ETSI TS 104 015 V1.1.1 (2025-02)

Cyber Security (CYBER); Quantum-Safe Cryptography (QSC); Efficient Quantum-Safe Hybrid Key Exchanges with Hidden Access Policies

Reference
DTS/CYBER-QSC-0023

Keywords
anonymity, authentication, encryption,
key exchange, quantum safe cryptography,
traceability

ETSI

650 Route des Lucioles
F-06921 Sophia Antipolis Cedex - FRANCE

Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Siret N° 348 623 562 00017 - APE 7112B
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° w061004871

Contents

Intellectual property rights

Essential patents

IPRs essential or potentially essential to normative deliverables may have been declared to ETSI. The declarations pertaining to these essential IPRs, if any, are publicly available for **ETSI members and non-members**, and can be found in ETSI SR 000 314: *"Intellectual Property Rights (IPRs); Essential, or potentially Essential, IPRs notified to ETSI in respect of ETSI standards"*, which is available from the ETSI Secretariat. Latest updates are available on the ETSI IPR online database.

Pursuant to the ETSI Directives including the ETSI IPR Policy, no investigation regarding the essentiality of IPRs, including IPR searches, has been carried out by ETSI. No guarantee can be given as to the existence of other IPRs not referenced in ETSI SR 000 314 (or the updates on the ETSI Web server) which are, or may be, or may become, essential to the present document.

Trademarks

The present document may include trademarks and/or tradenames which are asserted and/or registered by their owners. ETSI claims no ownership of these except for any which are indicated as being the property of ETSI, and conveys no right to use or reproduce any trademark and/or tradename. Mention of those trademarks in the present document does not constitute an endorsement by ETSI of products, services or organizations associated with those trademarks.

DECT™, **PLUGTESTS™**, **UMTS™** and the ETSI logo are trademarks of ETSI registered for the benefit of its Members. **3GPP™**, **LTE™** and **5G™** logo are trademarks of ETSI registered for the benefit of its Members and of the 3GPP Organizational Partners. **oneM2M™** logo is a trademark of ETSI registered for the benefit of its Members and of the oneM2M Partners. **GSM®** and the GSM logo are trademarks registered and owned by the GSM Association.

Foreword

This Technical Specification (TS) has been produced by ETSI Technical Committee Cyber Security (CYBER).

Modal verbs terminology

In the present document "**shall**", "**shall not**", "**should**", "**should not**", "**may**", "**need not**", "**will**", "**will not**", "**can**" and "**cannot**" are to be interpreted as described in clause 3.2 of the ETSI Drafting Rules (Verbal forms for the expression of provisions).

"**must**" and "**must not**" are **NOT** allowed in ETSI deliverables except when used in direct citation.

Executive summary

The present document will provide a framework to build Key Encapsulation Mechanisms (KEMs) retaining both pre-quantum and post-quantum security through hybridization, with additional features for practical use, while being efficient enough for browser or mobile use. Namely, keys will be encapsulated with respect to user-attributes, the encapsulations will be anonymous, and any user having attributes fulfilling the encapsulation policy will be able to retrieve the keys, while those who are not authorized will not be able to. Since many users could have the same attributes, the scheme includes an optional tracing feature, in which a tracing authority would have the means to distinguish users with the same attributes, to possibly later deactivate rights in case of abuse.

Introduction

Key Encapsulation Mechanisms (KEMs) provide the most efficient practical instantiations of Public-Key Encryption (PKE) mechanisms when combined with a Data Encapsulation Mechanism (DEM) to encrypt large amounts of data. The KEM-DEM paradigm introduced by Shoup in [i.4] wisely combines a public-key scheme together with a symmetric encryption scheme to create a scheme with ciphertexts of similar size to plaintexts.

In short, KEMs provide a tool to transmit session keys. First, a user runs the encapsulation procedure with respect to a recipient or set of recipients, generating a session key and an encapsulation of the latter. The recipients are provided with this encapsulation and, if they were among the set of intended recipients, are then able to derive the session key from it. The payload is then encrypted/decrypted under this session key using a DEM, which can be implemented by any authenticated encryption mechanism.

In order to provide additional confidence during the post-quantum migration, it is possible to hybridize two KEMs so that the security of the scheme relies on the stronger of two component algorithms. That is, if one component KEM algorithm is vulnerable to a cryptographic attack, then the privacy of the encapsulated keys is nonetheless maintained. This can be done with one pre-quantum and one post-quantum secure scheme (post-quantum schemes are resistant to adversaries with a cryptographically relevant quantum computer).

Moreover, for fine-grained access-control of users able to decrypt the payload, one just needs fine-grained access-control on the KEM part. Attribute-Based Encryption (ABE) - which often first generates an encapsulated session key, in the KEM-DEM paradigm- has been proposed to control decryption with respect to attributes and policies in ciphertexts and a user's keys [i.2]. More advanced ABE schemes have been proposed in the literature to handle complex access policies, but at a high computational cost and large ciphertexts, in particular when one wants post-quantum security.

To give two specific examples, a first work [i.3] proposed key-policy ABE, where a Boolean formula (the policy) is associated to the user's key, and attributes associated to the ciphertext, so that the user can decrypt if and only if the Boolean formula accepts on the ciphertext's attributes. The more general work [i.2] also defines ciphertext-policy ABE, where a Boolean formula (the policy) is associated to the ciphertext, and attributes associated to the user's key, so that the user can decrypt if and only if Boolean formula in the ciphertext accepts on the user's attributes.

The present document follows the approach of ciphertext-policy ABE, with policies with particular properties for efficiency reasons.

The scheme specified in the present document targets particular access-structures, with several orthogonal dimensions, with a hybrid KEM, providing fine-grained access control, key rotation to allow dynamicity of users and user's rights and an optional traceability feature that allows detection of abuse by individual users. It is described in a black box model, allowing component cryptographic algorithms to be selected according to the preferences of the implementer.

