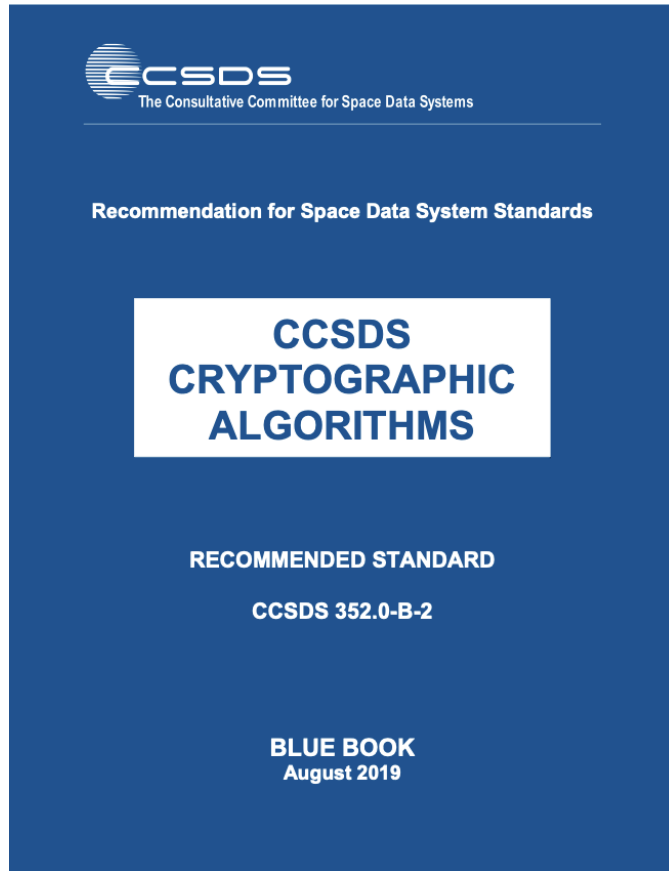


# CCSDS Cryptographic Algorithms - Proposed Updates and Rationale (for 3KEM Needs)

---

Oana Graur (ESA TEC-SES)  
Antonios Atlasis (ESA TEC-SES)

09/07/2025



- Addition of Elliptic Curve Cryptography (required for hybridization of 3KEM)
  - Elliptic Curve Diffie – Hellman (ECDH)
  - Curve selection for ECDH
- Addition of KEM Combiners (as per NIST SP 800 227 ipd or IETF work)
- Addition of KDFs (Two-Step Key Derivation as per NIST SP 800 56C Section 5, HKDF)

# ECDH

When using hybridized KEMs, ECDH can be used as the traditional KEM.

One option of implementing ECDH is as per **Section 5.7.1.2** of **NIST SP 800-56A rev 3**. This would automatically be aligned with **ETSI TS 103 744 V1.1.1. (2020-12)**.



This is an approved mechanism for FIPS 140-3 certification.

## 5.7.1.2 Elliptic Curve Cryptography Cofactor Diffie-Hellman (ECC CDH) Primitive

A shared secret  $Z$  is computed using the domain parameters  $(q, FR, a, b\{, SEED\}, G, n, h)$ , the other party's public key, and one's own private key. This primitive is used in [Section 6](#) by the Full Unified Model, Ephemeral Unified Model, One-Pass Unified Model, One-Pass Diffie-Hellman and Static Unified Model schemes. Assume that the party performing the computation is party A, and the other party is party B. Note that party A could be either party U or party V.

### Input:

1.  $(q, FR, a, b\{, SEED\}, G, n, h)$ : Domain parameters,
2.  $d_A$ : One's own private key, and
3.  $Q_B$ : The other party's public key.

### Process:

1. Compute the point  $P = hd_AQ_B$ .
2. If  $P = \emptyset$ , destroy all intermediate values used in the attempted computation of  $P$ , then output an error indicator, and exit this process without further processing.
3. Else, set  $z = x_P$ , where  $x_P$  is the  $x$ -coordinate of  $P$ , and convert  $z$  to  $Z$ , using the field-element-to-byte string conversion routine defined in [Appendix C.2](#).
4. Destroy the results of all intermediate calculations used in the computation of  $Z$  (including  $P$  and  $z$ ).
5. Output  $Z$ .

**Output:** The shared secret  $Z$  or an error indicator.

ESA proposes NIST SP 800-56A rev section 5.7.1.2 for the 3KEM “default profile” ECDH implementation (when hybridization used).

## 7.5 Elliptic Curve Diffie-Hellman (ECDH)

### 7.5.1 ECDH description

One of the key-agreement schemes shall be elliptic curve Diffie-Hellman defined in clause 5.7.1.2 of NIST SP800-56Ar3 [1]. A shared secret  $k$  is computed between two entities over the same elliptic curve domain parameters. The key-agreement scheme, for a given set of elliptic curve domain parameters, consists of a pair of algorithms:

- $ECKeyGen()$ , a probabilistic algorithm that returns a private and public key pair  $(d, Q)$ .
- $ECDH(d_A, Q_B)$ , a deterministic algorithm that takes  $d_A$ , entity A's private key and  $Q_B$ , entity B's public key as input and computes a shared secret  $k$ , or  $\perp$  an error indicator.

Similarly, party B computes the same shared secret by computing  $ECDH(d_B, Q_A)$ .

**ETSI TS 103 744 V1.1.1 (2020-12)**

NIST SP 800-56A rev 3



When using hybridized KEMs, ECDH can be used as the traditional KEM.

Another option for implementing ECDH is as per ANSI X9.63.

ANSI X9.63 aligns with BSI recommendation in TR-03111.

ANSI X9.63 “The modified Diffie-Hellman primitive also produces a shared secret value, but has been modified to resist small subgroup attacks in a way that may, in some cases, be more efficient. (See [8].)”



## DE NSA (BSI) – TR-03111 Elliptic Curve Cryptography

### 4.3. The Elliptic Curve Key Agreement Algorithm – ECKA

This section describes the Elliptic Curve Key Agreement Algorithm (ECKA), key derivation functions, and the key agreement protocols of Diffie-Hellman (ECKA-DH) and ElGamal (ECKA-EG). The description of ECKA is in conformance with [3].

**Note:** To prevent attacks based on invalid (ephemeral) public keys it MUST be checked that a received public key is indeed a point on the elliptic curve. This validation is already part of the point decoding algorithms (cf. Section 3.2). In addition to this, small subgroup attacks are prevented by using (compatible) cofactor multiplication in the key agreement algorithms.

[3] ANSI X9.63. *Public Key Cryptography for the Financial Services Industry: Key Agreement and Key Transport Using Elliptic Curve Cryptography*, 2011.

#### 5.4.2 Modified Diffie-Hellman Primitive

The shared secret value shall be calculated as follows:

**Prerequisites:** The prerequisite is a set of EC domain parameters  $D = (q, FR, a, b, ECCSEED, G, n, h)$ , which has been validated using the techniques described in Sections 5.1.1.2 and 5.1.2.2.

**Input:** The modified Diffie-Hellman primitive takes as input:

1. an EC private key  $d$  owned by  $A$ .
2. an EC public key  $Q$  owned by  $B$ .

