

Key Encapsulation Platform and Key Exchange Protocol

FIELD

5 [001] The present invention relates to a key encapsulation platform that uses a key derivation function, along with a plurality of existing key encapsulation mechanisms (KEM). This key derivation function could be a randomness extractor or randomness blender function.

BACKGROUND

10

[002] It is envisaged that quantum computing may threaten almost all widely adopted cryptography algorithms. Public key cryptography algorithms (e.g. RSA, Diffie-Hellman, DSA and ECC) relying on hardness of factorization problem or discrete log problem, may be broken by Shor's algorithm in polynomial time with powerful quantum computers. Symmetric cryptography algorithms may have their security level halved (e.g. AES128 provides at most 64 bits security, thus insecure, and AES256 provides at most 128 bits security against attackers with powerful quantum computers), due to Grover's algorithm.

15 [003] NIST (National Institute of Standards and Technology) competition of post quantum cryptography started in 2017, plans to select the winner PQC (post quantum cryptography) algorithms and publish the first draft standard of post quantum cryptography tentatively between 2022 and 2024. It is difficult to predict which candidate scheme will win the NIST competition of post quantum cryptography. As such, one strategy is to combine all of these candidate schemes, and make sure the combined scheme is secure if any one of the candidate schemes is secure.

20 [004] Combining n crypto primitives together in a naive way, will introduce n times overhead in complexity. There is thus a need to construct a robust combiner for key encapsulation mechanism (or key exchange protocol), aiming to optimize the (amortized) performance while preserving security.

SUMMARY OF THE INVENTION

30 [005] According to a first aspect of the present invention, there is provided a blender module configured to: receive a plurality of input messages, with at least one having uniform randomness; aggregate the received plurality of input messages to propagate the uniform randomness of the at least one input message with uniform randomness; and generate, in a randomness blender function mode: a plurality of output messages, each having uniform randomness, wherein total entropy of the plurality of output messages is less than or equal to total entropy of all input messages; or when at least one of the plurality of input messages lacks uniform randomness, a plurality of output messages, each with uniform randomness and dependency amongst all output messages where the uniform randomness of

the plurality of output messages has a total entropy less than or equal to the total entropy of all input messages; or generate, in a randomness extractor mode: an output message with uniform randomness.

[006] There is also disclosed a key encapsulation platform comprising: a plurality of key encapsulation modules with at least one being secure, each of the key encapsulation modules configured to generate a plaintext and ciphertext pair, with the ciphertext resulting from encapsulating the plaintext using a public key; and the blender module of the first aspect being in electrical communication with the plurality of key encapsulation modules to receive the plurality of plaintexts as its plurality of input messages, wherein the platform is configured to: compute at least one shared secret, from the plurality of output messages from the blender module, the plurality of output plaintexts being generated from the plurality of plaintexts; and compute an encapsulation from the plurality of ciphertexts, the encapsulation for transmission and from which the at least one shared secret is derived.

[007] According to a second aspect of the present invention, there is provided a key exchange module configured to: perform a first key exchange protocol with a terminal for communication so that the key exchange module and the terminal each have a first shared key; establish a first secure channel using the first shared key; perform a second key exchange protocol with the terminal over the first secure channel so that the key exchange module and the terminal each have a second shared key; and derive a session key from the first shared secret key and the second shared secret key, the session key being used to establish a second secure channel for payload communication.

BRIEF DESCRIPTION OF THE DRAWINGS

[008] Representative embodiments of the present invention are herein described, by way of example only, with reference to the accompanying drawings, wherein:

[009] Figure 1 shows a first construction of a key encapsulation platform that combines a plurality of KEM schemes using a blender module to fulfil its key derivation function through a randomness blender function.

[010] Figure 2 shows a second construction of a key encapsulation platform that combines a plurality of KEM schemes using a randomness extractor to fulfil the key derivation function of the blender module.

[011] Figure 3 shows hash modules present in the first construction of the key encapsulation platform of Figure 1.

[012] Figure 4 shows a schematic of a key exchange protocol in a series combiner framework.

[013] Figure 5 shows a schematic of a key exchange protocol in a parallel combiner framework.

DETAILED DESCRIPTION

[014] In the following description, various embodiments are described with reference to the drawings, where like reference characters generally refer to the same parts throughout the different views.

[015] The present application finds relevance in post quantum cryptography to counter quantum computing that can efficiently solve integer factorization problem and discrete logarithm problem in polynomial time, while also being robust to classical attacks.

[016] Herein disclosed is a key encapsulation platform which, at its core, has a module responsible for combining cryptography schemes to ensure that session keys can be securely exchanged between terminals to ensure secure communication therebetween. This module is called a “blender module” which performs a key derivation function. In one mode, the key derivation function is instantiated with a randomness extractor. A randomness extractor converts a long but weakly random input string with more than k bits entropy to a shorter output string with k bits entropy. In another mode, the key derivation function is a randomness blender function that converts an input string with k bits entropy somewhere to an output string with k bits entropy everywhere, through the use of algorithms that can extract randomness presenting in the input string. In one implementation, the blender module comprises a processor having electronics that realise numerous elementary logic gates of AND, OR, XOR, NOT, NAND, NOR and XNOR configuration that are suitably connected to execute these algorithms.

[017] The blender module receives a plurality of input messages. A message refers to unencrypted data providing information in a form that is in the clear, i.e. can be viewed or used without requiring a key or decryption. At least one of these messages has uniform randomness. A message with uniform randomness is perceived by an adversary to be random and not constant. In one implementation, uniform randomness is achieved by a sequence of numbers or symbols that cannot be reasonably predicted by having a uniform distribution of values in all dimensions and have no detectable pattern. The blender module aggregates the received plurality of input messages to propagate the uniform randomness of the at least one input message with uniform randomness. This propagation creates a resource pool of output messages, from which each output message will have uniform randomness, specifically to an adversary.

[018] When the blender module is operating in the randomness blender function mode, the quality of the output messages that the blender module generates, as a result of the aggregation and propagation, depends on the number of input messages that have uniform randomness. When at least one of the input messages has uniform randomness, the plurality of output messages generated by the blender module each has uniform randomness, wherein total entropy of the plurality of output messages is less than or equal to total entropy of the input messages. When at least one of the input messages has uniform randomness, with at least one of the input messages lacking uniform randomness (i.e. at least one input message is not uniformly random), the blender module generates a plurality of output messages, each with uniform randomness and dependency amongst all output messages where the plurality of output messages has a total entropy less than or equal to the total entropy of all the input messages. Independence in the uniform randomness in the plurality of output messages is in accordance with the following condition: for an input of n messages and an output of n messages, if m of the input messages are independent and uniformly random then any subset m of the output messages will be

independent and uniformly random, wherein $1 \leq m \leq n$. As an example, if there is a total of 40 input messages, of which three have uniform randomness jointly, any three of the output messages will be independent and uniformly random, while four or more of the output messages will have dependency amongst them.

5 [019] When the blender module is operating in the randomness extractor mode, the output message has uniform randomness, wherein total entropy of the output message is less than or equal to total entropy of all input messages.

[020] The blender module allows for combining cryptography schemes used to generate session keys to ensure that the output of the blender module is a secure session key provided one of the cryptography
10 schemes is secure. This is particularly advantageous to keep pace with cryptography schemes being developed with multiple layers of defence, where the blender module can be used with a developed scheme that is found to be secure.

[021] The blender module can be incorporated into a key encapsulation platform by being in electrical communication (e.g. by way of coupling, possibly with electronic component blocks therebetween)
15 with a plurality of key encapsulation modules, with the blender module being located downstream of the key encapsulation modules. Each of the key encapsulation modules is configured to generate a plaintext and ciphertext pair, with the ciphertext resulting from encapsulating the plaintext using a public key. The electrical communication between the plurality of key encapsulation modules and the blender module allows for the blender module to receive the plurality of plaintexts as its plurality of
20 input messages. At least one of these key encapsulation modules is required to be secure, so that the plurality of key encapsulation modules provides at least one plaintext that has pseudo-randomness as one of the input messages to the blender module. Having functionality to propagate uniform randomness, as explained earlier, the blender module similarly propagates the pseudo-randomness of the at least one plaintext, so that the blender module can generate a plurality of output messages, each
25 having pseudo-randomness. Specifically, the plurality of output messages is perceived by an adversary to have pseudo-randomness.

[022] The platform is configured to compute at least one secret message from the plurality of output messages from the blender module, the plurality of output messages being generated from the plurality of plaintexts. This secret message is kept by the platform, to be used to generate a shared session key
30 for establishing a secure encryption channel with another terminal. The platform also computes an output ciphertext (called as “encapsulation”) from the plurality of ciphertexts, the encapsulation for transmission and from which the at least one secret message (called as “shared secret”) is derived. The encapsulation can thus be considered as an encoded shared secret or secret message that is to be transmitted to a terminal with which communication is to be established, from the terminal deriving the
35 shared session key from the shared secret encoded inside the encapsulation. The shared secret is used to generate a shared session key for establishing a secure encryption channel for payload communication.

[023] Upon receipt of an encapsulation, which is transmitted by another terminal seeking to establish a secure encryption channel, the key encapsulation platform is configured to: compute a plurality of ciphertexts from the received encapsulation. Each of the key encapsulation modules is further configured to: decapsulate one of the ciphertexts using a private key to obtain a plaintext, the private key corresponding to a public key used to encapsulate the plaintext, so that the blender module receives a plurality of the decapsulated plaintexts as its plurality of input messages. The platform then computes at least one shared secret or secret message from the plurality of output messages from the blender module, the plurality of output messages being generated from the plurality of decapsulated plaintexts.

[024] The key encapsulation platform is described in greater detail below, by first introducing the following definitions.

1.1 Robust Combiner

Definition 1 (Robust Combiner): $A(k, n)$ robust combiner for a cryptographic primitive $\mathcal{P}$ is a construction that takes as input n ingredient schemes for $\mathcal{P}$ and combines them into one scheme such that if at least k of the candidates indeed implement $\mathcal{P}$ then the combiner also implements $\mathcal{P}$.

1.2 Security Formulation of Key Exchange Protocol

[025] In a security formulation, an adversary is characterized in two orthogonal dimensions:

- 1) Information
 - a) what information is given to the adversary, e.g. ciphertext only attack, known plaintext attack, side channel leakage;
 - b) what information that the adversary can feed into the crypto system, e.g. (adaptively) chosen plaintext/ciphertext attack, fault injection attack.
- 2) Computation power. E.g. Probabilistic polynomial algorithm running in classical Turing machine equivalent computer, or PPT (probabilistic polynomial time) algorithm running in quantum computer.

[026] The typical security formulations for Key Encapsulation Mechanism or Key Exchange Protocol include computational indistinguishability under chosen plaintext or ciphertext attack model (a.k.a. IND-CPA, IND-CCA), and forward/backward secrecy. The formulation of IND-CPA and IND-CCA for KEM can be found in F. Giacon, F. Heuer, and B. Poettering, “KEM combiners,” in Public-Key Cryptography – PKC 2018, 2018, pp. 190–218, T. Matsuda and J. Schuldt, “A new key encapsulation combiner,” in International Symposium on Information Theory and Its Applications (ISITA), 2018, pp. 698–702. Our definition of forward secrecy and backward secrecy for key exchange protocols is as follows.