1 Scope

The present document specifies methods to efficiently build and instantiate Key Encapsulation Mechanisms (KEMs) with hidden access policies, while having the privacy of encapsulated keys relying on the best security of two hybridized schemes, namely with an instantiation where the privacy relies on the Computational Diffie-Hellman (CDH) classical assumption and the Learning With Errors (LWE) post-quantum assumption. Both problems have to be broken to endanger the privacy of the encapsulated key.

2 References

2.1 Normative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

Referenced documents which are not found to be publicly available in the expected location might be found in the ETSI docbox.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long term validity.

The following referenced documents are necessary for the application of the present document.

[1] NIST SP 800-186: "Recommendations for Discrete Logarithm-Based Cryptography: Elliptic Curve Domain Parameters".

[2] IETF RFC 7748: "Elliptic Curves for Security".

[3] NIST SP800-185: "SHA-3 Derived Functions: cSHAKE, KMAC, TupleHash and ParallelHash".

[4] FIPS PUB 180-4: "Secure Hash Standard (SHS)".

[5] FIPS PUB 202: "SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions".

[6] FIPS PUB 203: "Module-Lattice-Based Key-Encapsulation Mechanism Standard".

2.2 Informative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long term validity.

The following referenced documents are not necessary for the application of the present document but they assist the user with regard to a particular subject area.

[i.1] Théophile Brézot, Paola de Perthuis, and David Pointcheval: "Covercrypt: an Efficient Early-Abort KEM for Hidden Access Policies with Traceability from the DDH and LWE". ESORICS 2023.

[i.2] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters: "Attribute-Based Encryption for Fine-Grained Access Control of Encrypted Data". ACM CCS 2006.

[i.3] Amit Sahai and Brent Waters: "Fuzzy identity-based encryption". EUROCRYPT 2005.

[i.4] ISO/IEC 18033-2: "Information technology — Security techniques — Encryption algorithms — Part 2: Asymmetric ciphers".

3 Definition of terms, symbols and abbreviations

3.1 Terms

For the purposes of the present document, the following terms apply:

adversary's advantage: probability for an adversary to distinguish two distributions

NOTE: Formally, for an adversary A, given two distributions D_0 and D_1, the advantage is defined as:

$$\mathrm{Adv}(A) = \Pr_{D_1}[A(x) = 1] - \Pr_{D_0}[A(x) = 1] = 2 \cdot \Pr_{b,D_b}[A(x) = b] - 1.$$

negligible probability in κ: probability that is smaller than the inverse of any polynomial in κ, for κ large enough

oracle access: efficient evaluation of a function for inputs of their choice

overwhelming probability in κ: probability p such that 1-p is negligible in κ

polynomial time: running time can be expressed as a polynomial in the security parameter

security parameter: number of bits in the security level

NOTE: If the security parameter is equal to κ, then the security should hold except with probability less than $2^{-\kappa}$.

3.2 Symbols

For the purposes of the present document, the following symbols apply:

1^κ	The security parameter κ taken as input to an algorithm
$\|\|$	The logical (non-exclusive) OR
&&	The logical AND
$\oplus$	The logical XOR
$x \leftarrow f(y)$	x is the output of the algorithm f applied to the input y. Unless stated otherwise, f is a randomized algorithm, implicitly also using an input of random coins
$x \xleftarrow{\$} S$	x is drawn from a uniform distribution on the finite set S
¬A	For an event A, the event in which A does not happen
D = { A : B }	The distribution of B given A (where A will specify the distribution from which B is taken). For instance, $D = \{ a \xleftarrow{\$} S : a \}$ denotes the distribution of a knowing that a is drawn from a uniform distribution on the finite set S
$f : X \rightarrow Y$	The function f takes input values in the space X and outputs values in Y
⊥	Output to an algorithm that indicates that it has failed and returns nothing, except for the indication that it did not terminate correctly

3.3 Abbreviations

For the purposes of the present document, the following abbreviations apply:

ABE	Attribute-Based Encryption
CCA	Chosen-Ciphertext Attacks
CDH	Computational Diffie-Hellman
CPA	Chosen-Plaintext Attacks
DEM	Data Encryption Mechanism
DNF	Disjunctive Normal Form
IND	INDistinguishability
KEM	Key Encapsulation Mechanism
KEMAC	KEM with Access Control
LWE	Learning With Errors
NIKE	Non-Interactive Key Exchange

PKE	Public-Key Encryption
PK-IND	Public-Key privacy INDistinguishability
PPT	Probabilistic Polynomial Time
SK-IND	Session-Key privacy INDistinguishability

4 Cryptographic primitives

4.1 Hash functions

Hash functions are used to produce a fixed length random output y from an arbitrary length input x:

- $\mathsf{H}(x) \rightarrow y$

Approved hash functions for the purpose of the present document are:

- SHA-256, SHA-384, SHA-512, SHA-512/256 as defined in FIPS PUB 180-4 [4].
- SHA3-256, SHA3-384, SHA3-512 as defined in FIPS PUB 202 [5].

4.2 Key Encapsulation Mechanisms (KEMs)

4.2.1 KEMs description

A Key Encapsulation Mechanism KEM is a public-key scheme defined by three algorithms:

- $\mathsf{KEM}.\mathrm{KeyGen}(1^{\kappa}) \rightarrow (\mathsf{pk}, \mathsf{sk})$: on input of a security parameter κ, returns a public key pk and a secret key sk;
- $\mathsf{KEM}.\mathrm{Enc}(\mathsf{pk}) \rightarrow (C, K)$: on input of the public key pk, generates a session key K, and its encapsulation C, and returns (C, K);
- $\mathsf{KEM}.\mathrm{Dec}(\mathsf{sk}, C) \rightarrow K$: on input of the encapsulation C and the secret key sk, returns the session key K encapsulated in C.

Correctness: A KEM is said to achieve correctness if the probability that KEM.Dec(sk,C) is not equal to K is negligible in κ, on the distribution of $\{(\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{KEM}.\mathrm{KeyGen}(1^{\kappa}), (C,K) \leftarrow \mathsf{KEM}.\mathrm{Enc}(\mathsf{pk})\}$.

