Security Analysis of Covercrypt: A Quantum-Safe Hybrid Key Encapsulation Mechanism for Hidden Access Policies

Théophile Brézot[1], Chloé Hébant[1], Paola de Perthuis[2], and David Pointcheval[1,3]

[1] Cosmian, Paris, France
[2] Centrum Wiskunde & Informatica (CWI), Nederland
[3] DIENS, École normale supérieure, université PSL, CNRS, Inria, Paris, France

Abstract. The ETSI Technical Specification 104 015 proposes a framework to build Key Encapsulation Mechanisms (KEMs) with access policies and attributes, in the Ciphertext-Policy Attribute-Based Encryption (CP-ABE) vein. Several security guarantees and functionalities are claimed, such as pre-quantum and post-quantum hybridization to achieve security against Chosen-Ciphertext Attacks (CCA), anonymity, and traceability.
In this paper, we present a formal security analysis of a more generic construction, with application to the specific Covercrypt scheme, based on the pre-quantum ECDH and the post-quantum ML-KEM KEMs. We additionally provide an open-source library that implements the ETSI standard, in Rust, with high effiency.

1 Introduction

Key Encapsulation Mechanisms (KEMs) are highly efficient when used in conjunction with Data Encryption Mechanisms (DEMs) to encrypt large volumes of data. The KEM-DEM paradigm, introduced by Shoup in [Sho01], indeed combines a public-key scheme with a symmetric encryption scheme, resulting in ciphertexts that are similar in size to plaintexts. In essence, KEMs facilitate the secure transmission of session keys. A user performs the encapsulation procedure for a recipient or group of recipients, generating a session key and its encapsulation (the ciphertext). The recipients, if intended, can derive the session key from the ciphertext. The payload is then encrypted and decrypted using this session key with a DEM, which can be any authenticated encryption mechanism. This is instrumental for non-interactive communications between a sender and a recipient. But KEMs may also be used as Interactive Key Exchange protocols, between two online parties.

To enhance security during the post-quantum transition, two KEMs may be hybridized, ensuring that the scheme's security depends on the best of both their securities. This approach maintains the privacy of encapsulated keys even if one KEM algorithm is compromised. Hybridization can involve one pre-quantum and one post-quantum secure scheme, with the latter being presumably resistant to quantum adversaries while the former relies on more classical security assumptions, even if insecure against potential future quantum adversaries.

For fine-grained access control, the KEM may be controlled to limit access to the payload. Attribute-Based Encryption (ABE), allows decryption based on attributes and policies in ciphertexts and user's keys [GPSW06]. Advanced ABE schemes support complex access policies but come with high computational costs and large ciphertexts, especially for post-quantum security.

ABE was initially introduced in [SW05], where decryption was possible if the number of common attributes in the key and ciphertext met a threshold. The more general work [GPSW06] introduced key-policy ABE (KP-ABE) and ciphertext-policy ABE (CP-ABE), associating a Boolean formula (policy) with either the user's key or the ciphertext. Decryption is possible if the Boolean formula accepts the attributes. However, it only proposed a concrete construction for KP-ABE, while the first CP-ABE scheme has been designed in [BSW07].

A desirable property for KP-ABE schemes (resp. CP-ABE) is for them to be *attribute-hiding* [OT12] (resp. *predicate-hiding*), meaning that ciphertexts should not allow users to learn

anything of the attributes or the policies they were generated with, outside of their ability (or not) to successfully decrypt. State-of-the-art constructions for this property only do so for the limited class of inner-product predicates on attributes, with pre-quantum [OT12], or more recent post-quantum versions [CW24].

State-of-the-Art. For general purpose ABE, current post-quantum versions with implementable parameters [DDP+18, Table II], implementing [BGG+14], led to encryption times in hundreds of milliseconds and decryption times (consisting in the application of the EvalCT, EvalPK, and Dec algorithms) of around a second on a CPU laptop, without attribute-hiding properties. A more recent post-quantum ABE-scheme was proposed last year [CW24] with a higher security, but incurring significantly higher costs than [BGG+14].

A first construction of ABE for polynomially-many logical attribute combinations, that can be built from any KEM, without pairings, and supporting efficient traditional/post-quantum hybridization, has been proposed two years ago [BdPP23], but with CPA security only. About functionalities, it was similar to [SW05], allowing decryption if some attributes are common in the key and ciphertext. It also attained a policy-hiding property ensuring that access policies in the ciphertexts were not revealed to users that did not fulfil them. More recently, a CCA construction has been standardized by ETSI [ETS25]. However, its CCA security had never been proven nor analyzed.

Contributions. This paper first presents a formal security analysis of a generalization of the ETSI standard [ETS25], and in particular, the CCA-secure $\mathsf{Covercrypt}$ scheme, based on the pre-quantum ECDH and the post-quantum ML-KEM. It targets specific access structures with multiple orthogonal dimensions, using a hybrid KEM for fine-grained access control, key rotation for dynamic user rights, and a traceability mechanism to detect user abuse. It is described in a black-box manner, allowing usage of various cryptographic algorithms. We have updated our library in Rust, from the CPA-version [BdPP23], that implements the standard, with additional features. The code is still open-source, and provides similar efficiency: half a millisecond for encrypting, and a millisecond for decrypting, for classical use, which is far better than [DDP+18].

2 Definitions

This paper targets Key Encapsulation Mechanism with Access Control (KEMAC), as introduced in [BdPP23]. We hereafter recall some formal definitions.

2.1 Computational Assumptions

Many security notions will be characterized by the computational indistinguishability between two distributions $\mathcal{D}_0$ and $\mathcal{D}_1$. It will be measured by the advantage an adversary $\mathcal{A}$ can have in distinguishing them:

$$\mathsf{Adv}(\mathcal{A}) = \Pr_{\mathcal{D}_1}[\mathcal{A}(x) = 1] - \Pr_{\mathcal{D}_0}[\mathcal{A}(x) = 1] = 2 \times \Pr_{\mathcal{D}_b}[\mathcal{A}(x) = b] - 1.$$

Then, we will denote $\mathsf{Adv}(\tau)$ the maximal advantage over all the adversaries with running-time bounded by τ. A first pair of distributions is used in the famous ElGamal encryption scheme and the Diffie-Hellman key exchange protocol, with Diffie-Hellman tuples in $\mathbb{G} = \langle P \rangle$, a group of prime order p, spanned by a generator P, and denoted additively.

Definition 1 (Decisional Diffie-Hellman Problem). *The DDH assumption in a group $\mathbb{G}$ ($\mathsf{DDH}_{\mathbb{G}}$) of prime order p, with a generator P, states that the distributions $\mathcal{D}_0$ and $\mathcal{D}_1$ are computationally hard to distinguish, where*

$$\mathcal{D}_0 = \{(a \cdot P, b \cdot P, ab \cdot P), a, b \xleftarrow{\$} \mathbb{Z}_p\} \qquad \mathcal{D}_1 = \{(a \cdot P, b \cdot P, c \cdot P), a, b, c \xleftarrow{\$} \mathbb{Z}_p\}$$

and we will denote $\mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}}(\mathcal{A})$ the advantage of an adversary $\mathcal{A}$ in distinguishing $\mathcal{D}_0$ and $\mathcal{D}_1$.

When studying the Kyber post-quantum encryption scheme [BDK+18], also known as ML-KEM [NIS22], we need another algebraic structure, with indistinguishable distributions. We will denote $\mathsf{R} = \mathbb{Z}[X]/(X^n+1)$ (resp. $\mathsf{R}_q = \mathbb{Z}_q[X]/(X^n+1)$) the ring of polynomials of degree at most $n-1$ with integer coefficients (resp. with coefficients in $\mathbb{Z}_q$, for a small prime q). We take n as power of 2, where X^n+1 is the $\frac{n}{2}$-th cyclotomic polynomial. We denote $\mathcal{B}_\eta$ the centered binomial distribution of parameter η. When a polynomial is sampled according to $\mathcal{B}_\eta$, it means each of its coefficients is sampled from that distribution. We will also use vectors $\mathbf{e} \in \mathsf{R}_q^k$ and matrices $\mathbf{A} \in \mathsf{R}_q^{m\times k}$ in R_q.

Definition 2 (Decisional Module Learning-with-Error Problem). *The DMLWE assumption in R_q ($\mathsf{DMLWE}_{\mathsf{R}_q,m,k,\eta}$) states that the distributions $\mathcal{D}_0$ and $\mathcal{D}_1$ are computationally hard to distinguish, where*

$$\mathcal{D}_0 = \{(\mathbf{A}, \mathbf{b}), \mathbf{A} \xleftarrow{\$} \mathsf{R}_q^{m\times k}, (\mathbf{s}, \mathbf{e}) \xleftarrow{\$} \mathcal{B}_\eta^k \times \mathcal{B}_\eta^m, \mathbf{b} \leftarrow \mathbf{A}\mathbf{s} + \mathbf{e}\}$$
$$\mathcal{D}_1 = \{(\mathbf{A}, \mathbf{b}), \mathbf{A} \xleftarrow{\$} \mathsf{R}_q^{m\times k}, \mathbf{b} \xleftarrow{\$} \mathcal{B}_\eta^m\}$$

We will denote $\mathsf{Adv}^{\mathsf{dmlwe}}_{\mathsf{R}_q,m,k,\eta}(\mathcal{A})$ the advantage of an adversary $\mathcal{A}$ in distinguishing $\mathcal{D}_0$ and $\mathcal{D}_1$.

Sometimes, the goal of the adversary $\mathcal{A}$ is more complex: $\mathcal{A}$ has to recover the exact solution to the problem, which is called the search problem, or the *one-wayness*.

Definition 3 (Computational Diffie-Hellman Problem). *The CDH assumption in a group $\mathbb{G}$ ($\mathsf{CDH}_\mathbb{G}$) of prime order p, with a generator P, states that given $a\cdot P$ and $b\cdot P$ for $a, b \xleftarrow{\$} \mathbb{Z}_p$, it is computationally hard to compute $ab\cdot P$. We denote $\mathsf{Adv}^{\mathsf{cdh}}_\mathbb{G}(\mathcal{A})$ the advantage of an adversary $\mathcal{A}$ in computing $ab\cdot P$.*

2.2 Cryptographic Primitives

Key Encapsulation Mechanism. A Key Encapsulation Mechanism KEM is defined by three algorithms:

- $\mathsf{KEM.KeyGen}(1^\kappa)$: the *key generation algorithm* outputs a pair of public and secret keys $(\mathsf{pk}, \mathsf{sk})$;
- $\mathsf{KEM.Enc}(\mathsf{pk})$: the *encapsulation algorithm* generates a session key K and a ciphertext C of it, and outputs the pair (C, K);
- $\mathsf{KEM.Dec}(\mathsf{sk}, C)$: the *decapsulation algorithm* outputs the key K encapsulated in C.

Correctness. A correct KEM satisfies $\mathsf{Adv}^{\mathsf{cor}}_{\mathsf{KEM}}(\kappa) = 1 - \Pr_\mathcal{D}[\mathsf{KEM.Dec}(\mathsf{sk}, C) = K] = \mathsf{negl}(\kappa)$, for the distribution probability $\mathcal{D} = \{(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KEM.KeyGen}(1^\kappa), (C, K) \leftarrow \mathsf{KEM.Enc}(\mathsf{pk}) : (\mathsf{sk}, C, K)\}$.