Definition 2 (Forward secrecy): Suppose we run a KEM scheme Φ to generate $(N + 1)$ session keys in sequence. Let TRANSCRIPT denote the collection of all public message exchanged during the generation of these $(N + 1)$ session keys. Let $\alpha(\cdot) \in [0,1]$ denote a real-valued function. We say the KEM scheme Φ provides $\alpha(N)$ -forward-secrecy, if the leakage of secret key (long term secret key and session key) at session $(N + 1)$, together with TRANSCRIPT, can lead to full or partial leakage of at most $(1 - \alpha(N)) \cdot N$ number of session keys before session $N + 1$. We say Φ provides *perfect forward secrecy*, if the corresponding $\alpha(N) = 1$.

Definition 3 (Backward Secrecy): Suppose we run a KEM scheme Φ to generate $(N + 1)$ sessions keys in sequence. Let TRANSCRIPT denote the collection of all public message exchanged during the generation of these $(N + 1)$ session keys. Let $\beta(\cdot) \in [0,1]$ denote a real-valued function. We say the KEM scheme Φ provides $\beta(N)$ -backward-secrecy, if the leakage of secret key (long term secret key and session key) at some session $i \in [1, N + 1]$, together with TRANSCRIPT, can lead to fully or partial leakage of at most $(1 - \beta(N)) \cdot N$ number of session keys after session i . We say Φ provides *perfect backward secrecy*, if the corresponding $\beta(N) = 1$.

1.3 Compression Entropy

[027] From Yao’s Entropy, and Xu and Zhou, J. Xu and J. Zhou, “Strong leakage-resilient encryption: enhancing data confidentiality by hiding partial ciphertext,” 2020, <https://doi.org/10.1007/s10207-020-00487-7>, our notion of “Compression Entropy”, is as follows:

Definition 4 (Compression Entropy or Extended Yao’s Entropy): Let function $f: \{0,1\}^{m_0} \times \{0,1\}^{m_1} \rightarrow \{0,1\}^{n_0} \times \{0,1\}^{n_1}$ be $f(x; a) = (y; b)$, where both x and y are kept private and, all of a and b and description of function f are public.

- Function f has at least k bits compression-entropy in its output w.r.t. the distribution $\mathcal{D}$ over $\{0,1\}^{m_0+m_1}$, denoted as $\Upsilon_{\epsilon,t}^{\mathcal{D}}(f_{m_0,n_0}) \geq k$, if for any t -time compression algorithm $\mathbb{C}$ and t -time decompression algorithm $\mathbb{D}$ with $\mathbb{C}(\dots) \in \{0,1\}^l$

$$\Pr_{(x,a) \sim \mathcal{D}} [\mathbb{D}(\mathbb{C}(x; a), a, b) = y] \leq 2^{l-k} + \epsilon \quad (1)$$

Shortly, we may say f has at least k bits compression-entropy (w.r.t. distribution $\mathcal{D}$).

- Function f has at most l bits compression entropy, in its output w.r.t. the distribution $\mathcal{D}$ over $\{0,1\}^{m_0+m_1}$, denoted as $\Upsilon_{\epsilon,t}^{\mathcal{D}}(f) \leq l$, if there exist some t -time compression algorithm $\mathbb{C}$ and t -time decompression algorithm $\mathbb{D}$ with $\mathbb{C}(\dots) \in \{0,1\}^l$,

$$\Pr_{(x,a) \sim \mathcal{D}} [\mathbb{D}(\mathbb{C}(x; a), a, b) = y] \geq 1 - \epsilon \quad (2)$$

Shortly, we may say f has at most l bits compression-entropy (w.r.t. distribution $\mathcal{D}$).

- Compression-entropy rate of function f w.r.t. distribution $\mathcal{D}$ is defined as

$$v_{\epsilon,t}^{\mathcal{D}} = \frac{\Upsilon_{\mathcal{D}}^{\epsilon,t}(f_{m_0,n_0})}{n_0} \quad (3)$$

Proposition 1 (Monotonic in parameter ϵ): If $\Delta > 0$, then

$$\Upsilon_{\epsilon,t}^{\mathcal{D}}(f) \leq \Upsilon_{\epsilon+\Delta,t}^{\mathcal{D}}(f). \quad (4)$$

5

Proposition 2 (Relation with Yao's Entropy (1)): Let identity function $I(x; \perp) = (x; \perp)$. We have

$$\Upsilon(I) = H^{\text{Yao}}(x) \quad (5)$$

Proposition 3 (Relation with Yao's Entropy (2)): For any function $f(x; a) = (y; b)$ as defined in

10 Definition 4, variable (x, a) follows the distribution $\mathcal{D}$, and distribution of y is determined by function f together with distribution $\mathcal{D}$, we have

$$\mathbf{H}_{\epsilon,t}^{\text{Yao}}(y) \geq \Upsilon_{\epsilon,t}^{\mathcal{D}}(f) \geq \mathbf{H}_{\infty}(y). \quad (6)$$

Note $\mathbf{H}_{\infty}(y)$ denotes the min-entropy of variable y .

15 *Proposition 4 (Relation with Conditional Yao's Entropy):* Let $f(\perp; a) = (y; \perp)$ and variable a follows distribution $\mathcal{D}$. We have

$$\mathbf{H}_{\epsilon,t}^{\text{Yao}}(y|a) = \Upsilon_{\epsilon,t}^{\mathcal{D}}(f). \quad (7)$$

20 *Lemma 1 (Separation from Yao's Entropy):* For any positive integer $c \geq \lambda$ and any polynomial $\text{poly}(c) \geq c$, there exists some polynomial time computable function $f(x; a) = (y; b)$, such that

$$\Upsilon(f) \leq c; \quad (8)$$

$$\mathbf{H}^{\text{Yao}}(y) \geq \text{poly}(\lambda) \quad (9)$$

1.4 Randomness Blender Function

25 [028] We define randomness blender function, which converts an input string with k bits entropy somewhere to an output string with k bits entropy everywhere.

Definition 5 (Randomness Blender): Let $(a_0, \dots, a_{n-1}) \in \{0,1\}^{\rho \cdot n}$ be a sequence of n independent random variables $a_i \in \{0,1\}^{\rho}$ and $(b_0, \dots, b_{n-1}) = f(a_0, \dots, a_{n-1}; r)$ with each $b_i \in \{0,1\}^{\rho'}$ ($\rho' \geq \rho$).

30 We say function f is (ϵ, δ) - randomness blender w.r.t. block size ρ , if the following conditions hold

- for any $l \in [1, n]$, if there are l distinct a_i 's with joint Shannon entropy equal to $l \cdot \rho$ bits, then with probability at least $(1 - \epsilon)$, any subset of l distinct b_j 's will have joint Shannon entropy $\geq (1 - \delta)l\rho$.

Lemma 2: Let polynomial $P_{\vec{a}}(x) = \sum_{i=0}^{n-1} a_i x^{i+1}$ over a finite field $GF(p)$ with prime order p . We define function F as

$$F(\vec{a}; r) = (P_{\vec{a}}(r), P_{\vec{a}}(r+1), \dots, P_{\vec{a}}(r+n-1)) \quad (10)$$

F is a (ϵ, δ) -randomness blender, where ϵ is negligible, and $\delta = 0$.

5 The above lemma can be proved from results in I. Shparlinski, “On the singularity of generalised vandermonde matrices over finite fields,” *Finite Fields and Their Applications*, vol. 11, pp. 193–199, April 2005

[029] Each of Figures 1 and 2 show the use of a blender module 102, 202 in a key encapsulation platform 100, 200. The blender module 102 of Figure 1 implements the randomness blender function of Definition 5, i.e. the blender module 102 is operating in a randomness blender function mode. The blender module 202 of Figure 2 is operating in a randomness extractor mode, described in greater detail in section 2.4. The blender module 102, 202 is configured to receive 104, 204 a plurality of input messages with at least one having uniform randomness. The blender module 102, 202 aggregate the received plurality of input messages to propagate the uniform randomness of the at least one input message with uniform randomness. Provided at least one of the input messages has uniform randomness, the blender module 102 generates a plurality of output messages 124, each having uniform randomness, wherein total entropy of the plurality of output messages 124 is less than or equal to total entropy of all the input messages. When at least one of the input messages has uniform randomness, with at least one of the input messages lacking uniform randomness, the blender module 102 generates a plurality of output messages 124, each with uniform randomness and dependency amongst all output messages, where the plurality of output messages 124 has a total entropy less than or equal to the total entropy of all input messages. Independence in the uniform randomness in the plurality of output messages 124 is in accordance with the following condition: for an input of n messages and an output of n messages, if m of the input messages are independent and uniformly random then any subset m of the output messages 124 will be independent and uniformly random, wherein $1 \leq m \leq n$. Provided at least one of the input messages has uniform randomness, the blender module 202 generates an output message 224 with uniform randomness.

30 2. Key Encapsulation Mechanisms

[030] To realise a key encapsulation platform, the blender module 102, 202 of Figures 1 and 2 is coupled with a plurality of key encapsulation modules 108, 110 and 112. There may be one or more electrical components between the plurality of key encapsulation modules 108, 110 and 112 and the blender module 102, 202, which are not shown for the sake of simplicity.

35 [031] Key encapsulation mechanism (KEM) is a public key cryptography primitive, closely related to public key encryption (PKE). A key encapsulation mechanism applies public key encryption on a

(uniformly) random plaintext. In the other direction, a public key encryption is constructed by simply combining key encapsulation mechanism and one-time pad cipher. Combining the two provides a generic way to construct CCA-(chosen ciphertext attack) secure PKE from CPA-(chosen plaintext attack) secure PKE.

5 *Definition 6 (Key Encapsulation Mechanism):* A KEM scheme consists of 3 algorithms (Gen, Encaps, Decaps), described as below

- $\text{Gen}(1^\lambda) \rightarrow (pk, sk)$.
- $\text{Encaps}(pk) \rightarrow (c, s)$.
- $\text{Decaps}(sk, c) \rightarrow s$.

10

[032] Given n number of ingredient KEM schemes $\Phi_i, i = 0, 1, 2, \dots, n - 1$, herein disclosed are two generic constructions, denoted as Φ and Ψ , of stateful KEM schemes, following parallel framework with core function (or key derivation function) F and G respectively, where functions F and G are described in greater detail below. In one implementation, our disclosed KEM schemes are robust

15 combiners for KEM. Without loss of generality, we assume: (1) each KEM scheme Φ_i will output l bits shared secret (i.e. s) in every invocation; (2) the two parties in communication have already authenticated the identity of each other before running KEM schemes; (3) each party may maintain separate long term status for different frequently contacted parties. It will be shown that good atomized complexity is achieved for two frequently contacted parties.

20 [033] Figures 1 and 2 each respectively shows the first construction and the second construction of a key encapsulation platform 100, 200 that combines a plurality of KEM schemes using a blender module 102, 202. The key encapsulation platform 100 of Figure 1 and the key encapsulation platform 200 of Figure 2 have the following common configuration of a plurality of key encapsulation module 108, 110 and 112 being coupled to a blender module 102, 202. At least one of the key encapsulation modules

25 108, 110 and 112 is secure.

[034] When the key encapsulation platform 100, 200 functions as a transmitter, each of the key encapsulation modules 108, 110 and 112 is configured to generate a plaintext and ciphertext pair 116, 118 and 120. The ciphertext (Figure 1: c_1, c_2 and c_3 ; Figure 2: $c_{1,1}, c_{2,1}, c_{3,1}$) results from encapsulating the plaintext (Figure 1: s_1, s_2 and s_3 ; Figure 2: $s_{1,1}, s_{2,1}, s_{3,1}$) using a public key pk_1, pk_2 and pk_3 .