Security: The main goal of a KEM is to encapsulate a session key K that can only be recovered from the encapsulation C with knowledge of the secret key. This is called Session Key Indistinguishability (SK-IND). One may also wish to protect the privacy of the recipient, meaning that an adversary cannot identify whether a particular encapsulation has been prepared for a specific public key. This is called Public Key Indistinguishability (PK-IND).

The adversary can be modelled as having access to an encapsulation oracle (equivalently, the KEM's public key), in which case the scheme should be resistant to a Chosen Plaintext Attack (CPA security), or having additional access to a decapsulation oracle, in which case the scheme should be resistant to a Chosen Ciphertext Attack (CCA security). The adversary is not allowed to submit any challenge values to the decapsulation oracle.

For a more detailed description of these properties, including the precise security games, see clause A.1.

Approved KEMs for the purpose of the present document are:

- ML-KEM [6].

4.2.2 KEMs with Access Control (KEMAC)

When several users are in a KEM system, a KEM with Access Control (KEMAC) can issue users keys according to a key-policy Y, and encapsulate session keys with respect to an encapsulation-policy X, so that a user with key-policy Y can decapsulate if and only if R(X,Y) evaluates to 1, for a fixed Boolean rule R. Said differently, the access control is defined with respect to the rule R on policies X and Y. For any user-policy Y and encapsulation-policy X, R(X,Y) evaluates to 1 if the user with keys corresponding to the policy Y is allowed to decapsulate an encapsulation made with the policy X; else, R(X,Y) evaluates to 0.

A KEMAC KEMAC is defined with the following algorithms:

- KEMAC.Setup(R,1^κ) → (MPK, MSK): on input of the rule R and the security parameter κ, outputs the global public parameters MPK and the master secret key MSK;
- KEMAC.KeyGen(MSK, Y) → USK: on input of the master secret key MSK and the user-policy Y, outputs a user secret key USK;
- KEMAC.Enc(MPK, X) → (C, K): on input of the global public parameters MPK and the encapsulation -policy X, outputs the session key K and an encapsulation C of K;
- KEMAC.Dec(USK, C) → K: on input of the user secret key USK and the encapsulation C, outputs the key K encapsulated in C.

Correctness. KEMAC is said to achieve correctness with respect to the rule R if for each user-policy Y and encapsulation-policy X such that R(X,Y)=1, given the security parameter κ, the distribution of user keys built with respect to Y, and of encapsulations C of keys K with respect to the policy X is such that except with probability negligible in κ, the decapsulation of C using these user keys is equal to K.

Security. The challenge setup consists of chosen policies X and Y according to R, a random key pair (MPK,MSK) ← KEM.Setup (R,1^κ), a random encapsulation (C,K_0)←KEM.Enc(MPK, X), a random bit *b*, and a random key K_1. For SK-IND-CPA security, given (C, K_b), no adversary, that can only ask keys for user-policies Y' such that R(X,Y')=0, can guess *b* with non-negligible advantage. Note that allowing key queries for a user-policy Y' such that R(X,Y')=1 would allow decapsulating C, and trivially guess *b*. For PK-IND-CCA security, the adversary has additional access to a decapsulation oracle, which provides the encapsulated key K for any encapsulation C' under a key USK, according to any user-policy Y', except for the challenge encapsulation C.

Traceability. An optional feature of a KEMAC is offering traceability in the case of a pirate decoder in which a particular user's key has been embedded. This is a recommended but not required feature, which gives each user distinct keys even if they have common attributes. Several levels of traceability exist. The simplest one is called *white-box tracing*, where from the key extracted in the pirate decoder one can trace back the traitor. In this case, the KeyGen algorithm takes an additional input U, the identity of the user. Then no adversary should be able to design a decapsulating algorithm that uses a key that does not correspond to a user U.

4.2.3 NIKE-based KEM

A Non-Interactive Key Exchange (NIKE) is defined by two algorithms:

- NIKE.KeyGen(1^κ) → (pk, sk): on input of a security parameter κ, returns a public key pk and a secret key sk;
- NIKE.SessionKey (sk, pk') → K: on input of a secret key sk and a public key pk', generates a session key K.

With the two properties:

- **Correctness**: for any (pk_0, sk_0), (pk_1, sk_1) ← NIKE.KeyGen(1^κ), NIKE.SessionKey(sk_1, pk_0) = NIKE.SessionKey(sk_0, pk_1);
- **Security**: for (pk_0, sk_0), (pk_1, sk_1) ← NIKE.KeyGen(1^κ), *K* ← NIKE.SessionKey(sk_1, pk_0), given (pk_0, pk_1) only, recovering *K* is hard.

Then one can derive a KEM:

- KEM.KeyGen(1^κ) → (pk, sk): for (pk, sk) ← NIKE.KeyGen(1^κ);

- KEM.Enc(pk) → (C, K): for (pk', sk') ← NIKE.KeyGen(1^κ) and K ← NIKE.SessionKey(sk', pk), then C ← pk';
- KEM.Dec(sk, C) → K': for K' = $\mathcal{H}$(K), with K ← NIKE.SessionKey(sk, pk'), where pk' ← C.

4.2.4 Key-Homomorphic NIKE (KH-NIKE)

A NIKE is called key-homomorphic, if there are two internal group-laws $\otimes$,$\odot$ on the secret and the public keys that make them correspond to each other: from (pk_0, sk_0), (pk_1, sk_1) ←NIKE.KeyGen(1^κ), the secret key $sk \leftarrow sk_0 \otimes sk_1$ corresponds to the public key $pk \leftarrow pk_0 \odot pk_1$. So, for any scalar x, the secret key $sk' \leftarrow x \,.\, sk = sk \otimes ... \otimes sk$ corresponds to the public key $pk' \leftarrow x \,.\, pk = pk \odot ... \odot pk$.