Session-Key Privacy. On the other hand, such a KEM is said to provide *session-key privacy* (denoted SK-IND, for *Session-Key Indistinguishability*) in the key space $\mathcal{K}$, if the encapsulated key is indistinguishable from a random key in $\mathcal{K}$. More formally, a KEM is SK-IND-CCA-secure (for *Session-Key Indistinguishability under Chosen-Ciphertext Attacks*), if for any adversary $\mathcal{A}$, $\mathsf{Adv}^{\mathsf{sk\text{-}ind\text{-}cca}}_{\mathsf{KEM}}(\mathcal{A}) = \mathsf{negl}(\kappa)$, in distinguishing $\mathcal{D}_0$ and $\mathcal{D}_1$, where

$$\mathcal{D}_b = \left\{ \begin{array}{l} (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KEM.KeyGen}(1^\kappa), \\ (C^*, K_0) \leftarrow \mathsf{KEM.Enc}(\mathsf{pk}), K_1 \xleftarrow{\$} \mathcal{K} \end{array} : (\mathsf{pk}, C^*, K_b) \right\}$$

with an unlimited access to a decapsulation oracle $\mathcal{O}\mathsf{Dec}(C)$ that outputs $\mathsf{KEM.Dec}(\mathsf{sk}, C)$, excepted on the challenge ciphertext C^*.

When the adversary is not allowed to ask decapsulation queries, the security notion is denoted SK-IND-CPA (for *Session-Key Indistinguishability under Chosen-Plaintext Attacks*).

Public-Key Privacy. One can additionally expect anonymity of the receiver, also known as *public-key privacy* (denoted PK-IND, for *Public-Key Indistinguishability*), if the ciphertext does not leak any information about the public key, first defined in [BBDP01]. More formally, as above, a KEM is PK-IND-CCA-secure (for *Public-Key Indistinguishability under Chosen-Ciphertext Attacks*), if for any adversary $\mathcal{A}$, $\mathsf{Adv}^{\mathsf{pk\text{-}ind\text{-}cca}}_{\mathsf{KEM}}(\mathcal{A}) = \mathsf{negl}(\kappa)$, in distinguishing $\mathcal{D}_0$ and $\mathcal{D}_1$, where

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

with an unlimited access to a decapsulation oracle $\mathcal{O}\mathsf{Dec}(i, C)$ that outputs $\mathsf{KEM.Dec}(\mathsf{sk}_i, C)$, for $i \in \{0, 1\}$, excepted on the challenge ciphertext $C^* = C_b$.

When the adversary is not allowed to ask decapsulation queries, the security notion is denoted PK-IND-CPA (for *Public-Key Indistinguishability under Chosen-Plaintext Attacks*).

ElGamal-based KEM. In a group $\mathbb{G}$ of prime order p, with a generator P:

- $\mathsf{EG.KeyGen}(1^\kappa)$ samples random $\mathsf{sk} = x \xleftarrow{\$} \mathbb{Z}_p$ and set $\mathsf{pk} = H \leftarrow x \cdot P$;
- $\mathsf{EG.Enc}(\mathsf{pk})$ samples a random $r \xleftarrow{\$} \mathbb{Z}_p$ and set $C \leftarrow r \cdot P$ together with $K \leftarrow r \cdot H$;
- $\mathsf{EG.Dec}(\mathsf{sk}, C)$ outputs $K \leftarrow x \cdot C$.

Remark 4. This is a folklore result that under the DDH assumption in $\mathbb{G}$, this KEM is both SK-IND-CPA and PK-IND-CPA with $\mathcal{K} = \mathbb{G}$. By replacing the key computation by $K \leftarrow \mathcal{H}(r \cdot H)$ during encapsulation, or $K \leftarrow \mathcal{H}(x \cdot C)$ during decapsulation, for a 2κ-bit hash function $\mathcal{H}$, under the CDH assumption in $\mathbb{G}$, this KEM is both SK-IND-CPA and PK-IND-CPA with $\mathcal{K} = \{0, 1\}^{2\kappa}$.

Key Encapsulation Mechanism with Access Control. A KEM with Access Control allows multiple users to access the encapsulated key K from C, according to a rule $\mathcal{R}$ applied on Y in the user's key usk and X in the ciphertext C. It is defined by four algorithms:

- $\mathsf{KEMAC.Setup}(\mathcal{R}, 1^\kappa)$ outputs the global public parameters MPK and the master secret key MSK;
- $\mathsf{KEMAC.KeyGen}(\mathsf{MSK}, Y)$ outputs the user's secret key usk according to Y;
- $\mathsf{KEMAC.Enc}(\mathsf{MPK}, X)$ generates a session key K and a ciphertext C of it according to X;
- $\mathsf{KEMAC.Dec}(\mathsf{usk}, C)$ outputs the key K encapsulated in C.

Correctness. One expects $\mathsf{KEMAC.Dec}(\mathsf{usk}, C) = K$ with overwhelming probability, for any X and Y such that $\mathcal{R}(X, Y) = 1$, where usk has been honestly generated for Y and K has been encapsulated in C for X, after an honest setup to generate $(\mathsf{MPK}, \mathsf{MSK})$.

1. $b \xleftarrow{\$} \{0, 1\}$
2. $(\mathsf{MPK}, \mathsf{MSK}) \leftarrow \mathsf{KEMAC.Setup}(1^\kappa)$
3. $b' \leftarrow \mathcal{A}^{\mathcal{O}\mathsf{KeyGen}(\cdot), \mathcal{O}\mathsf{Dec}(\cdot), \mathcal{O}\mathsf{EncChal}(\cdot)}(\mathsf{MPK})$
4. if there is a Y such that $\mathcal{R}(X^*, Y) = 1$, for X^* asked to $\mathcal{O}\mathsf{EncChal}$, asked either to $\mathcal{O}\mathsf{KeyGen}$ or to $\mathcal{O}\mathsf{Dec}$ on the challenge C^*, then output a random bit $\beta \xleftarrow{\$} \{0, 1\}$, else output $\beta \leftarrow (b' = b)$

1. $b \xleftarrow{\$} \{0, 1\}$
2. $(\mathsf{MPK}, \mathsf{MSK}) \leftarrow \mathsf{KEMAC.Setup}(1^\kappa)$
3. $b' \leftarrow \mathcal{A}^{\mathcal{O}\mathsf{KeyGen}(\cdot),,\mathcal{O}\mathsf{Dec}(\cdot), \mathcal{O}\mathsf{EncChal}(\cdot,\cdot)}(\mathsf{MPK})$
4. if there is a Y such that $\mathcal{R}(X_0, Y) = 1$ or $\mathcal{R}(X_1, Y) = 1$, for (X^0, X^1) asked to $\mathcal{O}\mathsf{EncChal}$, asked either to $\mathcal{O}\mathsf{KeyGen}$ or to $\mathcal{O}\mathsf{Dec}$ on the challenge C^*, then output a random bit $\beta \xleftarrow{\$} \{0, 1\}$, else output $\beta \leftarrow (b' = b)$

Fig. 1. Security Games for SK-IND-CCA and AC-IND-CCA for a KEMAC

Session-Key Privacy. As for the basic KEM, one may expect some privacy properties. Session-key privacy is modeled by indistinguishability of the actual session key from the ciphertext, even if the adversary has received some decryption keys and some decapsulations, as soon as some associated Y are incompatible with the challenge input X^* ($\mathcal{R}(X^*, Y) = 0$). More precisely, such a KEMAC is said to be $\mathsf{SK\text{-}IND\text{-}CCA}$-secure in the key space $\mathcal{K}$ if for any adversary $\mathcal{A}$, that can ask *any* key usk using oracle $\mathcal{O}\mathsf{KeyGen}(Y)$, that outputs $\mathsf{KEMAC.KeyGen}(\mathsf{MSK}, Y)$, *any* decapsulated key K using oracle $\mathcal{O}\mathsf{Dec}(Y, C)$, that outputs $\mathsf{KEMAC.Dec}(\mathsf{usk}, C)$, for an ephemeral key usk obtained from $\mathcal{O}\mathsf{KeyGen}(Y)$, and one challenge on any X^*, using $\mathcal{O}\mathsf{EncChal}(X^*)$, that runs $\mathsf{KEMAC.Enc}(\mathsf{MPK}, X^*)$ to get (C^*, K_0), chooses $K_1 \xleftarrow{\$} \mathcal{K}$, and outputs (C^*, K_b), for an initial random bit b, the advantage $\mathsf{Adv}^{\mathsf{sk\text{-}ind\text{-}cca}}_{\mathsf{KEMAC}}(\mathcal{A})$ is negligible in the game presented on the left of Figure 1. We stress that the bad situation where there is a Y such that $\mathcal{R}(X^*, Y) = 1$, for X^* asked to $\mathcal{O}\mathsf{EncChal}$, asked either to $\mathcal{O}\mathsf{KeyGen}$ or to $\mathcal{O}\mathsf{Dec}$ on the challenge ciphertext C^* should be avoided by the adversary: as this leads to a trivial guess, this is considered as a non-legitimate attack, and it reduces its advantage.

As usual, we can restrict to chosen-plaintext attacks, where the adversary cannot ask for the decapsulation oracle.

Access-Control Privacy. One can also extend the anonymity notion, by hiding the parameter X used in the ciphertext C even if the adversary $\mathcal{A}$ can ask some decryption keys and some decapsulations, as soon as some associated Y are incompatible with the challenge input X^* ($\mathcal{R}(X, Y) = 0$). A KEMAC is said to be $\mathsf{AC\text{-}IND\text{-}CCA}$-secure if for any adversary $\mathcal{A}$, that can ask *any* key usk using oracle $\mathcal{O}\mathsf{KeyGen}(Y)$, that outputs $\mathsf{KEMAC.KeyGen}(\mathsf{MSK}, Y)$, *any* decapsulated key K using oracle $\mathcal{O}\mathsf{Dec}(Y, C)$, that outputs $\mathsf{KEMAC.Dec}(\mathsf{usk}, C)$, for an ephemeral key usk obtained from $\mathcal{O}\mathsf{KeyGen}(Y)$, and one ciphertext on any pair (X_0, X_1), using $\mathcal{O}\mathsf{EncChal}(X_0, X_1)$, that runs $\mathsf{KEMAC.Enc}(\mathsf{MPK}, X_b)$ to get (C^*, K), for an initial random bit b, $\mathsf{Adv}^{\mathsf{ac\text{-}ind\text{-}cca}}_{\mathsf{KEMAC}}(\mathcal{A})$ is negligible in the game presented on the right of Figure 1. As above, we can restrict to chosen-plaintext attacks, where the adversary cannot ask for the decapsulation oracle.

Traceability. In any multi-user setting, to avoid abuse of the decryption keys, one may want to be able to trace users (or their personal keys) from the decryption mechanism, and more generally from any *useful* pirate decoder, either given access to the key material in the device (white-box tracing) or just interacting with the device (black-box tracing) [CFN94]. Without any keys, one expects session-key privacy, but as soon as one knows a key, one can distinguish the session-key. Then, we will call a *useful* pirate decoder $\mathcal{P}$ a good distinguisher against session-key privacy, that behaves differently with the real and a random key. But of course, this pirate decoder can be built from multiple user' keys, called traitors, and one would like to be able to trace at least one traitor from the collusion.

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

- $\mathsf{NIKE.KeyGen}(1^\kappa)$: on input of a security parameter κ, outputs a pair of public and secret keys $(\mathsf{pk}, \mathsf{sk})$;
- $\mathsf{NIKE.SessionKey}(\mathsf{sk}, \mathsf{pk}')$: on input of a secret key sk and a public key pk', generates a session key K.