30 The coupling between the blender module 102, 202 and the plurality of key encapsulation modules 108, 110 and 112 is such that the blender module 102, 202 receives 104, 204 the plurality of plaintexts (Figure 1: s_1, s_2 and s_3 in a single session; Figure 2: $s_{1,1}, s_{2,1}$, and $s_{3,1}$ during a first session; $s_{2,1}, s_{3,1}$ and $s_{1,2}$ during a second session; and $s_{3,1}, s_{1,2}$ and $s_{2,2}$ during a third session) as its plurality of input messages. The platform 100, 200 is then configured to compute 106, 206 at least one shared secret

35 (Figure 1: s'_1, s'_2, s'_3 in the single session; Figure 2: s'_1 during the first session; s'_2 during the second

session; and s'_3 during the third session) from the plurality of output messages 124, 224 from the blender module 102, 202, the plurality of output messages 124, 224 being generated from the plurality of plaintexts. The platform 100, 200 also computes 126, 226 an encapsulation c from the plurality of ciphertexts (Figure 1: c_1, c_2 and c_3 in the single session; Figure 2: $c_{1,1}, c_{2,1}$ and $c_{3,1}$ during the first session; $c_{2,1}, c_{3,1}$ and $c_{1,2}$ during the second session; and $c_{3,1}, c_{1,2}$ and $c_{2,2}$ during the third session), the encapsulation c for transmission and from which the at least one shared secret (Figure 1: s'_1, s'_2, s'_3 in the single session; Figure 2: s'_1 during the first session; s'_2 during the second session; and s'_3 during the third session) is derived.

[035] When the key encapsulation platform 100, 200 functions as a receiver, the platform 100, 200 computes a plurality of ciphertexts (Figure 1: c_1, c_2 and c_3 in a single session; Figure 2: $c_{1,1}, c_{2,1}$, and $c_{3,1}$ during a first session; $c_{2,1}, c_{3,1}$, and $c_{1,2}$ during a second session; and $c_{3,1}, c_{1,2}$, and $c_{2,2}$ during a third session) from a received encapsulation c (analogous to the encapsulation c for transmission). Each of the key encapsulation modules 108, 110 and 112 is configured to: decapsulate one of the ciphertexts using a private key sk_1, sk_2, sk_3 to obtain a plaintext, the private key corresponding to a public key pk_1, pk_2, pk_3 used to encapsulate the plaintext, so that the blender module 102, 202 receives a plurality of the decapsulated plaintexts (Figure 1: s_1, s_2 and s_3 in the single session; Figure 2: $s_{1,1}, s_{2,1}$, and $s_{3,1}$ during the first session; $s_{2,1}, s_{3,1}$, and $s_{1,2}$ during the second session; and $s_{3,1}, s_{1,2}$, and $s_{2,2}$ during the third session) as its plurality of input messages. The platform 100, 200 is further configured to: compute 124, 224 at least one shared secret (Figure 1: s'_1, s'_2, s'_3 in the single session; Figure 2: s'_1 during the first session; s'_2 during the second session; and s'_3 during the third session) from the plurality of output messages 124, 224 from the blender module 102, 202, the plurality of output messages 124, 224 being generated from the plurality of decapsulated plaintexts.

2.1 First Generic Construction Φ with Key Derivation Function F

[036] With reference to Figure 1, the key encapsulation platform 100 uses as building blocks: a blender module 102 that uses a construction denoted as $\Phi[F, n, m]$, which takes a function $F: (\{0,1\}^{\ell_0})^n \times \{0,1\}^{\ell_0} \rightarrow (\{0,1\}^{\ell_1})^m$; and a plurality of key encapsulation modules to provide n number of KEM schemes Φ_i 's, $i \in [0, n-1]$, although only three are shown 108, 110 and 112. The key encapsulation platform 100 generates and caches m number of bit-strings in $\{0,1\}^{\ell_1}$ at once. Optionally, a message authentication code (MAC) will be generated for every ciphertext, in order to achieve security under chosen ciphertext attack (CCA) model. The three algorithms Gen, Encaps, Decaps are defined as below.

$\Phi.\text{Gen}(1^\lambda)$

- $\forall i \in [0, n-1], (pk_i, sk_i) \leftarrow \Phi_i.\text{Gen}(1^\lambda).$

- Let $pk \leftarrow (\dots, pk_i, \dots)_{i \in [0, n-1]}$.
- Let $sk \leftarrow (\dots, sk_i, \dots)_{i \in [0, n-1]}$.
- Output (pk, sk) .

$\Phi.\text{Encaps}(pk; r)$

- 5 • $\forall i \in [0, n-1], (c_i, s_i) \leftarrow \Phi_i.\text{Encaps}(pk_i)$.
- Compute $\bar{r} \leftarrow h_0(r, c_0, \dots, c_{n-1})$.
- Compute $(x_0, \dots, x_{m-1}) \leftarrow F(s_0, \dots, s_{n-1}, \bar{r})$. With reference to Figure 1, the blender module 102 aggregates the plurality of plaintexts s_1, s_2 and s_3 that is generated by the plurality of key encapsulation modules 108, 110 and 112, so as to propagate the uniform randomness present in at least one input plaintext, s_1, s_2, s_3 . A plurality of output messages 124 is generated by the blender module 102 from the aggregation.
- 10 • Compute $\sigma \leftarrow \text{MAC}_{s_0, \dots, s_{n-1}}(r, c_0, \dots, c_{n-1})$. An optional message authentication code for the shared secret may be generated for accompanying the encapsulation c for transmission.
- Compute $s \leftarrow (h_1(x_0), \dots, h_1(x_{m-1}))$. With reference to Figure 1, the key encapsulation platform 100 computes 106 at least one shared secret (s'_1, s'_2, s'_3) , from the plurality of output messages 124 from the blender module 102. In one approach, a hash function h_1 is used in this computation 106, such as SHA256, SHA512 or SHA3.
- 15 • Compute $c \leftarrow (\sigma, r, c_0, \dots, c_{n-1})$. With reference to Figure 1, the key encapsulation platform 100 computes 126 an encapsulation c from the plurality of ciphertexts c_1, c_2 and c_3 .
- 20 • Output (s, c) . With reference to Figure 1, the key encapsulation platform 100 stores the at least one shared secret (s'_1, s'_2, s'_3) and transmits the encapsulation c , from which the at least one shared secret (s'_1, s'_2, s'_3) is derived.

$\Phi.\text{Decaps}(sk, c)$

- 25 • $(\sigma, r, c_0, \dots, c_{n-1}) \leftarrow c$.
- $\forall i \in [0, n-1], s_i \leftarrow \Phi_i.\text{Decaps}(sk_i, c_i)$. With reference to Figure 1, each of the plurality of key encapsulation modules 108, 110 and 112 decapsulates one of the ciphertexts c_1, c_2, c_3 , obtained from the input ciphertext c that is received by the key encapsulation platform 100, using a private key sk_1, sk_2, sk_3 to obtain a plaintext s_1, s_2, s_3 . The private key that is used corresponds to a public key used to encapsulate the plaintext. This results in the blender module 102 receiving 104 a plurality of the decapsulated plaintexts s_1, s_2 , and s_3 as its plurality of input messages.
- 30 • Compute $\bar{r} \leftarrow h_0(r, c_0, \dots, c_{n-1})$.
- Compute $(x_0, \dots, x_{m-1}) \leftarrow F(s_0, \dots, s_{n-1}, \bar{r})$. With reference to Figure 1, the blender module 102 aggregates the plurality of decapsulated plaintexts s_1, s_2 and s_3 , so as to propagate the uniform randomness present in at least one of the input plaintext, s_1, s_2, s_3 . A plurality of output messages 124 is generated by the blender module 102 from the aggregation.
- 35 • If σ is a valid MAC for message $(r, c_0, \dots, c_{n-1})$ w.r.t. key $(s_0, \dots, s_{n-1})$, compute and output $s \leftarrow (h_1(x_0), \dots, h_1(x_{m-1}))$. With reference to Figure 1, the key encapsulation platform 100 computes 106 at least one shared secret s'_1, s'_2, s'_3 , from the plurality of output messages 124 from the blender module 102. In one approach, a hash function h_1 is used in this computation 106, such as SHA256, SHA512 or SHA3. In the case where the input ciphertext c has a message authentication code, this computation 106 is performed in response to successful validation of the message authentication code. Otherwise, output $\perp$.
- 40

[037] The plurality of key encapsulation modules 108, 110 and 112 is further configured to simultaneously operate when providing the plurality of plaintexts s_1, s_2 and s_3 to the blender module 100, whereby the platform 100 computes the plurality of shared secrets s'_1, s'_2 , and s'_3 from the simultaneous operation. Upon receipt of a command to refresh the shared session key, the plurality of key encapsulation modules 108, 110 and 112 is further configured to simultaneously operate to generate a plurality of new plaintext and ciphertext pairs. The new plaintext and ciphertext pairs are then used by the platform 100 to compute a plurality of new secret messages and a new encapsulation for transmission from which the plurality of new secret messages is derived, following the steps described in $\Phi.\text{Encaps}(pk; r)$ and $\Phi.\text{Decaps}(sk, c)$.

[038] With reference to Figure 3, three of the output messages x_1, x_2, x_3 of the plurality of output messages 124 are shown. Hash modules 306 are also shown, where each outputs a hashed message y_1, y_2, y_3 with $s = (y_i)_i$, so that computation 106 of the at least one secret message s'_1, s'_2, s'_3 is achieved. The computation of the encapsulation c for transmission is denoted as $C = (c_i)_i$.

2.2 Second Generic Construction Ψ with Key Derivation Function G

[039] With reference to Figure 2, the key encapsulation platform 200 uses as building blocks: a blender module 102 that uses a construction denoted as $\Psi[G, n]$, which takes a function $G: (\{0,1\}^{\ell_0})^n \times \{0,1\}^{\ell_0} \rightarrow \{0,1\}^{\ell_1}$ and a plurality of key encapsulation modules to provide n number of KEM schemes Φ_i 's, $i \in [0, n-1]$, although only three are shown 108, 110 and 112. The three algorithms (Gen, Encaps, Decaps) are defined below.

$\Psi.\text{Gen}(1^\lambda)$

- $\forall i \in [0, n-1], (pk_i, sk_i) \leftarrow \Phi_i.\text{Gen}(1^\lambda).$
- Let $pk \leftarrow (\dots, pk_i, \dots)_{i \in [0, n-1]}.$
- Let $sk \leftarrow (\dots, sk_i, \dots)_{i \in [0, n-1]}.$
- Output $(pk, sk).$

$\Psi.\text{Encaps}(pk; r)$

- This algorithm maintains two state variables:
 - *COUNT*, an integer initialized to 0 when this algorithm is invoked for the first time, and
 - *CACHE*, a list of exactly n elements from $\{0,1\}^{\ell_1}$.
- If *COUNT* is zero, initialize the state variable *CACHE* as below:
 - $\forall i \in [0, n-1], (c_i, s_i) \leftarrow \Phi_i.\text{Encaps}(pk_i).$
 - Initialize $\text{CACHE} = (s_0, \dots, s_{n-1})$. With reference to Figure 2, during initialisation (also referenced as a “1st session”), the plurality of key encapsulation modules 108,

110 and 112 is configured to simultaneously operate to provide the plurality of plaintexts $s_{1,1}$, $s_{2,1}$, and $s_{3,1}$ to the blender module 202.

