/upKLCp0PEoN3Jyz89Z2/FY3MbJsShpm3IRZFwBW1XaX8UQ7
gamjRBK7e/BfMydXWlkR3TAdYFOGfzwwgHEfG/EVh7C7KYQnayaF53ViEOSz+JVT
hCMeVYxvUQyR4PxWtdGIX/KUnpWka8G+4fpx9QJ+EMRDsOkdD9dED0Z6JyISEuiP
XGumQpbK4NIHv8YPiMfPtcRaoYOdGMs3xFhD5UJqSpDIArZCj5U8NZxKwGA0UvrA
tzYeL9NdzIhakhRdT8oBWPG31wtLzRGOSipBVEON8xDESpobmepBWQcmeoiwYkJB
V5wXIvRu1hwuPspUXJlwUXF1OZuADbJdo5WT0GSQ1xQsAOiNLbBH6YmL23rLftkH
9uMEFswN5UokLAohJjAvXVTIW8Zqwvg8eXlFtQZ8qkK9LgwZypdQblB6sKXJ9WM3
CEmcGfJK7FE705A6XXO27EmR98cuuZHBw3iJgFyx6jigzAIXayfFjWOM5aMmaEV8
+bm+AnygIUBXlxcl1UEC6JlnFusq2CNFO2BbhVNwsbIbOTLN7UFgqplzx+uuWsR2
TZTPfMlQbwd7rXMBLbtKyBQKOHRkEuszyVFFliBfcHY1hiIX2bYJGMYmjZNEkVuE
eiR2waJw8VSlyEI0FlrPyGk5hwLOqemgfnsOmeqb3LeEH+nA+iXIM4CSVho+3dxw
AfR4rWV4GmAkqtFl2baXmtrESKRGL1ZGhVJ/diQ0/ppCWoRDe0VzkuyoDJE1BhUe
OhMjnzQvynZVtuquhFoiHOs+Z/VjnGGT9v3u9X45m4CLfzqitXQKre2QFj3F13XJ
+vfx+9B12rNE6dfRRmRygfu6ezxWyv1YM7epMOxCBufDptd2T+gdeg==
-----END PUBLIC KEY-----
```


Appendix E. ASN.1 Module

This appendix includes the ASN.1 module [X680] for X-Wing.

```
<CODE BEGINS>
XWing-KEM-2024
   { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1)
     pkcs-9(9) smime(16) modules(0) id-mod-XWing-kem-2024(TBD) }

DEFINITIONS IMPLICIT TAGS ::= BEGIN

-- EXPORTS ALL;

IMPORTS

  KEM-ALGORITHM
    FROM KEMAlgorithmInformation-2023  -- [I-D.housley-lamps-cms-kemri]
       { iso(1) identified-organization(3) dod(6) internet(1)
         security(5) mechanisms(5) pkix(7) id-mod(0)
```

```
            id-mod-kemAlgorithmInformation-2023(109) }

     AlgorithmIdentifier{}, PUBLIC-KEY, SMIME-CAPS
       FROM AlgorithmInformation-2009  -- [RFC5912]
         { iso(1) identified-organization(3) dod(6) internet(1)
           security(5) mechanisms(5) pkix(7) id-mod(0)
           id-mod-algorithmInformation-02(58) } ;


   -- XWing KEM Algorithm

   id-XWing OBJECT IDENTIFIER ::= { iso(1) identified-organization(3)
              dod(6) internet(1) private(4) enterprise(1) 62253 25722 }

   XWingPublicKey ::= OCTET STRING

   kema-XWing KEM-ALGORITHM ::= {
      IDENTIFIER id-XWing
      PARAMS ARE absent
      PUBLIC-KEYS { pk-XWing }
      UKM ARE optional
      SMIME-CAPS { IDENTIFIED BY id-XWing } }

   pk-XWing PUBLIC-KEY ::= {
     IDENTIFIER id-XWing
     -- KEY no ASN.1 wrapping --
     PARAMS ARE absent
     -- PRIVATE-KEY no ASN.1 wrapping --
     CERT-KEY-USAGE {keyEncipherment} }


   -- Updates for the KEM-ALGORITHM Set from rfc5990bis

   KeyEncapsulationMechanism ::=
      AlgorithmIdentifier { KEM-ALGORITHM, {KEMAlgorithms} }

   KEMAlgorithms KEM-ALGORITHM ::= { kema-XWing, ... }


   -- Updates for the SMIME-CAPS Set from RFC 5911

   SMimeCapsSet SMIME-CAPS ::= {kema-XWing.&smimeCaps, ... }

   END
   <CODE ENDS>
```

Appendix F. Acknowledgments

TODO acknowledge.

Appendix G. Change log

RFC Editor's Note: Please remove this section prior to publication of a final version of this document.

G.1. Since draft-connolly-cfrg-xwing-kem-07

* Elaborate on relation between randomized and derandomized functions.

* Update implementations section.

G.2. Since draft-connolly-cfrg-xwing-kem-06

* Add asn.1 module.

* To match FIPS 202, we request number of bits from SHAKE-256 instead of number of bytes. #27

* Update implementations section.

* Correct PEM header. #25

G.3. Since draft-connolly-cfrg-xwing-kem-05

* Fix several typos.

* Change HPKE/TLS codepoint requests to the memorable 25519 + 203.

* Add instruction for use in X.509. #21

G.4. Since draft-connolly-cfrg-xwing-kem-04

* Note that ML-KEM decapsulation key check is not required.

* Properly refer to FIPS 203 dependencies. #20

* Move label at the end. As everything fits within a single block of SHA3-256, this does not make any difference.

* Use SHAKE-256 to stretch seed. This does not have any security or performance effects: as we only squeeze 96 bytes, we perform a single Keccak permutation whether SHAKE-128 or SHAKE-256 is used. The effective capacity of the sponge in both cases is 832, which gives a security of 416 bits. It does require less thought from anyone analysing X-Wing in a rush.

* Add HPKE codepoint.

* Don’t mark TLS entry as recommended before it has been through the IETF consensus process. (Obviously the authors recommend X-Wing.)

G.5. Since draft-connolly-cfrg-xwing-kem-03

* Mandate ML-KEM encapsulation key check, and stipulate effect on TLS and HPKE integration.

* Add provisional TLS codepoint. (Not assigned, yet.)

G.6. Since draft-connolly-cfrg-xwing-kem-02

* Use seed as private key.

* Expand on caching decapsulation key values.

* Expand on binding properties.

G.7. Since draft-connolly-cfrg-xwing-kem-01

* Add list of implementations.

* Miscellaneous editorial improvements.

* Add Python reference specification.

* Correct definition of ML-KEM-768.KeyGenDerand(seed).

G.8. Since draft-connolly-cfrg-xwing-kem-00

* A copy of the X25519 public key is now included in the X-Wing decapsulation (private) key, so that decapsulation does not require separate access to the X-Wing public key. See #2.

Authors’ Addresses

Deirdre Connolly
SandboxAQ
Email: durumcrustulum@gmail.com

Peter Schwabe
MPI-SP & Radboud University
Email: peter@cryptojedi.org

Bas Westerbaan
Cloudflare
Email: bas@cloudflare.com