CoverCrypt: Early-Abort KEM for Hidden Fine-Grained Access Policies

Abstract. In this paper, we describe a practical solution to the problem of delivering encrypted contents such that only authorized users can decrypt, without revealing the target set. Our scheme is inspired by the Subset Cover framework where the users' rights are organized as subsets and a content is encrypted with respect to a subset covering of the target set. While some information leaks, such as the number of subsets, one can hide any additional information about the target set. Even if this makes the decryption procedure more complex, we also provide an early-abort technique to quickly select the correct part in the ciphertext.

1 Introduction

2 Definitions

Public-key encryption aims at encrypting a message so that only the recipient can decrypt. To make the scheme as much independent of the format of the message to be signed, the usual paradigm for encryption is KEM-DEM [Sho01], where one first encapsulates a session key that only the recipient can recover, and then the payload is encrypted under that key. The former step uses a Key Encapsulation Mechanism (KEM) and the latter a Data Encoding Method (DEM), that is usually instantiated with an Authenticated Encryption, such as AES256-GCM[1], that provides both the privacy and the authenticity of the plaintext.

This paper focuses on variants of the KEM part. We thus recall some formal definitions.

2.1 Notations

In the following, 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.

In the following, we denote $\mathbb{G} = \langle g \rangle$ a group of prime order p (that has to be exponential), spanned by a generator g. It will be denoted multiplicatively.

[1] `https://docs.rs/aes-gcm/latest/aes_gcm/`

On the other hand, we 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 B_η the centered binomial distribution of parameter η. When a polynomial is sampled according to B_η, it means each of its coefficient 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.

Decisional Diffie-Hellman Problem (DDH$_{\mathbb{G}}$). The DDH assumption in a group $\mathbb{G}$ of prime order p, with a generator g, states that the distributions $\mathcal{D}_0$ and $\mathcal{D}_1$ are hard to distinguish for any polynomial-time adversary, where

$$\mathcal{D}_0 = \{(g^a, g^b, g^{ab}), a, b \overset{\$}{\leftarrow} \mathbb{Z}_p\} \qquad \mathcal{D}_1 = \{(g^a, g^b, g^c), a, b, c \overset{\$}{\leftarrow} \mathbb{Z}_p\}$$

We will denote $\mathsf{Adv}_{\mathbb{G}}^{\mathsf{ddh}}(\mathcal{A})$ the advantage of an adversary $\mathcal{A}$ and $\mathsf{Adv}_{\mathbb{G}}^{\mathsf{ddh}}(\tau)$ the best advantage of any adversary $\mathcal{A}$ within time τ.

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

$$\begin{aligned}\mathcal{D}_0 &= \{(\mathbf{A}, \mathbf{b}), \mathbf{A} \overset{\$}{\leftarrow} \mathsf{R}_q^{m\times k}, (\mathbf{s}, \mathbf{e}) \overset{\$}{\leftarrow} B_\eta^k \times B_\eta^m, \mathbf{b} \leftarrow \mathbf{A}\mathbf{s} + \mathbf{e}\} \\ \mathcal{D}_1 &= \{(\mathbf{A}, \mathbf{b}), \mathbf{A} \overset{\$}{\leftarrow} \mathsf{R}_q^{m\times k}, \mathbf{b} \overset{\$}{\leftarrow} B_\eta^m\}\end{aligned}$$

We will denote $\mathsf{Adv}_{\mathsf{R}_q,m,k,\eta}^{\mathsf{dmlwe}}(\mathcal{A})$ the advantage of an adversary $\mathcal{A}$ and $\mathsf{Adv}_{\mathsf{R}_q,m,k,\eta}^{\mathsf{dmlwe}}(\tau)$ the best advantage of any adversary $\mathcal{A}$ within time τ.

2.2 Key Encapsulation Mechanism

A Key Encapsulation Mechanism (KEM) is defined by 3 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 an encapsulation 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 should satisfy $\Pr_{\mathcal{D}}[\mathsf{Ev}] = 1 - \mathsf{negl}()$, for

$$\begin{aligned}&\mathcal{D} = \{(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KEM.KeyGen}(1^\kappa), (C, K) \leftarrow \mathsf{KEM.Enc}(\mathsf{pk}) : (\mathsf{sk}, C, K)\} \\ &\mathsf{Ev} = [\mathsf{KEM.Dec}(\mathsf{sk}, C) = K]\end{aligned}$$

Session-Key Privacy. On the other hand, such a KEM is said to provide *session-key privacy* (denoted $\mathsf{SK\text{-}IND}$) in the key space $\mathcal{K}$, if the encapsulated key is indistinguishable from a random key in $\mathcal{K}$. More formally, a KEM is $\mathsf{SK\text{-}IND}$-secure if for any adversary $\mathcal{A}$, $\mathsf{Adv}^{\mathsf{sk\text{-}ind}}_{\mathsf{KEM}}(\mathcal{A}) = \mathsf{negl}()$ for

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

$$\mathsf{Adv}^{\mathsf{sk\text{-}ind}}_{\mathsf{KEM}}(\mathcal{A}) = 2 \times \Pr_{\mathcal{D}_b}[\mathcal{A}(\mathsf{pk}, C, K_b) = b] - 1.$$

Public-Key Privacy. One can also expect anonymity of the receiver, also known as *public-key privacy* (denoted $\mathsf{PK\text{-}IND}$), if the encapsulation does not leak any information about the public key. More formally, a KEM is $\mathsf{PK\text{-}IND}$-secure if for any adversary $\mathcal{A}$, $\mathsf{Adv}^{\mathsf{pk\text{-}ind}}_{\mathsf{KEM}}(\mathcal{A}) = \mathsf{negl}()$ for

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

$$\mathsf{Adv}^{\mathsf{pk\text{-}ind}}_{\mathsf{KEM}}(\mathcal{A}) = 2 \times \Pr_{\mathcal{D}_b}[\mathcal{A}(\mathsf{pk}_0, \mathsf{pk}_1, C_b) = b] - 1.$$

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

- $\mathsf{EG.KeyGen}(1^\kappa)$: sample random $\mathsf{sk} = x \xleftarrow{\$} \mathbb{Z}_p$ and set $\mathsf{pk} = h \leftarrow g^x$;
- $\mathsf{EG.Enc}(\mathsf{pk})$: sample a random $r \xleftarrow{\$} \mathbb{Z}_p$ and set $C \leftarrow g^r$ together with $K \leftarrow h^r$;
- $\mathsf{EG.Dec}(\mathsf{sk}, C)$: output $K \leftarrow C^x$.

Under the DDH assumption in $\mathbb{G}$, this KEM is $\mathsf{SK\text{-}IND}$ with $\mathcal{K} = \mathbb{G}$. In addition, as the encapsulation $C = g^r$ is independent of the public key, it perfectly provides $\mathsf{PK\text{-}IND}$. The formal security proofs for an extended version of this scheme will be given later, with the postpone the analysis of this scheme.

2.3 Key Encapsulation Mechanism with Access Control

A Key Encapsulation Mechanism with Access Control allows multiple users to access the encapsulated key K from C, according to a rule $\mathcal{R}$ applied on X in the user's key usk and Y in the encapsulation C. It is defined by 4 algorithms:

- $\mathsf{KEMAC.Setup}(1^\kappa)$: the *initialisation algorithm* outputs the global public parameters PK and the master secret key MSK;
- $\mathsf{KEMAC.KeyGen}(\mathsf{MSK}, Y)$: the *key generation algorithm* outputs the user's secret key usk according to Y;
- $\mathsf{KEMAC.Enc}(\mathsf{PK}, X)$: the *encapsulation algorithm* generates a session key K and an encapsulation C of it according to X;
- $\mathsf{KEMAC.Dec}(\mathsf{usk}, C)$: the *decapsulation algorithm* outputs the key K encapsulated in C.

Correctness. A correct KEMAC should satisfy $\Pr_{\mathcal{D}}[\mathsf{Ev}] = 1 - \mathsf{negl}()$, for

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

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

Session-Key Privacy. As for the basic KEM, one may expect some privacy properties. Session-key privacy is modeled by indistinguishability of ciphertexts, even if the adversary has received some decryption keys, as soon as associated Y_i are incompatible with X ($\mathcal{R}(X, Y_i) = 0$). Such a KEMAC is said to be $\mathsf{SK\text{-}IND}$-secure in the key space $\mathcal{K}$ if for any adversary $\mathcal{A}$, that can ask any key usk_i, using oracle $\mathcal{O}\mathsf{KeyGen}(Y_i)$ that stores Y_i in the set $\mathcal{Y}$ and outputs $\mathsf{KEMAC.KeyGen}(\mathsf{MSK}, Y_i)$, $\mathsf{Adv}^{\mathsf{sk\text{-}ind}}_{\mathsf{KEMAC}}(\mathcal{A}) = \mathsf{negl}()$ for

$$\mathcal{D}_b = \left\{ \begin{array}{l} (\mathsf{PK}, \mathsf{MSK}) \leftarrow \mathsf{KEMAC.Setup}(1^\kappa), \\ (\mathsf{state}, X) \leftarrow \mathcal{A}^{\mathcal{O}\mathsf{KeyGen}(\cdot)}(\mathsf{PK}), \\ (C, K_0) \leftarrow \mathsf{KEMAC.Enc}(\mathsf{PK}, X), K_1 \xleftarrow{\$} \mathcal{K} \end{array} : (\mathsf{state}, C, K_b) \right\}$$

$$\mathsf{BadXY} = [\exists Y_i \in \mathcal{Y}, \mathcal{R}(X, Y_i) = 1]$$

$$\mathsf{Adv}^{\mathsf{pk\text{-}ind}}_{\mathsf{KEMAC}}(\mathcal{A}) = 2 \times \Pr_{\mathcal{D}_b}[\mathcal{A}^{\mathcal{O}\mathsf{KeyGen}(\cdot)}(\mathsf{state}, C, K_b) = b \wedge \neg\mathsf{BadXY}] - 1.$$

We note the bad event BadXY (decided at the end of the game) should be avoided by the adversary, as it would lead to a trivial guess, and is thus considered as a non-legitimate attack.

Access-Control Privacy. In addition, one could want to hide the parameter X used in the encapsulation C even if the adversary $\mathcal{A}$ can ask any key usk_i for Y_i such that $\mathcal{R}(X_0, Y_i) = \mathcal{R}(X_1, Y_i) = 0$ for all i, using oracle $\mathcal{O}\mathsf{KeyGen}(Y_i)$ that stores Y_i in the set $\mathcal{Y}$ and outputs $\mathsf{KEMAC.KeyGen}(\mathsf{MSK}, Y_i)$. A KEMAC is said to be $\mathsf{AC\text{-}IND}$-secure if for any adversary $\mathcal{A}$, that can ask any key usk_i, using oracle $\mathcal{O}\mathsf{KeyGen}(Y_i)$ that stores Y_i in the set $\mathcal{Y}$ and outputs $\mathsf{KEMAC.KeyGen}(\mathsf{MSK}, Y_i)$, $\mathsf{Adv}^{\mathsf{ac\text{-}ind}}_{\mathsf{KEMAC}}(\mathcal{A}) = \mathsf{negl}()$ for

$$\mathcal{D}_b = \left\{ \begin{array}{l} (\mathsf{PK}, \mathsf{MSK}) \leftarrow \mathsf{KEMAC.Setup}(1^\kappa), \\ (\mathsf{state}, X_0, X_1) \leftarrow \mathcal{A}^{\mathcal{O}\mathsf{KeyGen}(\cdot)}(\mathsf{PK}), \\ (C_i, K_i) \leftarrow \mathsf{KEMAC.Enc}(\mathsf{PK}, X_i), \text{ for } i = 0, 1 \end{array} : (\mathsf{state}, C_b) \right\}$$

$$\mathsf{BadXY} = [\exists Y_i \in \mathcal{Y}, \mathcal{R}(X_0, Y_i) = 1 \vee \mathcal{R}(X_1, Y_i) = 1]$$

$$\mathsf{Adv}^{\mathsf{ac\text{-}ind}}_{\mathsf{KEMAC}}(\mathcal{A}) = 2 \times \Pr_{\mathcal{D}_b}[\mathcal{A}^{\mathcal{O}\mathsf{KeyGen}(\cdot)}(\mathsf{state}, C_b) = b \wedge \neg\mathsf{BadXY}] - 1.$$

Again, the bad event BadXY (decided at the end of the game) denotes non-legitimate attacks, where there is a trivial guess, at least in case of monotonous access structures.