Approved KEMs for the purpose of the present document are based on the Diffie-Hellman NIKE in a group (**G**, P, p), where P is a generator of **G**, of prime order p.

The DH algorithms are:

- DH.KeyGen(1^κ) → (pk, sk): for $sk \xleftarrow{\$} [\![1; p-1]\!]$ and pk ← sk.P;
- DH.SessionKey (sk_1, pk_2) → Q: where Q ← sk_1.pk_2.

The security relies on the Computational Diffie-Hellman (CDH) problem, and it provides key homomorphism.

Approved NIKEs for the purpose of the present document are the above DH scheme on elliptic curves where **G** is instantiated with the P-256, P-384 and P-521 [1] or the Curve25519 and Curve448 [2] elliptic curves.

5 Hybrid Traceable KEMAC (HTKEMAC)

5.1 Description

After the definitions given in previous clauses, this clause specifies the KEM instantiation recommended in the present document, combining hybridization, access control and traceability, from a set Ω of rights (which are combinations of attributes, as shown below) that defines the rule: for any pair (X, Y) of subsets of Ω, $\mathsf{R}(X, Y) = 1$ if and only X and Y have a non-empty intersection. As already explained, X and Y will be the encapsulation-policy and the user-policy, respectively, as lists of rights or equivalently lists of the indices of the rights in the set Ω.

It makes use of a key-homomorphic NIKE (with secret keys in $[\![1; p-1]\!]$) and a KEM with output session keys in $\mathsf{K} = \{0,1\}^{256}$. It will use the following notations:

- $\Omega = \{S_1, \ldots, S_N\}$ is the set of rights, as described in clause 6;
- **G** is a group of prime order p, in which the CDH is assumed to be hard. It will be instantiated with the P-256, P-384 and P-521 [1] or the Curve25519 and Curve448 [2] elliptic curves;
- KEM is a KEM scheme achieving SK-IND-CCA and PK-IND-CCA security. It will be implemented with ML-KEM [6];
- $\mathcal{G}$, $\mathcal{H}$, $\mathcal{J}$ are hash functions, mapping elements to $[\![0; p-1]\!]$, 256-bit strings and 384-bit string respectively where p is the order of group **G**, an elliptic curve field defined by the curve. They will be implemented with SHAKE [3], [5].

The algorithms are:

- HTKEMAC.Setup(Ω, 1^κ) → (MPK, MSK): for **G** a group of prime order p corresponding to the security parameter κ, and P a generator of **G**:
  - the algorithm samples $(H, s), (P_1, s_1), (P_2, s_2) \leftarrow \text{NIKE.KeyGen}(1^\kappa)$;
  - the set of user identities ID, is initialized as an empty set, the tracing secret key is then set to: tsk ← $(s, s_1, s_2, \mathsf{ID})$ and the tracing public key to: tpk ← (P, H, P_1, P_2);

- the set of users' secret keys showing their permissions is initialized as an empty set with $\mathsf{UP} \leftarrow \emptyset$;
- for each right S_i of index i in Ω, the algorithm samples $(\mathrm{pk_i}, \mathrm{sk_i}) \leftarrow \mathrm{KEM.KeyGen}(1^\kappa)$, $(X_i, x_i) \leftarrow \mathrm{NIKE.KeyGen}(1^\kappa)$, computes $H_i \leftarrow \mathrm{NIKE.SessionKey}(s, X_i)$, and sets $\mathrm{pk_i}' \leftarrow (X_i, \mathrm{pk_i})$ and $\mathrm{sk_i}' \leftarrow (x_i, \mathrm{sk_i})$;
- finally, the global public key is set to $\mathsf{MPK} \leftarrow (\mathrm{tpk}, \{\mathrm{pk_i}'\}_i)$, and the master secret key to $\mathsf{MSK} \leftarrow (\mathrm{tsk}, \{\mathrm{sk_i}'\}_i, \mathsf{UP})$.

The algorithm returns (MPK, MSK).

- HTKEMAC.KeyGen(MSK, U, Y) → (USK, MSK', tsk'): on input a username U, along with Y a set of indices corresponding to U's rights in Ω, parsing the master secret key MSK as an output of the Setup algorithm:
  - draws $\alpha[\![1; p-1]\!]$, sets β to be the representative of $((s - \alpha s_1)/s_2 \ mod\ p)$ that is in $[\![0; p-1]\!]$, so that $s = (\alpha.s_1) \otimes (\beta.s_2) \ mod\ p$ and thus $H = (\alpha.P_1) \odot (\beta.P_2)$, and sets U's secret identifier to $\mathrm{uid} \leftarrow (\alpha, \beta)$;
  - updates the tracing secret key by setting tsk' to be equal to tsk in which (U, uid) is added to ID;
  - U's secret key is defined as $\mathsf{USK} \leftarrow (\mathrm{uid}, \{\mathrm{sk'_j}\}_{j \in Y})$, and the master secret key is updated to MSK' equal to MSK in which USK was added to UP.

  Finally, the algorithm outputs (USK, MSK', tsk').
- HTKEMAC.Enc(MPK, X) → (C, K): parsing the public key MPK as an output of the Setup algorithm, and X as a set of indices of rights in Ω:
  - denoting K the key space of KEM, the encryption algorithm draws $S \xleftarrow{\$} \mathsf{K}$, $r \leftarrow \mathcal{G}(\mathrm{S})$, sets $c_1 \leftarrow r.P_1$ $c_2 \leftarrow r.P_2$ and $c \leftarrow (c_1, c_2)$, and for each index i in X, sets $K_i \leftarrow \mathrm{NIKE.SessionKey}(r, H_i)$, $(E_i, K'_i) \leftarrow \mathrm{KEM.Enc}(\mathrm{pk_i})$, and $F_i \leftarrow S \oplus \mathcal{H}(K_i, K'_i, c, \{E_l\}_{l \in X})$
  - the algorithm then computes $(K, V) \leftarrow \mathcal{J}(S, c, \{E_i, F_i\}_{i \in X})$, sets the encapsulation as $C \leftarrow (c, \{E_i, F_i\}_{i \in X}, V)$ and the encapsulated key to be *K*.