The public key  $Q$  shall have been validated as specified in Section 5.2.2.

**Actions:** The following actions are taken:

1. Compute the point  $P = hdQ$ .
2. Check that  $P \neq \mathcal{O}$ . If  $P = \mathcal{O}$ , output ‘invalid’ and stop.
3. Set  $z = x_P$ , where  $x_P$  is the  $x$ -coordinate of  $P$ .

**Output:**  $z \in F_q$  as the shared secret value.



R22

Mécanismes d'établissement de clé

Les mécanismes d'établissement de clé DH et EC-DH sont recommandés.

Primitive	Mécanisme	R/O	Notes
FF-DLOG	DH [SP800-56A, ISO11770-3]	R	6.4.a
EC-DLOG	EC-DH [SP800-56A, ISO11770-3]	R	6.4.a

[Mécanismes cryptographiques | ANSSI](#)



Elliptic curve selected candidates:

- **NIST P-384/secp384r1** defined in FIPS PUB 800-186
- **brainpoolP384r1** defined in clause 3.6 of IETF RFC 5639
- **Curve25519** defined in clause 4.1 of IETF RFC 7748
- **Curve448** defined in clause 4.2 of IETF RFC 7748
- **NIST P-521** in FIPS PUB 800-186
- **brainpoolP521r1** defined in clause 3.7 in IETF RFC 5639



NIST SP 800-186  
February 2023

Discrete Logarithm-based Cryptography:  
Elliptic Curve Domain Parameters

3. Recommended Curves for U.S. Federal Government Use

This section specifies the elliptic curves recommended for U.S. Federal Government use and contains choices for the private key length and underlying fields. This includes elliptic curves over prime fields (Section 3.2) and elliptic curves over binary fields (Section 3.3), where each curve takes one of the forms described in Section 3 (referred to as “Type” below).

Each recommended curve is uniquely defined by its domain parameters  $D$ , which indicate the field  $GF(q)$  over which the elliptic curve is defined, the parameters of its defining equation, and principal parameters, such as the cofactor  $h$  of the curve, the order  $n$  of its prime-order subgroup, and a designated point  $G$  on the curve of order  $n$  (i.e., the “base point”). In the case  $q = 2^m$ , the domain parameters also include a description of the representation chosen for  $GF(q)$ .

Table 1. Approximate Security Strength of the Recommended Curves

Security Strength	Recommended Curves
112	P-224, K-233, B-233

Security Strength	Recommended Curves
128	P-256, W-25519, Curve25519, Edwards25519, K-283, B-283
192	P-384, K-409, B-409
224	W-448, Curve448, Edwards448, E448
256	P-521, K-571, B-571

Each elliptic curve specified in this recommendation is allowed for specific NIST approved cryptographic functions. The allowed usages for each curve are summarized in Table 2.

Table 2. Allowed Usage of the Specified Curves

Specified Curves	Allowed Usage
K-233, B-233 K-283, B-283 K-409, B-409 K-571, B-571	Deprecated
P-224 P-256 P-384 P-521	ECDSA, EC key establishment (see [SP_800-56A])
Edwards25519 Edwards448	EdDSA
Curve25519, W-25519 Curve448, E448, W-448	Alternative representations included for implementation flexibility. Not to be used for ECDSA or EdDSA directly.



Appendix H. Other Allowed Elliptic Curves

H.1. Brainpool Curves

This standard also allows the curves specified in *Elliptic Curve Cryptography (ECC) Brainpool Standard Curves and Curve Generation* [RFC\_5639] for ECDSA signatures as well as EC key establishment, which support a security strength of 112 bits or higher. In particular, this includes brainpoolP224r1, brainpoolP256r1, brainpoolP320r1, brainpoolP384r1 and brainpoolP512r1. These curves were pseudorandomly generated and are allowed to be used for interoperability reasons.

H.2. The Curve secp256k1

This standard also allows the curve secp256k1 specified in *SEC 2: Recommended Elliptic Curve Domain Parameters* [SEC\_2], which supports a security strength of 128 bits. This curve is a Koblitz curve with coefficients selected for efficiency reasons. The curve secp256k1 is allowed to be used for blockchain-related applications.

Curve25519 was specified in IETF 7748 by the Crypto Forum Research Group (CFRG).<sup>9</sup>



In TLS 1.3, client and server can use the extension “supported\_groups” to inform each other about the Diffie-Hellman groups they want to use for (EC)DHE.

Table 9: Recommended Diffie-Hellman groups for TLS 1.3

**Note:** In general, the Brainpool curves are recommended.

[Technical Guideline TR-02102-2: Use of Transport Layer Security \(TLS\) \(bund.de\)](#)

The system parameters listed in Table B.3 are recommended:

- Table B.3: Recommended EC system parameters for asymmetric schemes that are based on elliptic curves.

### 3 Recommendations

When using elliptic curves, cryptographically strong curves over finite fields of the form  $F_p$  ( $p$  prime) are always recommended. In addition, it is recommended to only use *named curves* (see Section “Supported Groups Registry” in [IANA]) in order to avoid attacks via unverified weak domain parameters. The following named curves are recommended:

- If these curves are not available, the following curves can also be used:

FR NSA (ANSSI)



Mécanisme conforme

- L'emploi de la courbe FRP256v1 – définie dans le journal officiel n° 241 du 16/10/2011 et dont les paramètres, validés par l'ANSSI, peuvent librement être intégrés dans tous les produits de sécurité – est conforme au référentiel. Il en est de même de l'emploi des courbes P-256, P-384 et P-521 définies dans le FIPS 186-4 de 2013, ainsi que des courbes brainpoolP256r1, brainpoolP384r1 et brainpoolP512r1 définies dans la RFC 5639.

R18

Paramètres de courbes elliptiques pour le DLOG

Les paramètres de courbes elliptiques P256r1, P384r1 et P512r1 de la famille Brainpool, les paramètres de courbes elliptiques P-256, P-384 et P-521 du NIST et les paramètres de courbes Curve25519 et Curve448 sont recommandés.

Familles de courbes	Courbes	R/O	Notes
Brainpool [RFC5639]	BrainpoolP256r1	R	5.3.a
	BrainpoolP384r1	R	
	BrainpoolP512r1	R	
NIST [FIPS186] (voir Annexe D.1.2)	NIST P-256	R	5.3.a
	NIST P-384	R	
	NIST P-521	R	
IETF [RFC7748]	Curve25519	R	5.3.b
	Curve448	R	

Mécanismes cryptographiques | ANSSI

TLS 1.3 a fait évoluer le protocole en donnant la possibilité au client de négocier les groupes DHE et les courbes elliptiques. Les groupes d'entiers et les courbes elliptiques utilisables ont également été inscrits dans les spécifications. Les groupes d'entiers retenus sont ceux définis dans [34] : ffdhe2048, ffdhe3072, ffdhe4096, ffdhe6144 et ffdhe8192. Les courbes elliptiques retenues sont secp256r1, secp384r1, secp521r1 (aussi appelées P-256, P-384 et P-521) ainsi que les courbes x25519, x448, brainpoolP256r, brainpoolP384r et brainpoolP512r1. La négociation de ces groupes est réalisée par l'extension supported\_groups, détaillée dans la section 2.3.