Traceability. In any multi-user setting, to avoid abuse of the decryption keys, one may want to be able to trace a user (or his personal key) from the decryption mechanism, and more generally from any *useful* decoder, either given access to the key material in the device (white-box tracing) or just interacting with the device (back-box tracing). 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 of the traitors.

A weaker variant of traceability is just a confirmation of candidate traitors, and we will target this goal: if a pirate decoder $\mathcal{P}$ has been generated from a list $\mathcal{T} = \{Y_i\}$ of traitors' keys, a confirmer algorithm $\mathcal{C}$ can output, from a valid guess $\mathcal{G}$ for $\mathcal{T}$, at least one traitor in $\mathcal{T}$. More formally, let us consider any adversary $\mathcal{A}$ that can ask for key generation through oracle $\mathcal{O}\mathsf{KeyGen}(Y_i)$, that gets $\mathsf{usk}_i \leftarrow \mathsf{KEMAC.KeyGen}(\mathsf{MSK}, Y_i)$, outputs nothing but appends the new user Y_i in $\mathcal{U}$, and then corrupt some users through the corruption oracle $\mathcal{O}\mathsf{Corrupt}(Y_i)$, that outputs usk_i and appends Y_i in $\mathcal{T}$, to build a *useful* pirate decoder $\mathcal{P}$, then there is a *correct* confirmer algorithm $\mathcal{C}$ that outputs a traitor T, with *negligible error*:

$$\begin{aligned}
&\text{Useful } \mathcal{P}: && 2 \times \Pr_{\mathcal{D},b}[\mathcal{P}(C, K_b) = b] - 1 \text{ is non } \mathsf{negl}() \\
&\text{Correct } \mathcal{C}: && \Pr_{\mathcal{D}}[T \in \mathcal{T} \,\big|\, T \leftarrow \mathcal{C}^{\mathcal{P}(\cdot,\cdot)}(\mathsf{MSK}, \mathcal{T})] = 1 - \mathsf{negl}(), \\
&\text{Negl-Error } \mathcal{C}: && \Pr_{\mathcal{D}}[T \notin \mathcal{T} \,\big|\, T \leftarrow \mathcal{C}^{\mathcal{P}(\cdot,\cdot)}(\mathsf{MSK}, \mathcal{G}) \wedge T \neq \perp] = \mathsf{negl}(), \forall \mathcal{G} \subset \mathcal{U},
\end{aligned}$$

for

$$\mathcal{D} = \left\{ \begin{array}{l} (\mathsf{PK}, \mathsf{MSK}) \leftarrow \mathsf{KEMAC.Setup}(1^\kappa), \\ \mathcal{P} \leftarrow \mathcal{A}^{\mathcal{O}\mathsf{KeyGen}(\cdot), \mathcal{O}\mathsf{Corrupt}(\cdot)}(\mathsf{PK}), \\ X \text{ such that } \forall Y_i \in \mathcal{T}, \mathcal{R}(X, Y_i) = 1, \\ (C, K_0) \leftarrow \mathsf{KEMAC.Enc}(\mathsf{PK}, X), \\ K_1 \xleftarrow{\$} \mathcal{K} \end{array} : (\mathsf{MSK}, \mathcal{P}, \mathcal{U}, \mathcal{T}, C, K_0, K_1) \right\}.$$

The above distribution initializes the system and lets the adversary generate user' keys (to fill the set $\mathcal{U}$) and corrupt some of them (to fill the set $\mathcal{T}$) to build a pirate decoder $\mathcal{P}$. Then, a key is encapsulated so that all the corrupted users can get it back. We say that the decoder $\mathcal{P}$ is *useful* if it can distinguish the real key from a random key with significant advantage. Then, from such a useful decoder, the confirmer $\mathcal{C}$ is *correct* if it outputs a traitor with overwhelming probability, when it starts from the correct set $\mathcal{T}$ of candidates. Eventually, it should not output an honest user, in any case, but with negligible probability.

The t-confirmation limits the number of corrupted users in $\mathcal{T}$ to t.

3 Early-Abort Technique

With public-key privacy, one cannot know who is the actual receiver. And then one needs to wait for using the obtained session key with an authenticated encryption scheme to check the validity. The latter check can be time-consuming with a large plaintext. We thus propose an early-abort approach, that remains compatible with public-key privacy.

```
$\mathsf{KEM}'.\mathsf{KeyGen}(1^\kappa)$:
1. $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KEM}.\mathsf{KeyGen}(1^\kappa)$
2. return $(\mathsf{pk}, \mathsf{sk})$

$\mathsf{KEM}'.\mathsf{Enc}(\mathsf{pk})$:
1. $(c, s) \leftarrow \mathsf{KEM}.\mathsf{Enc}(\mathsf{pk})$
2. $U \| V \leftarrow G(s, k + \ell)$
3. $C \leftarrow (c, V)$, $K \leftarrow U$
4. return $(C, K)$

$\mathsf{KEM}'.\mathsf{Dec}(\mathsf{sk}, C = (c, V))$:
1. $s \leftarrow \mathsf{KEM}.\mathsf{Dec}(\mathsf{sk}, c)$
2. $U' \| V' \leftarrow G(s, k + \ell)$
3. if $V = V'$, return $U'$, otherwise reject
```

Fig. 1. Early-Abort KEM'

To this aim, one can use a Pseudo-Random Generator (PRG) G, where $G(s, \ell)$ expands a seed s into an ℓ-bit string. Such a function G is called a PRG if for any adversary $\mathcal{A}$ and any polynomially-bounded ℓ, $\mathsf{Adv}_G^{\mathsf{prg}}(\mathcal{A}, \ell) = \mathsf{negl}()$ for

$$\mathcal{D}_b = \left\{ s \overset{\$}{\leftarrow} \mathcal{S}, K_0 \leftarrow G(s, \ell), K_1 \overset{\$}{\leftarrow} \{0,1\}^\ell \ : K_b \right\}$$
$$\mathsf{Adv}_G^{\mathsf{prg}}(\mathcal{A}) = 2 \times \Pr_{\mathcal{D}_b}[\mathcal{A}(K_b) = b] - 1.$$

3.1 Early-Abort Key Encapsulation Mechanism

Such a KEM' with early abort just differs with the decapsulation algorithm that might reject. And we expect to accept a wrong key with negligible probability, so that we do not have to wait the Authenticated Encryption to detect invalid decryption, which can take time for a large payload.

In the scheme below (see Figure 1), we use a KEM generating keys in $\mathcal{S}$ and two security parameters: k is the length of the encapsulated key and ℓ is the length of the verification tag:

- $\mathsf{KEM}'.\mathsf{KeyGen}(1^\kappa)$: one runs $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KEM}.\mathsf{KeyGen}(1^\kappa)$;
- $\mathsf{KEM}'.\mathsf{Enc}(\mathsf{pk})$: one runs $(c, s) \leftarrow \mathsf{KEM}.\mathsf{Enc}(\mathsf{pk})$ and gets $U \| V \leftarrow G(s, k + \ell)$. One then outputs $C \leftarrow (c, V)$ together with the encapsulated key $K \leftarrow U$;

– $\mathsf{KEM}'.\mathsf{Dec}(\mathsf{sk}, C)$: for $C = (c, V)$, one runs $s \leftarrow \mathsf{KEM.Dec}(\mathsf{sk}, c)$, gets $U' \| V' \leftarrow G(s, k+\ell)$, and checks whether $V = V'$. In the positive case, one outputs $K' \leftarrow U'$, or otherwise rejects the ciphertext.

Note that one can make to decapsulate a wrong key if $V = V'$ whereas s is not the correct seed: this might happen with probability bounded by $1/2^{-\ell} + \mathsf{Adv}^{\mathsf{prg}}_{\mathsf{PRG}}(\tau, k+\ell)$, where τ is the maximal time of the adversary generating the fake encapsulation.

Note that if $\mathcal{S} = \{0,1\}^{k+\ell}$, then the identity function is a PRG with $\mathsf{Adv}^{\mathsf{prg}}_{G}(\tau, k+\ell) = 0$, for any powerful adversary. If G is implemented by a hash function modelled by a a random oracle, $\mathsf{Adv}^{\mathsf{prg}}_{G}(\tau, k+\ell) = 0$ too.

3.2 Security Analysis

For the above KEM' scheme, we can show that it keeps the initial security properties of the KEM scheme:

Theorem 1 (Session-Key Privacy). *If KEM is SK-IND-secure, and G a secure* PRG*, KEM′ is SK-IND-secure:* $\mathsf{Adv}^{\textsf{sk-ind}}_{\mathsf{KEM}'}(\tau) \leq \mathsf{Adv}^{\textsf{sk-ind}}_{\mathsf{KEM}}(\tau) + \mathsf{Adv}^{\mathsf{prg}}_{G}(\tau)$.

Proof. We present a sequence of games, starting from the initial $\mathsf{SK\text{-}IND}$ security game against KEM':

Game $\mathbf{G}_0$: In the initial game, challenger runs $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KEM.KeyGen}(1^\kappa)$, $(c, s) \leftarrow \mathsf{KEM.Enc}(\mathsf{pk})$, and evaluates $U \| V \leftarrow G(s, k+\ell)$ to output either $(\mathsf{pk}, C = (c, V), U)$ or $(\mathsf{pk}, C = (c, V), K)$, for a random $K \xleftarrow{\$} \{0,1\}^k$. The adversary outputs its guess b'. We denote P_0 the probability of event $b' = b$, which is $(1 + \mathsf{Adv}^{\textsf{sk-ind}}_{\mathsf{KEM}'}(\mathcal{A}))/2$.

Game $\mathbf{G}_1$: In this game, we use $s \xleftarrow{\$} \mathcal{S}$ to evaluate $U \| V \leftarrow G(s, k+\ell)$. The difference is the $\mathsf{SK\text{-}IND}$-game on the underlying KEM. Hence, $P_0 - P_1 \leq \mathsf{Adv}^{\textsf{sk-ind}}_{\mathsf{KEM}}(\tau)$, where τ is the maximum running-time of adversary $\mathcal{A}$.

Game $\mathbf{G}_2$: In this game, we use $U \| V \xleftarrow{\$} \{0,1\}^{k+\ell}$ instead of evaluating G. The difference is the PRG-game, hence $P_1 - P_2 \leq \mathsf{Adv}^{\mathsf{prg}}_{G}(\tau)$. In this final game, this is clear that $P_2 = 1/2$, as U and K are both randomly drawn in $\{0,1\}^k$.