  The algorithm outputs (*K*, *C*).
- HTKEMAC.Dec(USK, *C*) → *K*: parsing USK as an output of the KeyGen algorithm, and $C = (c = (c_1, c_2), \{E_i, F_i\}_{i \in X}, V)$ as an output of the Enc algorithm, for each index $i \in X$ with a pair (E_i, F_i) in C, and for each index $j \in Y$ with an element $\mathrm{sk_j}'$ in USK, the decryption algorithm:
  - runs $K'_{i,j} \leftarrow \mathrm{KEM.Dec}\ (\mathrm{sk_j}, \mathrm{E_i})$;
  - computes $K_j \leftarrow \mathrm{NIKE.SessionKey}(x_j, (\alpha.\mathrm{c_1}) \odot (\beta.c_2))$;
  - computes $S_{i,j} \leftarrow F_i \oplus \mathcal{H}(K_j, K'_{i,j}, c, \{E_l\}_{l \in X})$;

    it computes both $r' \leftarrow \mathcal{G}(S_{i,j})$ and $(U'_{i,j}, V'_{i,j}) \leftarrow \mathcal{J}(S_{i,j}, c, \{E_i, F_i\}_{i \in X})$, and checks whether $c = (r'.P_1, r'.P_2)$ and $V'_{i,j} = V$, returns $K \leftarrow U'_{i,j}$ and stops. Otherwise, it continues with the next pair (*i*,*j*).

  If for all indices i and j, no key was returned, the algorithm returns ⊥.

When instantiated as above, the HTKEMAC hybridizes a pre-quantum NIKE scheme and a post-quantum KEM scheme to obtain the best of both their secret-key privacy: it provides CCA security for secret-key privacy under the CDH or the MLWE problems, and CCA security for access-control privacy under both the CDH and the MLWE problems.

5.2 Parameter sets

Parameter sets consist of tuples of specific choices for the hash function SHAKE [3], [5], the ML-KEM [6], and the elliptic curves [1], [2]:

- SHAKE128_P256_ML-KEM-512

- SHAKE128_Curve25519_ML-KEM-512
- SHAKE256_P384_ML-KEM-768
- SHAKE256_Curve448_ML-KEM-768
- SHAKE256_P521_ML-KEM-1024

6 Access structure

6.1 High-level description

The previous clause describes an access control from two subsets X and Y of rights in Ω associated to the encapsulations and the user's keys respectively, so that the latter can decrypt the former if and only if $X \cap Y \neq \emptyset$. This clause explains how to transform an access structure and a Boolean policy into the set of all the possible rights Ω and the desired subsets X and Y.

The access structure is specified by rights, which are combinations of attributes along different dimensions. For the sake of clarity, one can consider a concrete case, where there are three **dimensions** of **attributes**:

- The countries `CTR={EN,FR}`
- The departments `DPT={DEV,MKG}`
- The security levels `SEC=(LOW,MED,HIG)`

This defines the following **qualified attributes** along the 3 dimensions: `CTR::FR`, `CTR::EN`, `DPT::DEV`, `DPT::MKG`, and `SEC::LOW`, `SEC::MED`, `SEC::HIG`. The two first dimensions `CTR` and `DPT` are defined by unordered sets, whereas the last security level `SEC` is defined by an ordered set, meaning that a user with the `SEC::HIG` attribute also possesses the `SEC::LOW` and `SEC::MED` qualified attributes, as $\texttt{SEC::HIG} \Rightarrow \texttt{SEC::MED} \Rightarrow \texttt{SEC::LOW}$, or equivalently $\texttt{SEC::LOW} \leq \texttt{SEC::MED} \leq \texttt{SEC::HIG}$, whereas attributes within the dimensions `CTR` and `DPT` are incomparable.

A right is a combination of attributes. Such a right is **valid** when represented as a list of attributes if it involves (some or none) attributes of different dimensions only. One can then define Ω as the set of valid rights, that are enough to define any monotonous access structure. One can note this includes fully defined rights, such as {`CTR::FR`, `DPT::MKG`, `SEC::MED`}, but also partially defined rights, such as {`CTR::FR`}, {`SEC::HIG`}, or even the empty one {}, which designates a broadcast. This notation is used to optimize the encapsulation size, with several kinds of broadcasts along the different dimensions: an encapsulation with right {`CTR::FR`} can be decrypted by users with rights {`CTR::FR`, `DPT::MKG`, `SEC::MED`} or {`CTR::FR`, `DPT::DEV`}, and even more. In order for decryption to be possible, a user-right has to contain all of the attributes required by an encapsulation-right. Hence, the sets X and Y will have to be carefully derived, as explained below, to allow decryption of an encapsulation under X by a user's key under Y if and only if $X \cap Y \neq \emptyset$.

The full ordering along some dimensions can be extended to a partial ordering between all the rights, in a trivial way: for the right r_1 to be smaller than the right r_2, denoted $r_1 \leq r_2$, one requires that along each dimension, the attribute in r_1 has to be absent, equal to or smaller (if comparable) than the attribute in r_2. For example, {`CTR::FR`, `SEC::MED`} $\leq$ {`CTR::FR`, `DPT::MKG`, `SEC::HIG`}, and {`CTR::FR`} $\leq$ {`CTR::FR`, `DPT::MKG`}, but {`CTR::FR`, `SEC::HIG`} and {`CTR::FR`, `DPT::MKG`, `SEC::MED`} are incomparable.

Hence, when one expresses the access control for a given encapsulation (in the Ciphertext-Policy ABE spirit), by a specific monotonous Boolean formula F, it can be converted into its Disjunctive Normal Form (DNF), that is a disjunction of clauses. Clauses are exactly the above rights. The encapsulation will be associated to all the rights/clauses in the DNF, whereas the user's key will be associated to all the rights/clauses owned by the user. The following explains how to find the smallest encapsulation within the above framework. This allows the minimization of encapsulation sizes.

