Algorithm 1 `Kyber.CCAKEM.KeyGen()`

$z \xleftarrow{\$} \mathcal{B}^{32}$ ▷ Randomly sample 32 bytes (256 bits)
$(\mathtt{pk}, \mathtt{sk}') \xleftarrow{\$}$ `Kyber.CPAPKE.KeyGen()`
$\mathtt{sk} = (\mathtt{sk}', \mathtt{pk}, H(\mathtt{pk}), z)$ ▷ H is instantiated with SHA3-256
return $(\mathtt{pk}, \mathtt{sk})$

Algorithm 2 `Kyber.CCAKEM.Encap`$^+$`(pk)`

$m \xleftarrow{\$} \mathcal{B}^{32}$
$m = H(m)$ ▷ Do not output system RNG directly
$(\bar{K}, K_{\mathtt{MAC}}, r) = G(m \| H(\mathtt{pk}))$ ▷ G is instantiated with SHA3-512
$c' \leftarrow$ `Kyber.CPAPKE.Enc(pk, m, r)` ▷ Because r is set, `CPAPKE` is deterministic
$t_1 = \mathtt{MAC}(K_{\mathtt{MAC}}, c')$
$K = \mathtt{KDF}(\bar{K} \| t_1)$
$t_2 = \mathtt{MAC}(K, t_1)$ ▷ `KDF` is instantiated with Shake256
$c \leftarrow (c', t_1, t_2)$
return (c, K)

Algorithm 3 `Kyber.CCAKEM.Decap`$^+(\mathtt{sk}, c)$

Require: Secret key $\mathtt{sk} = (\mathtt{sk}', \mathtt{pk}, H(\mathtt{pk}), z)$
Require: Ciphertext $c = (c', t_1, t_2)$
$(\mathtt{sk}', \mathtt{pk}, h, z) \leftarrow \mathtt{sk}$ ▷ Unpack the secret key; h is the hash of `pk`
$(c', t_1, t_2) \leftarrow c$
$m' = \mathtt{Kyber.CPAPKE.Dec}(\mathtt{sk}', c')$
$(\overline{K}, K_{\mathtt{MAC}}, r) = G(m' \| h)$
$t_1' = \mathtt{MAC}(K_{\mathtt{MAC}}, c')$
if $t_1' = t_1$ **then**
  $K = \mathtt{KDF}(\bar{K} \| t_1)$
  $t_2' = \mathtt{MAC}(K, t_1)$
end if
if $t_2' = t_2$ **then**
  return K
else
  Abort
end if

Unfortunately, the `Kyber+` construction is not `IND-CCA2` secure when combined with Kyber's `CPAPKE`. This is because there exists an efficient plaintext-checking attack against `CPAPKE` that can recover the secret key, and an `IND-CCA2` adversary against `KYBER+` can use the decapsulation oracle to simulate the plaintext-checking oracle, which means that an `IND-CCA2` adversary can run the plaintext-checking attack as a subroutine and efficiently recover the secret key.

Let B denote the plaintext-checking attack routine. It is given the public key `pk` (since the public key is identical between `CPAPKE` and `CCAKEM` there is no need to disambiguate) and access to the plaintext-checking oracle `PCO` (see figure 1). After a number of plaintext checking queries, B will output the $\mathtt{sk}'$ that corresponds with the `pk` that it receives.

Algorithm 4 $\texttt{PCO}(m, c')$

return $[\![m = \texttt{Kyber.CPAPKE.Dec}(\texttt{sk}', c')]\!]$

Figure 1: Plaintext checking oracle

We now construct a chosen-ciphertext attack against `Kyber+`, which we will denote by A. A will use B as a sub-routine, which means that A needs to

1. give B a public key
2. service B's plaintext checking query

Because `Kyber+`'s public key is identical to `Kyber.CPAPKE`'s public key, requirement 1 is trivial: A just gives its own public key to B. To service B's plaintext checking query $(\tilde{m}, \tilde{c}')$, A will use the decapsulation oracle $\texttt{Decap}(c)$ (figure 2).

Algorithm 5 $\texttt{Decap}(c)$

return $\texttt{Kyber.CCAKEM.Decap}^{+}(\texttt{sk}, c)$

Figure 2: The decapsulation oracle

A can run through the steps of $\texttt{Kyber.CCAKEM.Encap}^{+}$ using $(\tilde{m}, \tilde{c}')$ and arrive at some (authenticated) ciphertext $c = (\tilde{c}', t_1, t_2)$ and session key K. A then sends the c to the decapsulation oracle, which might return some session key is c is valid, or abort if c is not valid. If $\tilde{m}$ is the decryption of $\tilde{c}'$, then A's output of t_1, t_2 will be correct, so the decapsulation oracle will accept. If $\tilde{m}$ is not the decryption of $\tilde{c}'$, then $(\hat{K}, K_{\texttt{MAC}}, r) \leftarrow G(\tilde{m} \| H(\texttt{pk}))$ does not produce the correct `MAC` key, which means that t_1, t_2 will be incorrect, and the decapsulation oracle will abort. See figure 3

Algorithm 6 $\texttt{PCO}_1(m, c')$

$(\hat{K}, K_{\texttt{MAC}}, r) \leftarrow G(m \| H(\texttt{pk}))$ ▷ Line 3 of algorithm 2
$t_1 \leftarrow \texttt{MAC}(K_{\texttt{MAC}}, c')$ ▷ Line 5
$K \leftarrow \texttt{KDF}(\hat{K} \| t_1)$ ▷ Line 6
$t_2 \leftarrow \texttt{MAC}(K, t_1)$ ▷ Line 7
$c \leftarrow (c', t_1, t_2)$ ▷ Line 8
return $[\![\texttt{Decap}(c) = K]\!]$

Figure 3: Simulated plaintext-checking oracle

Here are the complete steps of the chosen-ciphertext attack

1. Challenger runs `CCAKEM.KeyGen` and gets $(\texttt{pk}, \texttt{sk})$ where $\texttt{sk} = (\texttt{sk}', \texttt{pk}, H(\texttt{pk}), z)$
2. Challenger gives `pk` to A, A then gives `pk` to B
3. For each of B's plaintext checking query, A runs $\texttt{PCO}_1$ and returns the result back to B
4. After B halts, it returns the recovered secret key $\texttt{sk}'$
5. Challenger generates the challenge ciphertext $(c^*, K_0) \xleftarrow{\$} \texttt{Kyber.CCAKEM.Encap}^{+}(\texttt{pk})$, samples a random session key $K_1 \xleftarrow{\$} \{0,1\}^{256}$, flips a coin $b \xleftarrow{\$} \{0,1\}$, then gives (c^*, K_b) to A
6. A unpacks $c^* = (c', t_1, t_2)$, uses the recovered secret key $\texttt{sk}'$ to decrypt c'. From here it is trivial to distinguish whether K_b is the correct session key or the random session key