Crypto Forum
Internet-Draft
Intended status: Informational
Expires: 17 June 2026

S. Kostin

14 December 2025

X-Change: High-Security Purpose Hybrid KEM Exchange
draft-kostin-xchange-kem-00

Abstract

This draft defines X-Change, a high-security-purpose post-quantum/traditional hybrid key encapsulation mechanism (PQ/T KEM) built on X25519 and ML-KEM-1024.

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/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 17 June 2026.

Table of Contents

1. Introduction

1.1. Motivation

There are many different choices for hybrid KEM exchanges with varying security and performance parameters. Most of them aim at the average performance-security trade-off, while this draft defines a security-first option.

The aim of X-Change is to provide a concrete and the most secure choice for post-quantum hybrid KEM that is suitable for the most sensitive environments.

1.2. Goals

Our design goal is to prioritize the security of the exchange above all else.

* Simplicity of Standardization: By adopting standardization formats for high-security hybrid KEM exchanges, we significantly reduce the risk of misuse of incorrect components. Our architecture tries to achieve as simple design as possible.

* Security Analysis: Since ML-KEM-1024 already meets security category 5 per [FIPS203], we do not need to complicate the analysis of X-Change by considering stronger models. Additionally, X25519 complies with traditional security proofs as outlined in [RFC7748].

* Performance: In the current design, we prioritize security over performance by using ML-KEM-1024 , while favoring the use of 512-bit (64-byte) keys and SHA3-512. In this design, performance trade-offs may be significant compared to simpler designs.

We aim for 256-bit security in most of the environments by adopting ML-KEM-1024 (which is classified as security category 5, equivalent to 256 bits of classical security) and using X25519 as the preferred traditional exchange. However, this may not be suitable for all applications or low-end environments.

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.

* 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-Change relies on the following primitives:

* ML-KEM-1024 post-quantum key-encapsulation mechanism (KEM):

  - ML-KEM-1024.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. d and z are both 32 byte values.

ML-KEM-1024.KeyGen_internal(d, z) MUST be as described at the [FIPS203] paragraph 6.1.

- ML-KEM-1024.Encaps_internal(pk_M, m): Deterministic algorithm to generate (ss_M, ct_M), an 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-1024.Encaps_internal(pk_M, m) MUST be as described at the [FIPS203] paragraph 6.2.

- ML-KEM-1024.Decap_internal(sk_M, ct): Deterministic algorithm to decapsulate ct using the previously generated sk_M and recover the shared key from ss_M.

  ML-KEM-1024.Decap_internal(sk_M, ct) MUST be as described at the [FIPS203] paragraph 6.3.

* X25519/X448 elliptic curve Diffie-Hellman key-exchange defined in [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(k, v) MUST be as described at the [RFC7748] paragraph 5.

  - X25519_BASE: the bytes string representing the standard base point of Curve25519.

    X25519_BASE MUST be as described at the [RFC7748] paragraph 6.

* Symmetric cryptography:

  - SHA3-512(message): The hash with that name defined in Section 6.1 of [FIPS202].

    SHA3-512(message) MUST be as described at the [FIPS202] paragraph 6.1.

5. X-Change Construction

5.1. Encoding and sizes

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

Decapsulation key (private): 96 bytes

Encapsulation key (public): 1600 bytes

Ciphertext: 1632 bytes

Proof: 64 bytes

Shared secret: 64 bytes

5.2. Static Key Derivation

Derivation of static keys within the X-Change framework is implemented as follows.

```
def GetStaticKeyPair(sk):
    (pk_M, sk_S) = ML-KEM-1024.KeyGen_internal(sk[0:32], sk[32:64])
    sk_X = sk[64:96]
    pk_X = X25519(sk_X, X25519_BASE)
    return (sk_M, sk_X, pk_M, pk_X)
```

The sk MUST be a randomly generated 96-byte value.

The GetStaticKeyPair(sk) function returns the decapsulation key from ML-KEM-1024, the secret key for X25519, the encapsulation key from ML-KEM-1024, and the public key from X25519.

5.3. Key Generation

An X-Change key pair (decapsulation key and encapsulation key) is generated as outlined below.

```
def GenerateKeyPair():
    sk = random(96)
    (sk_M, sk_X, pk_M, pk_X) = GetStaticKeyPair(sk)
    return (sk, concat(pk_M, pk_X))
```

The GenerateKeyPair() function returns a 96-byte secret key sk and a 1600-byte encapsulation key pk.