Dans le cas d'ECDHE, pour une protection des données au-delà de 2020, le RGS préconise l'utilisation de groupes d'ordre multiple d'un nombre premier long d'au moins 256 bits [35]. L'ensemble des courbes retenues dans TLS 1.3 respectent le RGS.

R7 Échanger les clés avec l'algorithme ECDHE

Les échanges de clés ECDHE doivent être privilégiés, à l'aide des courbes secp256r1, secp384r1 secp521r1. Les courbes x25519 et x448 constituent des variantes acceptables. Les courbes brainpoolP256r, brainpoolP384r et brainpoolP512r1 sont également acceptables.

Dans le cas de DHE, la sécurité de l'échange est liée à l'ordre du groupe multiplicatif en jeu. L'attaque Logjam [4] a illustré l'insuffisance des groupes de taille 512-bits, et pousse à déconseiller l'utilisation de groupes 1024-bits. Le RGS préconise l'utilisation de groupes 3072-bits ou plus, et tolère les groupes 2048-bits pour une protection des données jusqu'en 2030.

R7 – Échanger les clés avec l'algorithme DHE

Les échanges de clés DHE sont tolérés en utilisant les groupes 2048-bits ou plus (3072-bits ou plus si l'information doit être protégée au-delà de 2030) définis dans [34].

ESA proposes NIST P-384 to be the curve chosen for the default 3KEM profile, but CCSDS Crypto Blue Book to include all 5 options. If a second profile shall be defined in 3KEM Blue Book Annex, we propose **Curve25519**.

- **NIST P-384/secp384r1** defined in FIPS PUB 800-186
- **brainpoolP384r1** defined in clause 3.6 of IETF RFC 5639
- **Curve25519** defined in clause 4.1 of IETF RFC 7748
- **Curve448** defined in clause 4.2 of IETF RFC 7748
- **NIST P-521** in FIPS PUB 800-186

5.1.3 DIFFIE-HELLMAN KEY EXCHANGE

- 5.1.3.1 For hybrid CCSDS key agreement implementations (see section 2.7), Elliptic Curve Diffie-Hellman (ECDH) as specified in Section 5.7.1.2 of reference [20] with curve P-384, specified in reference [18], shall be used.
- 5.1.3.2 Curve brainpoolP384r1 (reference [21]) may also be used.
- 5.1.3.3 Hybrid CCSDS key agreement implementations may alternatively use ECDH implementations such as "X25519" (with underlying "Curve25519" elliptic curve [19]) when security strength of 128 bits is acceptable for a mission, or "X448" (with underlying "Curve448" elliptic curve) [19] when security strength of 224 bits is required.
- 5.1.3.4 Hybrid CCSDS key establishment implementations which require 256 bits of security may use P-521 (reference[18]) or brainpoolP521r1 (reference [21])

In the table below, the recommended curves versus their security strength are listed:

Security Strength	Recommended Curve
128	Curve25519
192	P-384 (default), brainpoolP384r1
224	Curve448
256	P-521, brainpoolP512r1

# Key Derivation Functions



# KDFs – What are the options?

Following the computation of a shared secret, an approved Key Derivation Function needs to be used to derive keying material from the shared secret (and some other info).

In the 3KEM protocol we will need to use a KDF in two places:

**Case 1:** For hybridized case: To derive a shared secret key from the ECDH (traditional) shared secret before passing the shared secret key to the KEM Combiner where it is combined with the PQ shared secret.

- a call to the KDF will happen every time (hybrid) .encaps is called (that is 3 times, once for each IKEM, RKEM, EKEM)

**Case 2:** To derive as many “final” keys as needed by a specific mission from the “combined” shared secret


Well established KDF options are:

- HKDF defined in **RFC 5869**
- one-step KDFs and two-step KDF Extract-then-Expand defined in **NIST SP 800-56C rev 2**
- X9.63-KDF defined in **ANSI X9.63**



HMAC based **two step KDF function** (HKDF) based on the “**extract-then-expand**” paradigm

**HKDF – Extract** (salt, IKM) - > PRK  
PRK = HMAC-Hash (salt, IKM)



The **first stage** takes the input keying material (which may not be uniformly distributed) and "extracts" from it a fixed-length pseudorandom key K.

#### Inputs:

- salt - optional salt value (a non-secret random value); if not provided, it is set to a string of HashLen zeros.
- IKM - input keying material

#### Output:

- PRK - a pseudorandom key (of HashLen octets)

**HKDF – Expand** (PRK, info, L) -> OKM

```
N = ceil(L/HashLen)
T = T(1) | T(2) | T(3) | ... | T(N)
OKM = first L octets of T where:
T(0) = empty string (zero length)
T(1) = HMAC-Hash(PRK, T(0) | info | 0x01)
T(2) = HMAC-Hash(PRK, T(1) | info | 0x02)
T(3) = HMAC-Hash(PRK, T(2) | info | 0x03) ... (where the
constant concatenated to the end of each T(n) is a single octet.)
```

The **second stage** "expands" the key K into several additional pseudorandom keys (the output of the KDF)

#### Inputs:

- PRK - a pseudorandom key of at least HashLen octets
- info - optional context and application specific information (can be a zero-length string)
- L - length of output keying material in octets (<= 255\*HashLen)

#### Outputs:

- OKM – output keying material (of L octets)



## HMAC based **two step KDF function** (HKDF) based on the “**extract-then-expand**” paradigm

**HKDF - Extract** (salt, IKM) -> PRK  
PRK = HMAC-Hash (salt, IKM)

HMAC key      HMAC input

## HKDF - Expand (PRK, info, L) -> OKM

```
N = ceil(L/HashLen)
T = T(1) | T(2) | T(3) | ... | T(N)
OKM = first L octets of T where:
T(0) = empty string (zero length)
T(1) = HMAC-Hash(PRK, T(0) | info | 0x01)
T(2) = HMAC-Hash(PRK, T(1) | info | 0x02)
T(3) = HMAC-Hash(PRK, T(2) | info | 0x03)
... (where the constant concatenated to the end of each T(n)
is a single octet.)
```

```
def finalize(state, m)
```

```
KDK = HMAC-SHA3-256(state , m)
key_IR_01 = KDF-HMAC-HASH-256(KDK, "" || info || 0x01) //Expand
key_IR_02 = KDF-HMAC-HASH-256(KDK, key_IR_01 || info || 0x02) //Expand
key_IR_03 = KDF-HMAC-HASH-256(KDK, key_IR_02 || info || 0x03) //Expand
key_RI_01 = KDF-HMAC-HASH-256(KDK, key_IR_03 || info || 0x04) //Expand
key_RI_02 = KDF-HMAC-HASH-256(KDK, key_RI_01 || info || 0x05) //Expand
key_RI_03 = KDF-HMAC-HASH-256(KDK, key_RI_02 || info || 0x06) //Expand
confirmation_key = KDF-HMAC-HASH-256(KDK, key_RI_03 || info || 0x07) //Expand
return key_IR_01, key_IR_02, key_IR_03, key_RI_01, key_RI_02, key_RI_03, confirmation_key
```