Hence, $\mathsf{Adv}^{\textsf{sk-ind}}_{\mathsf{KEM}'}(\mathcal{A}) \leq \mathsf{Adv}^{\textsf{sk-ind}}_{\mathsf{KEM}}(\tau) + \mathsf{Adv}^{\mathsf{prg}}_{G}(\tau)$. □

Theorem 2 (Public-Key Privacy). *If KEM is both SK-IND and PK-IND-secure, and G a secure* PRG*, KEM′ is PK-IND-secure:* $\mathsf{Adv}^{\textsf{pk-ind}}_{\mathsf{KEM}'}(\mathcal{A}) \leq \mathsf{Adv}^{\textsf{pk-ind}}_{\mathsf{KEM}}(\tau) + 2 \times \mathsf{Adv}^{\textsf{sk-ind}}_{\mathsf{KEM}}(\tau) + 2 \times \mathsf{Adv}^{\mathsf{prg}}_{G}(\tau)$.

Proof. We present a sequence of games, starting from the initial $\mathsf{PK\text{-}IND}$ security game against KEM':

Game $\mathbf{G}_0$: In the initial game, challenger runs $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KEM.KeyGen}(1^\kappa)$, $(c_i, s_i) \leftarrow \mathsf{KEM.Enc}(\mathsf{pk}_i)$, and evaluates $U_i \| V_i \leftarrow G(s_i, k+\ell)$, for $i = 0, 1$, to output $(\mathsf{pk}_0, \mathsf{pk}_1, C_b = (c_b, V_b))$, for a random $b \xleftarrow{\$} \{0,1\}$. The adversary outputs its guess b'. We denote P_0 the probability of event $b' = b$, which is $(1 + \mathsf{Adv}^{\textsf{pk-ind}}_{\mathsf{KEM}'}(\mathcal{A}))/2$.

Game $\mathbf{G}_1$: In this game, we use $s_i \xleftarrow{\$} \mathcal{S}$ to evaluate $U_i \| V_i \leftarrow G(s_i, k+\ell)$, for $i = 0, 1$. The difference is the SK-IND-game on the underlying KEM, for both $i = 0, 1$. Hence, $P_0 - P_1 \leq 2 \times \mathsf{Adv}^{\mathsf{sk\text{-}ind}}_{\mathsf{KEM}}(\tau)$, where τ is the maximum running-time of adversary $\mathcal{A}$.

Game $\mathbf{G}_2$: In this game, we use $U_i \| V_i \xleftarrow{\$} \{0,1\}^{k+\ell}$ instead of evaluating G, for $i = 0, 1$. The difference is the PRG-game, hence $P_1 - P_2 \leq 2 \times \mathsf{Adv}^{\mathsf{prg}}_{G}(\tau)$.

Game $\mathbf{G}_3$: We alter the simulation, by running $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KEM.KeyGen}(1^\kappa)$, $(c_i, s_i) \leftarrow \mathsf{KEM.Enc}(\mathsf{pk}_i)$, for $i = 0, 1$, to output $(\mathsf{pk}_0, \mathsf{pk}_1, C_b = (c_b, V))$, for a random $b \xleftarrow{\$} \{0,1\}$ and a random $V \xleftarrow{\$} \{0,1\}^\ell$. This simulation is perfectly indistinguishable from the previous game: $P_2 = P_3$. And this final game is exactly the PK-IND-game on the underlying KEM: $P_3 = (1 + \mathsf{Adv}^{\mathsf{pk\text{-}ind}}_{\mathsf{KEM}}(\mathcal{A}))/2$.

Hence, $\mathsf{Adv}^{\mathsf{pk\text{-}ind}}_{\mathsf{KEM}'}(\mathcal{A}) \leq \mathsf{Adv}^{\mathsf{pk\text{-}ind}}_{\mathsf{KEM}}(\tau) + 2 \times \mathsf{Adv}^{\mathsf{sk\text{-}ind}}_{\mathsf{KEM}}(\tau) + 2 \times \mathsf{Adv}^{\mathsf{prg}}_{G}(\tau)$. □

4 Subset-Cover KEMAC

The above notion of access control is quite general and includes both key-policy ABE and ciphertext-policy ABE, where one can have policies $\mathcal{P}$ and attributes such that given a subset of attributes, this defines a list of Boolean B (according to the presence or not of the attribute), and $\mathcal{P}(B)$ is either true or false.

4.1 Basic Subset-Cover KEMAC

For efficiency consideration, we will focus on the subset-cover approach: during the Setup, one defines multiple sets S_i; when generating a user key usk_j, a list A_j of subsets if specified, which implicitly means user $U_j \in S_i$ for all $i \in A_j$; at encapsulation time, a target set T is given by B, such that $T = \cup_{i \in B} S_i$.

Intuitively, S_i's are subsets of the universe of users, and to specify the receivers, one encapsulates the key K for a covering of the target set T. A KEMAC, for a list Σ of sets S_i, can then be defined from any KEM in $\mathcal{K}$ that is a group with internal law denoted $\oplus$:

- $\mathsf{KEMAC.Setup}(\Sigma)$: for each $S_i \in \Sigma$, one runs $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KEM.KeyGen}(1^\kappa)$. The global public parameters are $\mathsf{PK} \leftarrow (\mathsf{pk}_i)_i$ and the master secret key is $\mathsf{MSK} \leftarrow (\mathsf{sk}_i)_i$;
- $\mathsf{KEMAC.KeyGen}(\mathsf{MSK}, A_j)$: one defines the user's secret key $\mathsf{usk}_j \leftarrow (i, \mathsf{sk}_i)_{i \in A_j}$;
- $\mathsf{KEMAC.Enc}(\mathsf{PK}, B)$: one generates a random session key $K \leftarrow \mathcal{K}$, and runs $(C_i, K_i) \leftarrow \mathsf{KEM.Enc}(\mathsf{pk}_i)$ for all $i \in B$, and outputs $C \leftarrow (i, C_i, E_i = K \oplus K_i)_{i \in B}$ together with the encapsulated key K;
- $\mathsf{KEMAC.Dec}(\mathsf{usk}_j, C)$: one looks for $i \in \mathsf{usk}_j \cap C$, to run $K'_i \leftarrow \mathsf{KEM.Dec}(\mathsf{sk}_i, C_i)$ and output $K \leftarrow K'_i \oplus E_i$.

In terms of attributes, one can consider that each S_i is associated to an attribute a_i, and being in S_i for a user U_j means owning the attribute a_i. At encapsulation time, B lists the attributes that allow to decrypt: as soon as a_i is in B, any user U_j owning a_i can decrypt.

For the above scheme, we can claim the SK-IND security, but unfortunately not the AC-IND security.

Theorem 3 (Session-Key Privacy). *If the underlying* KEM *is* $\mathsf{SK\text{-}IND}$*-secure, the above basic subset-cover* KEMAC *is* $\mathsf{SK\text{-}IND}$*-secure, for selective key-queries:* $\mathsf{Adv}^{\mathsf{sk\text{-}ind}}_{\mathsf{KEMAC}}(\tau) \leq 2q_k \times \mathsf{Adv}^{\mathsf{sk\text{-}ind}}_{\mathsf{KEM}}(\tau)$*, where* q_k *is the number of key-queries.*

Proof. In the selective setting, the adversary asks, from the beginning, the keys it wants to get, before seeing the global public parameters PK.

Game $\mathbf{G}_0$: In the initial game, the adversary thus asks for the keys it wants: for several sets A_j. One calls $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KEM.KeyGen}(1^\kappa)$, for each $S_i \in \Sigma$, and provides PK together with all the asked keys sk_i, for $i \in A = \cup A_j$ (all the asked sets). The adversary answers with a set B, but with the constraint that $A \cap B = \emptyset$, and the challenger flips a random bit $b \xleftarrow{\$} \{0,1\}$, generates two random session keys $K'_0, K'_1 \leftarrow \mathcal{K}$, runs $(C_i, K_i) \leftarrow \mathsf{KEM.Enc}(\mathsf{pk}_i)$ for all $i \in B$, and outputs $C \leftarrow (i, C_i, E_i = K'_0 \oplus K_i)_{i \in B}$ together with the challenged key K'_b (that is either the really encapsulated key if $b = 0$ or a random key if $b = 1$). The adversary outputs its guess b'. We denote P_0 the probability of event $b' = b$, which is $(1 + \mathsf{Adv}^{\mathsf{sk\text{-}ind}}_{\mathsf{KEMAC}}(\mathcal{A}))/2$.

Game $\mathbf{G}_1$: In this game, we replace all the K_i's by $K_i \xleftarrow{\$} \mathcal{K}$ in the generation of E_i. To show this game is indistinguishable from the previous one, we define a sequence of hybrid games, for index I, such that for all $i < I$, one replaces K_i by a random element in $\mathcal{K}$. For $I = 1$, this is $\mathbf{G}_0$, whereas for $I = q_k + 1$, where q_k is the maximal number of keys, this is $\mathbf{G}_1$. And the gap between I and $I + 1$ is the $\mathsf{SK\text{-}IND}$-game on the underlying KEM. Hence, $P_0 - P_1 \leq q_k \times \mathsf{Adv}^{\mathsf{sk\text{-}ind}}_{\mathsf{KEM}}(\tau)$, where τ is the maximum running-time of adversary $\mathcal{A}$.

Game $\mathbf{G}_2$: In this game, we replace all the E_i's by $E_i \xleftarrow{\$} \mathcal{K}$, which is perfectly indistinguishable from $K'_0 \oplus K_i$ for a random K_i, under the group-law property. Hence, $P_1 = P_2$. In this final game, this is clear that $P_2 = 1/2$, as K'_0 and K'_1 do not appear anymore in C, and so K'_b is just a random key.

Hence, $\mathsf{Adv}^{\mathsf{sk\text{-}ind}}_{\mathsf{KEMAC}}(\mathcal{A}) \leq 2q_k \times \mathsf{Adv}^{\mathsf{sk\text{-}ind}}_{\mathsf{KEM}}(\tau)$. □

However, we need B to be provided in the ciphertext: indices i are given. We then definitely exclude access-control privacy. Alternatively, one can wait for the authenticity check performed by the Authenticated Encryption of the payload to identify the correct session key. But this introduces a high computational cost, in particular if the payload is large.

4.2 Anonymous Subset-Cover KEMAC with Early Abort

To avoid sending B together with the ciphertext, but still being able to quickly find the correct matching indices in the ciphertext and the user's key, one can use a KEM' with Early Abort:

- $\mathsf{KEMAC.Setup}(\Sigma)$: for each $S_i \in \Sigma$, one runs $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KEM}'.\mathsf{KeyGen}(1^\kappa)$. The global public parameters are $\mathsf{PK} \leftarrow (\mathsf{pk}_i)_i$ and the master secret key is $\mathsf{MSK} \leftarrow (\mathsf{sk}_i)_i$;

- $\mathsf{KEMAC.KeyGen}(\mathsf{MSK}, A_j)$: one defines the user's secret key $\mathsf{usk}_j \leftarrow (\mathsf{sk}_i)_{i \in A_j}$;
- $\mathsf{KEMAC.Enc}(\mathsf{PK}, B)$: one generates a random session key $K \stackrel{\$}{\leftarrow} \{0,1\}^k$, and, for all $i \in B$, runs $(C_i, K_i) \leftarrow \mathsf{KEM}'.\mathsf{Enc}(\mathsf{pk}_i)$ and outputs $C \leftarrow (C_i, E_i = K \oplus K_i)_{i \in B}$ together with the encapsulated key K;
- $\mathsf{KEMAC.Dec}(\mathsf{usk}, C)$: for all sk_i in usk and all (C_j, E_j) in C, one runs $K' \leftarrow \mathsf{KEM}'.\mathsf{Dec}(\mathsf{sk}_i, C_j)$. It stops for the first valid K', with pair (i, j), and outputs $K \leftarrow K' \oplus E_j$.