- Compute $s = h_1(G(CACHE, r))$. With reference to the 1st session of Figure 2, the blender module 202 aggregates the plurality of plaintexts $s_{1,1}$, $s_{2,1}$, and $s_{3,1}$, generated by the plurality of key encapsulation modules 108, 110 and 112, so as to propagate the uniform randomness present in at least one input plaintext, $s_{1,1}$, $s_{2,1}$, $s_{3,1}$. A plurality of output messages 224 is generated by the blender module 202 from the aggregation. The key encapsulation platform 200 then computes 206 at least one secret message (s'_1), from the plurality of output messages 224 from the blender module 102. In one approach, a hash function h_1 is used in this computation 106, such as SHA256, SHA512 or SHA3. The key encapsulation platform 100 stores the secret message s'_1 .
- Increment *COUNT* by 1 and output $c = (0, r, c_0, \dots, c_{n-1})$ and s . With reference to the 1st session of Figure 2, the key encapsulation platform 200 computes 226 an encapsulation c from the plurality of ciphertexts $c_{1,1}$, $c_{2,1}$, and $c_{3,1}$. The encapsulation platform 200 transmits the encapsulation c , from which the at least one secret message s'_1 is derived.

[040] From the above, during initialisation, the plurality of key encapsulation modules 108, 110 and 112 is configured to simultaneously operate to provide the plurality of plaintexts $s_{1,1}$, $s_{2,1}$, and $s_{3,1}$ to the blender module, whereby the platform 200 computes the at least one secret message from the simultaneous operation.

- If *COUNT* ≥ 1 :

- Let $i = COUNT \pmod n$, and compute $(c'_i, s'_i) \leftarrow \Phi_i. \text{Encaps}(pk_i)$.
- Update the item s_i in *CACHE*: $s_i \leftarrow s'_i$. During a 2nd session, one of the key encapsulation modules 108, 110 and 112 is configured to generate a new plaintext and ciphertext pair. With reference to Figure 2, this is the key encapsulation module 108, which generates the new plaintext and ciphertext pair $s_{1,2}$ and $c_{1,2}$. The new plaintext and the existing plaintexts are received 204 by the blender module 202, so that the blender module receives a plurality of plaintexts $c_{2,1}$, $c_{3,1}$, and $c_{1,2}$ during the second session.
- Compute $s = h_1(G(CACHE; r))$. With reference to the 2nd session of Figure 2, a plurality of output messages 224 is generated by the blender module 202 from the aggregation of the plurality of plaintexts $c_{2,1}$, $c_{3,1}$, and $c_{1,2}$. The key encapsulation platform 200 then computes 206 at least one secret message s'_2 from the plurality of output messages 224 from the blender module 102. In one approach, a hash function h_1 is used in this computation 106, such as SHA256, SHA512 or SHA3. The key encapsulation platform 100 stores the secret message s'_2 .
- Output $c = (COUNT, c'_i, r)$ and s . Increment *COUNT* by 1. With reference to the 2nd session of Figure 2, the key encapsulation platform 200 computes 226 an encapsulation c from the new ciphertext $c_{1,2}$ and the existing ciphertext $c_{2,1}$, $c_{3,1}$. The encapsulation platform 200 transmits the encapsulation c , from which the at least one secret message s'_2 is derived.

[041] From the above, upon receipt of a command to refresh the shared session key, each of the key encapsulation modules 108, 110 and 112 is configured to generate a new plaintext and ciphertext pair in turn, whereby the platform computes a new shared secret from the new plaintext and existing plaintexts and computes a new encapsulation for transmission from the new ciphertext and existing ciphertexts.

$\Psi.\text{Decaps}(sk, c)$

- This algorithm maintains a state variable *CACHE* which is a list of exactly n elements from $\{0,1\}^{\ell_1}$.
- If $c = (0, r, c_0, \dots, c_{n-1})$:
 - 10 – $\forall i \in [0, n-1], s_i \leftarrow \Phi_i.\text{Decaps}(sk_i, c_i)$. With reference to the 1st session of Figure 2, each of the plurality of key encapsulation modules 108, 110 and 112 decapsulates one of the ciphertexts $c_{1,1}, c_{2,1}, c_{3,1}$ obtained from an encapsulation c that is received by the key encapsulation platform 200, using a private key sk_1, sk_2, sk_3 , to obtain a plaintext $s_{1,1}, s_{2,1}, s_{3,1}$. The private key that is used corresponds to a public key used to encapsulate the plaintext.
 - 15 – Initialize the state variable $CACHE \leftarrow (\dots s_i \dots)_{i \in [0, n-1]}$. With reference to the 1st session of Figure 2, the plurality of key encapsulation modules 108, 110 and 112 is configured to simultaneously operate to provide the plurality of decapsulated plaintexts $s_{1,1}, s_{2,1}$, and $s_{3,1}$ to the blender module 202.
 - 20 – Compute $s = h_1(G(CACHE, r))$ and output s . With reference to the 1st session of Figure 2, the blender module 202 aggregates the plurality of decapsulated plaintexts $s_{1,1}, s_{2,1}$, and $s_{3,1}$, generated by the plurality of key encapsulation modules 108, 110 and 112, so as to propagate the uniform randomness present in at least one input plaintext, $s_{1,1}, s_{2,1}, s_{3,1}$. A plurality of output messages 224 is generated by the blender module 202 from the aggregation. The key encapsulation platform 200 then computes 206 at least one secret message (s'_1), from the plurality of output messages 224 from the blender module 102. In one approach, a hash function h_1 is used in this computation 106, such as SHA256, SHA512 or SHA3. The key encapsulation platform 100 stores the secret message s'_1 .
 - 25 – Compute and output $s = h_1(G(CACHE, r))$ and output s . With reference to the 1st session of Figure 2, the blender module 202 aggregates the plurality of decapsulated plaintexts $s_{1,1}, s_{2,1}$, and $s_{3,1}$, generated by the plurality of key encapsulation modules 108, 110 and 112, so as to propagate the uniform randomness present in at least one input plaintext, $s_{1,1}, s_{2,1}, s_{3,1}$. A plurality of output messages 224 is generated by the blender module 202 from the aggregation. The key encapsulation platform 200 then computes 206 at least one secret message (s'_1), from the plurality of output messages 224 from the blender module 102. In one approach, a hash function h_1 is used in this computation 106, such as SHA256, SHA512 or SHA3. The key encapsulation platform 100 stores the secret message s'_1 .
- 30 • If c is $(COUNT, c'_i, r)$:
 - Let $i = COUNT \pmod n$ and compute $s'_i = \Phi_i.\text{Decaps}(sk_i, c'_i)$. With reference to the 2nd session of Figure 2, an encapsulation c is received by the key encapsulation platform 200 in which a new ciphertext $c_{1,2}$ is present. The key encapsulation module 108 decapsulates the new ciphertext $c_{1,2}$ using its private key sk_1 to obtain a new plaintext $s_{1,2}$.
 - 35 – Update the item s_i in *CACHE*: $s_i \leftarrow s'_i$. With reference to the example in Figure 2, the state variable *CACHE* is reset to $(s_{1,2}, s_{2,1}, s_{3,1})$, with old plaintext $s_{1,1}$ being replaced by the new plaintext $s_{1,2}$.
 - Compute and output $s = h_1(G(CACHE, r))$. With reference to the 2nd session of Figure 2, a plurality of output messages 224 is generated by the blender module 202 from the aggregation of the plurality of decapsulated plaintexts $s_{1,2}, s_{2,1}, s_{3,1}$. The key encapsulation platform 200 then computes 206 at least one secret message s'_2 .
 - 40 – Compute and output $s = h_1(G(CACHE, r))$. With reference to the 2nd session of Figure 2, a plurality of output messages 224 is generated by the blender module 202 from the aggregation of the plurality of decapsulated plaintexts $s_{1,2}, s_{2,1}, s_{3,1}$. The key encapsulation platform 200 then computes 206 at least one secret message s'_2 .

from the plurality of output messages 224 from the blender module 102. In one approach, a hash function h_1 is used in this computation 106, such as SHA256, SHA512 or SHA3.

5 2.3 **Instantiation of Function F and G**

[042] SHA256, SHA512, or SHA3 function may be used for the hash function h_0, h_1 in the above constructions.

2.4 **Randomness Extractor and Pseudorandom Function**

10 [043] We observe that, the collection of all secret s_i generated by ingredient KEM Φ_i , follows block-fixing distribution, in the view of a computationally bounded adversary: If KEM scheme Φ_i is secure against the adversary, the secret value s_i will be pseudo-random; otherwise, the value s_i is considered as a constant in the view of the adversary.

[044] In our second construction, one possible choice of function $G(CACHE, r)$ is a randomness extractor function Ext with input random variable $CACHE$ and random seed r .

$$G_0(\dots, s_i, \dots, r) \& = \text{Ext}(\dots, s_i, \dots; r) \quad (11)$$

[045] We denote the second construction instantiated with the above function G_0 as $\Psi[\text{Ext}, n]$.

[046] One possible choice for function F in our first construction is a combination of randomness extractor and cryptographically secure pseudorandom function Prf.

$$20 \quad F_0(\dots, s_i, \dots, r) = (\dots, \text{Prf}_u(i), \dots)_{i \in [0, m-1]} \quad (12)$$

where $u \leftarrow G_0(\dots s_i \dots, r)$.

[047] We will denote the first construction instantiated with the above function F_0 as $\Phi[\text{Ext} + \text{Prf}, n, m]$.

25 2.3.2 **Randomness Blender Function**

[048] An alternative choice of F and G is a randomness blender function, defined as follows:

$$F_1(\dots, s_i, \dots, r) = \text{Blender}(\dots, s_i, \dots, r) \quad (13)$$

$$G_1(\dots, s_i, \dots, r) = \text{Blender}(\dots, s_i, \dots, r) \quad (14)$$

where notation $V[0]$ denotes the first element in the vector V .

30 [049] We construct G_1 and F_1 as below.

$$G_1(\dots, s_i, \dots, r) = \sum_{i=0}^{n-1} s_i \cdot r^{i+1} \quad (15)$$

$$\begin{aligned} F_1(\dots, s_i, \dots, r) &= (\dots G_1(\dots, s_i, \dots, r + j) \dots)_{j \in [0, t-1]} \\ &= (\dots, s_i, \dots) \times M_r \end{aligned} \quad (16)$$

where the above computation is over a finite field (e.g. $GF(p)$ with prime order p) and M_r is a *Generalized Vandermonde Matrix*. F_1 is the randomness blender function defined in Lemma 2.

[050] In a prototype implementation, we choose Mersenne prime in the form of $p = 2^q - 1$ with both p and q being prime numbers, to achieve faster modulo operation with modulus p .

5

2.4 Security

Theorem 3: Assume h_0 is a cryptographically secure hash function and h_1 be an identity function. Then $\Phi[\text{Ext}, n, 1]$ is a $(1, n)$ -Robust Combiner for KEM w.r.t. IND-CPA/CCA security formulation, where the function F in scheme Φ is instantiated with a randomness extractor Ext .