(FIPS-140) approved methods for key derivation (KDMs) from a shared secret are specified in **NIST SP 800-56C**. They follow the structure:

$$\text{KDM}(Z, \text{OtherInput})$$

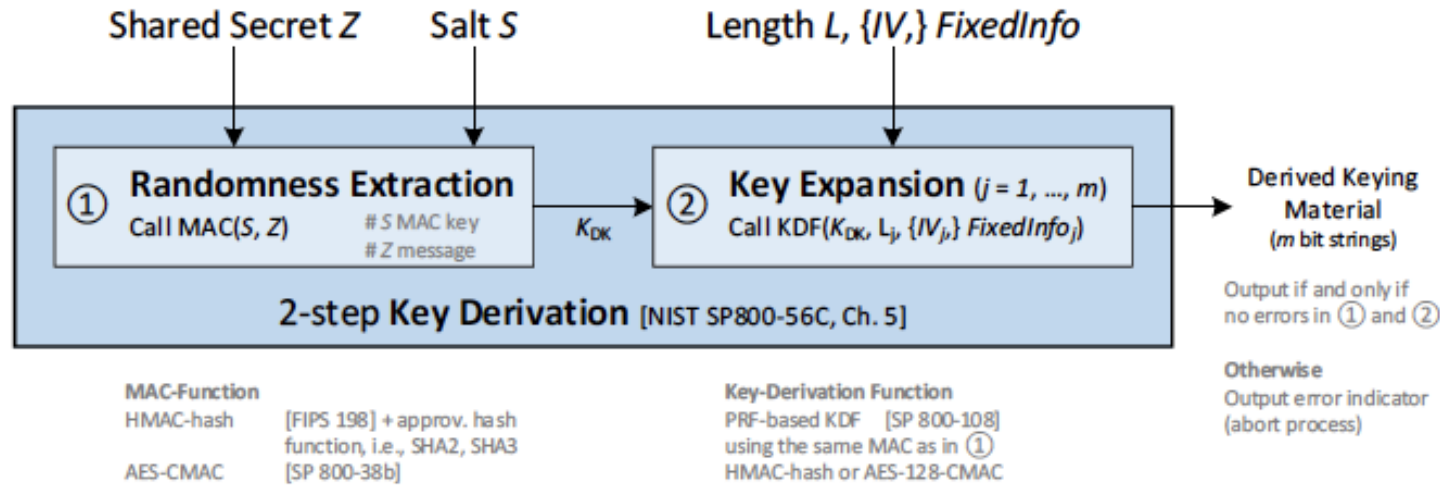
- $Z$  is a byte string that represents the shared secret
- $\text{OtherInput}$  consists of additional input information that may be required by a given key-derivation method, for example:
  - $L$  – an integer that indicates the length (in bits) of the secret keying material to be derived.
  - salt – a byte string
  - IV – a bit string used as an initialization value
  - FixedInfo – a bit string of context-specific data

## NIST options in SP 800-56C:

- One-step key derivation
- Two-step key derivation (Extract-then-Expand)

# KDFs – NIST SP 800-56C – Two Step KDF

## Two-step key derivation (Extract-then-Expand)



*FixedInfo* is a bit string that contains:

- Label is a binary string that identifies the purpose of the derived keying material. The encoding method for the label is defined in a larger context, for example, in a protocol using the key-derivation method.
- Context is a binary string containing information relating to the derived keying material.
- $[L]_2$  is a binary string that specifies the length (in bits) of the keying material to be derived.

### NIST SP 800-108

- KDF in Counter Mode
- KDF in Feedback Mode
- KDF in Double Pipeline Mode
- KDF using KMAC

#### 4.2. KDF in Feedback Mode

This section specifies a family of KDFs that employs a feedback mode. In the feedback mode, the output of the PRF is computed using iteration-dependent input data that consists of the result of the previous iteration and, optionally, a counter. The mode is defined as follows. (Note that when  $L \leq h$ ,  $IV = \emptyset$ , and the counter is used, the feedback mode will generate an output that is identical to the output of the counter mode specified in Section 4.1.)

##### Parameters:

- $h$  – The length of the output of a single invocation of the PRF in bits
- $r$  – The length of the binary representation of the counter  $i$ .  $r$  is specified only when a counter is used as an input

**Input:**  $K_{IN}$ , *Label*, *Context*, *IV*, and  $L$

##### Process:

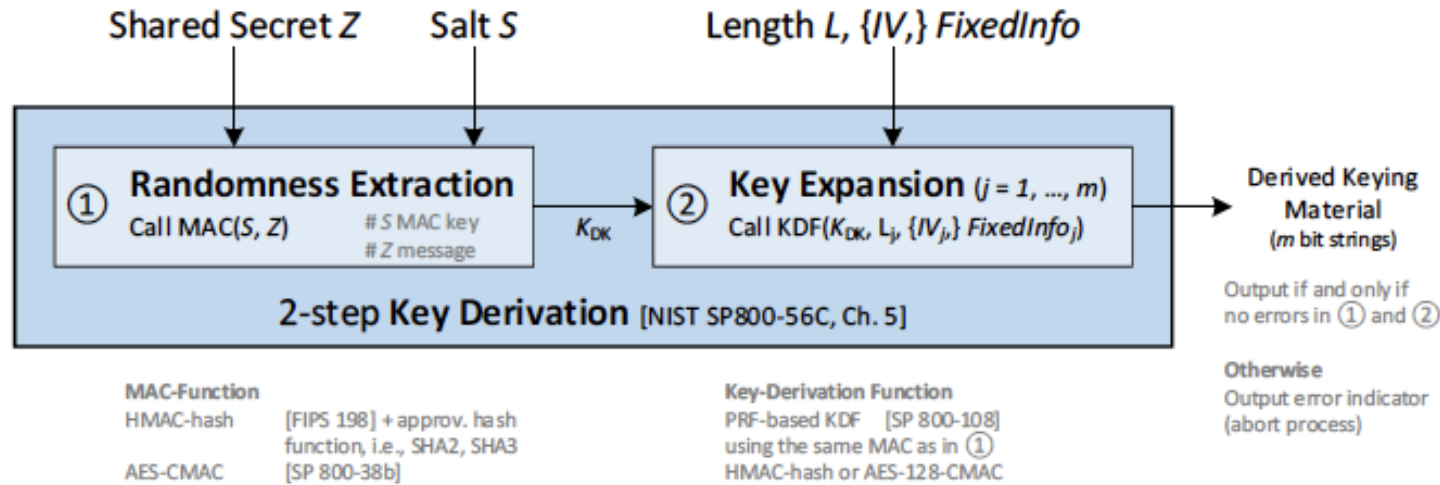
1.  $n := \lceil L/h \rceil$ .
2. If  $n > 2^{32} - 1$ , output an error indicator and stop (i.e., skip steps 3, 4, and 5).
3.  $result := \emptyset$  and  $K(0) := IV$ .
4. For  $i = 1$  to  $n$ , do
  - a.  $K(i) := \text{PRF}(K_{IN}, K(i-1) \parallel [i]_2 \parallel \underbrace{Label \parallel 0x00 \parallel Context \parallel [L]_2}_{FixedInfo})$ .
  - b.  $result := result \parallel K(i)$ .
5.  $K_{OUT} :=$  the leftmost  $L$  bits of  $result$ .