Correctness. A correct NIKE satisfies

$$\mathsf{Adv}^{\mathsf{cor}}_{\mathsf{NIKE}}(\kappa) = 1 - \Pr_{\mathcal{D}}[\mathsf{NIKE.SessionKey}(\mathsf{sk}_1, \mathsf{pk}_0) = \mathsf{NIKE.SessionKey}(\mathsf{sk}_0, \mathsf{pk}_1)] = \mathsf{negl}(\kappa),$$

for the distribution probability

$$\mathcal{D} = \{(\mathsf{pk}_0, \mathsf{sk}_0), (\mathsf{pk}_1, \mathsf{sk}_1) \leftarrow \mathsf{NIKE.KeyGen}(1^\kappa) : (\mathsf{sk}_0, \mathsf{pk}_0, \mathsf{sk}_1, \mathsf{pk}_1)\}.$$

Session-Key Privacy. We will consider a weak version of session-key privacy, in the one-wayness vein instead of indistinguishability, as it will be enough for our purpose: such a NIKE is said to be SK-OW-secure (for *Session-Key One-Wayness*) in the key space $\mathcal{K}$ if for any adversary $\mathcal{A}$, $\mathsf{Adv}^{\mathsf{sk\text{-}ow}}_{\mathsf{NIKE}}(\mathcal{A}) = \mathsf{negl}(\kappa)$, in extracting the session key K from the public view $(\mathsf{pk}_0, \mathsf{pk}_1)$, according to the following distribution probability:

$$\mathcal{D} = \left\{ \begin{array}{l} (\mathsf{pk}_0, \mathsf{sk}_0), (\mathsf{pk}_1, \mathsf{sk}_1) \leftarrow \mathsf{NIKE.KeyGen}(1^\kappa), \\ K \leftarrow \mathsf{NIKE.SessionKey}(\mathsf{sk}_1, \mathsf{pk}_0) \end{array} : (\mathsf{pk}_0, \mathsf{pk}_1, K) \right\}$$

Then, one can derive a KEM that is both SK-IND-CPA-secure and PK-IND-CPA-secure, in the random oracle model, for a hash function $\mathcal{H}$, from the set of the keys of NIKE into $\mathcal{K} = \{0,1\}^{2\kappa}$:

- $\mathsf{KEM.KeyGen}(1^\kappa)$ runs and outputs $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{NIKE.KeyGen}(1^\kappa)$;
- $\mathsf{KEM.Enc}(\mathsf{pk})$ runs $(\mathsf{pk}', \mathsf{sk}') \leftarrow \mathsf{NIKE.KeyGen}(1^\kappa)$, and gets $K \leftarrow \mathcal{H}(\mathsf{NIKE.SessionKey}(\mathsf{sk}', \mathsf{pk}))$, outputting $(C = \mathsf{pk}', K)$;
- $\mathsf{KEM.Dec}(\mathsf{sk}, C)$ runs $K' \leftarrow \mathsf{NIKE.SessionKey}(\mathsf{sk}, \mathsf{pk}')$, where $\mathsf{pk}' = C$, and outputs $K = \mathcal{H}(K')$.

Key-Homomorphic NIKE (KH-NIKE). A NIKE is said to be key-homomorphic, if the secret keys are in a ring $(\mathcal{R}, +, \times)$ and there is an internal group-law $\otimes$ and an external law $\odot$ on the public keys that make them correspond to each other: from $(\mathsf{pk}_0, \mathsf{sk}_0), (\mathsf{pk}_1, \mathsf{sk}_1) \leftarrow \mathsf{NIKE.KeyGen}(1^\kappa)$, the secret key $\mathsf{sk} \leftarrow \mathsf{sk}_0 + \mathsf{sk}_1$ corresponds to the public key $\mathsf{pk} \leftarrow \mathsf{pk}_0 \otimes \mathsf{pk}_1$, and the secret key $\mathsf{sk}' \leftarrow \mathsf{sk}_0 \times \mathsf{sk}_1$ corresponds to the public key $\mathsf{pk}' \leftarrow \mathsf{sk}_0 \odot \mathsf{pk}_1$. Furthermore, for any $(\mathsf{pk}'', \mathsf{sk}'') \leftarrow \mathsf{NIKE.KeyGen}(1^\kappa)$, we have

$$\mathsf{NIKE.SessionKey}(\mathsf{sk}'', \mathsf{sk}_0 \odot \mathsf{pk}_1) = \mathsf{NIKE.SessionKey}(\mathsf{sk}_0 \times \mathsf{sk}'', \mathsf{pk}_1).$$

A classical KH-NIKE is the Diffie-Hellman NIKE, with secret keys in the particular ring $\mathcal{R} = \mathbb{Z}_p$, and public keys in the group $(\mathbb{G}, P, p)$, where P is a generator of $\mathbb{G}$, of prime order p. The DH algorithms are

- $\mathsf{DH.KeyGen}(1^\kappa)$: on input of a security parameter κ, it outputs a pair of public and secret keys $(\mathsf{pk}, \mathsf{sk})$, where $\mathsf{sk} \leftarrow \mathbb{Z}_p$ and $\mathsf{pk} \leftarrow \mathsf{sk} \cdot P$;
- $\mathsf{DH.SessionKey}(\mathsf{sk}, \mathsf{pk}')$: on input of a secret key sk and a public key pk', it generates a session key $K = \mathsf{sk} \cdot \mathsf{pk}'$.

The SK-OW security relies on the Computational Diffie-Hellman (CDH) problem, and it provides key homomorphism, in $(\mathbb{Z}_p, +, \times)$ for the secret keys, the internal group-law $\otimes$ being the addition in $\mathbb{G}$, and the external law $\odot$ being the scalar multiplication in $\mathbb{G}$.

3 Security Analysis of Covercrypt

In this section, we first present a generalization of Covercrypt, the ETSI TS 104 015 [ETS25], that proposes to use a hybrid construction, combining a pre-quantum key-homomorphic NIKE and a post-quantum KEM, where the KEM is instantiated by the ML-KEM [NIS22] and the NIKE is instantiated by the Hashed-ECDH. This provides hybridization of the security, with the best of both worlds, the post-quantum security of the KEM and the pre-quantum security of the KH-NIKE, at least for the session-key privacy.

3.1 Hybrid Traceable KEMAC with CCA Security

We thus describe a generalization of Covercrypt, that combines any KH-NIKE with any KEM, the former being SK-OW-secure, and the latter being SK-IND-CCA and PK-IND-CCA-secure.

Detailled Description. Using the following notations:

- $\Omega = \{S_1, \ldots, S_N\}$ is the set of rights;
- NIKE is a KH-NIKE scheme achieving $\mathsf{SK\text{-}OW}$ security, with secret keys in a ring $\mathcal{R}$ and public keys in a group $\mathbf{G}$;
- KEM is a KEM scheme achieving $\mathsf{SK\text{-}IND\text{-}CCA}$ and $\mathsf{PK\text{-}IND\text{-}CCA}$ security;
- $\mathcal{G}$, $\mathcal{H}$, and $\mathcal{J}$ are hash functions, mapping elements to $\mathcal{R}$ elements, 2κ-bit strings, and 3κ-bit strings respectively,

the scheme $\mathsf{HTKEMAC}$ is defined as follows:

- $\mathsf{HTKEMAC.Setup}(\Omega, t, 1^\kappa)$: for a set Ω of rights and a threshold t for traceability:
  1. the algorithm samples $(P_1, s_1), \ldots, (P_t, s_t) \leftarrow \mathsf{NIKE.KeyGen}(1^\kappa)$;
  2. the algorithm samples $\alpha_1, \ldots, \alpha_t \xleftarrow{\$} \mathcal{R}$, and sets $s = \sum_k \alpha_k \cdot s_k$ and $H = \otimes_k(\alpha_k \odot P_k)$, with the constraint that s is invertible in $\mathcal{R}$;
  3. the set of user identities $\mathcal{ID}$ is initialized as an empty set, the tracing secret key is then set to $\mathsf{tsk} = (s, (s_k)_k, \mathcal{ID})$ and the tracing public key to $\mathsf{tpk} = (H, (P_k)_k)$;
  4. the set of users' secret keys showing their permissions is initialized as an empty set with $\mathcal{UP} \leftarrow \emptyset$;
  5. for each right S_i of index i in Ω, the algorithm samples $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KEM.KeyGen}(1^\kappa)$, $(X_i, x_i) \leftarrow \mathsf{NIKE.KeyGen}(1^\kappa)$, computes $H_i \leftarrow s \odot X_i$, and sets $\mathsf{pk}'_i \leftarrow (H_i, \mathsf{pk}_i)$ and $\mathsf{sk}'_i \leftarrow (x_i, \mathsf{sk}_i)$;
  6. finally, the global public key is set to $\mathsf{MPK} \leftarrow (\mathsf{tpk}, \{\mathsf{pk}'_i\}_i)$, and the master secret key to $\mathsf{MSK} \leftarrow (\mathsf{tsk}, \{\mathsf{sk}'_i\}_i, \mathcal{UP})$.

  The algorithm returns $(\mathsf{MSK}, \mathsf{MPK})$.
- $\mathsf{HTKEMAC.KeyGen}(\mathsf{MSK}, U, Y)$: on input a username U, along with Y a set of indices corresponding to U's rights Ω, parsing the master secret key $\mathsf{MSK} = (\mathsf{tsk}, \{\mathsf{sk}'_i\}_i, \mathcal{UP})$ as an output of the Setup algorithm:
  1. it draws a random tuple $(\beta_k)_k$ such that $s = \sum_k \beta_k \cdot s_k$, and sets U's secret identifier to $\mathsf{uid} \leftarrow (\beta_k)_k$;
  2. it updates the tracing secret key by setting tsk' to be equal to tsk in which (U, uid) is added to $\mathcal{ID}$;
  3. U's secret key is defined as $\mathsf{usk} \leftarrow (\mathsf{uid}, \{\mathsf{sk}'_j\}_{j \in Y})$, and the master secret key is updated to MSK' equal to MSK in which usk was added to $\mathcal{UP}$.

  Finally, the algorithm outputs $(\mathsf{usk}, \mathsf{MSK}', \mathsf{tsk}')$.
- $\mathsf{HTKEMAC.Enc}(\mathsf{MPK}, X)$: parsing the public key $\mathsf{MPK} = (\mathsf{tpk}, \{\mathsf{pk}'_i\}_i)$ as an output of the Setup algorithm, and X as a set of indices of rights in Ω:
  1. denoting $\mathcal{K}$ the key space of KEM, the encryption algorithm draws $S \xleftarrow{\$} \mathcal{K}$, sets $r \leftarrow \mathcal{G}(S)$ and $(c_k \leftarrow r \odot P_k)_k$, and $c \leftarrow (c_k)_k$
  2. for each index $i \in X$, the algorithm sets $K_i \leftarrow \mathsf{NIKE.SessionKey}(r, H_i)$, $(E_i, K'_i) \leftarrow \mathsf{KEM.Enc}(\mathsf{pk}_i)$, and $E \leftarrow (E_\ell)_{\ell \in X}$;
  3. for each index $i \in X$, the algorithm sets $F_i \leftarrow S \oplus \mathcal{H}(K_i, K'_i, c, E)$, and $F \leftarrow (F_\ell)_{\ell \in X}$;
  4. the algorithm then computes $(K, V) \leftarrow \mathcal{J}(S, c, E, F)$, and sets the ciphertext as $C \leftarrow (c, E, F, V)$, and the encapsulated key to be K.

  The algorithm outputs (K, C).
- $\mathsf{HTKEMAC.Dec}(\mathsf{usk}, C)$: parsing usk as an output of the KeyGen algorithm, and $C = (c = (c_k)_k, E = (E_\ell)_\ell, F = (F_\ell)_\ell, V)$ as an output of the Enc algorithm, for a list of pairs (E_i, F_i), without knowing the corresponding keys. For each index i with such a pair (E_i, F_i) in C, and for each index $j \in Y$ with an element sk_j in usk, the decryption algorithm:
  1. runs $K'_{i,j} \leftarrow \mathsf{KEM.Dec}(\mathsf{sk}_j, E_i)$;
  2. computes $K_j \leftarrow \mathsf{NIKE.SessionKey}(x_j, \otimes_k(\beta_k \odot c_k))$;
  3. computes $S_{i,j} \leftarrow F_i \oplus \mathcal{H}(K_j, K'_{i,j}, c, E)$;
  4. computes both $r' \leftarrow \mathcal{G}(S_{i,j})$ and $(U'_{i,j}, V'_{i,j}) \leftarrow \mathcal{J}(S_{i,j}, c, E, F)$;
  5. checks whether $c = (r' \odot P_k)_k$ and $V'_{i,j} = V$: in the positive case, it 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 $\perp$.