10 *Theorem 4:* Assume h_0 is a cryptographically secure hash function and h_1 is a random oracle. Then $\Phi[\text{Blender}, n, n]$ is a $(1, n)$ -Robust Combiner for KEM w.r.t. IND-CPA/CCA security formulation, where the function F in scheme Φ is instantiated with a randomness blender function Blender .

Theorem 5: Assume h_0 is a cryptographically secure hash function and h_1 is a random oracle. Then $\Phi[\text{Ext} + \text{Prf}, n, n]$ is a $(1, n)$ -Robust Combiner for KEM w.r.t. IND-CPA/CCA security formulation, where the function F in scheme Φ is instantiated with a combination of randomness extractor Ext and a secure pseudorandom function Prf .

Theorem 6: Assume h_0 is a cryptographically secure hash function and h_1 is a random oracle. Then $\Psi[\text{Blender}[0], n]$ is a $(1, n)$ -Robust Combiner for KEM w.r.t. IND-CPA/CCA security formulation, where the function G in scheme Ψ is instantiated with the first element $\text{Blender}[0]$ in the output vector of function Blender .

Theorem 7: Let $f(sk; c) = (\Phi[\text{Blender}, n, n]. \text{Decaps}(sk, c), \perp)$. The compression entropy $Y_{\epsilon, t}(f) \geq m \times \log(1/\epsilon)$ if m number of ingredients KEM schemes are (t, ϵ) -secure (i.e. any probabilistic t -time adversary can have advantage at most ϵ to break the targeted scheme).

Theorem 8: Let $f(sk; c) = (\Phi[\text{Ext} + \text{Prf}, n, n]. \text{Decaps}(sk, c), \perp)$. The compression entropy $Y_{\epsilon, t}(f) \leq \ell_1$ if m number of ingredient KEM schemes are (t, ϵ) -secure (i.e. any probabilistic t -time adversary can have advantage at most ϵ to break the targeted scheme).

[051] The above two theorems show that our randomness blender function is much better than the combination of randomness extractor and pseudorandom function, when constructing robust combiner of KEM by following our first construction framework.

[052] We compare our proposed robust combiner of KEMs with related works in Table 1.

Table 1: Comparison of construction of KEM combinars in existing works

Scheme	core function W	tag function T	Security
[7]	$\bigoplus_i s_i$ $h(\bigoplus_i s_i, c_0, \dots, c_{n-1})$ $h(\dots s_i \dots c_i \dots)$ $h(\pi_{s_{n-1}}(\dots \pi_{s_0}(0)), c_0 \parallel \dots \parallel c_{n-1})$ $\bigoplus_i \text{Prf}(s_i, c_0, \dots, c_{n-1})$	$\perp$ $\perp$ $\perp$ $\perp$ $\perp$	CPA, SM CCA, ROM CCA, ROM CCA, ROM or ICM CCA, SM
[1], [8], [10]	k_0 where $(k_0, k_1) \leftarrow \bigoplus_i s_i$	$\text{MAC}_{k_1}(\dots c_i \dots)$	CCA, SM
[1]	$\text{Prf}(\text{dualPrf}(s_0, s_1), c_0, c_1)$	$\perp$	CCA, SM
[3], [4]	$\text{Kdf}(s_0 \parallel s_1)$	$\perp$	No claims
[4]	$\text{Kdf}(s_0 \oplus s_1)$	$\perp$	No claims
Our constructions Φ, Ψ	Randomness Extractor + Prf, or Randomness Blender function	MAC of all ciphertexts	CCA, ROM

[1] N. Bindel, J. Brendel, M. Fischlin, B. Goncalves, and D. Stebila, “Hybrid key encapsulation mechanisms and authenticated key exchange,” in International Conference on Post-Quantum Cryptography, 2019, <https://eprint.iacr.org/2018/903>

5 [3] J. W. Bos, C. Costello, M. Naehrig, and D. Stebila, “Post-quantum key exchange for the tls protocol from the ring learning with errors problem,” in IEEE Security & Privacy, 2015, <https://eprint.iacr.org/2014/599>

[4] E. Crockett, C. Paquin, and D. Stebila, “Prototyping post-quantum and hybrid key exchange and authentication in tls and ssh,” Cryptology ePrint Archive, Report 2019/858, 2019, <https://eprint.iacr.org/2019/858>

[7] F. Giacon, F. Heuer, and B. Poettering, “Kem combinars,” in Public-Key Cryptography – PKC 2018, 2018, pp. 190–218

10 [8] G. HANAOKA, T. MATSUDA, and J. C. N. SCHULDT, “A new combiner for key encapsulation mechanisms,” IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, vol. E102.A, no. 12, pp. 1668–1675, 2019

[10] T. Matsuda and J. Schuldt, “A new key encapsulation combiner,” in International Symposium on Information Theory and Its Applications (ISITA), 2018, pp. 698–702

15 3. Public Key Exchange Protocol

[053] Public key exchange protocol aims to establish a shared secret key between two parties (Alice and Bob) by exchanging some messages over an insecure public communication channel. It can be implemented by applying key encapsulation mechanism in a straightforward manner with one round of communication (as in Table 2).

20

Table 2: Public Key Exchange between Alice and Bob using Key Encapsulation Mechanism

Setup: Alice generates a pair of keys $(pk, sk) \leftarrow \text{KeyGen}(1^\lambda)$, and distributes public key pk to Bob (via a PKI system). B1: Bob computes $(c, s) \leftarrow \text{Encaps}(pk)$ and sends the ciphertext c to Alice (typically via an insecure public communication channel). A1: Alice recovers the secret $s \leftarrow \text{Decaps}(sk, c)$.
--

[054] Alternatively, key exchange protocol can also be implemented using different methods and with one or more rounds of communication, for example, Quantum Key Distribution protocol, based on quantum physical theory. Compared with key encapsulation mechanism, public key exchange protocol may have more security requirements beyond IND-CPA or IND-CCA security:

25

[055] Typically, public key exchange protocols may be stateless: The two parties Alice and Bob do not maintain a long term dynamic secret internal status, beyond the possible long term static private key. TLS Session Resumption (<https://tools.ietf.org/html/rfc5077>) is an example of stateful key exchange protocol, but is not widely adopted yet.

30

[056] We are interested in key exchange between two parties who want to communicate securely and frequently. The key encapsulation platform 100, 200 of Figures 1, 2 and 3 use a stateful hybrid key exchange protocol with good atomized complexity by combining existing key encapsulation mechanisms/public key exchange protocols. There may be one or more rounds of communication.

5 [057] Typically, there will be two different settings of Key Exchange Protocols, which may have different requirements in performance:

- Server-Client mode: In this setting, the goal is to maximize the number of concurrent secure connection to the server. Thus, in a key exchange protocol between a server and a client, we may attempt to distribute more workload to client side and reduce the burden on server side. Examples include HTTPS, SSH, and SFTP.
- Peer to Peer mode: In this setting, we may attempt to distribute almost equal workload to the two peers. For example, point to point encryption gateways between two data centres or two branch offices of the same organization.

15 [058] Our disclosed hybrid key exchange protocol refines and optimizes details in Table 2 to achieve better performance and/or security.

3.1 Generation of Private Keys

[059] In existing KEM combiners, one party (e.g. Alice) generates all private keys $sk \leftarrow \{sk_i\}_i$, and the other party (e.g. Bob) has access to the public keys $pk \leftarrow \{pk_i\}_i$. To generate a new shared key between Alice and Bob, in the first step, Bob will compute $(c, s) \leftarrow \text{Encaps}(pk)$ and passes c to Alice; in the second step, Alice will recover the same value $s \leftarrow \text{Decaps}(sk, c)$. This naive solution has some drawbacks:

- from security point of view, Alice did not authenticate the identity of Bob;
- from performance point of view, Alice's computation of Decaps has to start after Bob completes his computation of Encaps, since the input c of Decaps is the output of Encaps.

[060] Our solution distributes the responsibility and workload of key generation to both parties. Particularly, we define a subset $S \subset [0, n - 1]$ of indices, and denote its complement set as $\bar{S} \leftarrow [0, n - 1] \setminus S$. For each $i \in S$, Alice generates the private key $s_i \leftarrow \Phi_i.\text{KeyGen}(1^\lambda)$; for each $i \in \bar{S}$, Bob generates the private key $s_i \leftarrow \Phi_i.\text{KeyGen}(1^\lambda)$. To generate a shared key, Alice and Bob can compute in parallel as below

- in the first step, Bob and Alice independently computes $(c_i, s_i) \leftarrow \Phi_i.\text{Encaps}(pk_i)$ for $i \in S$ and $i \in \bar{S}$ respectively;
- in the second step, Alice and Bob independently computes $s_i \leftarrow \Phi_i.\text{Decaps}(sk_i, c_i)$ for $i \in S$ and $i \in \bar{S}$ respectively.

35 The next question is how to find a proper set S ? From security point of view, both set $\{\Phi_i: i \in S\}$ and the complement set $\{\Phi_i: i \in \bar{S}\}$ should contain KEM schemes Φ_i based on various hard problems in lattice, coding theory, multi-variable polynomial, etc. From performance point of view, the choice of

set S should roughly equally distribute the computation workload to Alice and Bob, in the peer to peer setting; or more workload to the client, in the server-client setting.

3.2 Delivery of Public Keys and Ciphertext

- 5 [061] In typical public key cryptography, the public key is simply made available to everyone, as the name ``public key" suggests. So is the ciphertext.
- [062] However, to allow KEM or KEP to work, it is not necessary to let any third party, beyond Alice and Bob, know about Alice's and Bob's public keys. Discussed in greater detail later, with reference to Figures 4 and 5, is a protocol that exchanges public keys between Alice and Bob using a secure channel,
- 10 established using another secure public key exchange protocol, to achieve two layer of defence, as set out in Table 3 below. This combination method can be treated as a generic series KEP combiner framework compared to the parallel KEM combiner framework.

Table 3: Series Combination of two Key Exchange Protocols Φ_0 and Φ_1

A1, B1: Alice and Bob interactively run key exchange protocol Φ_0 to establish a secure and authenticated channel with session key k_0. A2, B2: Within the secure and authenticated channel with session key k_0, Alice and Bob interactively run key exchange protocol Φ_1 to establish another secure channel with session key k_1. Then Alice and Bob will communicate securely over this secure channel with session key k_1. Both public key and ciphertext of Φ_1 will be delivered over the secure channel with session key k_0.
--

- 15 [063] In the series combination of key exchange protocol showed in Table 3, the ingredient KEP scheme Φ_0 and Φ_1 can be any key exchange protocol:
1. Key Exchange Protocol derived from Key Encapsulation Mechanism, or
 2. Quantum Key Distribution (QKD), or
 3. other ad-hoc KEP.
- 20 [064] In case that QKD and One Time Pad are applied to establish the first secure channel with session key k_0 , which is (quantum) informationally secure, adversary cannot get the public keys or ciphertext for the second KEP scheme Φ_1 from sniffed information, even with unlimited computation power, let alone the corresponding private keys. By combining the QKD and Post Quantum Crypto Key Exchange protocol in series combination framework (as in Table 3), such that QKD is applied only once to
- 25 securely deliver the public key and first ciphertext of the PQC KEP scheme, and the PQC KEP scheme runs again and again to generate sessions keys for different sessions, we achieve good balance between security and cost: The customer could enjoy quantum informational security by leasing the QKD hardware for a short time with much lower cost, rather than purchasing them.