**Output:**  $K_{OUT}$  (or an error indicator)

In each iteration of a PRF execution in step 4 above, the fixed input data is the string  $Label \parallel 0x00 \parallel Context \parallel [L]_2$ . The iteration-dependent input data is  $K(i-1) \parallel [i]_2$ . The KDF in feedback mode is illustrated in Fig. 2.

# KDFs – NIST SP 800-56C – Two Step KDF

## Two-step key derivation (Extract-then-Expand)



The Feedback mode, specified in detail in NIST SP 800-108 is of particular interest for our 3KEM use case.  $K(i)$  is a function (among others) of  $K(i-1)$ . This means that the last key  $K(n)$  can be used as a confirmation key for all the other “final” keys derived in the same session.

This comes in handy in an error-prone environment such as space, where bitflips due to radiation occur with significantly higher probability than on ground. A spacecraft can use this to confirm to/inform ground that **all keys up to the last one** have been correctly derived (and 3KEM was successfully executed).

### NIST SP 800-108

- KDF in Counter Mode
- KDF in Feedback Mode
- KDF in Double Pipeline Mode
- KDF using KMAC

#### 4.2. KDF in Feedback Mode

This section specifies a family of KDFs that employs a feedback mode. In the feedback mode, the output of the PRF is computed using iteration-dependent input data that consists of the result of the previous iteration and, optionally, a counter. The mode is defined as follows. (Note that when  $L \leq h$ ,  $IV = \emptyset$ , and the counter is used, the feedback mode will generate an output that is identical to the output of the counter mode specified in Section 4.1.)

##### Parameters:

- $h$  – The length of the output of a single invocation of the PRF in bits
- $r$  – The length of the binary representation of the counter  $i$ .  $r$  is specified only when a counter is used as an input

**Input:**  $K_{IN}$ ,  $Label$ ,  $Context$ ,  $IV$ , and  $L$

##### Process:

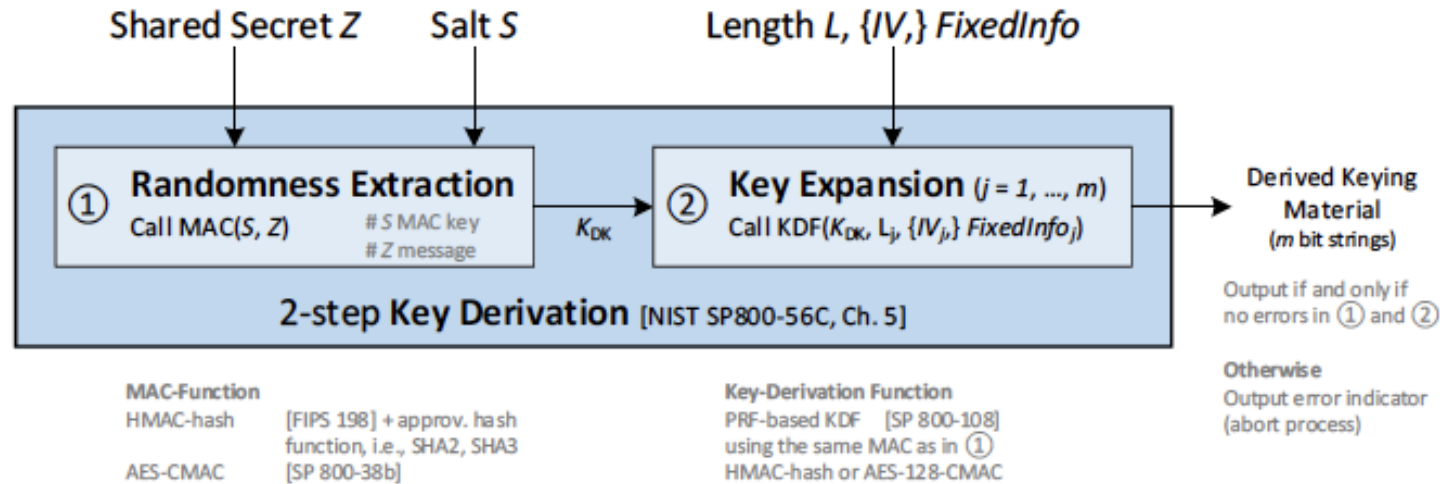
1.  $n := \lceil L/h \rceil$ .
2. If  $n > 2^{32} - 1$ , output an error indicator and stop (i.e., skip steps 3, 4, and 5).
3.  $result := \emptyset$  and  $K(0) := IV$ .
4. For  $i = 1$  to  $n$ , do
  - a.  $K(i) := \text{PRF}(K_{IN}, K(i-1) \parallel [i]_2 \parallel \underbrace{Label \parallel 0x00 \parallel Context \parallel [L]_2}_{FixedInfo})$ .
  - b.  $result := result \parallel K(i)$ .
5.  $K_{OUT} :=$  the leftmost  $L$  bits of  $result$ .

**Output:**  $K_{OUT}$  (or an error indicator)

In each iteration of a PRF execution in step 4 above, the fixed input data is the string  $Label \parallel 0x00 \parallel Context \parallel [L]_2$ . The iteration-dependent input data is  $K(i-1) \parallel [i]_2$ . The KDF in feedback mode is illustrated in Fig. 2.

# KDFs – NIST SP 800-56C – Two Step KDF

## Two-step key derivation (Extract-then-Expand)



### Case 2

For the 3KEM “default profile”, for final key derivation (finalize function) we propose to use **NIST SP 800 56C Two-Step KDF** with HMAC-SHA3-256:

- Randomness extraction: HMAC-SHA3-256
- Key Expansion: HMAC-SHA3-256 in Feedback Mode
- HMAC: **FIPS 198**
- KDF in Feedback Mode: **NIST SP 800-108**

### NIST SP 800-108

- KDF in Counter Mode
- KDF in Feedback Mode
- KDF in Double Pipeline Mode
- KDF using KMAC

#### 4.2. KDF in Feedback Mode

This section specifies a family of KDFs that employs a feedback mode. In the feedback mode, the output of the PRF is computed using iteration-dependent input data that consists of the result of the previous iteration and, optionally, a counter. The mode is defined as follows. (Note that when  $L \leq h$ ,  $IV = \emptyset$ , and the counter is used, the feedback mode will generate an output that is identical to the output of the counter mode specified in Section 4.1.)

##### Parameters:

- $h$  – The length of the output of a single invocation of the PRF in bits
- $r$  – The length of the binary representation of the counter  $i$ .  $r$  is specified only when a counter is used as an input

**Input:**  $K_{IN}$ ,  $Label$ ,  $Context$ ,  $IV$ , and  $L$

##### Process:

1.  $n := \lceil L/h \rceil$ .
2. If  $n > 2^{32} - 1$ , output an error indicator and stop (i.e., skip steps 3, 4, and 5).
3.  $result := \emptyset$  and  $K(0) := IV$ .
4. For  $i = 1$  to  $n$ , do
  - a.  $K(i) := PRF(K_{IN}, K(i-1) \parallel [i]_2 \parallel Label \parallel 0x00 \parallel Context \parallel [L]_2)$ .
  - b.  $result := result \parallel K(i)$ .
5.  $K_{OUT} :=$  the leftmost  $L$  bits of  $result$ .