Correctness. First of all, let us check the correctness of the scheme, during decryption of a ciphertext $C^* = (c^* = (c^*_k)_k, E^* = (E^*_\ell)_\ell, F^* = (F^*_\ell)_\ell, V^*)$ with $\mathsf{usk} = ((\beta_k)_k, \{(x_j, \mathsf{sk}_j)\}_{j\in Y})$: if there is an acceptable index $j \in Y$, that corresponds to an index $i \in X^*$ used for ciphertext, one

1. runs $K'_{i,j} \leftarrow \mathsf{KEM.Dec}(\mathsf{sk}_j, E^*_i)$, which corresponds to K'^*_i;
2. computes $K_j \leftarrow \mathsf{NIKE.SessionKey}(x_j, \otimes_k(\beta_k \odot c^*_k))$. As $\otimes_k(\beta_k \odot c^*_k) = \otimes_k((\beta_k \cdot r^*) \odot P_k) = r^* \odot (\otimes_k(\beta_k \odot P_k)) = r^* \odot H$, then $K_j = \mathsf{NIKE.SessionKey}(x_j, r^* \odot H) = \mathsf{NIKE.SessionKey}(r^*, x_j \odot H) = \mathsf{NIKE.SessionKey}(r^*, H_j) = \mathsf{NIKE.SessionKey}(r^*, H_i) = K^*_i$;
3. computes $S_{i,j} \leftarrow F^*_i \oplus \mathcal{H}(K_j, K'_{i,j}, c^*, E^*) = F^*_i \oplus \mathcal{H}(K^*_i, K'^*_i, c^*, E^*) = S^*$;

Then, $r' \leftarrow \mathcal{G}(S_{i,j}) = \mathcal{G}(S^*) = r^*$ and $(U'_{i,j}, V'_{i,j}) \leftarrow \mathcal{J}(S_{i,j}, c^*, E, F) = \mathcal{J}(S^*, c^*, E^*, F^*) = (K^*, V^*)$. As a consequence, both verifications $c^* = (r^* \odot P_k)_k$ and $V'_{i,j} = V^*$ succeed, and so one returns $K \leftarrow U'_{i,j} = K^*$.

3.2 Security Analysis

Let us prove the CCA security for the HTKEMAC scheme, for both session-key privacy and access-control privacy.

Session-Key Privacy. For the session-key privacy, we consider the SK-IND-CCA-game, where the adversary has access to the decryption oracle, excepted on the challenge ciphertext: HTKEMAC is said to be SK-IND-CCA-secure in the key space $\mathcal{K}$ if for any adversary $\mathcal{A}$, that can ask:

- any key usk, for access rights Y under user U, using the oracle $\mathcal{O}\mathsf{KeyGen}(U, Y)$ that outputs $\mathsf{HTKEMAC.KeyGen}(\mathsf{MSK}, U, Y)$,
- any decryption on ciphertext C under access rights Y, using the oracle $\mathcal{O}\mathsf{Dec}(Y, C)$ that first gets usk from $\mathsf{HTKEMAC.KeyGen}(\mathsf{MSK}, U, Y)$, and then outputs $\mathsf{HTKEMAC.Dec}(\mathsf{usk}, C)$,
- and one ciphertext on any access rights X^*, using the oracle $\mathcal{O}\mathsf{EncChal}(X^*)$ that runs $\mathsf{HTKEMAC.Enc}(\mathsf{MPK}, X^*)$ to get (C^*, K^*_0), chooses $K^*_1 \xleftarrow{\$} \mathcal{K}$, and outputs (C^*, K^*_b), for the initial random bit b,

$\mathsf{Adv}^{\mathsf{sk\text{-}ind\text{-}cca}}_{\mathsf{HTKEMAC}}(\mathcal{A})$ is negligible in the security game presented on the left of Figure 1, where the 4-th line is more precisely

4. if there is a Y, either asked to $\mathcal{O}\mathsf{KeyGen}$ or with (Y, C^*) asked to $\mathcal{O}\mathsf{Dec}$, such that $X^* \cap Y \neq \emptyset$, for X^* asked to $\mathcal{O}\mathsf{EncChal}$, then output a random bit $\beta \xleftarrow{\$} \{0, 1\}$, else output $\beta \leftarrow (b' = b)$.

Theorem 5 (Session-Key Privacy against Chosen-Ciphertext Attacks). HTKEMAC *achieves SK-IND-CCA security under either the SK-IND-CCA security of the underlying KEM or the SK-OW of the underlying KH-NIKE, in the random oracle model for $\mathcal{G}$, $\mathcal{H}$, and $\mathcal{J}$.*

Proof. To make the proof of HTKEMAC, we start from the initial security game:

Game $\mathbf{G}_0$: At setup time, it chooses all the secret keys associated to all the rights in Ω (including s and $(s_k)_k$ that will be known all along this proof, as their privacy will only be used for tracing).

- For the $\mathcal{O}\mathsf{EncChal}$ query on $X^* \subseteq \Omega$, we denote $C^* = (c^* = (c^*_k)_k, E^* = (E^*_\ell)_\ell, F^* = (F^*_\ell)_\ell, V^*)$ the challenge ciphertext, honestly generated on X^*, with $S^* \xleftarrow{\$} \mathcal{K}$, $r^* \leftarrow \mathcal{G}(S^*)$, $(K^*_i \leftarrow \mathsf{NIKE.SessionKey}(r^*, H_i))_i$, $((E^*_i, K'^*_i) \leftarrow \mathsf{KEM.Enc}(\mathsf{pk}_i))_i$, and $(F^*_i \leftarrow S^* \oplus \mathcal{H}(K^*_i, K'^*_i, c^*, E^*)_i$. Eventually, K^* will denote the claimed encapsulated key, that is either real or random.
- For the $\mathcal{O}\mathsf{KeyGen}$ queries, one first generates a random tuple $(\beta_k)_k$ such that $\sum_k \beta_k \cdot s_k = s$, and then concatenates the secret keys generated in the setup;

– For a decryption query $\mathcal{O}\mathsf{Dec}(Y, C)$: one first generates a random tuple $(\beta_k)_k$ such that $\sum_k \beta_k \cdot s_k = s$; for each $j \in Y$ and every E_i in the ciphertext, it runs $\mathsf{KEM.Dec}(\mathsf{sk}_j, E_i)$ to get $K'_{i,j}$ or $\perp$, as well as $K_j = \mathsf{NIKE.SessionKey}(x_j, \otimes(\beta_k \odot c_k))$ and computes the candidate $S_{i,j} = F_i \oplus \mathcal{H}(K_j, K'_{i,j}, c, E)$. It then computes both $r' \leftarrow \mathcal{G}(S_{i,j})$ and $U'_{i,j}||V'_{i,j} \leftarrow \mathcal{J}(S_{i,j}, c, E, F)$, and checks whether $c = (r' \odot P_k)_k$ and $V'_{i,j} = V$. In the positive case, it returns $U'_{i,j}$, otherwise it continues on the i, j indices.

In the end, the adversary must guess whether K^* is real or random.

Note that from the constraints on the $\mathcal{O}\mathsf{KeyGen}$-queries and $\mathcal{O}\mathsf{Dec}$-queries, for the former, the Y's are disjoint from X^*, and for the latter, if the ciphertext is exactly the challenge ciphertext C^*, then $Y \cap X^* = \emptyset$ too.

We stress that if for a given ciphertext C, $c = c^*$, then under the assumption public keys of NIKE have a huge entropy (negligible probability of collisions), and the absence of collisions for $\mathcal{G}$, $S = S^*$, to ensure the validity of the test $c^* = c = (r' \odot P_k)_k$ for $r' \leftarrow \mathcal{G}(S)$. Without having asked any of the challenge $\mathcal{H}$ queries, and thus on correct (K^*_i, K'^*_i), S^* is hidden and the final check with $\mathcal{J}(S^*, c^*, E, F)$ would fail. This will be the intuition all along this proof.

Game $\mathbf{G}_1$: In this game, for decryption queries, we stop and continue looping on the i, j indices if $\mathcal{J}(S_{i,j}, c, E, F)$ has not already been asked by the adversary. Indeed, without such query $V'_{i,j} = V$ would likely fail, as it cannot come from the challenge ciphertext. This makes no difference from the previous game. We stress that such $\mathcal{J}$ queries are specific to each ciphertext query.

Game $\mathbf{G}_2$: As a consequence, for a decryption query C, we can enumerate on all the candidates S asked to $\mathcal{J}$, with answers U', V', and $S \leftarrow \mathcal{G}(S)$ such that $c = (r' \odot P_k)_k$ and $V' = V$. For each of these promising candidates S, and for each $j \in Y$, one computes $K_j \leftarrow \mathsf{NIKE.SessionKey}(r', H_j)$, and for $K'_{i,j} \leftarrow \mathsf{KEM.Dec}(\mathsf{sk}_j, E_i)$. Then one eventually checks whether $S = F_i \oplus \mathcal{H}(K_j, K'_{i,j}, c, E)$. When it works for one candidate S and one index $j \in Y$, one returns U' as the decapsulated key.

This makes no difference with the previous game as we were already expecting the $\mathcal{J}$-queries to be asked.

Game $\mathbf{G}_3$: During the challenge ciphertext, we replace $\mathcal{H}(K^*_i, K'^*_i, c^*, E^*)$ by a random value, for all the indices i. This can only be detected by the adversary if it asks for one of theses $\mathcal{H}$-queries. We thus denotes AskH the event that some of these $\mathcal{H}$-queries is asked.

Hence, unless event AskH happens, this game is perfectly indistinguishable from the previous one. We will now show this event is negligible.

Lemma 6. *Denoting the two events following events, during the above game,*
– Ev *the event that some* NIKE *key K^*_i generated during the challenge generation has been queried to the hash function $\mathcal{H}(K^*_i, *, *, *)$ by the adversary;*
– Ev' *the event that some* KEM *key K'^*_i generated during the challenge generation has been queried to the hash function $\mathcal{H}(*, K'^*_i, *, *)$.*

we can state that
– *if* Ev *is non-negligible, one can break the* SK-OW *security of* NIKE*;*
– *if* Ev' *is non-negligible, one can break the* SK-IND-CCA *security of* KEM*.*

We postpone the proof of this lemma to the end of the proof of Theorem 5. But as we clearly have $\mathsf{AskH} \Rightarrow \mathsf{Ev}$ and $\mathsf{AskH} \Rightarrow \mathsf{Ev}'$, if AskH would be non-negligible, then one could break both SK-OW security of NIKE and SK-IND-CCA security of KEM. Hence, AskH is negligible.

Game $\mathbf{G}_4$: Eventually, we can replace all the F^*_i's by random strings in $\{0,1\}^{2\kappa}$, for each index $i \in X^*$, in the challenge ciphertext. As in the previous game, they were all computed from S^* masked by truly independent random values, this makes no difference.

In this game, S^* is not used anymore, and thus, the probability to ask the $\mathcal{J}$-query is negligible: K^* is unpredictable. Hence, the advantage of the adversary in this last game is negligible, which concludes the proof. □

Let us now prove the Lemma 6.

Proof (Proof of Lemma 6). Intuitively, for Ev to happen, the adversary must be able to break the SK-OW security of NIKE, and for Ev' to happen, the adversary must be able to break the SK-IND-CCA security of KEM. We thus split the proof according to these two cases.

Case 1: if Ev *is non-negligible, one can break the SK-OW security of NIKE.* We thus describe the simulation $\mathcal{B}$ of the challenger of the SK-IND-CCA of HTKEMAC in front of the adversary $\mathcal{A}$, using a challenger $\mathcal{C}$ of the SK-OW of NIKE. Eventually, the combination of $\mathcal{B}$ and $\mathcal{A}$ will be an adversary against the SK-OW security of NIKE. It will not exploit the output guess of the adversary $\mathcal{A}$, but the occurrence of event Ev.