3.3 Construction of Authenticated Key Exchange Protocol

[065] The Diffie-Hellman key exchange protocol is not authenticated and thus suffers from man-in-the-middle attack. If two parties communicate very frequently, starting from the second session, we could deliver all messages of the key exchange protocol using the secure and authenticated channel established in the previous session. For the first session, the authentication of each party can be done with help of public key infrastructure (PKI).

3.4 Derivation of a Session Key

[066] Let k_i denote the session key for the i -th session. Let y_{i+1} denote the output of KEM scheme for the $(i + 1)$ -th session. We derive the session key k_{i+1} as below

$$k_{i+1} \leftarrow h(k_0, \dots, k_i, y_{i+1}). \quad (17)$$

[067] The advantage of the above method is that, the current session key is dependent on all previous session keys, which are in turn dependent on all messages exchanged between Alice and Bob.

[068] The naive method to compute Equation (17), requires Alice and Bob to keep record all of previous session keys as an internal state, which will have linear storage complexity for internal secret status, and injure the forward secrecy.

[069] Our key exchange protocol solution of Figures 4 and 5 maintains only a constant size of aggregation value of all previous history session keys, from which we derive the new session key as defined in equation (17), as stated in assertion1 below.

Assertion 1: Let $h \in \{\text{SHA256}, \text{SHA512}, \text{SHA3}\}$. There exists 3 efficient functions f_0, f_1, f_2 with constant output size, such that, for any bit string x and y ,

$$h(x) = f_1(f_0(x))$$

$$f_0(x||y) = f_2(f_0(x), y)$$

[070] The above assertion is related to but not identical to the length extension attack on SHA2. It is a natural consequence of any efficient hash function which requires only constant memory and consume all input bits in one pass from left to right.

3.5 Choosing Ingredient KEM or KEP Schemes

[071] Typically, there are two reasons to combine multiple KEM/KEP schemes.

3.5.1 Construction of a secure KEM/KEP Scheme

[072] Since the hard problems behind post quantum cryptography are quite young, compared to factorization of large integer and discrete logarithm problem, one approach is to combine post quantum KEM schemes based on various quantum resistant hard problem assumptions in lattice, coding theory, etc. One may choose not only candidate schemes from on-going NIST competition, but also works in post quantum KEM/KEP, which are not submitted to the NIST competition (e.g. proposed after the deadline of NIST competition).

3.5.2 Compliance with Future Standards

[073] A NIST standard may come out tentatively between 2022 to 2024. At present, the NIST competition of post quantum cryptography is in its second round with 17 short listed candidate KEM/KEP proposals. We can combine all these KEM schemes with a classical KEM (e.g. RSA, DH) to comply with both current and future standards.

3.6 Achieving Forward/Backward Secrecy

[074] Just like ephemeral Diffie-Hellman key exchange protocol provides perfect forward secrecy, our KEM scheme can also achieve forward/backward secrecy by frequently refreshing the public/private key pairs for KEM scheme, especially for our second construction Ψ : In every session, run an ingredient KEM scheme and refresh its public/private key pair. So a fast key generation method is desirable for this purpose. The performance can be found in Table 18 and Table 20.

3.7 Variant Version of first construction Φ and S second construction Ψ Our Proposed Scheme

[075] Similar to RSA scheme, our first construction Φ in Section 2.1 and second construction Ψ in Section 2.2 cannot achieve forward or backward secrecy.

[076] To achieve forward/backward secrecy, one approach is to keep refreshing our public/private key pairs, just like Diffie-Hellman key exchange protocol. Let Ψ' denote a variant version of our second construction Ψ , such that all refinement and optimization ideas in this Section 3 applies and one party refreshes the public/private key pair for ingredient scheme $\Phi_{i \pmod n}$ at the beginning of each session i and sends the new public key to the other party via the existing secure and authenticated channel. We compare our scheme Ψ' with existing key exchange protocols from a security aspect in Table 4 and from a performance aspect in Table 5. The workload of 17 candidate PQC KEMs are distributed to client and server as in Table 6.

Table 4: Comparison of security features of various key exchange protocols against classical or quantum attacks

Scheme	Classical adversary					Quantum adversary				
	IND-CCA/CPA	Forward secrecy	Backward secrecy	Authentication	Comply with Standard	IND-CCA/CPA	Forward secrecy	Backward secrecy	Authentication	Comply with Standard
eDH	Yes	Yes	Yes	No	Yes	No	No	No	No	No
RSA	Yes	No	No	Yes	Yes	No	No	No	No	No
Hybrid ephemeral KEP(eDH + 1 PQC KEP)	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Probably no
Our ephemeral Hybrid KEP (eDH + 17 PQC KEP)	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Probably yes

eDH: ephemeral Diffie-Hellman

Table 5: Comparison of performance of various key exchange protocols

Scheme (NIST Security Level 1)	Latency (ms)	Computation time (ms) on Server side	Computation time (ms) on client side
ephemeral Diffie-Hellman (eDH)	19.24	19.24	19.24
RSA-KEM	175.46	175.46	0.84
Hybrid KEP(eDH + 1 fastest ephemeral PQC KEP (SABER))	19.40	19.38	19.40
Hybrid KEP(eDH + 1 slowest ephemeral PQC KEP (SIKE))	441.74	441.74	370.62
Our Hybrid KEP Ψ' (eDH + 17 ephemeral PQC KEP)	30.43 (amortized)	26.81 (amortized)	30.43 (amortized)

5

Table 6: The deployment mode of the 17 KEMS in Ψ'

Deployment Mode	KEMS
Encaps on client side Decaps on server side	KYBER, NewHope
Encaps on server side Decaps on client side	The other KEMS

3.7.1 Security Analysis

Theorem 9: Assume at least 1 out of n ingredient KEM schemes is secure against both classical and quantum attacks. The variant version Ψ' of our second construction achieves $\alpha(N)$ -forward secrecy and $\beta(N)$ -backward secrecy with

$$\alpha(N) = 1; \quad (20)$$

$$\beta(N) \geq 1 - (n - 1)/N. \quad (21)$$

against polynomial time classical or quantum adversary.

4 Integration with Existing Systems

[077] Integrating post-quantum cryptography (e.g. key exchange protocol) with existing widely deployed protocols, like TLS/SSL, IPsec, and SSH, may have a large impact on how quickly and widely post-quantum cryptography can be adopted.

20

[078] Herein disclosed are two simple and almost blackbox approaches to integrate any new Key Exchange Protocol (e.g. our hybrid key exchange protocol) with existing protocols, like TLS/SSL, SSH, and IPsec:

- One follows a parallel combiner framework and
- 5 The other follows a series combiner framework.

[079] First, we implement our key exchange protocol independently from existing protocols (e.g. TLS/SSL, IPsec): We decide the step by step interactions between two parties, and we choose how to represent our data (e.g. crypto key, ciphertext, parameters) and send them over Internet. Furthermore,
10 the implementation of our key exchange protocol is not necessary to change adaptively with every new version of implemented existing protocols.

4.1 Series Combination

[080] Figure 4 shows a schematic of our key exchange protocol in the series combiner framework
15 400. The procedure of a series combination using existing security protocol (e.g. TLS/SSL, SSH, IPsec) is described below.

- 1) Each terminal 402, 404 runs a first key exchange protocol (denoted as " $\Pi.kep$ ") to establish a first secure channel 406 (also denoted as " $\Pi.sec_channel(k_0)$ ") with a first shared key k_0 . If
20 the existing protocol adopts RSA or Diffie-Hellman as the first key exchange protocol, this first secure channel 406 will not be quantum-resistant.
- 2) Within the first secure channel 406 with the first shared key k_0 , the terminals 402, 404 run a second key exchange protocol KEP2 to establish a second shared key k_1 .
- 3) As a result, the terminals 402 and 404 establish a second secure channel 408 with a session key
25 k_2 derived from the first shared key k_0 and the second shared key k_1 . The two shared keys k_0 and k_1 may be combined, for example, using a key derivation function.
- 4) The terminals 402, 404 deliver their payload over the second secure channel 408 with the session key k_2

30 [081] From the above, each of the two terminals 402, 404 hosts a key exchange module configured to execute the key exchange protocol 400. From the perspective of the terminal 402, its key exchange module performs a first key exchange protocol ($\Pi.kep$) with the other terminal 404, so that the key exchange module of the terminal 402 and the key exchange module of the other terminal 404 each have a first shared key k_0 . A first secure channel 406 between the terminals 402 and 404 is established using
35 the first shared secret key k_0 . A second key exchange protocol KEP2 is performed with the other terminal 404 over the first secure channel 406 so that the key exchange module of the terminal 402 and the key exchange module of the other terminal 404 each have a second shared key k_1 . A session key is derived from the first shared secret key k_0 and the second shared secret key k_1 , whereby the session

key is used to establish a second secure channel 408 for payload communication between the two terminals 402 and 404.

4.2 Parallel Combination

5 [082] Figure 5 shows a schematic of our key exchange protocol in the parallel combiner framework 500. The procedure of a parallel combination using existing security protocol (e.g. TLS/SSL, SSH, IPsec) is described below.

- 10 1) Each terminal 402, 404 runs a first key exchange protocol (denoted as " $\Pi.kep$ ") to establish a first secure channel with a first shared key k_0 .
- 2) The terminals 402 and 404 run a second key exchange protocol KEP2 independently and concurrently to establish a second shared key k_1 .
- 3) As a result, the terminals 402 and 404 establish a second secure channel 508 with session key k_2 derived from the first shared key k_0 and the second shared key k_1 . The two shared keys k_0 and
15 k_1 may be combined, for example, using a key derivation function.
- 4) The terminals 402, 404 deliver their payload over the second secure channel 508 with the session key k_2 .

[083] In both series and parallel combinations, Phase 1 and Phase 4 are identical with the original protocols (TLS/SSL, SSH, IPsec, etc). An extra Phase 2 and 3 is introduced between the Phases 1 and
20 4. These two combinations share the same Phase 3, but have a different Phase 2. The series combination and parallel combination, described above, treat the implementation of existing security protocol (denoted as Π in Figures 4 and 5) as a blackbox, except that we will read and overwrite the session key of existing security protocol.

[084] Reading and overwriting of the session key of an existing security protocol may be performed
25 as follows. With reference to section 4.1, the first key exchange protocol of Phase 1 is considered performed and the first secure channel 406 is considered established when an existing secure channel, established using an existing session key, is present between the terminal 402 and the terminal 404. The key exchange module of either one of the terminals 402, 404 is configured to use the existing session key as the first shared key k_0 and use the existing secure channel as the first secure channel 406,
30 The second key exchange protocol of Phase 2 is performed over the existing secure channel to obtain the second shared key k_1 . Analogous to Phase 3, the existing secure channel is then replaced with the second secure channel 408, wherein the session key k_2 for the second secure channel 408 is derived from the existing session key and the second shared key k_1 .

35 5 Experimental Evaluations

[085] In this section, we evaluate performance of our hybrid KEMs/KEPs and compare them with related work F. Giacon, F. Heuer, and B. Poettering, "Kem combiners," in Public-Key Cryptography –

PKC 2018, 2018, pp. 190–218, T. Matsuda and J. Schuldt, “A new key encapsulation combiner,” in International Symposium on Information Theory and Its Applications (ISITA), 2018, pp. 698–702. Our hybrid KEM/KEP could combine one traditional KEM, say RSA KEM, and several candidate post-quantum KEM from second round of NIST competition of post-quantum cryptography.