6.1.1 Attributes, dimensions and hierarchies

Let a dimension be a set of attributes. Each dimension is constrained to be either:

- a *hierarchy* H, which defines a set of ordered attributes:

$$\forall (A, B) \in H^2, (A \leq B) \text{ or } (B \leq A)$$

- an *anarchy* D, which defines a set of incomparable attributes:

$$\forall (A, B) \in D^2, (A = B) \text{ or } (\neg(A \leq B) \text{ and } \neg(B \leq A))$$

A Boolean policy can then be described as a disjunction and/or conjunction of attributes.

Each dimension is further constrained to be uniquely named, and each attribute has to have a unique name across its dimension. Then, the following context-free grammar can be used to parse an access policy:

```
<access-policy>       = <qualified-attribute>
                        | <access-policy> <op> <access-policy>
<qualified-attribute> = <dimension>::<attribute>
<dimension>           = <name>
<attribute>           = <name>
<op>                  = `&&' | `||'
<name>                = [^ `&&' `||' `::']
```

where `[^ '&&' '||' '::']` stands for any non-empty combination of characters (`name`) in which neither '`&&`', '`||`' nor '`::`' occur.

6.1.2 Spatial representation

Without loss of generality, a right can be defined as a conjunction of attributes belonging to different dimensions. The universe Ω can then be defined as the set of all those conjunctions. Moreover, there is a one-to-one correspondence between an access policy on an access structure with d dimensions, and a set of points in a d-dimensional space. Each right r is associated to a point in the d-dimensional space:

- the *associated point* P_r of the right r is the only point that belongs to this access policy, but in no further restriction of it.

EXAMPLE: One can associate rights to points, by associating attributes to a specific integer along the dimension. The absence of an attribute in a dimension is then represented by 0:

- `CTR::EN && DPT::DEV` to $(1,1)$;
- `CTR::EN && DPT::MKG` to $(1,2)$;
- `CTR::FR && DPT::DEV` to $(2,1)$;
- `CTR::FR && DPT::MKG` to $(2,2)$;
- `CTR::EN` to $(1,0)$;
- `CTR::FR` to $(2,0)$;
- `DPT::DEV` to $(0,1)$;
- `DPT::MKG` to $(0,2)$.

The total number of rights in a 2-dimensional universe Ω is therefore 8 and becomes 9 if one adds a right that maps to the origin 0_Ω, corresponding to the global broadcast, denoted right `BROADCAST`. These points will be used for an encapsulation-policy X. On the other hand, for a user-policy, they will be expanded for efficiency reasons, in order to represent the Boolean policy on rights in the key. This allows the use of the smaller of the two sets for encapsulation generation, which minimizes the size.

One can furthermore define a partial ordering on the rights and/or points. This was explained above for rights and can be translated as follows for points:

- For the rights: $r_1 \leq r_2$, if and only if for each dimension, the attribute in r_1 shall be absent, equal to or smaller, for a hierarchical dimension, than the attribute in r_2;
- For the points: $P_1 = (a_i)_i \leq P_2 = (b_i)_i$ if and only if for each dimension i: $a_i = 0$ or $a_i \leq b_i$ for a hierarchical dimension.

6.2 Efficiency considerations

As explained above, a Boolean formula F expresses the access control for a given encapsulation. After the DNF conversion, this leads to the set X with all the clauses (or points associated to the rights) that appear in the DNF. However, building the set Y associated to the attributes for the user's key is more complex.

6.2.1 Encapsulation rights

Once converted into its DNF, the Boolean formula F associated to an encapsulation is a list of rights from Ω. This is the set X.

6.2.2 User's key rights

Two additional spaces have to be defined to build the set Y:

- the *semantic space* $\mathbf{sem}(\Omega, r)$ of a right r is the subspace of points in which the attributes involved in this right can be expressed.

EXAMPLE: In the 3-dimensional space, the semantic space of **`CTR::EN`**, associated to $(1,0,0)$, is the 1-dimensional subspace of Ω generated by the **`CTR`** dimension: $< (1,0,0) >$. The semantic space of the right **`CTR::FR && DPT::DEV && SEC::MED`**, associated to $(2,1,2)$, is the whole space $\Omega = < (1,0,0), (0,1,0), (0,0,1) >$.

- the *complementary space* $\mathbf{comp}(\Omega, r)$ of a right r is defined as $\mathbf{comp}(\Omega, r) = \{P_{r'}, r' \leq r\} + (\Omega \setminus \mathbf{sem}(\Omega, r))$.

EXAMPLE: The complementary space of **`CTR::EN`** is the 2-dimensional space $< (0,1,0), (0,0,1) >$ generated by **`DPT`** and **`SEC`** which origin is $(1,0,0)$:
$\{(1,0,0), (1,1,0), (1,1,2), (1,1,3), (1,2,0), (1,2,1), (1,2,2), (1,2,3)\}$.

In case of a hierarchical dimensions, one has to consider all the origins that are implied by the right r: the complementary space of **`CTR::FR && DPT::DEV && SEC::MED`** is the combination of the 0-dimensional spaces which origins are $(2,1,1), (2,1,2)$: $\{(2,1,1), (2,1,2)\}$.

As a final example, the complementary space of **`CTR::FR && SEC::MED`** is the combination of the 1-dimensional spaces $< (0,1,0) >$ generated by **`DPT`** which origins are $(2,0,1), (2,0,2)$:
$\{(2,0,1), (2,1,1), (2,1,2), (2,0,2), (2,1,2), (2,2,2)\}$.

In the user-policy set Y, all the points in all the subsets associated to the rights are concatenated, plus the `BROADCAST`, associated to the origin point 0_Ω, that is $(0,0,0)$ in the 3-dimensional case.

6.2.3 Cardinality of an encapsulation

Following the previous description, the number of rights used in an encapsulation should be equal to the number of clauses in the DNF of its associated access policy, since each clause is either a broadcast to a subspace of Ω, or to a singleton.