From the SK-OW challenger $\mathcal{C}$, the simulator $\mathcal{B}$ receives two public keys (A, R) coming from $(A, a), (R, r) \leftarrow \mathsf{NIKE.KeyGen}(1^\kappa)$. Its goal is to find $K = \mathsf{NIKE.SessionKey}(a, R) = \mathsf{NIKE.SessionKey}(r, A)$.

First, before the setup, the simulator $\mathcal{B}$ guesses for which index $I \in X^* \subseteq \Omega$, the event Ev happens (*i.e.*, K_I^* is queried to $\mathcal{H}$). As the real challenger would do, $\mathcal{B}$ generates all the secret keys s, $(s_k)_k$, and $(\mathsf{pk}_i, \mathsf{sk}_i)_i$, for all the rights in Ω. It also generates all the NIKE keys honestly, as $(X_i, x_i) \leftarrow \mathsf{NIKE.KeyGen}(1^\kappa)$, excepted $X_I \leftarrow A$ that comes from the SK-OW challenger $\mathcal{C}$. During the generation of the challenge ciphertext, $\mathcal{B}$ first runs by itself $(E_i^*, K'^*_i) \leftarrow \mathsf{KEM.Enc}(\mathsf{pk}_i)$. After the choice of S^*, we implicitly set $r^* \leftarrow r/s$, as the output of the hash function $\mathcal{G}(S^*)$. Note that this is the reason why s must be invertible in $\mathcal{R}$. Then, we set $c_k^* \leftarrow (s_k/s) \odot R$, which is the correct value, as $(s_k/s) \odot R = (s_k \cdot r/s) \odot P = r^* \odot (s_k \cdot P) = r^* \odot P_k$. For the K_i^*'s, we do not need to explicitly compute $\mathcal{H}(K_i^*, K'^*_i, c^*, E^*)$, to mask S^* in the F_i's, as is has been replaced by a random value.

We stress that in this case, no information leaks about S^*, and whereas we do not know the answer, we should never be asked for $\mathcal{G}(S^*)$. Similarly, we should never be asked for $\mathcal{J}(S^*, c, E, F)$. Thus the above simulation of decapsulation queries can be used by $\mathcal{B}$, exploiting the knowledge of the sk_i's. This makes $\mathcal{A}$ behaving exactly as in an SK-IND-CCA game, until event Ev happens. And we actually do not care whether the simulation is not perfect after event Ev is raised.

As we assumed Ev to be non-negligible, in the initial game, it is still in this game: when $\mathcal{A}$ stops (or after a pre-determined polynomial-time limit) we can output a random element queried to $\mathcal{H}$. If Ev happened and the guess I was correct, we output the correct value $K = K_I^*$ with non-negligible probability: the success probability is $\Pr[\mathsf{Ev}]/qn$ where n is the size of Ω (for the guess of I) and q the number of $\mathcal{H}$ queries (for the guess of the query). If Ev is non-negligible, we have built an efficient adversary (the combination of our simulator $\mathcal{B}$ and the adversary $\mathcal{A}$) against the SK-OW of NIKE. Indeed, $K_I^* = \mathsf{NIKE.SessionKey}(r^*, H_I) = \mathsf{NIKE.SessionKey}(r/s, s \odot X_I)$, which is equal to $\mathsf{NIKE.SessionKey}(r, X_I) = \mathsf{NIKE.SessionKey}(r, A)$.

Case 2: if Ev' *is non-negligible, one can break the SK-IND-CCA security of KEM.* We thus describe the simulation $\mathcal{B}$ of the challenger of the SK-IND-CCA of HTKEMAC in front of the adversary $\mathcal{A}$, using a challenger $\mathcal{C}$ of the SK-IND-CCA of KEM. Eventually, the combination of $\mathcal{B}$ and $\mathcal{A}$ will be an adversary against the SK-IND-CCA security of KEM. Again, it will not exploit the output guess of the adversary $\mathcal{A}$, but the occurrence of event Ev'.

First, the simulator $\mathcal{B}$ guesses for which index $I \in X^* \subseteq \Omega$, the event Ev' happens (K'^*_I is queried to $\mathcal{H}$). As the real challenger would do, $\mathcal{B}$ generates all the secret keys s, $(s_k)_k$, and $(X_i, x_i)_i$, for all the rights in Ω.

It also generates all the KEM keys honestly, as $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KEM.KeyGen}(1^\kappa)$, excepted pk_I that comes from the SK-IND-CCA challenger $\mathcal{C}$. During the generation of the challenge ciphertext, $\mathcal{B}$ first runs by itself $(E_i^*, K'^*_i) \leftarrow \mathsf{KEM.Enc}(\mathsf{pk}_i)$, excepted for $(E_I^* = E, K)$ that comes from the SK-IND-CCA challenger $\mathcal{C}$, if $I \in X^*$, with K that is either real or random. After the choice of S^*, $\mathcal{B}$ can generate r^* and $(c_k^*)_k$. But eventually, we can choose random values instead of $\mathcal{H}(K_i^*, K'^*_i, c^*, E^*)$, to mask S^* in the F_i's, as in the previous case.

This simulation of the challenge ciphertext by $\mathcal{B}$ makes no difference to the adversary unless it asks for some $\mathcal{H}(K_i^*, K'^*_i, c^*, E^*)$, for the real value K'^*_i. Then, the above simulation of decapsulation queries can be used by $\mathcal{B}$, exploiting the knowledge of the sk_i's, and K as the decapsulation of E_I^* under unknown sk_I. Indeed,

- if E_I^* is not involved, the simulation is easy, as in the previous game, possibly asking the decapsulation oracle under sk_I^* from the SK-IND-CCA security game against KEM.
- If E_I^* is involved, one uses K as the decapsulation of E_I^*.

In the real case, the simulation is perfect, as we decrypt E_I^* into the correct session key, until event Ev' happens. And we actually do not care whether the simulation is not perfect after event Ev' is raised. In the random case, the probability to ask K is negligible, as K is random and with no leakage. One may only reject a valid ciphertext if $\mathcal{H}(K_I, K'^*_I, c, E)$ has been queried, with correct K'^*_I. When the game ends (or after a pre-determined polynomial-time limit), the simulator $\mathcal{B}$ stops and outputs its answer (its guess whether K is real or random) against the SK-IND-CCA challenge of KEM as follows:

- if K has been queried to $\mathcal{H}$, then the simulator outputs 1 (meaning this is the real key)
- otherwise, it outputs a random bit.

If we are in an execution with event Ev' for $I \in X^*$, the simulator outputs 1 in the real case, but a random bit in the random case. In any situation, in the random case, the probability for K to have been queried is negligible, as it is random with no leakage, and so a random bit will be returned. So the advantage of our distinguisher (the combination of $\mathcal{B}$ and $\mathcal{A}$) is at least $\Pr[\mathsf{Ev}']/n$, where n is the size of Ω. If Ev' is non-negligible, we have built a successful distinguisher against the SK-IND-CCA security of KEM.

Access-Control Privacy. In addition, we want to hide the set X^* used in the ciphertext C^*. More precisely, HTKEMAC is said to be AC-IND-CCA-secure if for any adversary $\mathcal{A}$, that can ask any key usk, for access rights Y, under user U, using oracle $\mathcal{O}\mathsf{KeyGen}(U, Y)$, and any decryption on ciphertext C under access rights Y, using oracle $\mathcal{O}\mathsf{Dec}(Y, C)$, as above, and one ciphertext on (X_0^*, X_1^*), using $\mathcal{O}\mathsf{EncChal}(X_0^*, X_1^*)$, that runs $\mathsf{HTKEMAC.Enc}(\mathsf{MPK}, X_b^*)$ to get (C^*, K^*), for the initial random bit b, $\mathsf{Adv}^{\mathsf{ac\text{-}ind\text{-}cca}}_{\mathsf{HTKEMAC}}(\mathcal{A})$ is negligible in the security game presented on the right of Figure 1, where the 4-th line is more precisely

4. if $|X_0^*| \neq |X_1^*|$ or there is an Y, either asked to $\mathcal{O}\mathsf{KeyGen}$ or with (Y, C^*) asked to $\mathcal{O}\mathsf{Dec}$, such that $X_0^* \cap Y \neq \emptyset$ or $X_1^* \cap Y \neq \emptyset$, for the pair (X_0^*, X_1^*) asked to $\mathcal{O}\mathsf{EncChal}$ then output a random bit $\beta \xleftarrow{\$} \{0, 1\}$, else output $\beta \leftarrow (b' = b)$.

We stress that the last step excludes trivial attacks, where the adversary would be able to check the challenge session key that helps to break privacy. As our ciphertexts are linear in X, the sets X_0^* and X_1^* must be of same sizes to expect privacy.

Theorem 7 (Access-Control Privacy against Chosen-Ciphertext Attacks). HTKEMAC *achieves AC-IND-CCA-security under the PK-IND-CCA security of KEM, the SK-IND-CCA security of KEM, and the SK-OW security of NIKE, in the random oracle model for $\mathcal{G}$, $\mathcal{H}$, and $\mathcal{J}$.*

Proof. To make the proof of HTKEMAC, we start from the initial security game:

Game $\mathbf{G}_0$: At setup time, the simulator of the challenger will generate all the secret keys, including s and $(s_k)_k$ that will be known all along this proof, as their privacy will only be used for tracing. Then, we denote $C^* = (c^* = (c_k^*)_k, E^* = (E_\ell^*)_\ell, F^* = (F_\ell^*)_\ell, V^*)$ the challenge ciphertext, and K^* the encapsulated key, for the random bit b.
For the $\mathcal{O}\mathsf{KeyGen}$ and $\mathcal{O}\mathsf{Dec}$ queries, one uses the secret keys. In the end, the adversary must guess whether X_0^* or X_1^* has been used to generate C^* (the bit b).

Game $\mathbf{G}_1$: As in the previous proof, under the SK-IND-CCA of KEM and/or SK-OW of NIKE, Ev or Ev′ is negligible, and so we can replace F_i^*'s by random values. This makes no difference to the adversary: S^* is perfectly hidden.

Game $\mathbf{G}_2$: As S^* is hidden, any E_i^* involved during a decapsulation query can be safely skipped: without previous call to $\mathcal{H}(K_i, K'^*_i, c, E)$, the computed S will be random and then rejected by the $\mathcal{J}$-query. So during the simulation of the decapsulation queries, we skip the cases that involve an E_i^*. So only new ciphertexts can be queried under the keys from X_0^* or X_1^*.

Game $\mathbf{G}_3$: As $|X_0^*| = |X_1^*|$, we can define a bijection B from X_0^* to X_1^*, where the common elements match with themselves, and different elements match with new elements.

Now, we can make an hybrid sequence, where we change the public keys from X_b^* by the keys from X_0^*: in the k-th step of the sequence, we change E_k^* (involving pk_k from X_0^* or pk'_k from X_1^*, where we have previously guessed, during the setup, the two rights that will correspond to pk_k and pk'_k, so that they are provided by the PK-IND-CCA-challenger), which is indistinguishable under the PK-IND-CCA security of KEM. Note that the difference between X_k from X_0^* and X'_k from X_1^* only impacts K_k^* which is never queried (as Ev is negligible, under SK-OW of NIKE). Again, we can always use the decryption oracle under pk_k from X_0^* or pk'_k from X_1^*, as E_k^* will never be queried during the simulation.

At the end of the sequence, all the public keys are from X_0^*.

In this last game, the challenge ciphertext does not depend on b anymore: the adversary has zero-advantage. This concludes the proof. □

We can stress that the security analysis still holds even if we compute $F_i \leftarrow S \oplus \mathcal{H}(K_i, K'_i)$, as this still raises event AskH, in the above simulations. In the next section, we detail both implementations, with and without the ciphertext in the hash function $\mathcal{H}$.