For this above scheme, we can claim both the SK-IND security and the AC-IND security, for selective key queries:

Theorem 4 (Session-Key Privacy). *If the underlying KEM′ is SK-IND-secure, the above subset-cover KEMAC is also SK-IND-secure, for selective key-queries:* $\mathsf{Adv}^{\textit{sk-ind}}_{\textit{KEMAC}}(\tau) \leq 2q_k \times \mathsf{Adv}^{\textit{sk-ind}}_{\textit{KEM}}(\tau)$*, where q_k is the number of key-queries.*

The same proof as above applies.

Theorem 5 (Access-Control Privacy). *If the underlying KEM′ is PK-IND-secure, the above subset-cover KEMAC is AC-IND-secure, for selective key-queries and constant-size sets B:* $\mathsf{Adv}^{\textit{ac-ind}}_{\textit{KEMAC}}(\tau) \leq 2S_B \times \mathsf{Adv}^{\textit{pk-ind}}_{\textit{KEM}}(\tau)$*, where S_B is the constant-size of the sets B.*

Proof. The proof is quite similar. In the selective setting, the adversary asks, from the beginning, the keys it wants to get, before seeing the global public parameters PK.

Game $\mathbf{G}_0$: In the initial game, the adversary thus asks for the keys it wants: for several sets A_j. One calls $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KEM}'.\mathsf{KeyGen}(1^\kappa)$, for each $S_i \in \Sigma$, and provides PK together with all the asked keys sk_i, for $i \in A = \cup A_j$ (all the asked sets). The adversary answers with two sets B_0 and B_1, but with the constraints that $A \cap B_0 = A \cap B_1 = \emptyset$ and $|B_0| = |B_1|$, and the challenger flips a random bit $b \stackrel{\$}{\leftarrow} \{0,1\}$, generates a random session key $K \leftarrow \mathcal{K}$, runs $(C_i, K_i) \leftarrow \mathsf{KEM}'.\mathsf{Enc}(\mathsf{pk}_i)$ for all $i \in B_0 \cup B_1$, and outputs $C \leftarrow (C_i, E_i = K \oplus K_i)_{i \in B_b}$ together with the challenged key K. The adversary outputs its guess b'. We denote P_0 the probability of event $b' = b$, which is $(1 + \mathsf{Adv}^{\mathsf{ac\text{-}ind}}_{\mathsf{KEMAC}}(\mathcal{A}))/2$.

Game $\mathbf{G}_1$: In this game, one always outputs $C \leftarrow (C_i, E_i = K \oplus K_i)_{i \in B_0}$. To show this game is indistinguishable from the previous one, we define a sequence of hybrid games, for index I, such that for the I first $(C_i, E_i = K \oplus K_i)$ in C are for indices in B_0 and the last are for indices in B_b. For $I = 0$, this is $\mathbf{G}_0$, whereas for $I = |B_0| = |B_1|$, this is $\mathbf{G}_1$. And the gap between I and $I + 1$ is the PK-IND-game on the underlying KEM'. Hence, $P_0 - P_1 \leq |B_0| \times \mathsf{Adv}^{\mathsf{pk\text{-}ind}}_{\mathsf{KEM}'}(\tau)$, where τ is the maximum running-time of adversary $\mathcal{A}$.
Furthermore, in this final game, this is clear that $P_1 = 1/2$, as the challenge C does not depend on b anymore.

Hence, $\mathsf{Adv}^{\mathsf{ac\text{-}ind}}_{\mathsf{KEMAC}}(\mathcal{A}) \leq 2S_B \times \mathsf{Adv}^{\mathsf{pk\text{-}ind}}_{\mathsf{KEM}'}(\tau)$, where S_B is the constant-size of the sets B. □

5 Hybrid KEM

While one can never exclude an attack against a cryptographic scheme, combining several independent approaches reduces the risks. This is the way one suggests to apply post-quantum schemes, in combination with classical schemes, in order to be sure to get the best security.

5.1 Hybrid KEM Construction

Let us first study the combination of two KEMs (KEM_1 and KEM_2), so that as soon as one of them achieves SK-IND security, the hybrid KEM achieves SK-IND security too. We need both KEMs to generate keys in $\mathcal{K}$, with a group structure and internal law denoted $\oplus$:

```
$\mathsf{KEM.KeyGen}(1^\kappa)$:
    $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KEM.KeyGen}_i(1^\kappa)$, for $i = 1, 2$
    $\mathsf{pk} \leftarrow (\mathsf{pk}_1, \mathsf{pk}_2)$; $\mathsf{sk} \leftarrow (\mathsf{sk}_1, \mathsf{sk}_2)$
    return $(\mathsf{pk}, \mathsf{sk})$

$\mathsf{KEM.Enc}(\mathsf{pk})$:
    $(C_i, K_i) \leftarrow \mathsf{KEM}_i.\mathsf{Enc}(\mathsf{pk}_i)$, for $i = 1, 2$
    $C \leftarrow (C_1, C_2)$; $K \leftarrow K_1 \oplus K_2$
    return $(C, K)$

$\mathsf{KEM.Dec}(\mathsf{sk}, C)$:
    $K_i \leftarrow \mathsf{KEM}_i.\mathsf{Dec}(\mathsf{sk}_i, C_i)$, for $i = 1, 2$
    $K \leftarrow K_1 \oplus K_2$
    return $K$
```

Fig. 2. Hybrid KEM

- $\mathsf{KEM.KeyGen}(1^\kappa)$: call $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KEM}_i.\mathsf{KeyGen}(1^\kappa)$, for $i \in \{0, 1\}$ and output $\mathsf{pk} \leftarrow (\mathsf{pk}_1, \mathsf{pk}_2)$ and $\mathsf{sk} \leftarrow (\mathsf{sk}_1, \mathsf{sk}_2)$;
- $\mathsf{KEM.Enc}(\mathsf{pk})$: parse pk as $(\mathsf{pk}_1, \mathsf{pk}_2)$, call $(C_i, K_i) \leftarrow \mathsf{KEM}_i.\mathsf{Enc}(\mathsf{pk}_i)$ for $i \in \{0, 1\}$, and output $(C = (C_1, C_2), K = K_1 \oplus K_2)$;
- $\mathsf{KEM.Dec}(\mathsf{sk}, C)$: parse sk as $(\mathsf{sk}_1, \mathsf{sk}_2)$ and C as (C_1, C_2), then call both $K_i \leftarrow \mathsf{KEM}_i.\mathsf{Dec}(\mathsf{sk}_i, C_i)$, and output $K = K_1 \oplus K_2$;

5.2 Security Properties

As expected, we can prove that as soon as one of them achieves SK-IND security, the hybrid KEM achieves SK-IND security too. However, for PK-IND security of KEM, we need both the underlying schemes to be PK-IND secure.

Theorem 6 (Session-Key Privacy). *If at least one of the underlying KEM_1 and KEM_2 is SK-IND-secure, the hybrid KEM is SK-IND-secure.*

$$\mathsf{Adv}^{\textit{sk-ind}}_{\textit{KEM}}(\tau) \leq \min\{\mathsf{Adv}^{\textit{sk-ind}}_{\textit{KEM}_1}(\tau), \mathsf{Adv}^{\textit{sk-ind}}_{\textit{KEM}_2}(\tau)\}.$$

Proof. We present the sequence of games, first exploiting the session-key privacy of KEM_1.

Game $\mathbf{G}_0$: In the initial game, the adversary receives $\mathsf{pk} \leftarrow (\mathsf{pk}_1, \mathsf{pk}_2)$, both keys having been generated by the respective key generation algorithms, together with $C = (C_1, C_2)$ and K that is either $K_1 \oplus K_2$ (each encapsulated in C_1 and C_2) or a random key from $\mathcal{K}$, according to the random bit b. The adversary outputs its guess b'. We denote P_0 the probability of event $b' = b$, which is $(1 + \mathsf{Adv}^{\mathsf{sk\text{-}ind}}_{\mathsf{KEM}}(\mathcal{A}))/2$.

Game $\mathbf{G}_1$: In this game, we replace K_1's by $K_1 \overset{\$}{\leftarrow} \mathcal{K}$ in the generation of $K_1 \oplus K_2$: $P_0 - P_1 \leq \mathsf{Adv}^{\mathsf{sk\text{-}ind}}_{\mathsf{KEM}_1}(t)$, where t is the maximum running-time of adversary $\mathcal{A}$. In this final game, this is clear that $P_1 = 1/2$, as K is truly random in $\mathcal{K}$ in both cases.

Hence, $\mathsf{Adv}^{\mathsf{sk\text{-}ind}}_{\mathsf{KEM}}(\mathcal{A}) \leq \mathsf{Adv}^{\mathsf{sk\text{-}ind}}_{\mathsf{KEM}_1}(t)$. In the same way, one can prove $\mathsf{Adv}^{\mathsf{sk\text{-}ind}}_{\mathsf{KEM}}(\mathcal{A}) \leq \mathsf{Adv}^{\mathsf{sk\text{-}ind}}_{\mathsf{KEM}_2}(t)$, Hence

$$\mathsf{Adv}^{\mathsf{sk\text{-}ind}}_{\mathsf{KEM}}(\mathcal{A}) \leq \min\{\mathsf{Adv}^{\mathsf{sk\text{-}ind}}_{\mathsf{KEM}_1}(\tau), \mathsf{Adv}^{\mathsf{sk\text{-}ind}}_{\mathsf{KEM}_2}(\tau)\}.$$

□

Theorem 7 (Public-Key Privacy). *If both underlying KEM_1 and KEM_2 are PK-IND-secure, the hybrid KEM is PK-IND-secure:*

$$\mathsf{Adv}^{\textit{pk-ind}}_{\textit{KEM}}(\tau) \leq \mathsf{Adv}^{\textit{pk-ind}}_{\textit{KEM}_1}(\tau) + \mathsf{Adv}^{\textit{pk-ind}}_{\textit{KEM}_2}(\tau).$$

Proof. The proof is quite similar to the previous one, but with the security of both schemes:

Game $\mathbf{G}_0$: In the initial game, the adversary receives $\mathsf{pk}^{(0)} \leftarrow (\mathsf{pk}_1^{(0)}, \mathsf{pk}_2^{(0)})$ and $\mathsf{pk}^{(1)} \leftarrow (\mathsf{pk}_1^{(1)}, \mathsf{pk}_2^{(1)})$, with keys having been generated by the respective key generation algorithms, together with $C = (C_1, C_2)$ and $K = K_1 \oplus K_2$ where $(C_i, K_i) \leftarrow \mathsf{KEM}_i.\mathsf{Enc}(\mathsf{pk}_i^{(b)})$, according to the random bit b. The adversary outputs its guess b'. We denote P_0 the probability of event $b' = b$, which is $(1 + \mathsf{Adv}^{\mathsf{pk\text{-}ind}}_{\mathsf{KEM}}(\mathcal{A}))/2$.

Game $\mathbf{G}_1$: In this game, we replace (C_1, K_1) by $(C_1, K_1) \leftarrow \mathsf{KEM}_1.\mathsf{Enc}(\mathsf{pk}_1^{(0)})$ in the generation of $C = (C_1, C_2)$ and $K = K_1 \oplus K_2$: $P_0 - P_1 \leq \mathsf{Adv}^{\mathsf{pk\text{-}ind}}_{\mathsf{KEM}_1}(t)$, where t is the maximum running-time of adversary $\mathcal{A}$.

Game $\mathbf{G}_2$: In this game, we replace (C_2, K_2) by $(C_2, K_2) \leftarrow \mathsf{KEM}_2.\mathsf{Enc}(\mathsf{pk}_2^{(0)})$ in the generation of $C = (C_1, C_2)$ and $K = K_1 \oplus K_2$: $P_1 - P_2 \leq \mathsf{Adv}^{\mathsf{pk\text{-}ind}}_{\mathsf{KEM}_2}(t)$. In this final game, this is clear that $P_2 = 1/2$, as (C, K) is independent of b.