6.2.4 Cardinality of a user secret key

The number of rights associated to a user secret key is the number of points in the complementary spaces generated by the rights associated to the user.

7 Specification

7.1 Introduction

This clause specifies all the objects and functions needed to implement Covercrypt, with their input/output types, where the brackets (such as `[* Name]`) stand for optional components.

7.2 Access Structure

7.2.1 Type

An `AccessStructure` is a set of dimensions, where a `Dimension` is an object that holds its name and attributes, which are themselves composed of their name and an id that is unique across this access structure:

```
AccessStructure = Set Dimension
Dimension       = Hierarchy (Name * OrderedSet (Id * Name))
                | Anarchy (Name * Set        (Id * Name))
```

Then a right is uniquely associated to each set of IDs from an access structure:

```
Right = Set Id
```

7.2.2 API

An object of type `AccessStructure` should expose the following API:

- `ap_to_usk_rights (AccessStructure * AccessPolicy) -> Set Right`
  Generates the set of USK rights described by the given access policy.

- `ap_to_enc_rights (AccessStructure * AccessPolicy) -> Set Right`
  Generates the set of ciphertext rights described by the given access policy.

- `add_anarchy (AccessStructure * Name) -> AccessStructure`
  Adds an anarchic dimension with the given name to the access structure.
  Requires USK refresh: On addition of a dimension, users will need to refresh keys in order to decrypt new encapsulations with access policies including the new dimension.

- `add_hierarchy (AccessStructure * Name) -> AccessStructure`
  Adds a hierarchic dimension with the given name to the access structure.
  Requires USK refresh: On addition of a dimension, users will need to refresh keys in order to decrypt new encapsulations with access policies including the new dimension.

- `del_dimension (AccessStructure * Name) -> AccessStructure`
  Removes the dimension with the given name from the access structure.
  Requires USK refresh: On deletion of a dimension, users that refresh their keys will lose the ability to decrypt old encapsulations with access policies including the old dimension.

- `add_attribute (AccessStructure * QualifiedAttribute [* Name]) -> AccessStructure`
  Adds the given qualified attribute to the access structure.
  If the dimension is hierarchical, specifying the name of an existing attribute of the same dimension sets the rank of the new attribute to be in-between this existing attribute and the next attribute if any. Gives the new attribute the lowest rank in case no such attribute name is specified. If this name does not match any valid attribute, an error is returned.
  Specifying an attribute name when adding an attribute to an anarchy has no effect.
  Requires USK refresh: On addition of an attribute, users will need to refresh keys in order to decrypt new encapsulations with access policies including the new attribute.

- `del_attribute (AccessStructure * QualifiedAttribute) -> AccessStructure`
  Removes the given qualified attribute from the access structure or returns an error if this attribute does not belong to this access structure.
  Requires USK refresh: On deletion of an attribute, users that refresh their keys will lose the ability to decrypt old encapsulations with access policies including the old attribute.

7.3 Master Secret Key

The master secret key has the following type:

```
MSK = AccessStructure * Keypairs [* SigningKey] [* Tracers]
```

where:

- the `AccessStructure` is described above and contains all necessary information to validate and translate an access structure into a set of rights;
- the `Keypairs` structure contains the history of the keypairs associated to each right. This includes old keys that are no longer in use for encryption, because encapsulations are not necessarily re-encrypted after a key refresh;
- the `SigningKey` is used to produce publicly verifiable signatures of the user secret keys and master public keys;
- the `Tracers` is used to allow tracing user keys.

7.4 Master Public Key

A master public key has the following type:

```
MPK = Version * AccessStructure * (Right -> PublicKey) [* Signature] [* TracingPoints]
```

where:

- `Version` holds a number that is increased at each generation of a new master public key;
- `AccessStructure` holds all necessary information for validating and translating an access structure into a set of rights;
- `Right -> PublicKey` maps each right to the latest version of its associated public key;
- `Signature` holds a publicly verifiable signature generated by the master secret key;
- `TracingPoints` holds public points required in the encapsulation process if the master secret key defines tracers.

7.5 User Secret Key

A user secret key has the following type:

```
USK = SecretKeys [* Signature] [* UserId]
```

where:

- `SecretKeys` contains the history of the secret keys associated to each right that have been given to the user;
- `Signature` holds a publicly verifiable signature generated by the master secret key;
- `UserID` contains a unique identifier required for opening encapsulations if the master secret key defines tracers.

7.6 Encapsulation

An encapsulation has the following type:

```
XEnc = List Encapsulation * Tag [* Traps] [* Version]
```

where:

- `List Encapsulation` contains the encapsulation associated to each right, in a cryptographically-random order. Using a `List` allows serialization of this order, which is needed since it matters when recomputing the `Tag`;
- `Tag` is the early-abort tag that allows determination of whether opening an encapsulation was successful;
- `Traps` contains public points required in the opening process if the master secret key defines tracers;
- `Version` holds the version number of the master public key used to generate this encapsulation.

7.7 Covercrypt

A `Covercrypt` implementation exposes the following API:

- `mk_gen (AccessStructure) -> MSK * MPK`

  It takes as input the access structure and generates new master secret and public keys:

  ```
  omega ← ap_to_usk_rights(AccessStructure, "*")
  (msk, mpk) ← HTKEMAC.setup(omega)
  ```

- `usk_gen (MSK * U * AccessPolicy) -> USK`

  It takes as input the master secret key, a user and an access policy and generates a new user secret key that holds secret keys as follows, or signals an error if this access policy is invalid w.r.t. this master secret key:

  ```
  y ← ap_to_usk_rights(msk.AccessStructure, ap)
  usk ← HTKEMAC.keygen(msk, u, y)
  ```

- `enc (MPK * AccessPolicy) -> (K * XEnc)`

  It takes as input a master public key and an access policy and generates an encapsulation of a random session key for the rights returned by `ap->enc-rights` or signals an error if this access policy is invalid w.r.t. this master public key:

  ```
  x ← ap_to_enc_rights(mpk.AccessStructure, ap)
  (k, c) ← HTKEMAC.enc(mpk, x)
  ```

- `dec (USK * XEnc) -> K`

  It takes as input a user secret key associated to an access policy AP_{usk} and an encapsulation associated to an access policy AP_{enc}, and returns a session key if $AP_{usk} \cap AP_{enc} \neq \emptyset$ and this user secret key holds the correct version of at least one of secret key associated to a right in this intersection:

  ```
  k ← HTKEMAC.dec(usk, c)
  ```

- `rotate (MSK * MPK * AccessPolicy) -> MSK * MPK`

  Generates new keypairs for all rights given by `ap_to_usk_rights`.