Traceability. [BdPP23] already provided a traceable analysis in a black-box way, with only confirmation of traitors, following [BF99], when $\mathcal{R}$ is a field $\mathbb{F}$, and then $\mathbb{F}^t$ is a vector space of dimension t over $\mathbb{F}$.

Actually, as explained in [BF99], from n keys $\vec{\beta}_i = (\beta_{i,k})_k \in \mathbb{F}^t$, that satisfy $s = \vec{\beta}_i \cdot \vec{s}$, any new key that must satisfy the same relation can only be a convex combination, unless one can break the hardness of finding secret keys from public keys. And in the particular case of NIKE, this would break the SK-OW security.

Following [BF99] (and [BdPP23]), any collusion of less than t traitors can be confirmed in a black-box way: if we have a set of possible traitors in mind, we can confirm, whether this guess is correct or not, just by interacting with the pirate decoder.

Using multiples of codewords, from a specific linear space tracing code [BF99] for the secret keys $(\beta_{i,k})_k \in \mathbb{F}^t$, and the Berlekamp algorithm for decoding, any collusion of less than $t/2$ traitors can be efficiently traced in a white-box way: from the key $(\gamma_k)_k \in \mathbb{F}^t$ used by the pirate decoder, we can find the convex combination used to build it, and then know the keys used.

3.3 Covercrypt: an Efficient HTKEMAC with Hybrid CCA Security

We thus propose an efficient instantiation with ECDH on any curve $\mathbb{G}$ of prime order p, spanned by a generator P, for the KH-NIKE, whose SK-OW security relies on the CDH, and any KEM that is both SK-IND-CCA and PK-IND-CCA-secure. In this case, the ring $\mathcal{R}$ is the finite field $\mathbb{F}_p$.

The first version (on Figure 2) follows the ETSI standard [ETS25], with the full input to the hash function $\mathcal{H}$, while the second version (on Figure 3) is optimized with a smaller input to the hash function $\mathcal{H}$. In both cases, we use three hash functions $\mathcal{G}$, $\mathcal{H}$, and $\mathcal{J}$, mapping elements to $\mathbb{Z}_p^*$, 2κ-bit strings, and 3κ-bit strings respectively.

$\mathsf{Setup}(\Omega, t, 1^\kappa)$, for a set Ω of rights and a threshold t for traceability

1. sample $s_k \xleftarrow{\$} \mathbb{Z}_p$ and sets $P_k \leftarrow s_k \cdot P$, for $k \in \{1, \ldots, t\}$
2. sample $\alpha_1, \ldots, \alpha_t \xleftarrow{\$} \mathbb{Z}_p$, and set $s = \sum_k \alpha_k \cdot s_k$ and $H = \sum_k \alpha_k P_k$, with the constraint that $s \neq 0$
3. the set of user identities $\mathcal{ID}$ is initialized as an empty set
4. the tracing secret key is set to $\mathsf{tsk} = (s, (s_k)_k, \mathcal{ID})$
5. the tracing public key is initialized to $\mathsf{tpk} = (H, (P_k)_k)$
6. the set of users' secret keys showing their permissions is initialized as an empty set $\mathcal{UP} \leftarrow \emptyset$
7. for each right S_i of index i in Ω, sample $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KEM.KeyGen}(1^\kappa)$, $x_i \xleftarrow{\$} \mathbb{Z}_p$, compute $X_i \leftarrow x_i \cdot P$, $H_i \leftarrow s \cdot X_i$, and set $\mathsf{pk}'_i \leftarrow (H_i, \mathsf{pk}_i)$ and $\mathsf{sk}'_i \leftarrow (x_i, \mathsf{sk}_i)$

The global public key is $\mathsf{MPK} \leftarrow (\mathsf{tpk}, \{\mathsf{pk}'_i\}_i)$, and the master secret key is $\mathsf{MSK} \leftarrow (\mathsf{tsk}, \{\mathsf{sk}'_i\}_i, \mathcal{UP})$

$\mathsf{KeyGen}(\mathsf{MSK}, U, Y)$, on input a username U, the set Y of U's rights, and the master secret key MSK

1. parse the master secret key $\mathsf{MSK} = (\mathsf{tsk}, \{\mathsf{sk}'_i\}_i, \mathcal{UP})$
2. draw a random tuple $(\beta_k)_k$ such that $s = \sum_k \beta_k \cdot s_k$, and sets U's secret identifier to $\mathsf{uid} \leftarrow (\beta_k)_k$
3. define U's secret key as $\mathsf{usk} \leftarrow (\mathsf{uid}, \{\mathsf{sk}'_j\}_{j \in Y})$
4. update the tracing secret key tsk by appending (U, uid) to $\mathcal{ID}$
5. update the master secret key MSK by appending usk to $\mathcal{UP}$

$\mathsf{Enc}(\mathsf{MPK}, X)$, on the set X of rights, and the public key MPK

1. parse the public key $\mathsf{MPK} = (\mathsf{tpk}, \{\mathsf{pk}'_i\}_i)$
2. denoting $\mathcal{K}$ the key space of KEM, draws $S \xleftarrow{\$} \mathcal{K}$
3. set $r \leftarrow \mathcal{G}(S)$, $(c_k \leftarrow r \cdot P_k)_k$, and $c \leftarrow (c_k)_k$
4. for each index $i \in X$, set $K_i \leftarrow r \cdot H_i$, $(E_i, K'_i) \leftarrow \mathsf{KEM.Enc}(\mathsf{pk}_i)$
5. compute $E \leftarrow (E_\ell)_{\ell \in X}$, $T \leftarrow \mathcal{H}(c, E)$
6. for each index $i \in X$, set $F_i \leftarrow S \oplus \mathcal{H}(K_i, K'_i, T)$
7. compute $F \leftarrow (F_\ell)_{\ell \in X}$, $L \leftarrow \mathcal{H}(T, F)$
8. set $(K, V) \leftarrow \mathcal{J}(S, L)$

The ciphertext is $C \leftarrow (c, E, F, V)$ for the key K

$\mathsf{Dec}(\mathsf{usk}, C)$: on a user's secret key usk and $(C = (c = (c_k)_k, E = (E_\ell)_\ell, F = (F_\ell)_\ell, V)$

Parse $\mathsf{usk} \leftarrow (\mathsf{uid}, \{\mathsf{sk}'_j\}_j)$, compute $E \leftarrow (E_\ell)_{\ell \in X}$, $T \leftarrow \mathcal{H}(c, E)$, $L \leftarrow \mathcal{H}(T, F)$, and for each index i with a pair (E_i, F_i) in C, and for each index j with an element $\mathsf{sk}'_j = (x_j, \mathsf{sk}_j)$ in usk

1. run $K'_{i,j} \leftarrow \mathsf{KEM.Dec}(\mathsf{sk}_j, E_i)$;
2. compute $K_j \leftarrow x_j \cdot (\sum_k \beta_k \cdot c_k)$ and $S_{i,j} \leftarrow F_i \oplus \mathcal{H}(K_j, K'_{i,j}, T)$
3. compute both $r' \leftarrow \mathcal{G}(S_{i,j})$ and $(U'_{i,j}, V'_{i,j}) \leftarrow \mathcal{J}(S_{i,j}, L)$
4. check whether $c = (r' \odot P_k)_k$ and $V'_{i,j} = V$
5. if both checks are accepted, return $K \leftarrow U'_{i,j}$ and stop, else continues with the next pair (i, j)

If for all indices i and j, no key was returned, return $\perp$.

Fig. 2. The Standardized Covercrypt Scheme.

$\mathsf{Setup}(\Omega, t, 1^\kappa)$, for a set Ω of rights and a threshold t for traceability

1. sample $s_k \xleftarrow{\$} \mathbb{Z}_p$ and sets $P_k \leftarrow s_k \cdot P$, for $k \in \{1, \ldots, t\}$
2. sample $\alpha_1, \ldots, \alpha_t \xleftarrow{\$} \mathbb{Z}_p$, and set $s = \sum_k \alpha_k \cdot s_k$ and $H = \sum_k \alpha_k P_k$, with the constraint that $s \neq 0$
3. the set of user identities $\mathcal{ID}$ is initialized as an empty set
4. the tracing secret key is set to $\mathsf{tsk} = (s, (s_k)_k, \mathcal{ID})$
5. the tracing public key is initialized to $\mathsf{tpk} = (H, (P_k)_k)$
6. the set of users' secret keys showing their permissions is initialized as an empty set $\mathcal{UP} \leftarrow \emptyset$
7. for each right S_i of index i in Ω, sample $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KEM.KeyGen}(1^\kappa)$, $x_i \xleftarrow{\$} \mathbb{Z}_p$, compute $X_i \leftarrow x_i \cdot P$, $H_i \leftarrow s \cdot X_i$, and set $\mathsf{pk}'_i \leftarrow (H_i, \mathsf{pk}_i)$ and $\mathsf{sk}'_i \leftarrow (x_i, \mathsf{sk}_i)$

The global public key is $\mathsf{MPK} \leftarrow (\mathsf{tpk}, \{\mathsf{pk}'_i\}_i)$, and the master secret key is $\mathsf{MSK} \leftarrow (\mathsf{tsk}, \{\mathsf{sk}'_i\}_i, \mathcal{UP})$

$\mathsf{KeyGen}(\mathsf{MSK}, U, Y)$, on input a username U, the set Y of U's rights, and the master secret key MSK

1. parse the master secret key $\mathsf{MSK} = (\mathsf{tsk}, \{\mathsf{sk}'_i\}_i, \mathcal{UP})$
2. draw a random tuple $(\beta_k)_k$ such that $s = \sum_k \beta_k \cdot s_k$ and $H = \sum_k \beta_k \cdot P_k$, and sets U's secret identifier to $\mathsf{uid} \leftarrow (\beta_k)_k$
3. define U's secret key as $\mathsf{usk} \leftarrow (\mathsf{uid}, \{\mathsf{sk}'_j\}_{j \in Y})$
4. update the tracing secret key tsk by appending (U, uid) to $\mathcal{ID}$
5. update the master secret key MSK by appending usk to $\mathcal{UP}$

$\mathsf{Enc}(\mathsf{MPK}, X)$, on the set X of rights, and the public key MPK

1. parse the public key $\mathsf{MPK} = (\mathsf{tpk}, \{\mathsf{pk}'_i\}_i)$
2. denoting $\mathcal{K}$ the key space of KEM, draws $S \xleftarrow{\$} \mathcal{K}$
3. set $r \leftarrow \mathcal{G}(S)$, $(c_k \leftarrow r \cdot P_k)_k$, and $c \leftarrow (c_k)_k$
4. for each index $i \in X$, set $K_i \leftarrow r \cdot H_i$, $(E_i, K'_i) \leftarrow \mathsf{KEM.Enc}(\mathsf{pk}_i)$ and $F_i \leftarrow S \oplus \mathcal{H}(K_i, K'_i)$
5. compute $E \leftarrow (E_\ell)_{\ell \in X}$ $F \leftarrow (F_\ell)_{\ell \in X}$, and $L \leftarrow \mathcal{H}(c, E, F)$
6. set $(K, V) \leftarrow \mathcal{J}(S, L)$

The ciphertext is $C \leftarrow (c, E, F, V)$ for the key K

$\mathsf{Dec}(\mathsf{usk}, C)$: on a user's secret key usk and $(C = (c = (c_k)_k, E = (E_\ell)_\ell, F = (F_\ell)_\ell, V)$

Parse $\mathsf{usk} \leftarrow (\mathsf{uid}, \{\mathsf{sk}'_j\}_j)$, compute $E \leftarrow (E_\ell)_{\ell \in X}$ $F \leftarrow (F_\ell)_{\ell \in X}$, and $L \leftarrow \mathcal{H}(c, E, F)$, and for each index i with a pair (E_i, F_i) in C, and for each index j with an element $\mathsf{sk}'_j = (x_j, \mathsf{sk}_j)$ in usk

1. run $K'_{i,j} \leftarrow \mathsf{KEM.Dec}(\mathsf{sk}_j, E_i)$;
2. compute $K_j \leftarrow x_j \cdot (\sum_k \beta_k \cdot c_k)$ and $S_{i,j} \leftarrow F_i \oplus \mathcal{H}(K_j, K'_{i,j})$
3. compute both $r' \leftarrow \mathcal{G}(S_{i,j})$ and $(U'_{i,j}, V'_{i,j}) \leftarrow \mathcal{J}(S_{i,j}, L)$
4. check whether $c = (r' \odot P_k)_k$ and $V'_{i,j} = V$
5. if both checks are accepted, return $K \leftarrow U'_{i,j}$ and stop, else continues with the next pair (i, j)

If for all indices i and j, no key was returned, return $\perp$.