- 5 [086] In our prototype of hybrid KEM/KEP, we chose all 17 candidate KEM proposals in the second round of NIST competition of post-quantum cryptography (<https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions>) and an RSA KEM as building blocks. The details of the RSA KEM are shown in Table 7.

10

Table 7: RSA KEM construction

Key Encap: $\langle ct, ss \rangle = \text{Encap}(pk)$	Key Decap: $ss = \text{Decap}(ct, sk)$
1 : $rr \leftarrow \text{RNG}$	
2 : $ct \leftarrow \text{RSA_encrypt}(rr, pk)$	1 : $rr \leftarrow \text{RSA_decrypt}(ct, sk)$
3 : $ss \leftarrow \text{SHA256}(rr)$	2 : $ss \leftarrow \text{SHA256}(rr)$

- [087] Each candidate KEM from NIST competition of PQC has several variants with different choices of parameters. As a result, our KEM combiner has many variants in practice. In our experiments, we evaluated only two of these variants named *PQCH1* and *PQCH5*, which represent combinations of building blocks with the lowest security strength (i.e. NIST Level 1) and the highest security strength (i.e. NIST Level 5), respectively. The detailed configuration of the two KEMs combiners are shown in Table 8 and Table 9.

20

Table 4: Building block selections of PQCH1-xxx-KEM

NIST PQC Round 2 Submissions	Variants/Parameters	Shared Key Size (Bits)	NIST Category
RSA KEM	RSA3072-SHA256	256	Level 1
BIKE ^[12]	BIKE1-128-CPA	256	Level 1
Classic McEliece ^[13]	mceliece348864	256	Level 1
CRYSTALS-KYBER	Kyber512	256	Level 1
FrodoKEM	FrodoKEM-640	128	Level 1
HQC	HQC-128-1	512	Level 1
LAC	LAC-128	256	Level 1
LEDAcrypt	LEDAcrypt-128-1	256	Level 1
NewHope	NewHope512-CPA	256	Level 1
NTRU	ntruhs2048509	256	Level 1
NTRU Prime	ntrupr653	256	Level 2
NTS-KEM	NTS-KEM(12,64)	256	Level 1
ROLLO	ROLLO-I-128	512	Level 1
Round5	R5N1_1KEM_0d	128	Level 1
RQC	RQC-I	512	Level 1
SABER	LightSaber-KEM	256	Level 1
SIKE	SIKEp434	128	Level 1
Three Bears	BabyBear	256	Level 2
Hybrid KEM	PQCH1	4,992 = 19.5 × 256	

Table 5: Building block selections of PQCH5-XXX-KEM

NIST PQC Round 2 Submission	Variants/Parameters	Shared Key Size (Bits)	NIST Category
RSA KEM	RSA15360-SHA256	256	Level 5
BIKE	BIKE1-256-CPA	256	Level 5
Classic McEliece	mceliece6688128	256	Level 5
CRYSTALS-KYBER	Kyber1024	256	Level 5
FrodoKEM	FrodoKEM-1344	256	Level 5
HQC	HQC-256-1	512	Level 5
LAC	LAC-256	256	Level 5
LEDACrypt	LEDACrypt-256-1	512	Level 5
NewHope	NewHope1024-CPA	256	Level 5
NTRU	ntruhs4096821	256	Level 5
NTRU Prime	ntrulpr857	256	Level 4
NTS-KEM	NTS-KEM(13,136)	256	Level 5
ROLLO	ROLLO-I-256	512	Level 5
Round5	RSN1_5KEM_0d	256	Level 5
RQC	RQC-III	512	Level 5
SABER	FireSaber-KEM	256	Level 5
SIKE	SIKEp751	256	Level 5
Three Bears	PapaBear	256	Level 5
Hybrid KEM	PQCH5	5,632 = 22 × 256	

[088] In our prototype, the key derivation function (KDF) of our schemes, denoted as “M257pX”, is the randomness blender function over finite field $GF(p)$ with Mersenne prime $p = 2^{257} - 1$, as defined in Lemma 2. Along with the two KDFs, SHA256 and M257pX, we have four KEM combiners in total, namely PQCH1-SHA256-KEM, PQCH1-M257pX-KEM, PQCH5-SHA256-KEM, and PQCH5-M257pX-KEM, where PQCH1-SHA256-KEM and PQCH5-SHA256-KEM are treated as the representative schemes in related work F. Giacon, F. Heuer, and B. Poettering, “Kem combiners,” in Public-Key Cryptography – PKC 2018, 2018, pp. 190–218, T. Matsuda and J. Schuldt, “A new key encapsulation combiner,” in International Symposium on Information Theory and Its Applications (ISITA), 2018, pp. 698–702. In addition, we also evaluate their performance in two modes, namely Φ and Ψ , as described in Section 2. So our experiments are performed essentially on 8 targets shown in Table 10.

15

Table 6: Naming of our schemes

Mode	KDF	Security Level	Proposed Hybrid KEM's Name
Φ	SHA256	NIST Level 1	PQCH1-SHA256-KEM
		NIST Level 5	PQCH5-SHA256-KEM
	M257pX	NIST Level 1	PQCH1-M257pX-KEM
		NIST Level 5	PQCH5-M257pX-KEM
Ψ	SHA256	NIST Level 1	PQCH1-SHA256-KEM- Ψ
		NIST Level 5	PQCH5-SHA256-KEM- Ψ
	M257pX	NIST Level 1	PQCH1-M257pX-KEM- Ψ
		NIST Level 5	PQCH5-M257pX-KEM- Ψ

[089] We remark that, the related works F. Giacon, F. Heuer, and B. Poettering, “Kem combiners,” in Public-Key Cryptography – PKC 2018, 2018, pp. 190–218, T. Matsuda and J. Schuldt, “A new key encapsulation combiner,” in International Symposium on Information Theory and Its Applications (ISITA), 2018, pp. 698–702 may adopt XOR function as KDF as showed in Table 1. However, the lengths of outputs of 17 candidate PQC KEM/KEP in second round of NIST competition, are not

20

identical, thus simple XOR of all 17 output bit-string of these candidate PQC KEM/KEP may not lead to a secure resulting KEM/KEP. So we replace $\text{XOR } \bigoplus_i x_i$ with hash $h(x_0|x_1|\dots)$ in our implementation of reference hybrid scheme. This treatment is reasonable for comparison of performance between our proposals and existing works, since for efficient KDF (like XOR, hash function, or our extractor or randomness blender function), the running time of KDF only takes an extremely small portion (e.g. smaller than 0.01%) of running time of the whole Decaps or Encaps algorithm, as confirmed in Table 11.

Table 7: PERFORMANCE OF PQCH1/5-SHA256/M257pX-KEM- Φ AND ITS KDF IN CPU CYCLE

	Encapsulation			Decapsulation		
	Total	KDF	KDF%	Total	KDF	KDF%
PQCH1-SHA256-KEM	1,656,699,869	245,170	0.0148%	1,512,655,832	245,170	0.0162%
PQCH1-M257pX-KEM	1,656,688,297	233,598	0.0141%	1,512,644,260	233,598	0.0154%
PQCH5-SHA256-KEM	7,295,282,612	531,164	0.0073%	7,209,529,868	531,164	0.0074%
PQCH5-M257pX-KEM	7,295,029,236	277,788	0.0038%	7,209,276,492	277,788	0.0039%

5.1 Experimental Set-ups

[090] Our experiments are performed on an Intel CPU. The CPU is equipped with 6 cores, but our experiments are performed only on a single CPU core.

[091] The source code of the 17 candidate KEMs are from NIST PQC Round 2 Submission, and the RSA KEM is implemented with RSA-OAEP and SHA-256 in OpenSSL v3.0.0. More details about the set-ups are listed in Table 12.

Table 8: Experimental set-ups

Name	Value
CPU	Intel Core i5-8500 @ 3.00 GHz
Memory Capacity	16 GB
Operating System	Linux 4.18.0 x86_64
Compiler	clang v7.0.1
Optimization Level	-O3

5.2 Experimental Results

5.2.1 PQCHx-SHA256-KEM vs. PQCHx-M257pX-KEM

[092] We evaluated the throughput (i.e. key generate rate) of our hybrid KEM with SHA256 and M257pX as its KDF, respectively. As showed in Table 15, PQCH1-M257pX-KEM gains speed-up of 18.56x and 18.54x respectively in Encaps and Decaps when compared with PQCH1-SHA256-KEM. At the same time, PQCH5-M257pX-KEM also gains much higher throughput than PQCH5-SHA256-KEM does, as shown in Table 16.

5.2.2 QCHx-M257pX-KEM vs. RSAx-SHA256-KEM

[093] We also compared the throughput of our hybrid KEMs with traditional KEMs, say RSA KEM as described in Table 7. As shown in Table 13 and Table 14, RSA KEMs with 3072 bits key gain much higher throughput than our hybrid KEMs. We remark that, some of the 17 candidate KEMs, like SIKE, Classic McEliece and Round5, are very time-consuming. For example, SIKE consumes more than 80% CPU cycles in PQCH1-M257pX-KEM. If it is not included in the hybrid KEMs, the throughput will be improved significantly.

5.2.3 PQCHx-xxx-KEM- Ψ

[094] Unlike in mode Φ , the hybrid KEMs in mode Ψ are stateful. It keeps a dynamic state. In each round, except the first one, only one candidate KEM runs to update the state, before a KDF outputs a shared key. Tables 17, 18, 19 and 20 show the performance of every round in this mode.

Table 9: Throughput and latency of RSA3072-SHA256-KEM and PQCH1-M257PX-KEM

KEM	Encapsulation (ms)	Decapsulation (ms)	Key Generation (ms)	Throughput (bps)
RSA3072-SHA256-KEM	0.14	3.63	265.32	67,905
PQCH1-M257pX-KEM	527.47	467.40	964.72	4,631

Table 10: Throughput and latency of RSA15360-SHA256-KEM and PQCH5-M257PX-KEM

KEM	Encapsulation (ms)	Decapsulation (ms)	Key Generation (ms)	Throughput (bps)
RSA15360-SHA256-KEM	0.97	178.14	61,843.89	1,429
PQCH5-M257pX-KEM	2,336.01	2,219.06	48,850.40	1,012

Table 11: Throughput (key generation rate) of PQCH1-SHA256-KEM and PQCH1-M257PX-KEM

	PQCH1-SHA256-KEM	PQCH1-M257pX-KEM	Speed-up ratio
Encap	463.56 bps	9,066.24 bps	18.56x
Decap	508.82 bps	9,944.45 bps	18.54x

Table 12: Throughput (key generation rate) of PQCH5-SHA256-KEM and PQCH5-M257PX-KEM

	PQCH5-SHA256-KEM	PQCH5-M257pX-KEM	Speed-up ratio
Encap	105.43 bps	2,321.14 bps	21.02x
Decap	106.76 bps	2,347.56 bps	20.99x