Hence, we can claim

$$\mathsf{Adv}^{\mathsf{pk\text{-}ind}}_{\mathsf{KEM}}(\mathcal{A}) \leq \mathsf{Adv}^{\mathsf{pk\text{-}ind}}_{\mathsf{KEM}_1}(\tau) + \mathsf{Adv}^{\mathsf{pk\text{-}ind}}_{\mathsf{KEM}_2}(\tau).$$

□

6 Traceable KEM

In a subset-cover-based KEMAC, a same decapsulation key sk_i is given to multiple users, for a public key pk_i. In case of abuse, one cannot know who is the defrauder. We propose a ElGamal-based KEM that allows some kind of traceability, in the same vein as [BF99].

6.1 Traceable ElGamal-based TKEM

Let $\mathbb{G}$ be a group of prime order q, with a generator g, in which the Computational Diffie-Hellman problem is hard. We describe below a TKEM with n multiple decapsulation keys for a specific public key, allowing to deal with collusions of less than t users:

- $\mathsf{TKEM.KeyGen}(1^\kappa, n, t)$: generates a public key pk and n different secret keys usk_j:
  - it samples random $s, s_k \xleftarrow{\$} \mathbb{Z}_q$, for $k = 1 \ldots, t$ and sets
  $$h_k \leftarrow g^{s_k} \qquad\qquad h \leftarrow g^s$$
  - for users U_j, for $j = 1 \ldots, n$, one samples random $(v_{j,k})_k \xleftarrow{\$} \mathbb{Z}_q^t$, such that $\sum_k v_{j,k} s_k = s$, for $j = 1 \ldots, n$. Then, $\mathsf{pk} \leftarrow ((h_k)_k, h)$, while each $\mathsf{usk}_j \leftarrow (v_{j,k})_k$.
- $\mathsf{TKEM.Enc}(\mathsf{pk} = ((h_k)_k, h))$: it samples a random $r \xleftarrow{\$} \mathbb{Z}_q$, and sets $C = (C_k \leftarrow h_k^r)_k$, as well as $K \leftarrow h^r$.
- $\mathsf{TKEM.Dec}(\mathsf{usk}_j = (v_{j,k})_k, C = (C_k)_k)$: it outputs $K \leftarrow \prod_k C_k^{v_{j,k}}$

One can note that

$$\prod_k C_k^{v_{j,k}} = \prod_k h_k^{r v_{j,k}} = \prod_k (g^r)^{s_k v_{j,k}} = g^{r \sum_k s_k v_{j,k}} = g^{sr} = h^r = K.$$

6.2 Security Properties

First, we will show that the above TKEM construction achieves both $\mathsf{SK\text{-}IND}$ and $\mathsf{PK\text{-}IND}$ security. But it also allows to confirm traitors, from a stateless pirate decoder $\mathcal{P}$ (in particular, this means that $\mathcal{P}$ never blocks itself after several invalid ciphertexts).

Theorem 8 (Session-Key Privacy). *The above TKEM achieves SK-IND security under the DDH assumption in* $\mathbb{G}$*:* $\mathsf{Adv}^{\mathit{sk\text{-}ind}}_{\mathit{TKEM}}(\tau) \leq \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}}(\tau)$.

Proof. From a Diffie-Hellman tuple (A, B, C), one can derive, for random $s_k \xleftarrow{\$} \mathbb{Z}_q$, for $k = 1 \ldots, t$

$$h_k \leftarrow g^{s_k} \qquad h \leftarrow B \qquad C_k \leftarrow A^{s_k} \qquad K \leftarrow C$$

where s, r are implicitly defined as $A = g^r$ and $B = g^s$. If C is indeed the Diffie-Hellman value for (g, A, B), then $K = C = g^{sr} = h^r$, we are in the real case $(b = 0)$. If C is a random value, we are in the random case $(b = 1)$:

$$\mathsf{Adv}^{\mathsf{sk\text{-}ind}}_{\mathsf{TKEM}}(\mathcal{A}) \leq \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}}(\tau).$$

Theorem 9 (Public-Key Privacy). *The above TKEM achieves PK-IND security under the DDH assumption in $\mathbb{G}$: $\mathsf{Adv}^{\textsf{pk-ind}}_{\textsf{TKEM}}(\tau) \leq \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}}(\tau)$.*

Proof. From a Diffie-Hellman tuple (A, B, C), one can derive, for random scalars $z_k^{(0)}, z_k^{(1)}, s_k^{(0)}, s_k^{(1)}, z^{(0)}, z^{(1)}, s^{(0)}, s^{(1)} \xleftarrow{\$} \mathbb{Z}_q$, for $k = 1 \ldots, t$

$$\begin{aligned} h_k^{(0)} &\leftarrow A^{z_k^{(0)}} \cdot g^{s_k^{(0)}} & h^{(0)} &\leftarrow A^{z^{(0)}} \cdot g^{s^{(0)}} \\ h_k^{(1)} &\leftarrow A^{z_k^{(1)}} \cdot g^{s_k^{(1)}} & h^{(1)} &\leftarrow A^{z^{(1)}} \cdot g^{s^{(1)}} \\ C_k &\leftarrow C^{z_k^{(b)}} \cdot B^{s_k^{(b)}} \end{aligned}$$

where r is implicitly defined as $B = g^r$. If C is indeed the Diffie-Hellman value for (g, A, B), then $C_k = A^{rz_k^{(b)}} \cdot g^{rs_k^{(b)}} = (A^{z_k^{(b)}} \cdot g^{s_k^{(b)}})^r = (h_k^{(b)})^r$. If C is a random value $C = A^{r+c}$, for a random $c \xleftarrow{\$} \mathbb{Z}_q$:

$$C_k = A^{(r+c)z_k^{(b)}} \cdot g^{rs_k^{(b)}} = (A^{z_k^{(b)}} \cdot g^{s_k^{(b)}})^r \cdot A^{cz_k^{(b)}} = (h_k^{(b)})^r \cdot A^{cz_k^{(b)}}.$$

As $z_k^{(b)}$ is perfectly hidden in the public key, C_k follows a uniform distribution in $\mathbb{G}$, independently of the public key, and thus of b:

$$\mathsf{Adv}^{\mathsf{pk\text{-}ind}}_{\mathsf{TKEM}}(\mathcal{A}) \leq \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}}(\tau).$$

Theorem 10 (Confirmation). *A collusion of less than t keys can be confirmed from a useful stateless pirate decoder $\mathcal{P}$: starting from a correct guess for $\mathcal{T}$, the traitors' keys used for building the pirate decoder $\mathcal{P}$, by accessing the decoder, one can confirm a traitor in $\mathcal{T}$, with negligible error.*

Proof. To prove this theorem, we first give a description of the confirmer algorithm $\mathcal{C}$, then we provide so indistinstinguishability analysis, and eventually prove $\mathcal{C}$ will give a correct answer.

Description of the Confirmer $\mathcal{C}$: The confirmer algorithm $\mathcal{C}$ can proceed as follows, for a candidate subset $\mathcal{G}$: $\{\mathsf{usk}_j = (v_{j,k})_k\}_{j \in \mathcal{G}}$, for $\mathcal{G}$ of size less than t: it chooses $(u_k)_k$ orthogonal to the subvector-space spanned by $\{(v_{j,k})_k\}_{j \in \mathcal{G}}$, which means that

$$\sum_k u_k v_{j,k} = 0, \forall j \in \mathcal{G}.$$

This is possible as $(v_{j,k})_{k \in [1,t], j \in \mathcal{G}}$ is of rang at most $t-1$ in $\mathbb{Z}_q^t$. Then the kernel is of dimension at least 1. One generates a fake ciphertext $C = (C_k)_k$, with $C_k \leftarrow h_k^r \cdot g^{u_k s'}$, for random $r, s' \xleftarrow{\$} \mathbb{Z}_q$, and then $K \leftarrow h^r$:

- Any key usk_j in $\mathcal{G}$ will lead to

$$\prod_k C_k^{v_{j,k}} = \prod_k g^{(rs_k + s'u_k) \cdot v_{j,k}} = g^{r \sum_k s_k v_{j,k} + s' \sum_k u_k v_{j,k}} = g^{rs + s' \times 0} = g^{rs} = K.$$

– Similarly, any key usk_j outside $\mathcal{G}$ will lead to

$$\prod_k C_k^{v_{j,k}} = K \times (g^{\sum_k u_k v_{j,k}})^{s'} \neq K.$$

we will show this allows to confirm at least one traitor from a candidat subset of traitors.

Indistinguishability Analysis. The above remark about the output key from a pirate decoder $\mathcal{P}$ assumes an honest behavior, whereas it can stop answering if it detects the fake ciphertext. We first need to show that, with the public key $\mathsf{pk} = ((h_k)_k, h)$ and only $\{\mathsf{usk}_j = (v_{j,k})_k\}_{j \in \mathcal{G}}$, one cannot distinguish the fake ciphertext from a real ciphertext, generated as above: from a Diffie-Hellman tuple $(A = g^a, B = g^r, C)$, one can derive, from random scalars $s, s'_k, u_k \overset{\$}{\leftarrow} \mathbb{Z}_q$, such that $\sum_k v_{j,k} s'_k = s$ and $\sum_k v_{j,k} u_k = 0$, for $j = 1 \ldots, n$:

$$h_k \leftarrow A^{u_k} \cdot g^{s'_k} = g^{a u_k + s'_k} \qquad h \leftarrow g^s \qquad \mathsf{usk}_j = (v_{j,k})_k \text{ for } j \in \mathcal{G}$$

where we implicitly define $s_k \leftarrow a u_k + s'_k$, that satisfy

$$\sum_k v_{j,k} s_k = \sum_k v_{j,k}(s'_k + a u_k) = \sum_k v_{j,k} s'_k + a \sum_k v_{j,k} u_k = s + 0 = s.$$

Then, one defines

$$C_k \leftarrow C^{u_k} \cdot B^{s'_k} \qquad\qquad K \leftarrow B^s$$

Let us note $C = g^{r-c}$, where c is either 0 (a Diffie-Hellman tuple) or random:

$$C_k = A^{(r+c)u_k} \cdot g^{r s'_k} = (A^{u_k} \cdot g^{s'_k})^r \cdot A^{c u_k} = h_k^r \cdot (A^c)^{u_k}.$$

One can remark that: when $c = 0$ (Diffie-Hellman tuple), $C = (C_k)_k$ is a normal ciphertext; when $c = s'$ (random tuple), this is a fake ciphertext. Under the DDH assumption, they are thus indistinguishable for an adversary knowing the keys $(\mathsf{usk}_i)_{i \in \mathcal{G}}$.

Confirmation of a Traitor. The above analysis shows that a pirate decoder $\mathcal{P}$ built from $(\mathsf{usk}_i)_{i \in \mathcal{G}}$ cannot distinguish the fake ciphertext from a real ciphertext. A useful pirate decoder should necessarily distinguish real key from random key. Then, several situations may appear, according to the actual set $\mathcal{T}$ of traitors' keys used to build the pirate decoder $\mathcal{P}$ by the adversary $\mathcal{A}$:

- If $\mathcal{T} \subseteq \mathcal{G}$, a useful decoder $\mathcal{P}$ can distinguish keys;
- If $\mathcal{T} \cap \mathcal{G} = \emptyset$, $\mathcal{P}$ cannot distinguish keys, as it can get several candidates, independent from the real or random keys.

Let us now assume we started from $\mathcal{G} \supseteq \mathcal{T}$, then the advantage of $\mathcal{P}$ in distinguishing real and random keys, denoted $p_{\mathcal{G}}$, is non-negligible, from the usefulness of the decoder. The following steps would also work if one starts with $\mathcal{G} \cap \mathcal{T} \neq \emptyset$, so that the advantage $p_{\mathcal{G}}$ is significant.