Fig. 3. The Optimized Covercrypt Scheme.

4 Access Structure

4.1 High-Level Description

A Ciphertext-Policy Attribute-Based Encryption (CP-ABE) should handle more general policies, whereas the previous sections focused on a specific case of ABE, where both keys and ciphertexts are associated to sets of rights, and decapsulation is possible if and only if the intersection of the two sets is not empty. The ETSI standard describes a way to derive such sets from more advances policies.

Let us first start with an example to illustrate the global approach, with an access structure described by three families, later called **dimensions**:

- `CTR={EN,FR}`, to deal with the countries England and France;
- `DPT={DEV,MKG}`, for the Development and Marketing departments;
- `SEC=(LOW,MED,HIG)`, for a hierarchy of security levels.

This defines the following **qualified attributes** along the 3 dimensions:

- along `CTR`, we have `CTR::EN` and `CTR::FR`
- along `DPT`, we have `DPT::DEV` and `DPT::MKG`
- along `SEC`, we have `SEC::LOW`, `SEC::MED`, and `SEC::HIG`

The two first dimensions `CTR` and `DPT` are defined by unordered sets (using `{...}`), whereas the last security level `SEC` is defined by an ordered set (using `(...)`), meaning that a user with the `SEC::HIG` attribute also possesses the `SEC::LOW` and `SEC::MED` qualified attributes, as `SEC::HIG` $\Rightarrow$ `SEC::MED` $\Rightarrow$ `SEC::LOW`, or equivalently `SEC::LOW` $\leq$ `SEC::MED` $\leq$ `SEC::HIG`, whereas attributes within the dimensions `CTR` and `DPT` are incomparable.

For backward compatibility reason, in each dimension, we introduce the *empty* attribute, leading to the qualified attributes `CTR::`, `DPT::` and `SEC::`. This will indeed allow to dynamically add dimensions in the future, without having to re-encrypt all the data: new users' keys will remain compatible with the existing ciphertexts, as they will implicitly associate the empty attribute to the new dimensions. In addition, the empty attribute is smaller than any attribute in the same dimension: `SEC::` $\Leftarrow$ `SEC::LOW`, and `CTR::` $\Leftarrow$ `CTR::EN` as well as `DPT::` $\Leftarrow$ `DPT::MKG`.

A **right** is a combination of attributes. Such a right is **valid** when represented as a conjunction 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 will be enough and necessary to define expected monotonous access policies in the specific context. In the general case, Ω contains all the possible combinations: this includes fully defined rights, such as `CTR::FR && DPT::MKG && SEC::MED` in the 3-dimension space, but also partially defined rights, such as `CTR::FR` which is equivalent to `CTR::FR && DPT:: && SEC::` or `SEC::HIG`, equivalent to `CTR:: && DPT:: && SEC::HIG`.

These expansions will be used to optimize the ciphertext size: we expect a ciphertext for the right `CTR::FR` to be decapsulated by users with various rights, such as `CTR::FR && DPT::DEV`, `CTR::FR && DPT::MKG && SEC::MED`, and more. The sets X and Y will have to be carefully derived to allow decapsulation of a ciphertext under X by a user key under Y if and only if $X \cap Y \neq \emptyset$.

On the one hand, when one expresses the access policy for a given ciphertext, by any monotonous Boolean formula F, it can be converted into its Disjunctive Normal Form (DNF), that is a disjunction of (conjunctive) clauses. Such conjunctive clauses are exactly the above valid rights. The ciphertext 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, and the initial universe Ω will contain all the meaningful rights. We will define Ω to optimize the size of the ciphertext. In particular, all the possible combinations is a good choice in the most natural cases, where the number of clauses should be not so large.

The ordering along each dimension can be extended to a partial ordering between the rights, when the order is the same along each dimension. For example:

$$\texttt{CTR::FR \&\& SEC::MED} \Leftarrow \texttt{CTR::FR \&\& DPT::MKG \&\& SEC::HIG}$$

as `SEC::MED` $\Leftarrow$ `SEC::HIG` and the absence of dimension `DPT` in the former, can be seen as `DPT::` $\Leftarrow$ `DPT::MKG`. However, `CTR::FR && SEC::HIG` and `CTR::FR && DPT::MKG && SEC::MED` are incomparable, as again `SEC::HIG` $\Rightarrow$ `SEC::MED` but `DPT::` $\Leftarrow$ `DPT::MKG`.

Rule 1, from F to X: Once the Boolean formula F associated to a ciphertext has been converted into a list of clauses (DNF), one first removes the smaller clauses, and only keeps the remaining greater clauses for X.

If some clause `CTR::FR && SEC::MED` (completed into `CTR::FR && DPT:: && SEC::MED`, because of the empty dimension `DPT::`) is among those remaining clauses in X, for encrypting the ciphertext, Alice should be able to decapsulate it if she owns any right that is equal or greater than this right, which means:

`CTR::FR && DPT::MKG && SEC::MED`, `CTR::FR && DPT::DEV && SEC::MED`,
`CTR::FR && DPT::MKG && SEC::HIG`, `CTR::FR && DPT::DEV && SEC::HIG`.

Hence, on the other hand, when generating keys for Alice, if she is explicitly given any of these 4 above rights, she should also implicitly own the right `CTR::FR && DPT:: && SEC::MED` to be able to decapsulate the above ciphertext without having to include more rights than the minimal ones in the ciphertext.

Rule 2, rights in Y (first hint): A user with explicit right R should also receive all the rights R' that are smaller than R in the partial order.

For example, a user with explicit right `CTR::FR && DPT::DEV && SEC::HIG` should receive keys for

- `CTR::FR && DPT::DEV && SEC::HIG`
- `CTR::FR && DPT::DEV && SEC::MED`
- `CTR::FR && DPT::DEV && SEC::LOW`
- `CTR::FR && DPT::DEV && SEC::`
- `CTR::   && DPT::DEV && SEC::HIG`
- `CTR::   && DPT::DEV && SEC::MED`
- `CTR::   && DPT::DEV && SEC::LOW`
- `CTR::   && DPT::DEV && SEC::`
- `CTR::FR && DPT::    && SEC::HIG`
- `CTR::FR && DPT::    && SEC::MED`
- `CTR::FR && DPT::    && SEC::LOW`
- `CTR::FR && DPT::    && SEC::`
- `CTR::   && DPT::    && SEC::HIG`
- `CTR::   && DPT::    && SEC::MED`
- `CTR::   && DPT::    && SEC::LOW`
- `CTR::   && DPT::    && SEC::`

Furthermore, a missing dimension in such an explicit user's right R means "no restriction along this dimension". Then theses missing dimensions can be completed by any value: if Alice is given explicit right `CTR::FR && SEC::MED`, with no `DPT` restriction, she should be able to decapsulate any ciphertext with lower security level, or no security level (as `SEC::` $\Leftarrow$ `SEC::LOW` $\Leftarrow$ `SEC::MED`) and `CTR::FR`, whatever `DPT` is.

Rule 2, rights in Y: A user with explicit right R should also receive all the rights R' that are smaller than R, and these rights $R' \Leftarrow R$ should also be completed by any value in the empty dimensions of R.

In our above example, a user with explicit right `CTR::FR && SEC::MED` should receive keys for (`DPT` can be filled with any value): all the rights $R' \Leftarrow R =$ `CTR::FR && SEC::MED`

- `CTR::FR && DPT::    && SEC::MED`
- `CTR::FR && DPT::    && SEC::LOW`
- `CTR::FR && DPT::    && SEC::`
- `CTR::   && DPT::    && SEC::MED`
- `CTR::   && DPT::    && SEC::LOW`
- `CTR::   && DPT::    && SEC::`
- `CTR::   && DPT::    && SEC::`

and all these rights R' where `DPT` is completed by any value

- `CTR::FR && DPT::MKG && SEC::MED`
- `CTR::FR && DPT::MKG && SEC::LOW`
- `CTR::FR && DPT::MKG && SEC::`
- `CTR::   && DPT::MKG && SEC::MED`
- `CTR::   && DPT::MKG && SEC::LOW`
- `CTR::   && DPT::MKG && SEC::`
- `CTR::   && DPT::MKG && SEC::`
- `CTR::FR && DPT::DEV && SEC::MED`
- `CTR::FR && DPT::DEV && SEC::LOW`
- `CTR::FR && DPT::DEV && SEC::`
- `CTR::   && DPT::DEV && SEC::MED`
- `CTR::   && DPT::DEV && SEC::LOW`
- `CTR::   && DPT::DEV && SEC::`
- `CTR::   && DPT::DEV && SEC::`

This is the main idea behind the optimization of the ciphertext size, and the definition of the user-key rights, that will be formalized in the following part.

4.2 Formal Description

We can now provide some terminology: **attribute** will be any name, along a **dimension** for each category. An attribute with a specific dimension will be a **qualified attribute**, and a conjunction of such qualified attributes along different dimensions with define a **right**. We will then denote Ω as the **universe** of the rights, with all the meaningful rights, and an **access policy** will be a subset of rights.

Access Policies. Covercrypt is a subset-cover algorithm: both ciphertexts and user secret keys are associated to a set of **rights**. A user's secret key associated to a set of rights Y can then open a ciphertext associated to a set of rights X if and only if $X \cap Y \neq \emptyset$. In this document, we call **access policy** the set of rights associated to either a user secret key or a ciphertext, and **access structure** the global structure defined at setup and against which access policies are validated: an access policy is said to be valid w.r.t. an access structure if the rights it defines belong to this access structure.

Our goal is now to present a way to efficiently express these sets of rights and to produce users' secret keys and compact ciphertexts, in the ciphertext-policy vein, which means that some attributes are provided to the users, while a Boolean formula is associated to the ciphertext. We first introduce the notion of **attribute** and of **dimension** that can be either:

- a **hierarchy**, which defines a set of strictly growing attributes:

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

A hierarchy H therefore defines a total order relation $<_H$ (or $\Leftarrow$) on its attributes.

– an **anarchy**, which defines a set of independent attributes:

$$\forall (A, B) \in D^2, (A = B) \text{ or } \neg (A \mathbin{\&\&} B).$$

Attributes from different dimensions are said to be **orthogonal** and attributes from the same anarchical dimension are said to be **mutually exclusive** while attributes from the same hierarchical dimension are said to be **mutually inclusive**.