**Output:**  $K_{OUT}$  (or an error indicator)

In each iteration of a PRF execution in step 4 above, the fixed input data is the string  $Label \parallel 0x00 \parallel Context \parallel [L]_2$ . The iteration-dependent input data is  $K(i-1) \parallel [i]_2$ . The KDF in feedback mode is illustrated in Fig. 2.



## A third KDF option is the X9.63 KDF defined in section 5.6.3 of ANSI X9.63 Public Key Cryptography for the Financial Services Industry Key Agreement and Key Transport Using Elliptic Curve Cryptography

### 5.6.3 Key Derivation Function (kdf)

This section specifies key derivation functions (KDFs) that shall be used by the schemes in this Standard.

Key derivation functions are used to derive keying data from a shared secret value that is represented as a bit string.

Key derivation functions are also used by the asymmetric encryption schemes in Section 5.8.

An Approved key derivation function shall be used. The concatenation key derivation function specified in NIST Special Publication 800-56A is Approved.

The following key derivation function is also Approved.

Keying data shall be calculated as follows:

**Prerequisites:** The prerequisite for the operation of the key derivation function is that an Approved hash function has been chosen as specified in Section 5.6.2.

**Input:** The input to the key derivation function is:

1. A bit string  $Z$  that is the shared secret value, of bit length at most  $maxhashlen - 32$ .
2. An integer  $keydatalen$  that is the length in bits of the keying data to be generated.  $keydatalen$  shall be less than  $(2^{32}-1) hashlen$
3. (Optional) A bit string  $SharedInfo$  that consists of some data shared by the two entities intended to share the secret value  $Z$ . The total bit length of  $Z$  and  $SharedInfo$  must be at most  $maxhashlen - 32$ .

**Ingredients:** The key derivation function employs one of the hash functions specified in Section 5.6.2.

**Actions:** The key derivation function is computed as follows:

1. Initiate a 32-bit, big-endian bit string  $counter$  as  $00000001_{16}$ .
2. For  $i=1$  to  $j=\lceil keydatalen/hashlen \rceil$ , do the following:
  - 2.1 Compute  $Hash_i = H(Z \parallel counter \parallel [SharedInfo])$ .
  - 2.2 Increment  $counter$ .
  - 2.3 Increment  $i$ .
3. Let  $HHash_j$  denote  $Hash_j$  if  $keydatalen/hashlen$  is an integer, and let it denote the  $(keydatalen - (hashlen)j)$  leftmost bits of  $Hash_j$  otherwise.
4. Set  $KeyData = Hash_1 \parallel Hash_2 \parallel \dots \parallel Hash_{j-1} \parallel HHash_j$ .

**Output:** The bit string  $KeyData$  of length  $keydatalen$  bits.

Note that the key derivation function produces keying data of length less than  $(2^{32}-1) hashlen$  bits. It is assumed that all key derivation function calls are for bit strings of length less than  $(2^{32}-1) hashlen$  bits. Any scheme attempting to call the key derivation function for a bit string of length greater than or equal to  $(2^{32}-1) hashlen$  bits shall output 'invalid' and stop.



## DE NSA (BSI) – TR-03111 Elliptic Curve Cryptography

**X9.63 Key Derivation Function.** ANSI X9.63 [3] describes a method for converting a shared secret to a cryptographic key. The algorithm  $KDF_{X9.63}()$  requires to select a hash function  $H()$  from Section 4.1.2. Let  $\ell$  denote the bit length of the hash value.

**Input:** The following inputs are needed:

1. An octet string  $Z_{AB}$ , which is the shared secret value.
2. An integer  $\kappa < \ell \cdot (2^{32} - 1)$ , which is the bit length of the keying data to be generated.
3. An octet string *SharedInfo*, which consists of some information shared between A and B (OPTIONAL).

**Output:** The octet string *KeyData* of length  $k = \lceil \kappa/8 \rceil$ .

**Actions:** The following actions are performed:

1. Let *counter* be a 32 bit, big-endian integer, initialized with 0x00000001.
2.  $j = \lceil \kappa/\ell \rceil$
3. For  $i = 1$  to  $j - 1$  do the following:
  - a)  $H_i = H(Z_{AB} \parallel \textit{counter} \parallel [\textit{SharedInfo}])$
  - b)  $\textit{counter} = \textit{counter} + 1$
  - c)  $i = i + 1$
4.  $l = \kappa - (\ell \cdot (j - 1))$
5.  $H_j = H_l(Z_{AB} \parallel \textit{counter} \parallel [\textit{SharedInfo}])$
6.  $\textit{KeyData} = H_1 \parallel H_2 \parallel \dots \parallel H_{j-1} \parallel H_j$
7. Output *KeyData*

An Approved hash function that offers 112 bits of security or more shall be used, i.e. an Approved hash function whose output is 224 bits or more. Possibilities, therefore, include the SHA-256 hash function



[73] H. Krawczyk and P. Eronen. HMAC-based Extract-and-Expand Key Derivation Function (HKDF). RFC 5869, 2010. <https://datatracker.ietf.org/doc/rfc5869/>.



## FR NSA (ANSSI)

ANSI recommends any of the following Key Derivation Functions: **HKDF, NIST SP800-56C, ANSI-X.96-KDF**

R14

## Mécanismes de dérivation de clés

Les mécanismes de dérivation de clés HKDF, NIST SP800-56 A, B, et C, ANSI-X9.63-KDF et PBKDF2 sont recommandés.

Schéma	R/O	Notes
HKDF [RFC5869]	R	
NIST SP800-56C [SP800-56C]	R	
ANSI-X9.63-KDF [ANSI-X9-63]	R	
PBKDF2 [RFC8018]	R	4.7.a



Not open/free

Mécanismes cryptographiques | ANSSI

This clause specifies a mapping from the KDF definition to HKDF with a specified hash function.

```
key_material = HKDF(secret, salt, info, length)
```

15

where the parameters and output shall be defined as follows.

*secret* - an octet string that constitutes the input for the key derivation function. It shall be present.

*info* - an octet string that contains application specific information. It may be a zero-length string.

*length* - the length in octets of the derived keying material. It shall be present.

*key\_material* - the derived keying material of length *length*.

$$\text{key\_material} = \text{KDF}(\text{secret}, \text{label}, \text{context}, \text{length}) = \text{HKDF}(\text{secret}, \text{label}, \text{context}, \text{length}).$$

- For the KDF in `.finalize` function in 3KEM, we recommend to follow NIST SP 800-56C two step KDF. For this specific use, it may be beneficial to choose the Feedback mode defined in NIST SP 800-108.
- For the KDF in the DHKEM (for the hybridized KEM case), the one-step KDF defined in NIST SP 800-56C may be sufficient. However, since the two-step KDF is used in `.finalize ()`, it may be simpler to reuse the same KDF.