  All user secret keys need to be refreshed, and older ciphertexts may be re-encrypted.

- `refresh (MSK * USK) -> USK`

  It takes as argument the master secret key and a user secret key and return an updated version of this user secret key that holds each newer version of the secret keys it previously held.

8 Conclusion

The present document shows the construction of a KEM which achieves privacy and correctness properties, while allowing the encapsulation of keys with respect to hidden access policies. As shown in [i.1], this scheme allows an order of magnitude speedup with respect to Attribute-Based Encryption implementations in practical use-cases in which the access-policies can be expressed with less than ten logical gates.

Annex A (informative): Security Definitions

A.1 KEM Security Definitions

Session-Key Privacy. KEM is said to achieve *session-key privacy* (denoted SK-IND or SK-IND-CPA) if session keys generated by the encapsulation algorithm are indistinguishable from elements taken uniformly at random in the session-key space K. In other words, KEM is SK-IND secure if and only if for any PPT adversary A, A has a negligible advantage in the security parameter κ in distinguishing between distributions D_0 and D_1 defined for a bit b with:

$$D_b = \left\{ \begin{array}{l} (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathrm{KEM.KeyGen}(1^\kappa), \\ (C, K_0) \leftarrow \mathrm{KEM.Enc}(\mathsf{pk}), K_1 \overset{\$}{\leftarrow} \mathrm{K} \end{array} : (\mathsf{pk}, C, K_b) \right\}$$

In the SK-IND-CCA security game, the adversary is given access to a KEM.Dec oracle, except for the challenge ciphertext.

Public-Key Privacy. Defined analogously to the anonymity of a PKE scheme, the public-key privacy (denoted PK-IND or PK-IND-CPA) of an encapsulation scheme similarly states the outputs of encapsulations using one or the other of two output public keys of the KeyGen algorithm will be indistinguishable except with probability negligible in the security parameter κ. More formally, for any PPT adversary A, defining for a bit b:

$$D_b = \left\{ \begin{array}{l} \text{For } i = 0,1: \\ (\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathrm{KEM.KeyGen}(1^\kappa), : (\mathsf{pk}_0, \mathsf{pk}_1, C_b, K_b) \\ (C_i, K_i) \leftarrow \mathrm{KEM.Enc}(\mathsf{pk}_i) \end{array} \right\}$$

The advantage of A in distinguishing between D_0 and D_1 is negligible in κ.

In the PK-IND-CCA security game, the adversary is given access to a KEM.Dec oracle, except for the challenge ciphertext.

A.2 KEMAC Security Definitions

Correctness. Formally, if defining the distribution D and event Ev as:

$$\mathrm{D} = \left\{ \begin{array}{l} \forall (X, Y) \text{ such that } \mathcal{R}(X, Y) = 1, \\ (\mathsf{MPK}, \mathsf{MSK}) \leftarrow \mathrm{KEMAC.KeyGen}(1^\kappa), \\ \mathsf{usk} \leftarrow \mathrm{KEMAC.KeyGen}(\mathsf{MSK}, Y), \\ (C, K) \leftarrow \mathrm{KEMAC.Enc}(\mathsf{PK}, X) \end{array} : (\mathsf{usk}, C, K) \right\}$$

$$\mathrm{Ev} = [\mathrm{KEMAC.Dec}(\mathsf{usk}, C) = K].$$

The probability that Ev happens on the distribution D is overwhelming in κ then the KEMAC is said to be correct.

Session-Key Privacy. KEMAC is said to be session-key private (denoted SK-IND or SK-IND-CPA) if any PPT adversary A provided with the public key MPK, and with an oracle access to the KeyGen algorithm for a set of user policies Y, choosing an encapsulation-policy X such that for each user-policy Y in Y, R(X,Y)=0, and given an encapsulation C of MPK under the policy X, can distinguish between the session-key encapsulated in C and a random element of the key-space K only with probability negligible in κ.

In the SK-IND-CCA security game, the adversary is given access to a KEMAC.Dec oracle, except for the challenge ciphertext.

In addition to avoiding leaking information about encapsulated session-keys to non-recipients, a scheme with this property will also not reveal any information about the policies used, which will be granted by the access-control privacy property defined hereafter.

Access-Control Privacy. KEMAC is said to be access-control private (denoted AC-IND-CPA) if in the same setting as in the session-key privacy security game, any PPT adversary A choosing two encapsulation policies X_0 and X_1 for which she cannot have user keys enabling decapsulation, cannot distinguish between encapsulations using one or the other policy.

In the AC-IND-CCA security game, the adversary is given access to a KEMAC.Dec oracle, except for the challenge ciphertext.

Traceability. Let *A* be a PPT adversary that can:

- ask for the generation of user keys through an oracle OKeyGen(·) taking usernames as inputs, and, on input *U*, running $USK_U \leftarrow$ KEMAC.KeyGen(MSK, *U*) and adding *U* and the corresponding key USK_U to the system;
- corrupt registered users with access to an oracle OCorrupt(·), taking as input a username *U*, adding it to the system if it is not in it yet, as well as in the list of traitors *T*, and returning *U*'s secret key USK_U to *A*.

A KEMAC is called *white-box traceable* if from the key extracted in a pirate decoder one can get the identity of the traitor.

History

Document history		
V1.1.1	February 2025	Publication