One then removes a user J from $\mathcal{G}$ to generate $\mathcal{G}'$ and new ciphertexts to evaluate $p_{\mathcal{G}'}$: if $J \notin \mathcal{T}$, usk_J is not known to the adversary, and so there is no way to check whether $\sum_k v_{J,k} s'_k = s$ and $\sum_k v_{J,k} u_k = 0$, even for a powerful adversary. So necessarily, $p_{\mathcal{G}'} = p_{\mathcal{G}}$.

On the other hand, we know that $p_{\emptyset} = 0$. So, one can sequentially remove users until a significant gap appears: this is necessarily for a user in $\mathcal{T}$.

Corrolary 1 *In the particular case of $t = 2$, one can efficiently trace one traitor, from a useful stateless pirate decoder: by trying $\mathcal{G} = \{J\}$ sequentially for each $J = 1, \ldots, n$, and evaluating $p_{\mathcal{G}}$, one should get either a significant advantage (for the traitor) or* 0 *(for honest keys).*

7 Our KEMAC Scheme

We have already presented a traceable KEM that is secure against classical adversaries. If we combine it with another scheme expected secure against quantum adversaries, we can thereafter combine them into an hybrid-KEM, that inherits security properties from both schemes, with still traceability against classical adversaries.

7.1 CRYSTALS-Kyber KEM

We recall the algorithms of CRYSTALS-Kyber key encapsulation mechanism from [ABD+21] whose both semantic security and anonymity rely on the hardness of Module-LWE [LS15]. We identify R_q with $\mathbb{Z}_q^n$ that contains $\{0,1\}^n$, and use two noise parameters $\eta_1 \geq \eta_2$, for the Gaussian distributions B_{η_1} and B_{η_2}:

- $\mathsf{Kyber.KeyGen}(1^\kappa)$: sample random $\mathbf{A} \overset{\$}{\leftarrow} \mathsf{R}_q^{k \times k}$ and $(\mathbf{s}, \mathbf{e}) \overset{\$}{\leftarrow} B_{\eta_1}^k \times B_{\eta_1}^k$, then set $\mathsf{pk} \leftarrow (\mathbf{A}, \mathbf{b} = \mathbf{A}\mathbf{s} + \mathbf{e})$ and $\mathsf{sk} \leftarrow \mathbf{s}$.
- $\mathsf{Kyber.Enc}(\mathsf{pk})$: sample $K \overset{\$}{\leftarrow} \{0,1\}^n \subset \mathsf{R}_q$, $\mathbf{r} \overset{\$}{\leftarrow} B_{\eta_1}^k$, and $(\mathbf{e}_1, e_2) \overset{\$}{\leftarrow} B_{\eta_2}^k \times B_{\eta_2}$, then set $\mathbf{u} = \mathbf{A}^T \mathbf{r} + \mathbf{e}_1$ and $v = \mathbf{b}^T \mathbf{r} + e_2 + \lceil \frac{q}{2} \rfloor \cdot K$, and return $(K, C = (\mathbf{u}, v))$.
- $\mathsf{Kyber.Dec}(\mathsf{sk}, C)$: compute $w \leftarrow v - \mathbf{s}^T \mathbf{u}$ and output $K = \lceil \frac{2}{q} \cdot w \rfloor$.

Correctness. Before proving the security properties, we first show that decapsulation indeed gives back K. To this aim, one can note that we have

$$\begin{aligned} w = v - \mathbf{s}^T \mathbf{u} &= (\mathbf{b}^T \mathbf{r} + e_2 + \lceil \frac{q}{2} \rfloor \cdot K) - \mathbf{s}^T \cdot (\mathbf{A}^T \mathbf{r} + \mathbf{e}_1) \\ &= (\mathbf{s}^T \mathbf{A}^T + \mathbf{e}^T) \cdot \mathbf{r} + e_2 + \lceil \frac{q}{2} \rfloor \cdot K - \mathbf{s}^T \cdot (\mathbf{A}^T \mathbf{r} + \mathbf{e}_1) \\ &= \mathbf{s}^T \mathbf{A}^T \mathbf{r} + \mathbf{e}^T \mathbf{r} + e_2 + \lceil \frac{q}{2} \rfloor \cdot K - \mathbf{s}^T \mathbf{A}^T \mathbf{r} + \mathbf{s}^T \mathbf{e}_1 \\ &= \mathbf{e}^T \mathbf{r} + e_2 + \lceil \frac{q}{2} \rfloor \cdot K + \mathbf{s}^T \mathbf{e}_1 = \lceil \frac{q}{2} \rfloor \cdot K + z \end{aligned}$$

with $z = \mathbf{e}^T\mathbf{r} + \mathbf{s}^T\mathbf{e}_1 + e_2$. Then,

$$\lceil \frac{2}{q} \cdot w \rfloor = \lceil \frac{2}{q} \cdot \Big(\lceil \frac{q}{2} \rfloor \cdot K + z \Big) \rfloor.$$

As $K \in \{0,1\}^b$, if $\|z\|_\infty < q/4$, then $\lceil \frac{2}{q} \cdot w \rfloor = K$. And since $\mathbf{e}, \mathbf{r}, \mathbf{s}$ are drawn according to B_{η_1}, and $\mathbf{e}_1, e_2$ according to B_{η_2}, with appropriate n, k, q, η_1, η_2, $\|z\|_\infty < q/4$, with overwhelming probability [ABD$^+$21], even with the additional compression function, that introduces an additional noise. And we omit it in the current description, as it is for efficiency purpose but does not impact the security notions we are interested in.

Security Properties. Let us now prove that Kyber satisfies both $\mathsf{SK\text{-}IND}$ and $\mathsf{PK\text{-}IND}$ security notions.

Theorem 11 (Session-Key Privacy of Kyber.). Kyber *is* $\mathsf{SK\text{-}IND}$*-secure under the decisional Module-LWE assumption:*

$$\mathsf{Adv}^{\textsf{sk-ind}}_{\mathsf{Kyber}}(\tau) \leq \mathsf{Adv}^{\mathsf{dmlwe}}_{\mathsf{R}_q,k,k,\eta_1}(\tau) + \mathsf{Adv}^{\mathsf{dmlwe}}_{\mathsf{R}_q,k+1,k,\eta_2}(\tau) \leq 2 \times \mathsf{Adv}^{\mathsf{dmlwe}}_{\mathsf{R}_q,k+1,k,\eta_2}(\tau).$$

Proof. We proceed with a sequence of games, so that the final game does not leak any information about the bit b used by the challenger.

Game $\mathbf{G}_0$: The initial game is the security experiment, where a public key $\mathsf{pk} = (\mathbf{A}, \mathbf{b})$ is generated, a ciphertext C is generated according to K_0, and another random K_1 is generated and eventually the adversary receives (pk, C, K_b).

Game $\mathbf{G}_1$: We change the way the challenger generates the public key pk, with randomly chosen $\mathbf{b} \overset{\$}{\leftarrow} \mathsf{R}_q$. The distance of the distributions of the outputs of the adversary is then bounded by $\mathsf{Adv}^{\mathsf{dmlwe}}_{\mathsf{R}_q,k,k,\eta_1}(\tau)$. But still, the ciphertext is built as

$$\begin{bmatrix} \mathbf{u} \\ v \end{bmatrix} = \begin{bmatrix} \mathbf{A} | \mathbf{b} \end{bmatrix}^T \times \mathbf{r} + \begin{bmatrix} \mathbf{e}_1 \\ e_2 \end{bmatrix} + \begin{bmatrix} 0 \\ \lceil \frac{q}{2} \rfloor \cdot K_0 \end{bmatrix}.$$

Game $\mathbf{G}_2$: We change now the simulation of the ciphertext, with $(\mathbf{w}, z) \overset{\$}{\leftarrow} \mathsf{R}_q^k \times \mathsf{R}_q = \mathsf{R}_q^{k+1}$, and

$$\begin{bmatrix} \mathbf{u} \\ v \end{bmatrix} \leftarrow \begin{bmatrix} \mathbf{w} \\ z \end{bmatrix} + \begin{bmatrix} 0 \\ \lceil \frac{q}{2} \rfloor \cdot K_0 \end{bmatrix}.$$

The distance of the distributions of the outputs of the adversary is then bounded by $\mathsf{Adv}^{\mathsf{dmlwe}}_{\mathsf{R}_q,k+1,k,\eta_2}(\tau)$, as $\mathbf{r} \overset{\$}{\leftarrow} B^k_{\eta_1}$ and $(\mathbf{e}_1, e_2) \overset{\$}{\leftarrow} B^k_{\eta_2} \times B_{\eta_2} = B^{k+1}_{\eta_2}$, and $\eta_1 \geq \eta_2$.

Game $\mathbf{G}_3$: We eventually replace C by a random sampling in $\mathsf{R}_q^k \times \mathsf{R}_q$, which is perfectly indistinguishable. And then, the ciphertext is perfectly independent from K_0.

Theorem 12 (Public-Key Privacy of Kyber.). Kyber *is* $\mathsf{PK\text{-}IND}$*-secure under the decisional Module-LWE assumption:*

$$\mathsf{Adv}^{\textsf{pk-ind}}_{\mathsf{Kyber}}(\tau) \leq 2 \times \mathsf{Adv}^{\mathsf{dmlwe}}_{\mathsf{R}_q,k,k,\eta_1}(\tau) + \mathsf{Adv}^{\mathsf{dmlwe}}_{\mathsf{R}_q,k+1,k,\eta_2}(\tau) \leq 3 \times \mathsf{Adv}^{\mathsf{dmlwe}}_{\mathsf{R}_q,k+1,k,\eta_2}(\tau).$$

Proof. We will show that the ciphertexts are statistically close to uniform in the ciphertext space, i.e. in $\mathsf{R}_q^k \times \mathsf{R}_q$n whatever the public key.

Game $\mathbf{G}_0$: The initial game is the security experiment, where two public keys $\mathsf{pk}_0 = (\mathbf{A}_0, \mathbf{b}_0)$ and $\mathsf{pk}_1 = (\mathbf{A}_1, \mathbf{b}_1)$ are generated and a ciphertext C is generated according to pk_b.

Game $\mathbf{G}_1$: We change the way the challenger generates the public keys pk_0 and pk_1, with randomly chosen $\mathbf{b}_0, \mathbf{b}_1 \overset{\$}{\leftarrow} \mathsf{R}_q$. The distance of the distributions of the outputs of the adversary is then bounded by $2 \times \mathsf{Adv}^{\mathsf{dmlwe}}_{\mathsf{R}_q,k,k,\eta_1}(\tau)$. But still, the ciphertext is built as

$$\begin{bmatrix} \mathbf{u} \\ v \end{bmatrix} = \begin{bmatrix} \mathbf{A}_b | \mathbf{b}_b \end{bmatrix}^T \times \mathbf{r} + \begin{bmatrix} \mathbf{e}_1 \\ e_2 \end{bmatrix} + \begin{bmatrix} 0 \\ \lceil \frac{q}{2} \rfloor \cdot K \end{bmatrix}.$$

Game $\mathbf{G}_2$: We change now the simulation of the ciphertext, with $(\mathbf{w}, z) \overset{\$}{\leftarrow} \mathsf{R}_q^k \times \mathsf{R}_q = \mathsf{R}_q^{k+1}$, and

$$\begin{bmatrix} \mathbf{u} \\ v \end{bmatrix} \leftarrow \begin{bmatrix} \mathbf{w} \\ z \end{bmatrix} + \begin{bmatrix} 0 \\ \lceil \frac{q}{2} \rfloor \cdot K \end{bmatrix}.$$

The distance of the distributions of the outputs of the adversary is then bounded by $\mathsf{Adv}^{\mathsf{dmlwe}}_{\mathsf{R}_q,k+1,k,\eta_2}(\tau)$, as $\mathbf{r} \overset{\$}{\leftarrow} B_{\eta_1}^k$ and $(\mathbf{e}_1, e_2) \overset{\$}{\leftarrow} B_{\eta_2}^k \times B_{\eta_2} = B_{\eta_2}^{k+1}$, and $\eta_1 \geq \eta_2$.