# KEM Combiners

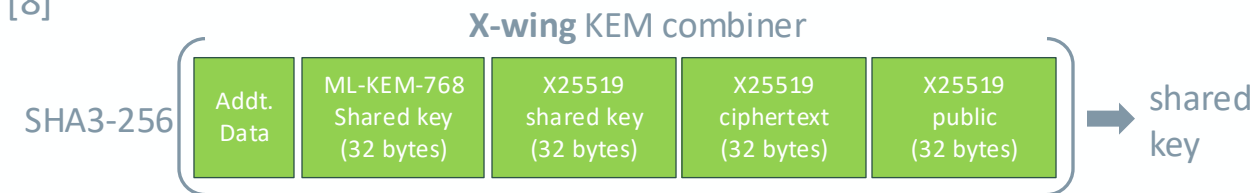
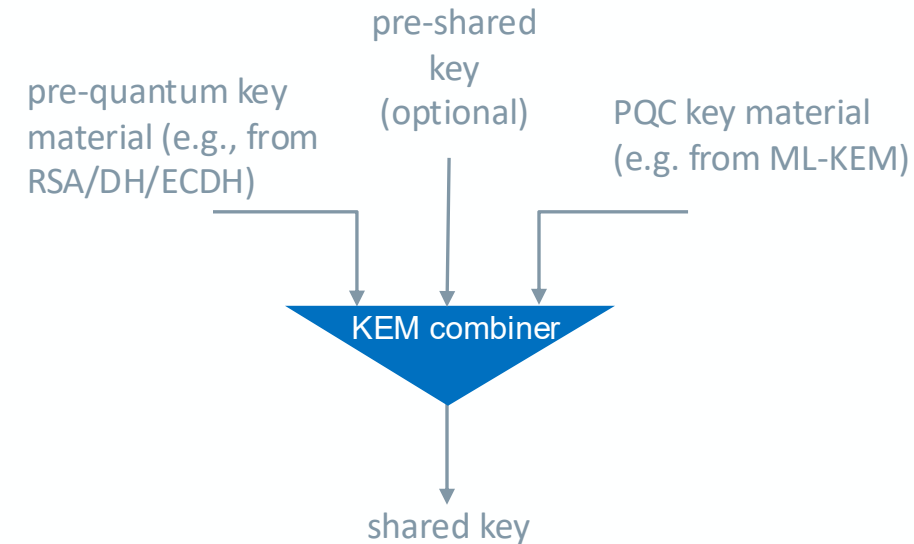


For 3KEM we want to use a **secure (standardized) KEM combiner** with a **validated security proof**.

A KEM combiner uses a Key Derivation Function to combine the inputs from the pre- and post-quantum schemes and produce a common shared secret.

KEM Combiner options:

- See NIST recent (draft) recommendations in [7]
- See ANSSI recommendations for KEM combiners in [1],[9]
- See BSI recommendations for KEM combiners in [5], [8]
- CEMPAT [3]
- X-wing (specific for Kyber) [2]
- draft-ounsworth-cfrg-kem-combiners [4]




**CHEMPAT KEM combiner**

```
H = SHA3-256
hybrid_pk = concat(receiver_pk_TKEM, receiver_pk_PQKEM)
hybrid_ct = concat(sender_ct_TKEM, sender_ct_PQKEM)
hybrid_ss = H(concat(ss_TKEM,
                    ss_PQKEM,
                    H(hybrid_ct),
                    H(hybrid_pk),
                    context))
```

- [1] [ANSSI views on Post-Quantum Cryptography Transition \(2023 follow-up\)](#)
- [2] IETF draft: X-Wing: general-purpose hybrid post-quantum KEM [draft-connolly-cfrg-xwing-kem-04](#)
- [3] IETF draft: [Chempat: Generic Instantiated PQ/T Hybrid Key Encapsulation Mechanisms](#)
- [4] IETF draft: [Combiner function for hybrid key encapsulation mechanisms \(Hybrid KEMs\)](#)
- [5] [https://pkic.org/events/2023/pqc-conference-amsterdam-nl/pkic-pqcc\\_stephan-ehlen\\_bsi\\_post-quantum-policy-and-roadmap-of-the-bsi.pdf](https://pkic.org/events/2023/pqc-conference-amsterdam-nl/pkic-pqcc_stephan-ehlen_bsi_post-quantum-policy-and-roadmap-of-the-bsi.pdf)
- [6] [Hybrid key exchange in TLS 1.3](#)
- [7] [NIST SP 800 227](#)
- [8] [BSI TR-02102-1 Jan 2025 Technical Guideline – Cryptographic Algorithms and Key Lengths](#)
- [9] [PQC Transition in France, ANSSI](#)



## KEM Combiner options:

1. See NIST recent (draft) recommendations in [7] --> see dedicated slide
2. See ANSSI recommendations for KEM combiners in [1], [9] - -> see dedicated slide
3. See BSI recommendations for KEM combiners in [5], [8] - -> see dedicated slide
4. CHEMPAT [3]  **NOT CHOSEN**: Generic **draft** but not explicitly recommended by neither NIST, ANSSI, BSI, etc
5. X-wing (specific for Kyber +x25519) [2] **NOT CHOSEN**: specific **draft** but not explicitly recommended by neither NIST, ANSSI, BSI, etc
6. draft-ounsworth-cfrg-kem-combiners [4] **NOT CHOSEN**: Generic **draft**, BSI involvement, but **expired** and not renewed



- [1] [ANSSI views on Post-Quantum Cryptography Transition \(2023 follow-up\)](#)
- [2] IETF draft: X-Wing: general-purpose hybrid post-quantum KEM [draft-connolly-cfrg-xwing-kem-04](#)
- [3] IETF draft: [Chempat: Generic Instantiated PQ/T Hybrid Key Encapsulation Mechanisms](#)
- [4] IETF draft: [Combiner function for hybrid key encapsulation mechanisms \(Hybrid KEMs\)](#)
- [5] [https://pkic.org/events/2023/pqc-conference-amsterdam-nl/pkic-pqcc\\_stephan-ehlen\\_bsi\\_post-quantum-policy-and-roadmap-of-the-bsi.pdf](https://pkic.org/events/2023/pqc-conference-amsterdam-nl/pkic-pqcc_stephan-ehlen_bsi_post-quantum-policy-and-roadmap-of-the-bsi.pdf)
- [6] [Hybrid key exchange in TLS 1.3](#)
- [7] [NIST SP 800 227](#)
- [8] [BSI TR-02102-1 Jan 2025 Technical Guideline – Cryptographic Algorithms and Key Lengths](#)
- [9] [PQC Transition in France, ANSSI](#)



DE NSA (BSI)

## 2.2. Key Derivation and Hybridisation

After a key agreement, both parties are in possession of a shared secret from which symmetric keys, for example for encryption and data authentication, can be derived using a key derivation function. For recommended mechanisms for key derivation, please refer to Section B.1.1.

The quantum-safe mechanisms recommended in this Technical Guideline are generally not yet trusted to the same extent as the established classical mechanisms, since they have not been as well studied with regard to side-channel resistance and implementation security. To ensure the long-term security of a key agreement, this Technical Guideline therefore recommends the use of a hybrid key agreement mechanism that combines a quantum-safe and a classical mechanism. An obvious hybridization is to perform two key agreements in parallel and to derive a combined key from the generated key material. The hybrid key agreement should remain secure as long as one of the methods used is secure. It is hereby important to consider in detail how the individual key material is combined with each other and that context-dependent information is included so that the aforementioned property is actually fulfilled.