5.4. Share Secret Derivation

This section details the derivation of a shared secret and its corresponding proof in the X-Change framework.

```
def KeyDerivation(salt, ss_X, ss_M, ct):
    ikm = concat(salt, ct)
    ss = concat(ss_X, ss_M)
    okm = SHA3-512(concat(ikm, ss, x-change-key-context))
    proof = SHA3-512(concat(okm, x-change-proof-context))
    return (okm, proof)
```

The salt MUST be a randomly generated 32-byte value.

The KeyDerivation() function returns a 64-byte final shared secret key, okm, along with a 64-byte proof. This proof assists in identifying various potential attack vectors, such as:

* Bit flips within the ciphertext, public key, or shared keys, which may not trigger immediate errors.

* Simple Man-in-the-Middle (MITM) attacks that do not replicate the full exchange.

The current setup ensures resilience against corruption in shared keys, public keys, or ciphertext, thereby enforcing an early exit when such issues are detected.

5.5. Encapsulation Key Derivation

This section explains the derivation of an X-Change encapsulation key pk.

```
def EncapsulationStatic(pk, seed):
    pk_M = pk[0:1568]
    pk_X = pk[1568:1600]
    ek_X = seed[0:32]
    salt = seed[64:96]

    ct_X = X25519(ek_X, X25519_BASE)
    ss_X = X25519(ek_X, pk_X)
    (ss_M, ct_M) = ML-KEM-1024.Encaps_internal(pk_M, seed[32:64])

    ct = concat(ct_M, ct_X, salt)
    (ss, proof) = KeyDerivation(salt, ss_X, ss_M, ct)
    return (ss, ct, proof)
```

The pk is 1600-byte value from GenerateKeyPair() function. The seed MUST be a randomly generated 96-byte value.

The EncapsulationStatic() function returns shared secret ss, ciphertext ct and proof used to verify keys after decapsulation.

5.6. Decapsulation Key Derivation

This section explains the derivation of an X-Change decapsulation key sk.

```
def Decapsulate(sk, ct, given_proof):
    (sk_M, sk_X, pk_M, pk_X) = GetStaticKeyPair(sk)
    ct_M = ct[0:1568]
    ct_X = ct[1568:1600]
    salt = ct[1600:1632]

    ss_M = ML-KEM-1024.Decap_internal(sk_M, ct_M)
    ss_X = X25519(sk_X, ct_X)
    (ss, proof) = KeyDerivation(salt, ss_X, ss_M, ct)

    if proof != given_proof:
        print("Proof mismatch!")
    return ss
```

The sk is private 1632-byte value. The ct is ciphertext we get from EncapsulationStatic() operation. The given_proof is proof we got from EncapsulationStatic() operation.

The Decapsulate() function returns shared secret ss, and checks proofs together to detect keys mismatch.

6. Security Considerations

Informally, X-Change is secure if SHA3-512 is secure, and either X25519 is secure, or ML-KEM-1024 is secure.

More precisely, if SHA3-512 may be modeled as a random oracle, then the IND-CCA security of X-Change is bounded by the IND-CCA security of ML-KEM-1024, and the gap-CDH security of Curve25519.

The security of X-Change relies crucially on the specifics of the Fujisaki-Okamoto transformation used in ML-KEM-1024: the X-Change 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.

X-Change also adds additional security features as proof. This proof is used to detect mismatches and protect against common errors, such as using the wrong decapsulating key on ciphertext. Additionally, the proof system could be integrated and sent over more secure channels than the ciphertext to guarantee Man-in-the-Middle (MITM) security.

7. IANA Considerations

This document do not requests/registers a new entry in the registry.

8. Use Cases

The X-Change mechanism provides a secure hybrid key encapsulation method suitable for a variety of applications, particularly in high-security environments. Below are some potential use cases:

Secure Communication This use case involves secure peer-to-peer communication channels, such as instant messaging apps or VoIP services, requiring robust confidentiality and integrity for message exchanges.

Cloud-based Security Solutions X-Change can enhance the high-security clouds computing environments, enabling secure key exchanges for encrypting data transmitted between users and cloud services.

Financial Sector Applications Financial institutions can utilize X-Change for secure transactions and communications of highest possible security level, reducing the risks associated with financial data breaches.

9. References

9.1. 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>.

[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>.

9.2. Informative References

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

[FIPS203] National Institute of Standards and Technology, "FIPS 203: Module-Lattice-Based Key-Encapsulation Mechanism Standard", <https://nvlpubs.nist.gov/nistpubs/fips/nist.fips.203.pdf>.

[RFC7748] Langley, A., Hamburg, M., and S. Turner, "Elliptic Curves for Security", January 2016.

Author’s Address

Stepan Kostin
Email: s_kostin@proton.me