Game $\mathbf{G}_3$: We eventually replace C by a random sampling in $\mathsf{R}_q^k \times \mathsf{R}_q$, which is perfectly indistinguishable. And then, the ciphertext is perfectly independent from the used public key.

7.2 Hybrid KEM

Using the ElGamal KEM that is SK-IND-secure under the DDH assumption, and unconditionally PK-IND-secure, together with the Kyber KEM that is both SK-IND and PK-IND-secure under the DMLWE assumption, the hybrid KEM is

- SK-IND-secure, as soon as either the DDH or the DMLWE assumptions hold;
- PK-IND-secure, under the DMLWE assumption.

It can be described as follows in a group $\mathbb{G}$ of prime order p, with generator g, and in the ring R_q, with a hash function $\mathcal{H}$ into $\{0,1\}^n$. Then, we can improve on the session key:

- $\mathsf{Hyb.KeyGen}(1^\kappa)$: sample a random scalar $x \overset{\$}{\leftarrow} \mathbb{Z}_p$, a random matrix $\mathbf{A} \overset{\$}{\leftarrow} \mathsf{R}_q^{k \times k}$ and $(\mathbf{s}, \mathbf{e}) \overset{\$}{\leftarrow} B_{\eta_1}^k \times B_{\eta_1}^k$, then set $\mathsf{pk} \leftarrow (h = g^x, \mathbf{A}, \mathbf{b} = \mathbf{A}\mathbf{s} + \mathbf{e})$ and $\mathsf{sk} \leftarrow (x, \mathbf{s})$.
- $\mathsf{Hyb.Enc}(\mathsf{pk})$: sample a random scalar $r \overset{\$}{\leftarrow} \mathbb{Z}_p$, a random key $K' \overset{\$}{\leftarrow} \{0,1\}^n$, and random noise vectors $\mathbf{r} \overset{\$}{\leftarrow} B_{\eta_1}^k$, and $(\mathbf{e}_1, e_2) \overset{\$}{\leftarrow} B_{\eta_2}^k \times B_{\eta_2}$, then set $K \leftarrow \mathcal{H}(h^r) \oplus K'$, $c \leftarrow g^r$, $\mathbf{u} \leftarrow \mathbf{A}^T\mathbf{r} + \mathbf{e}_1$ and $v \leftarrow \mathbf{b}^T\mathbf{r} + e_2 + \lceil \frac{q}{2} \rfloor \cdot K'$, and return $(K, C = (c, \mathbf{u}, v))$.
- $\mathsf{Hyb.Dec}(\mathsf{sk}, C)$: compute $w \leftarrow v - \mathbf{s}^T\mathbf{u}$ and $K' = \lceil \frac{2}{q} \cdot w \rfloor$, and then output $K = K' \oplus \mathcal{H}(c^x)$.

7.3 Hybrid Traceable KEMAC

We can also applies the generic combination to build an anonymous subset-cover KEMAC with early abort, with the traceable ElGamal KEM and Kyber to get a Key Encapsulation Mechanism with Access Control and Black-Box traceability (without collusion), where all the privacy notions (message-privacy and target set privacy) hold as soon as at least the DDH or the DMLWE assumptions hold, while traceability works under the DDH assumption: **david:** To be concluded

- $\mathsf{Setup}^{\mathtt{hybrid}}(\mathcal{S} = (S_i)_i)$:
  run $\mathsf{Setup}(\mathcal{S} = (S_i)_i)$ from Section **??**. For each set S_i, invoke $\mathsf{Kyber.cpa.KeyGen}(1^k)$ which outputs an $(\mathsf{pk}_i^{\mathsf{pq}}, \mathsf{sk}_i^{\mathsf{pq}})$.
  Let $\mathsf{MSK} \leftarrow (u, v, s, (x_i, \mathsf{sk}_i^{\mathsf{pq}})_i)$ and $\mathsf{mpk} \leftarrow (\mathbb{G}, g, U, V, H, (H_i, \mathsf{pk}_i^{\mathsf{pq}})_i)$.
- $\mathsf{KeyGen}^{\mathtt{hybrid}}(\mathsf{MSK}, j, A_j)$: same as in Section **??**. And we assume that mpk is known to everybody.
- $\mathsf{Enc}^{\mathtt{hybrid}}(K, B)$: it takes as input a bitstring $K \in \{0,1\}^n$ to encrypt by invoking $\mathsf{Enc}(K, B)$. Let $E = (C, D, (E_i = \mathcal{H}(K_i) \oplus K)_{i \in B})$ be the encapasulation output by the pre-quantum scheme from Section **??**; and

  1. Concatenation: for each $i \in B$, it invokes $\mathsf{Kyber.cpa.Encrypt}(K)$ which gives $(K_i^{\mathsf{pq}}, E_i^{\mathsf{pq}})$. The ciphertext consists of:
  $$(C, D, (E_i^{\mathsf{pq}}, E_i = \mathcal{H}(K_i) \oplus \mathcal{H}(K_i^{\mathsf{pq}}) \oplus K)_{i \in B})$$
  2. Optimized version: for each $i \in B$, it invokes $\mathsf{Kyber.cpa.Encrypt}(E_i)$ which gives $(K_i^{\mathsf{pq}}, E_i^{\mathsf{pq}})_{i \in B}$ and then it returns:
  $$(C, D, E_i^{\mathsf{pq}})_{i \in B}$$

- $\mathsf{Dec}()$:

  1. Concatenation: $\mathsf{Dec}(\mathsf{usk}_j, (C, D, (E_i^{\mathsf{pq}}, E_i^{\mathsf{pq}'} = \mathcal{H}(K'_i) \oplus \mathcal{H}(K_i) \oplus K)_{i \in B}$: it takes as input a user's secret key and a ciphertext, it outputs the encrypted key K.
     - the user first chooses an index $i \in B \cap A_j$, in both its set of rights A_j and the rights B of the ciphertext, and then uses $x_i = \mathsf{sk}_i \in \mathsf{usk}_j$;
     - it can compute $K_i = (C^{a_j} D^{b_j})^{x_i}$;
     - it retrieves K'_i as $\mathsf{Dec}(\mathsf{sk}_i^{\mathsf{pq}}, E_i^{\mathsf{pq}})$;
     - and it extracts $K = E_i^{\mathsf{pq}'} \oplus \mathcal{H}(K_i) \oplus \mathcal{H}(K'_i)$.
  2. Optimized version: $\mathsf{Dec}(\mathsf{usk}_j, (C, D, (E_i^{\mathsf{pq}})_{i \in B}$: it takes as input a user's secret key and a ciphertext, it outputs the encrypted key K.
     - the user first chooses an index $i \in B \cap A_j$, in both its set of rights A_j and the rights B of the ciphertext, and then uses $x_i = \mathsf{sk}_i, \mathsf{sk}_i^{\mathsf{pq}} \in \mathsf{usk}_j$;
     - it can compute $K'_i = (C^{a_j} D^{b_j})^{x_i}$;
     - it can compute $K_i^{\mathsf{pq}} = \mathsf{Kyber.cpa.Decrypt}(\mathsf{sk}_i^{\mathsf{pq}}, E_i^{\mathsf{pq}})$;
     - and it extracts $K = K_i^{\mathsf{pq}} \oplus K'_i$.

8 Implementation

We propose an implementation [**??**] of the hybrid CoverCrypt (hybrid anonymous Subset-Cover KEM-AC with Early-Abort) scheme with optimization for a security of 128 bits. We use Kyber768 and an El-Gammal-based KEM scheme built on the Curve25519. The hash algorithm used to generate the Early-Abort tag and hash the keys generated by the KEM to the given length is Shake256.

8.1 Data structures

The data structures used are given bellow, where Scalar and $\mathsf{GroupElement}$ are the scalar and group element types for the El-Gammal based KEM, $\mathsf{ByteArray}(n)$ is an array of n bytes and t is the length of the EAKEM tag.

$$\mathsf{MasterPublicKey} : \begin{cases} U : \mathsf{GroupElement} \\ V : \mathsf{GroupElement} \\ H : \mathsf{HashMap}\left\langle i, (\mathsf{PublicKey}^{\mathsf{pq}}, \mathsf{GroupElement})\right\rangle \end{cases}$$

$$\mathsf{MasterSecretKey} : \begin{cases} u : \mathsf{Scalar} \\ v : \mathsf{Scalar} \\ s : \mathsf{Scalar} \\ x : \mathsf{HashMap}\left\langle i, (\mathsf{SecretKey}^{\mathsf{pq}}, \mathsf{Scalar})\right\rangle \end{cases}$$

$$\mathsf{UserSecretKey} : \begin{cases} a : \mathsf{Scalar} \\ b : \mathsf{Scalar} \\ x : \mathsf{HashSet}\left\langle (\mathsf{SecretKey}^{\mathsf{pq}}, \mathsf{Scalar})\right\rangle \end{cases}$$

$$\mathsf{Ciphertext} : \begin{cases} C : \mathsf{GroupElement} \\ D : \mathsf{GroupElement} \\ T : \mathsf{ByteArray}(t) \\ E : \mathsf{HashSet}\left\langle \mathsf{ByteArray}(\mathtt{sizeof}(\mathsf{PublicKey}^{\mathsf{pq}}))\right\rangle \end{cases}$$

Since the length of the hashmaps and hashsets used is not fixed, we need to write their length in the serialization of the data structures defined above. In order to optimize this serialization, we write this length using the LEB128 format [**??**]. We define the operator `leb128_sizeof` as below:

$$\mathtt{leb128\_sizeof} \begin{cases} \mathbb{N} \to \mathbb{N} \\ n \mapsto \text{Size of the LEB128 serialization of } n \end{cases}$$

Hence, the size of a serialized ciphertext is:

$$2 \times \mathtt{sizeof}(\mathsf{GroupElement}) + t + \mathtt{leb128\_sizeof}(|B|) \times \mathtt{sizeof}(\mathsf{PublicKey}^{\mathsf{pq}})$$

where $|B|$ is the length of the target set used for encapsulation and thus the length of the encapsulation's hashset. The size of the other serializations can be computed in a similar way.

8.2 Algorithms

The 4 algorithms used in our implementation are given in Appendice B.

8.3 Benchmarks

The following benchmarks are performed on an Intel(R) Core(TM) i7-10750H CPU @ 3.20GHz.

Encapsulation time The table 8.3 gives the time required to generate a CoverCrypt encapsulation for a 32-bytes symmetric key depending on the size of the target set B.

Size of B	1	2	3	4	5
Encapsulation time (μs)	239	358	482	606	728

Table 1. Encapsulation time (in μs) given the size of the target set

Decapsulation The table 8.3 gives the time required to decapsulate a CoverCrypt encapsulation for a 32-bytes symmetric key depending on the size of the target set B.

Size of B	1	2	3	4	5
Decapsulation time (μs)	197	235	273	311	350

Table 2. Decapsulation time (in μs) given the size of the target set

9 Additional Features

9.1 Updates

Adding users and rights. When a new user joins the system, he will receive secret keys associated to its rights from the master secret key. When a new right corresponding to a new set S_i is added to the system, a new x_i is sampled at random. This x_i is added to MSK and $H_i \leftarrow H^{x_i}$ is added to mpk. Users cannot decrypt ciphertexts computed using the new policies as long as their secret key has not been updated (and if they have the corresponding right).