Table 13: Performance of PQCH1-SHA256-KEM- Ψ on our testbed

#Round	Running Components	Keypair Gen (CPU cycles)	Encap (CPU cycles)	Decap (CPU cycles)
0	All 18 KEMs + SHA256	2,390,137,940	1,591,766,461	1,404,702,143
1,19,37...	RSA3072-KEM + SHA256	641,517,718	694,329	11,241,2287
2,20,38...	BIKE1-128-CPA + SHA256	181,564	498,855	2,386,032
3,21,39...	FrodoKEM-640 + SHA256	1,505,725	9,581,588	9,517,207
4,22,40...	ntruhs2048509 + SHA256	5,168,458	797,193	1,440,639
5,23,41...	ntrupr653 + SHA256	15,337,196	28,633,984	42,647,700
6,24,42...	Kyber512 + SHA256	96,863	427,357	534,831
7,25,43...	LightSaber + SHA256	66,707	382,024	387,941
8,26,44...	NewHope512-CPA + SHA256	81,375	415,210	330,863
9,27,45...	HQC-128-1 + SHA256	284,381	962,155	1,473,931
10,28,46...	RQC-I + SHA256	577,339	1,332,349	5,456,514
11,29,47...	ROLLO-I-128 + SHA256	1,229,599	646,389	1,601,661
12,30,48...	LAC-128 + SHA256	271,663	776,838	1,053,696
13,31,49...	R5N1_KEM_0d + SHA256	488,055,171	495,873,684	6,192,681
14,32,50...	LEDACrypt-128-1 + SHA256	31,425,954	2,024,190	8,836,204
15,33,51...	NTS-KEM(12,64) + SHA256	144,639,183	577,656	4,774,845
16,34,52...	BabyBear + SHA256	504,226	1,017,000	1,761,595
17,35,53...	mceliece348864 + SHA256	332,683,415	474,676	44,251,048
18,36,54...	SIKEp434 + SHA256	743,551,151	1,046,866,793	1,261,130,331
1 - 18	Mean	133,732,093	88,443,459	78,056,608

Table 14: Performance of pqchl-m257px-kem- Ψ on our testbed

#Round	Running Components	Keypair Gen (CPU cycles)	Encap (CPU cycles)	Decap (CPU cycles)
0	All 18 KEMs + M257pX	2,382,522,623	1,584,404,820	1,407,962,720
1,19,37...	RSA3072-KEM + M257pX	950,336,794	412,970	10,908,323
2,20,38...	BIKE1-128-CPA + M257pX	182,663	220,986	2,219,171
3,21,39...	FrodoKEM-640 + M257pX	1,504,188	8,959,004	9,249,746
4,22,40...	ntruhs2048509 + M257pX	5,273,087	498,212	1,159,798
5,23,41...	ntrupr653 + M257pX	15,274,764	28,363,474	42,233,018
6,24,42...	Kyber512 + M257pX	97,340	145,367	164,839
7,25,43...	LightSaber + M257pX	67,977	100,359	104,331
8,26,44...	NewHope512-CPA + M257pX	83,394	133,520	47,022
9,27,45...	HQC-128-1 + M257pX	280,206	676,206	1,189,589
10,28,46...	RQC-I + M257pX	573,778	1,046,960	5,078,244
11,29,47...	ROLLO-I-128 + M257pX	1,225,143	367,861	1,319,040
12,30,48...	LAC-128 + M257pX	272,181	493,641	770,998
13,31,49...	R5N1_KEM_0d + M257pX	487,529,032	494,836,153	5,921,7461
14,32,50...	LEDACrypt-128-1 + M257pX	31,239,090	1,747,088	8,391,436
15,33,51...	NTS-KEM(12,64) + M257pX	146,269,843	300,845	4,502,248
16,34,52...	BabyBear + M257pX	503,478	735,022	1,478,099
17,35,53...	mceliece348864 + M257pX	337,022,509	195,131	44,602,601
18,36,54...	SIKEp434 + M257pX	740,952,652	1,043,300,145	1,267,288,893
1 - 18	Mean	151,038,228	87,918,496	78,146,063

5

[095] In conclusion, we gave the security formulation of key encapsulation mechanism or key exchange protocol, including IND-CCA/CPA security, forward/backward security, authentication, leakage resilience.

[096] We then disclosed generic constructions of two robust combiners of KEM. The two robust
 10 combiners for key encapsulation mechanisms (KEM), follow a parallel framework, by combining multiple KEMs together, in order to be more secure and comply with future NIST standard of post quantum cryptography. Our robust combiners are optimal in amortized time complexity. In our first construction of robust combiner, we follow a parallel framework using different key derivation function. We generalize key derivation functions in existing works (F. Giacon, F. Heuer, and B.
 15 Poettering, “Kem combiners,” in Public-Key Cryptography – PKC 2018, 2018, pp. 190–218; G.

Hanaoka, T. Matsuda, And J. C. N. Schuldt, “A new combiner for key encapsulation mechanisms,” IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, vol. E102.A, no. 12, pp. 1668–1675, 2019; and T. Matsuda and J. Schuldt, “A new key encapsulation combiner,” in International Symposium on Information Theory and Its Applications (ISITA), 2018, pp. 698–702) to randomness extractor. Furthermore, we use our randomness blender function as a better key derivation function, to achieve optimal amortized time complexity. In one approach, we use the combination of randomness extractor and secure pseudorandom function as the key derivation function. In our second construction of robust combiner, we modify the parallel framework of robust combiner of KEMs, so that we could run a single ingredient KEM scheme in turn, rather than all n ingredient KEM schemes, in every session. Compared to the first construction which generates $256 \cdot n$ bits shared secret string at once by revoking all of n ingredient KEM schemes, our second construction generates 256 bits shared secret string in every session by revoking a single ingredient KEM scheme in turn, out of the n KEM schemes.

[097] Adding more refinement and optimization in many aspects of details in KEM, we disclosed practical hybrid stateful key exchange protocols. We construct practical and secure hybrid stateful key exchange protocols based on our disclosed KEM schemes, together with refinement and optimization in many detailed aspects of key exchange protocols: We also gave suggestions on integration with existing solutions like TLS/SSL, IKEv2. We also implement our schemes and run various experiments and recorded the experiment data.

[098] While this invention has been described with reference to exemplary embodiments, it will be understood by those skilled in the art that various changes can be made and equivalents may be substituted for elements thereof, without departing from the spirit and scope of the invention. In addition, modification may be made to adapt the teachings of the invention to situations and materials, without departing from the essential scope of the invention. Thus, the invention is not limited to the examples that are disclosed in this specification, but encompasses all embodiments falling within the scope of the appended claims.

CLAIMS

1. A blender module configured to:
 - receive a plurality of input messages, with at least one having uniform randomness;
 - 5 aggregate the received plurality of input messages to propagate the uniform randomness of the at least one input message with uniform randomness; and
 - generate, in a randomness blender function mode:
 - a plurality of output messages, each having uniform randomness, wherein total entropy of the plurality of output messages is less than or equal to total entropy of all input messages; or
 - 10 when at least one of the plurality of input messages lacks uniform randomness, a plurality of output messages, each with uniform randomness and dependency amongst all output messages where the uniform randomness of the plurality of output messages has a total entropy less than or equal to the total entropy of all input messages; or
 - generate, in a randomness extractor mode:
 - 15 an output message with uniform randomness.
2. The blender module of claim 1, operating in the randomness blender function mode, wherein for an input of n messages and an output of n messages, if m of the input messages are independent and uniformly random then any subset m of the output messages will be independent and uniformly random,
 - 20 wherein $1 \leq m \leq n$.
3. A key encapsulation platform comprising:
 - a plurality of key encapsulation modules with at least one being secure, each of the key encapsulation modules configured to generate a plaintext and ciphertext pair, with the ciphertext
 - 25 resulting from encapsulating the plaintext using a public key; and
 - the blender module of claim 1 being in electrical communication with the plurality of key encapsulation modules to receive the plurality of plaintexts as its plurality of input messages,
 - wherein the platform is configured to:
 - compute at least one shared secret, from the plurality of output messages from the
 - 30 blender module, the plurality of output plaintexts being generated from the plurality of plaintexts; and
 - compute an encapsulation from the plurality of ciphertexts, the encapsulation for transmission and from which the at least one shared secret is derived.
- 35 4. The key encapsulation platform of claim 3, further configured to:
 - compute a plurality of ciphertexts from a received encapsulation,
 - wherein each of the key encapsulation modules is further configured to:

decapsulate one of the ciphertexts using a private key to obtain a plaintext, the private key corresponding to a public key used to encapsulate the plaintext, so that the blender module receives a plurality of the decapsulated plaintexts as its plurality of input messages, wherein the platform is further configured to:

5 compute at least one shared secret from the plurality of output messages from the blender module, the plurality of output messages being generated from the plurality of decapsulated plaintexts.

10 5. The key encapsulation platform of claim 4, wherein the received encapsulation and the encapsulation for transmission comprises a message authentication code for the shared secret.

15 6. The key encapsulation platform of claim 4, further configured to compute the shared secret in response to successful validation of the message authentication code, when the message authentication code is present in the received encapsulation.

20 7. The key encapsulation platform of any one of the claims 3 to 6, wherein the plurality of key encapsulation modules is further configured to simultaneously operate when providing the plurality of plaintexts to the blender module, whereby the platform computes a plurality of shared secrets from the simultaneous operation.

25 8. The key encapsulation platform of claim 7, wherein upon receipt of a refresh command, the plurality of key encapsulation modules is further configured to simultaneously operate to generate a plurality of new plaintext and ciphertext pairs from which the platform respectively computes a plurality of new shared secrets and a new encapsulation for transmission from which the plurality of new shared secrets are derived.

30 9. The key encapsulation platform of any one of the claims 3 to 8, wherein the blender module comprises a randomness blender function, constructed by combining a randomness extractor function and a pseudorandom function.

35 10. The key encapsulation platform of any one of the claims 3 to 8, wherein the blender module comprises a randomness blender function, constructed using a generalized Vandermonde matrix.

 11. The key encapsulation platform of claim 3 or 4, wherein during initialisation, the plurality of key encapsulation modules is configured to simultaneously operate to provide the plurality of plaintexts to the blender module, whereby the platform computes the at least one shared secret from the simultaneous operation.

12. The key encapsulation platform of claim 11, wherein upon receipt of a refresh command, each of the key encapsulation modules is configured to generate a new plaintext and ciphertext pair in turn, whereby the platform computes a new shared secret from the new plaintext and existing plaintexts and
5 computes a new encapsulation for transmission from the new ciphertext and existing ciphertexts.
13. The key encapsulation platform of claim 11 or 12, wherein the blender module comprises a randomness extractor function.
- 10 14. The key encapsulation platform of any one of the claims 3 to 13, wherein the shared secret is used to generate a shared session key for establishing a secure encryption channel for payload communication.
15. A key exchange module configured to:
15 perform a first key exchange protocol with a terminal for communication so that the key exchange module and the terminal each have a first shared key;
establish a first secure channel using the first shared key;
perform a second key exchange protocol with the terminal over the first secure channel so that the key exchange module and the terminal each have a second shared key; and
20 derive a session key from the first shared secret key and the second shared secret key, the session key being used to establish a second secure channel for payload communication.
16. The key exchange module of claim 15, wherein the first key exchange protocol is considered performed and the first secure channel is considered established when an existing secure channel,
25 established using an existing session key, is present between the key exchange module and the terminal, wherein the key exchange module is further configured to:
use the existing session key as the first shared key;
use the existing secure channel as the first secure channel, so that the second key exchange protocol is performed over the existing secure channel to obtain the second shared key; and
30 replace the existing secure channel with the second secure channel, wherein the session key for the second secure channel is derived from the existing session key and the second shared key.