The following mechanism is recommended for hybridisation in this Technical Guideline:

CatKDF, see [32].

Table 2.1: Recommended hybridisation mechanisms.

**Remark 2.5** • In [106], a comparable construction is defined. It is intended that future versions of this Technical Guideline will incorporate constructions from [106].

- The above recommendations are of a general nature. In addition, there are mechanisms tailored to specific protocols that are not covered in this part of the Technical Guideline.

[32] ETSI. ETSI TS 103 744: CYBER; Quantum-safe Hybrid Key Exchanges. V1.1.1, 2020. [https://www.etsi.org/deliver/etsi\\_ts/103700\\_103799/103744/01.01.01\\_60/ts\\_103744v010101p.pdf](https://www.etsi.org/deliver/etsi_ts/103700_103799/103744/01.01.01_60/ts_103744v010101p.pdf).

[106] National Institute of Standards and Technology. Special Publication NIST SP 800-227 ipd: Recommendations for Key Encapsulation Mechanisms, 2025. <https://doi.org/10.6028/NIST.SP.800-227.ipd>.

[4] IETF draft: [Combiner function for hybrid key encapsulation mechanisms \(Hybrid KEMs\)](#).

[5] [https://pkic.org/events/2023/pqc-conference-amsterdam-nl/pkic-pqcc\\_stephan-ehlen\\_bsi\\_post-quantum-policy-and-roadmap-of-the-bsi.pdf](https://pkic.org/events/2023/pqc-conference-amsterdam-nl/pkic-pqcc_stephan-ehlen_bsi_post-quantum-policy-and-roadmap-of-the-bsi.pdf)

[8] [BSI TR-02102-1 Jan 2025 Technical Guideline – Cryptographic Algorithms and Key Lengths](#)

BSI recommends the **ETSI CatKDF** defined in ETSI TS 103 744, as well as indicates that future constructions from **NIST SP 800 227** (currently draft) will be incorporated into BSI TR-02102

The CatKDF mode shall be defined as follows.

**Fixed values:**

*KDF* - The key derivation function being used from clause 7.4.

*f* - The context formatting function being used from clause 7.2 and the underlying hash function *hash* and *digest\_len*.

**Input:**

*psk* - a secret key. It may be present. If not present this value shall be the empty octet string,  $\emptyset$ .

$(k_1, k_2, \dots, k_n)$  - *n*-tuple of octet strings containing shared secrets  $k_i$ , exchanged through a hybrid key exchange, see Figure 4.

*MA, MB* - octet string of a pair of exchanged messages in establishment of the shared secrets  $k_i$ .

*context* - octet string context set by the instance of the key exchange transaction - this may include a transcript of additional exchanged messages.

*label* - an octet string that specifies a separation of use for the application or instance of the key-exchange. Any labels used in the key exchange should not be provided as an argument to the same hash function for another purpose in the application.

*length* - the length in octets of the derived key material *key\_material*.

**Process:**

- 1) Form *secret* = *psk* ||  $k_1$  ||  $k_2$  || ... ||  $k_n$ .
- 2) Set *f\_context* = *f*(*context*, *MA*, *MB*), where *f* is a context formatting function.
- 3) *key\_material* = *KDF*(*secret*, *label*, *f\_context*, *length*).
- 4) Return *key\_material*.

**Output:**

*key\_material* - derived key material.

**NOTE:** For a given set of key exchange mechanisms, the lengths of the  $k_i$ 's are independent of the execution of the protocol, i.e.  $k_1$  will be *length*<sub>1</sub>,  $k_2$  will be *length*<sub>2</sub>, etc.

A pre-shared key, *psk*, for this method and the cascade method below may be established using a previous session or an alternative key-establishment method like QKD.







NL NCSC

[https://www.ncsc.nl/binaries/ncsc/documenten/publicaties/2022/juli/guidelines-for-quantum-safe-transport-layer-encryption/guidelines-for-quantum-safe-transport-layer-encryption/Guidelines\\_for\\_PQC\\_-\\_Kyber\\_archief.pdf](https://www.ncsc.nl/binaries/ncsc/documenten/publicaties/2022/juli/guidelines-for-quantum-safe-transport-layer-encryption/guidelines-for-quantum-safe-transport-layer-encryption/Guidelines_for_PQC_-_Kyber_archief.pdf)



Gearchiveerde publicatie

Deze publicatie wordt niet meer actief onderhouden door het NCSC.  
De informatie in deze publicatie kan daarom verouderd zijn.



National Cyber Security Centre  
Ministry of Justice and Security

## Guidelines for quantum-safe transport-layer encryption

These guidelines are written for an audience of architects responsible for specifying cryptographic requirements. They can also be used in R&D and prototyping as well as for contract negotiations. For a more general introduction, see NLNCSA's [brochure](#) and our own [factsheet](#). For further details, follow [NIST](#), [ETSI](#), [IETF](#), and [ISO](#) standardisation efforts and read publications by [ENISA](#) and [TNO](#).

Apply one of the following key derivation mechanisms to get a hybrid construction:

- Concatenation of shared secrets (as specified by NIST in SP 800-56C Rev. 2) using HKDF-256
- Cascade of shared secrets (as specified by ETSI in TS 103 744) using HKDF-256



## → THE EUROPEAN SPACE AGENCY

-

## Proposal for default profile instantiation (and options):

- ## Proposal for default profile instantiation (and options):

		Default Profile		Option 1 (for missions that are very bandwidth constrained)	Option 2* (for unclassified institutional missions in EU)	Option 3
PQKEM		NIST FIPS 203 ML-KEM (Kyber): ML-KEM-768	↔	Classic McEliece parameter set TBC	Frodo parameter set TBC	ML-KEM-1024
Hybridization HKEM (PQKEM +DHKEM)		Recommended	↔	X25519 RFC 7748	ECDH - NIST SP 800 – 56Ar3 section 5.7.1.2	
In case of hybridization	TKEM / DHKEM	ECDH – NIST SP 800 – 56Ar3 section 5.7.1.2	↔	Recommended	Mandatory in EU	
	ECDH curves	NIST P-384 (NIST SP 800 -186) / secp384r1	↔	Curve 25519 RFC 7748	NIST P-384 (NIST SP 800 -186) or brainpoolP384r1 (RFC 8734)	Curve 448 RFC 7748
	KEM Combiner	NIST SP 800 227 with a two step KDF from NIST SP 800 56C	↔	NIST SP 800 227 with a two step KDF from NIST SP 800 56C	ETSI CatKDF, CASCADE KDF or NIST SP 800 227	
KDF (two step)		NIST SP 800 56 two-step KDF with HMAC-SHA3-256 in Feedback Mode (NIST SP 800 108) – for “final” key derivation, as well as for the derivation of the ECDH shared key from the ECDH shared secret	↔	Same as default profile		

\* Note that in the case of institutional missions, we recommend that mission designers **consult the responsible NSAs directly**. The “institutional” options presented here are compiled from publicly available Technical Reports and Recommendations published by various EU NSAs at the time of writing, and considered “common ground”.