In particular, rotations can be implemented for a given set S_i by replacing the corresponding x_i and H_i in $(\mathsf{MSK}, \mathsf{mpk})$ by freshly generated ones. Another way of seeing it to add a new set S_i in place of the old one. Users can then ask the master authority for a key update in order to get the new x_i. Users can also keep the old x_i in order to be able to decrypt old ciphertexts.

Rotations allow producing ciphertexts that cannot be decrypted by old user secret keys without modifying the global partition. The following section presents a way to implement the same functionality by adding one dimension to the partition space.

Revocation. For revocation, one can use time-periods, and then a three-dimensional space, with $\mathsf{sk}_{t,i}$. When the time-period changes, users receives the keys $\mathsf{sk}_{t,i}$ associated to their right. The tracing part (a_j, b_j) does not need to be updated.

On the other hand, stored data does not need to be re-encrypted, but the key K must be re-encrypted under the new $\mathsf{sk}_{t,i}$. It can be done without any private information: but we need a slight modification with $\mathsf{Enc}(K, B)$ that now takes as input a key $K \in \mathbb{G}$, while $H_{t,i} = H^{x_{t,i}}$ depends on the time-period t

- it samples a random $r \overset{\$}{\leftarrow} \mathbb{Z}_q$;
- it sets $C \leftarrow U^r$ and $D \leftarrow V^r$;
- for every $i \in B$, it generates $K_{t,i} \leftarrow H^r_{t,i}$ and $E_{t,i} = K \times K_{t,i}$

The ciphertext of K is thus $(C, D, (E_{t,i})_i)$, and data is encrypted with the session key $\mathcal{H}(K) \in \{0,1\}^n$.

When updating the keys: $H_{t+1,i} \leftarrow H_{t,i} \cdot U^{\Delta_{t,i}} = H^{x_{t,i}+\Delta_{t,i}\cdot u/s}$. The ciphertext should be updated so that

$$E_{t+1,i} = K \times H^r_{t+1,i} = K \times H^r_{t,i} \cdot U^{r\cdot\Delta_{t,i}} = E_{t,i} \cdot C^{\Delta_{t,i}}$$

References

[ABD+21] Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. Crystals-kyber algorithm specifications and supporting documentation. `https://pq-crystals.org/kyber/resources.shtml`, 2021.

[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, Heidelberg, August 1999.

[LS15] Adeline Langlois and Damien Stehlé. Worst-case to average-case reductions for module lattices. *Des. Codes Cryptogr.*, 75(3):565–599, 2015.

[Sho01] Victor Shoup. A proposal for an iso standard for public key encryption. `https://shoup.net/papers/iso-2_1.pdf`, 2001.

Appendix

A CRYSTALS-Kyber

We recall the algorithms of CRYSTALS-Kyber public key encryption scheme from [ABD$^+$21] whose semantic security relies on the hardness of Module-LWE [].

We denote by $\mathsf{R} = \mathbb{Z}[X]/X^n + 1$ the ring of polynomials of degree at most $n-1$ with integer coefficients and R_q the ring of polynomials of degree at most $n-1$ with coefficients in $\mathbb{Z}_q$. We take n as power of 2 and X^n+1 is the $\frac{n}{2}$-th cyclotomic polynomial. The modulus q is chosen so that multiplications in R_q can be performed efficiently. Given a polynomial $f = \sum_{i=0}^{n-1} f_i X^i \in \mathsf{R}_q$, the NTT$(\cdot)$ of f is given by a vector of linear polynomials (in the case of Kyber, 128 polynomials of degree 1) represented in the NTT domain by: $\hat{f} = \sum_{i=0}^{n-1} \hat{f}_i X^i$. The multiplication of $a \in \mathsf{R}_q$ and $b \in \mathsf{R}_q$ is computed as $\mathrm{NTT}^{-1}(\mathrm{NTT}(f) \circ \mathrm{NTT}(g))$, where $\circ$ is the usual multiplication. $\mathsf{Compress}_q(x,d)$ and $\mathsf{Decompress}_q(x,d)$ are defined such that $\mathsf{Decompress}_q(\mathsf{Compress}_q(x,d),d) \approx x$ where d is the amount of precision loss of the compression. When a polynomial in R_q is sampled according to the central binomial distribution (CBD), denoted B_η of parameter η, each of its coefficient is sampled from that distribution. We write CBD_η the function which allows to sample polynomial according to the CBD distribution of parameter from a pseudorandom bit string. $\mathsf{PRF}(\cdot)$ is implemented based on a hash function with input length todo: length PRF. The algorithm of Kyber CPA-PKE scheme will be by $\mathsf{Kyber.cpa.KeyGen}$, $\mathsf{Kyber.cpa.Encrypt}$, $\mathsf{Kyber.cpa.Decrypt}$.

```
$\mathsf{KeyGen}(1^\lambda) \to (\mathsf{pk}, \mathsf{sk})$
1. Generate $\hat{\mathbf{A}} \in \mathsf{R}_q^{k \times k}$ in NTT domain;
2. Sample $\mathbf{s} \in \mathsf{R}_q^k$ from $B_{\eta_1}$, $\hat{\mathbf{s}} = \mathrm{NTT}(\mathbf{s})$;
3. Sample $\mathbf{e} \in \mathsf{R}_q^k$ from $B_{\eta_1}$, $\hat{\mathbf{e}} = \mathrm{NTT}(\mathbf{e})$;
4. $\hat{\mathbf{t}} = \hat{\mathbf{A}} \circ \hat{\mathbf{s}} + \hat{\mathbf{e}}$;
5. $\mathsf{pk} = (\hat{\mathbf{t}}, \hat{\mathbf{A}})$ and $\mathsf{sk} = \hat{\mathbf{s}} \in \mathsf{R}_q^k$.
```

```
$\mathsf{Dec}(\mathsf{sk}, c) \to m$
1. $\mathbf{u} = \mathsf{Decompress_q}(c_1, d_u)$;
2. $v = \mathsf{Decompress_q}(c_2, d_v)$;
3. $\hat{z} = \hat{\mathbf{s}}^T \circ \mathrm{NTT}(\mathbf{u})$;
4. $w = v - \mathrm{NTT}^{-1}(\hat{\mathbf{s}}^T \circ \mathrm{NTT}(\mathbf{u}))$;
5. return $m = \mathsf{Compress}_q(w, 1)$.
```

```
$\mathsf{Enc}(\mathsf{pk}, m; \sigma) \to c$
1. Sample $\mathbf{r}$ according to $B_{\eta_1}$;
2. Sample $\mathbf{e}_1, e_2$ according to $B_{\eta_2}$;
3. $\hat{\mathbf{r}} = \mathrm{NTT}(\mathbf{r})$;
4. $\mathbf{u} = \mathrm{NTT}^{-1}(\hat{\mathbf{A}}^T \circ \hat{\mathbf{r}}) + \mathbf{e}_1$;
5. $v = \mathrm{NTT}^{-1}(\hat{\mathbf{t}^T} \circ \hat{\mathbf{r}}) + e_2 + \mathsf{Decomp}_q(m, 1)$;
6. $c_1 = \mathsf{Compress}_q(\mathbf{u}, d_u)$;
7. $c_2 = \mathsf{Compress}_q(\mathbf{v}, d_v)$;
8. return $c = (c_1, c_2)$.
```

Fig. 3. CRYSTALS Kyber IND-CPA PKE. In the encryption algorithm, the randomness σ is used as a seed for PRF computations whose output is used as input for all the following CBD sampling functions.

B CoverCrypt Algorithms

Algorithm 1

$\mathsf{Setup}(\Omega : \mathsf{HashSet}\,\langle\mathsf{int}\rangle) \rightarrow (\mathsf{MasterSecretKey}, \mathsf{MasterPublicKey})$

```
$(u, v, s) \xleftarrow{\$} \mathbb{Z}_q^3$
$S \leftarrow g^s$
$x \leftarrow \mathsf{HashMap} :: \mathsf{new}()$
$H \leftarrow \mathsf{HashMap} :: \mathsf{new}()$
for $i \in \Omega$ do
    $(\mathsf{sk}_i^{\mathsf{pq}}, \mathsf{pk}_i^{\mathsf{pq}}) \leftarrow \mathsf{CPAKyber.KeyGen}()$
    $x_i \xleftarrow{\$} \mathbb{Z}_q$
    $x.\mathsf{insert}(i, (\mathsf{sk}_i^{\mathsf{pq}}, x_i))$
    $H.\mathsf{insert}(i, (\mathsf{pk}_i^{\mathsf{pq}}, S^{x_i}))$
end for
$msk \leftarrow \{u, v, s, x\}$
$mpk \leftarrow \{g^u, g^v, H\}$

return $(msk, mpk)$
```

Algorithm 2

$\mathsf{KeyGen}(msk : \mathsf{MasterSecretKey}, A : \mathsf{HashSet}\,\langle\mathsf{int}\rangle) \rightarrow \mathsf{UserSecretKey}$

```
$a \xleftarrow{\$} \mathbb{Z}_q$
$b \leftarrow \left(msk.s \times a^{-msk.u}\right)^{-msk.v}$        ▷ such that $a^{msk.u} \times b^{msk.v} = msk.s$
$x \leftarrow \mathsf{HashMap} :: \mathsf{new}()$
for $i \in A$ do
    $v_i \leftarrow msk.x.\mathsf{get}(i)$
    if $v_i \neq nil$ then                                  ▷ this is a valide indice
        $x.\mathsf{insert}(v_i)$
    end if
end for

return $\{a, b, x\}$
```

Algorithm 3
$\mathsf{Enc}(mpk : \mathsf{MasterPublicKey}, B : \mathsf{HashSet}\,\langle\mathsf{int}\rangle) \rightarrow (\mathsf{ByteArray}(k),\ \mathsf{Ciphertext})$

```
$K \xleftarrow{\$} \mathcal{K}$

$r \xleftarrow{\$} \mathbb{Z}_q$
$(C, D) \leftarrow (mpk.U^r, mpk.V^r)$
$E \leftarrow \mathsf{HashSet} :: \mathsf{new}()$
for $i \in B$ do
    if $nil \neq msk.x.\mathsf{get}(i)$ then                    ▷ this is a valide indice
        $(\mathsf{pk}_i^{\mathsf{pq}}, H_i) \leftarrow msk.x.\mathsf{get}(i)$
        $E_i \leftarrow K \oplus \mathcal{H}(H_i^r)$
        $E.\mathsf{insert}(\mathsf{CPAKyber.Encrypt}(\mathsf{pk}_i^{\mathsf{pq}}, E_i))$
    end if
end for

$K1 \mid\mid K2 \leftarrow \mathcal{H}(K)$
$ct \leftarrow \{C, D, K1, E\}$

return $(K2,\ ct)$
```

Algorithm 4
$\mathsf{Dec}(usk : \mathsf{UserSecretKey}, ct : \mathsf{Ciphertext}) \rightarrow \mathsf{ByteArray}(k)$

```
$precomputation \leftarrow C^{usk.a} \cdot D^{usk.b}$
for $E_i^{\mathsf{pq}} \in ct.E$ do
    for $(\mathsf{sk}_j^{\mathsf{pq}}, x_j) \in usk.x$ do
        $E_j \leftarrow \mathsf{CPAKyber.Decrypt}(\mathsf{sk}_j^{\mathsf{pq}}, E_i^{\mathsf{pq}})$
        $K_j \leftarrow precomputation^{x_j}$
        $K' \leftarrow E_j \oplus K_j$
        $K1' \mid\mid K2' \leftarrow \mathcal{H}(K')$
        if K1' = ct.T then
            return $K2'$
        end if
    end for
end for
return Error "No access right"
```