An access structure is therefore described by the following grammar:

```
<access-structure> = [<dimension>]
<dimension>        = <anarchy> | <hierarchy>
<anarchy>          = <name> '{' [attribute]+ '}'
<hierarchy>        = <name> '(' [attribute]+ ')'
<attribute>        = <name>
<name>             = [^ '&&' '||' '::']
```

where the attributes of hierarchical dimensions are given in increasing order, and the notation `[^ '&&' '||' '::']` stands for any non-empty combination of characters in which neither `&&`, `||`, nor `::` occur, to avoid any ambiguity. An access policy is described by:

```
<access-policy>       = <broadcast>
                        | <qualified-attribute>
                        | <access-policy> <op> <access-policy>
<qualified-attribute> = <dimension>::<attribute>
<dimension>           = <name>
<attribute>           = <name>
<broadcast>           = '*'
<op>                  = '&&' | '||'
```

Finally, we define one right per intersection of orthogonal attributes, such that the set of expressible subset of rights is exactly the set of such intersections. That way, deriving the set of rights expressed by an access policy is equivalent to expressing this access policy in its Disjunctive Normal Form (DNF) as described in the previous section. In order to guard against trivial reductions of an access policies, we add a rule to forbid parsing an empty access policy. For example, given a pair (A, B) of exclusive attributes, their intersection is empty which is forbidden. Parsing `A && B` should therefore return an error. The special `<broadcast>` syntax is associated to a right given to each user secret key, thus allowing for efficient broadcast encapsulations.

Deriving Y from an Access Policy. As we are in the ciphertext-policy setting, the access policy of a user is a set of rights, that can simply be seen as a disjunction of clauses. Each clause is expanded (as illustrated above): in every non-empty dimension of the clause, the attribute A is replaced by the disjunction of A with all the $B \Leftarrow A$ (including the empty one); and every empty dimension is filled with a disjunction of all the possible attributes (including the empty one). Hence, the initial conjunctive clause is expanded into a conjunction of disjunctions along all the dimensions. This CNF is then developed into its DNF: with disjunctions of conjunctive clauses that only involve orthogonal attributes along all the dimension, *i.e.* a disjunction of rights, in which the empty attributes can then be removed. The right with only empty attributes is the broadcast right "$*$". The resulting set of rights is then the set Y of rights associated to the user.

Deriving X from an Access Policy. In the ciphertext-policy setting, the ciphertext is associated to a Boolean formula on attributes: the access policy is any monotonous Boolean formula, that can first be converted into its DNF: a disjunction of clauses. Its smallest version

can be found by removing the clauses that are bigger than any other clause: clauses are compared to each-other (which is quadratic in the number of clauses), and clauses which are bigger than any other one are removed before converting the clauses into rights. The resulting set of rights is then the set X of rights associated to the ciphertext.

4.3 Efficiency Considerations

Running Time. It is easy to see that the running-time of a Covercrypt encapsulation is linear in the size of the target access policy: $T^{cc}_{enc} = O(|X|)$.

However, in order to guarantee the policy-privacy and the attribute-hiding properties, in addition to the indistinguishability of the public keys, we need to avoid leakage during decryption: in our implementation, the rights in X and Y are first randomly permuted, so that timing attacks cannot help guessing which right has led to the decryption. However, as we stop as soon as decryption succeeds, the decryption time is not constant, but depends on the number of rights in X and Y, and the size of the $X \cap Y$.

Let S be the random variable that counts the number of user's rights that are tried before a right allowing to open the encapsulation is tried. This random variable follows a negative hypergeometric distribution of parameter $N = |Y|$, $K = |Y| - |X \cap Y|$ and $r = 1$. The expected number of rights drawn from $Y \setminus X \cap Y$ before getting a success in $X \cap Y$ is therefore:

$$E(S) = \frac{|Y| - |X \cap Y|}{|X \cap Y| + 1}.$$

Since each right drawn from Y is tried against each right from X, if the time of a trial is T_u, then the expected running time $E(T^{cc}_{dec})$ of a Covercrypt decapsulation of a ciphertext associated to a set of rights X with a user's secret key associated to a set of rights Y is:

$$E(T^{cc}_{dec}) = \left(\frac{|Y| - |X \cap Y|}{|X \cap Y| + 1} + \frac{1}{2} \right) \cdot |X| \cdot T_u.$$

In this formula, the roles of X and Y are not symmetrical, which results in the fact that testing each elements from the smaller set against each element from the bigger one has a smaller expected running time as soon as $X \cap Y$ is greater than one. Since our scheme is expected to produce smaller sets of rights for encapsulations, the most efficient way to perform a decapsulation is to try each encapsulation against each secret from the user's secret key.

Storage Efficiency. Given d dimensions each of n_d attributes, and since one right per intersection of orthogonal attributes is defined, the total number of rights is $R = \prod_{d \in \mathcal{D}} (|n_d + 1|)$, where the $+1$ counts clauses that involve dimensions with an empty attribute. If U is the number of users in the system, the size of the master secret key is thus in $O(R + U)$. The size of the ciphertext is linear in $|X|$, while the size of users' secret is linear in $|Y|$.

Additional Rights. Covercrypt is inspired by the subset-cover notion, as introduced with NNL [NNL01], for revocation in broadcast encryption. In the above description, we consider all the subsets that are singletons, with a unique right, and users receive several of them, while ciphertexts need a minimal covering to express the Boolean formula. For better efficiency, we already implicitly introduced a first alias for the full space, where all the dimensions have empty attributes. But one could introduce many other aliases, for any union of rights. For example, in the case of a unique dimension, the complete subtree, or the subtree-difference [NNL01] could be used.

Dynamic Access Structure and Users. Covercrypt [ETS25] has been designed to allow a dynamic evolution of the access structure, by adding new dimensions, or new attributes in existing dimensions, as well as new users, without having to re-encrypt all the data. The specification precises when users' keys have to be renewed or refreshed, after some modification of the access structure. But in order not to loose old rights on old ciphertexts, our implementation considers users' secret keys as sets of keys for each right, keeping the old keys for the old rights, and adding new keys for the new rights. For decryption, the user tries each encapsulation against each secret for the current time period. If there is no matching, he tries again with the secrets for the previous time period, etc. For efficiency reasons, one could truncate each set of keys for each right.

5 Experimental Results

We have implemented the standard [ETS25] in Rust, with additional features[4]. It uses the `ml-kem` library[5]) and Diffie-Hellman, as well as SHA3 for the hash function. Thanks to the black-box design, we propose ML-KEM 512 or 768, and Diffie-Hellman on Ristretto255 (built on top of Curve25519) or P-256. More schemes can be easily added.

Timings reported in table 1 were measured on an Intel(R) Xeon(R) CPU @ 2.30GHz and correspond to a Covercrypt encapsulation and decapsulation, with ML-KEM 512 and Ristretto255, for a 32-byte symmetric key in function of $|X|$ and $|Y|$, with $|X \cap Y|$ growing one-by-one with the increase in $|Y|$ from 1 to $|X|$: the darker is the cell, the larger is the intersection (making the early-aborts approach more profitable), from 1 to 5. A comparison with timings measured using the GPSW pairing-based KEM [GPSW06] is also provided[6]. The CCA version is about

Size of X	1	2	3	4	5
Covercrypt	271	378	515	652	794
[BdPP23] (CPA only)	191	272	329	401	487
GPSW KEM	4793	5431	6170	6607	7245

Covercrypt encapsulation time (in μs)

$\lvert Y\rvert \downarrow \quad \setminus \lvert X\rvert \rightarrow$	1	2	3	4	5
12 [BdPP23] (CPA only)	508	896	1276	1688	2062
12	1100	1922	2640	3420	4360
18	1515	1520	2429	3214	4304
24	1908	2022	1955	2780	3547
30	2380	2324	2370	2394	3484
36	2828	2891	2776	2829	2817

$\lvert X \cap Y\rvert$	1	2	3	4	5

Covercrypt decapsulation time (in μs)

Table 1. Comparisons of Covercrypt and GPSW encapsulation/decapsulation times. For decapsulation, GPSW has a constant runtime of approximately 3880 μs.

twice as slow as the CPA version due to reliance on the Fujisaki-Okamoto transform both on the pre- and post-quantum encapsulations. Part of the slowdown can also be explained by the change of the underlying Kyber library.

[4] `https://github.com/Cosmian/cover_crypt/releases/tag/v15.0.0`
[5] `https://docs.rs/ml-kem/latest/ml_kem/`
[6] `https://github.com/Cosmian/abe_gpsw`

References

BBDP01. Mihir Bellare, Alexandra Boldyreva, Anand Desai, and David Pointcheval. Key-privacy in public-key encryption. In Colin Boyd, editor, *ASIACRYPT 2001*, volume 2248 of *LNCS*, pages 566–582. Springer, Berlin, Heidelberg, December 2001.

BDK+18. Joppe W. Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS - kyber: A CCA-secure module-lattice-based KEM. In *2018 IEEE European Symposium on Security and Privacy*, pages 353–367. IEEE Computer Society Press, April 2018.

BdPP23. 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. In Gene Tsudik, Mauro Conti, Kaitai Liang, and Georgios Smaragdakis, editors, *ESORICS 2023, Part I*, volume 14344 of *LNCS*, pages 372–392. Springer, Cham, September 2023.

BF99. Dan Boneh and Matthew K. Franklin. An efficient public key traitor tracing scheme. In Michael J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 338–353. Springer, Berlin, Heidelberg, August 1999.

BGG+14. Dan Boneh, Craig Gentry, Sergey Gorbunov, Shai Halevi, Valeria Nikolaenko, Gil Segev, Vinod Vaikuntanathan, and Dhinakaran Vinayagamurthy. Fully key-homomorphic encryption, arithmetic circuit ABE and compact garbled circuits. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 533–556. Springer, Berlin, Heidelberg, May 2014.

BSW07. John Bethencourt, Amit Sahai, and Brent Waters. Ciphertext-policy attribute-based encryption. In *2007 IEEE Symposium on Security and Privacy*, pages 321–334. IEEE Computer Society Press, May 2007.

CFN94. Benny Chor, Amos Fiat, and Moni Naor. Tracing traitors. In Yvo Desmedt, editor, *CRYPTO'94*, volume 839 of *LNCS*, pages 257–270. Springer, Berlin, Heidelberg, August 1994.

CW24. Valerio Cini and Hoeteck Wee. Unbounded ABE for circuits from LWE, revisited. In Kai-Min Chung and Yu Sasaki, editors, *ASIACRYPT 2024, Part IV*, volume 15487 of *LNCS*, pages 238–267. Springer, Singapore, December 2024.

DDP+18. Wei Dai, Yarkin Doröz, Yuriy Polyakov, Kurt Rohloff, Hadi Sajjadpour, Erkay Savas, and Berk Sunar. Implementation and Evaluation of a Lattice-Based Key-Policy ABE Scheme. *IEEE Trans. Inf. Forensics Secur.*, 13(5):1169–1184, 2018.

ETS25. ETSI. TS 104 015: Efficient Quantum-Safe Hybrid Key Exchanges with Hidden Access Policies, February 2025. Cyber Security (CYBER); Quantum-Safe Cryptography (QSC); `https://www.etsi.org/deliver/etsi_ts/104000_104099/104015/01.01.01_60/ts_104015v010101p.pdf`.

GPSW06. Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors, *ACM CCS 2006*, pages 89–98. ACM Press, October / November 2006. Available as Cryptology ePrint Archive Report 2006/309.

NIS22. NIST. Module-Lattice-Based Key-Encapsulation Mechanism Standard. `https://nvlpubs.nist.gov/nistpubs/fips/nist.fips.203.pdf`, 2022.

NNL01. Dalit Naor, Moni Naor, and Jeffery Lotspiech. Revocation and tracing schemes for stateless receivers. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 41–62. Springer, Berlin, Heidelberg, August 2001.

OT12. Tatsuaki Okamoto and Katsuyuki Takashima. Fully secure unbounded inner-product and attribute-based encryption. In Xiaoyun Wang and Kazue Sako, editors, *ASIACRYPT 2012*, volume 7658 of *LNCS*, pages 349–366. Springer, Berlin, Heidelberg, December 2012.

Sho01. Victor Shoup. A Proposal for an ISO Standard for Public Key Encryption. `https://shoup.net/papers/iso-2_1.pdf`, December 2001.

SW05. Amit Sahai and Brent R. Waters. Fuzzy identity-based encryption. In Ronald Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 457–473. Springer, Berlin, Heidelberg, May 2005.