Practical Quantum-Safe Stateful Hybrid Key Exchange Protocol

Jia Xu
NUS-Singtel Cyber Security R&D Lab
Email: jiaxu2001@gmail.com

Yiwen Gao
NUS-Singtel Cyber Security R&D Lab
Email: yiwen.gao@trustwave.com

Hoonwei Lim
NUS-Singtel Cyber Security R&D Lab
Email: hoonwei.lim@trustwave.com

***Abstract*—Shor's quantum algorithm, running in quantum computers, can efficiently solve integer factorization problem and discrete logarithm problem in polynomial time. This poses an urgent and serious threat to long-term security with recent accelerated evolution of quantum computing. However, National Institute of Standards and Technology (NIST) plans to release its standard of post-quantum cryptography between 2022 and 2024. It is crucially important to propose an early solution, which is likely secure against quantum attacks and classical attacks, and likely to comply with the future NIST standard. A robust combiner combines a set of 2 or more cryptography primitives into a new primitive of the same type, and guarantees that if anyone of the ingredient primitive is secure, then the resulting primitive is secure. This work proposes the first construction of robust combiner for Key Encapsulation Mechanism (KEM), with optimal amortized performance. From our robust combiner of KEMs, we construct efficient stateful hybrid Key Exchange Protocol (KEP), which is more suitable for two parties who will communicate with each other frequently.**

***Index Terms*—Key Exchange Protocol, Key Encapsulation Mechanism, Robust Combiner, Security and Performance, Parallel Combination, Series Combination**

1. Introduction

Quantum computers would bring critical threat to almost all widely adopted cryptography algorithms:

- All of public key cryptography algorithms (e.g. RSA, Diffie-Hellman, DSA and ECC) relying on hardness of factorization problem or discrete log problem, will be broken by Shor's algorithm in polynomial time with powerful quantum computers;
- all symmetric cryptography algorithms will 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.

In the face to quantum threats, the research community recommends to:

- phase out traditional public key cryptography algorithms, and switch to post-quantum cryptography (PQC), based on hardness of math problems in lattice, or coding theory, etc, and quantum key distribution (QKD), based on quantum physics theory;
- phase out symmetric ciphers with short keys (e.g. 128 bits).

NIST competition of post quantum cryptography started in 2017, and plans to select the winner PQC algorithms and publish the first draft standard of post quantum cryptography tentatively between 2022 and 2024.

Since the powerful quantum computer which is capable to execute Shor's algorithm or Grover's algorithm, is still under active development, the generic threat from quantum attacks seems to be in the future rather than on today. However, the quantum threat on long term security is immediate and urgent: The attackers can sniff and archive ciphertext of our today's data in the motion, and crack them when powerful quantum computer becomes available [1] in the future (say 10 years later), then we will fail to achieve long term (11+ years) security protection on our today's data in the motion. It is an urgent and important task to offer post quantum security from today, even before NIST standard or any other International or national standards of post quantum cryptography come out.

It is difficult and risky to predict which candidate scheme will win NIST competition of post quantum cryptography. A wise strategy, is to combine all of these candidate schemes, and make sure the combined scheme is secure if any one of the candidate scheme is secure. Such black-box combination is called "Robust Combiner", Dodis [6] and Harnik et al. [9] started to construct robust combiner for public key encryption and oblivious transfer, respectively, 14 years ago.

Combining n crypto primitives together in a naive way, will introduce n times overhead in complexity. In this work, we will propose new constructions of robust combiner for key encapsulation mechanism (or key exchange protocol), aiming to optimize the (amortized) performance while preserving security. We will evaluate the security property of proposed schemes under various security formulation, including (1)IND-CPA/CCA security formulation and (2)

This document has applied for provisional patent in March 2020.

1. For example, cloud service of quantum computing, like IBM Q System One https://www.ibm.com/quantum-computing/

forward/backward security and (3) leakage resilient cryptography. In contrast, the related works [7], [10] only evaluate security under IND-CPA/CCA security formulation.

1.1. Contributions

Our main contributions in this work can be summarized as below.

1) We propose a notion called "randomness blender function", as a special type of randomness extractor. We construct the first randomness blender function using *generalized vandermonde matrix*.
2) We propose two robust combiners for key encapsulation mechanisms (KEM), following parallel framework, by combining multiple KEMs [2] together, in order to be more secure and comply with future NIST standard of post quantum cryptography. Our robust combiners are almost optimal in amortized time complexity. Particularly, in existing works [7], [8], [10], the robust combiners of n number of ingredient KEM schemes, is about n times slower than the average of these n KEM schemes. In contrast, the amortized time complexity of our robust combiners is about as efficient as average of these n KEM schemes, indicating an almost n times speedup.
    - In our first construction of robust combiner, we follow the existing parallel framework [7] using different key derivation function (a.k.a core function in [7]). We generalize the key derivation function in existing works [7], [8], [10] to randomness extractor. Furthermore, we propose to use our randomness blender function as a better key derivation function, to achieve almost optimal amortized time complexity. In addition, we also propose to 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.
3) We construct practical and secure hybrid stateful key exchange protocols based on our proposed KEM schemes, together with refinement and optimization in many detailed aspects of key exchange protocols, which may have been overlooked in previous works: For example,
    - Should one party (Alice or Bob) generate all public/private key pairs for the n ingredient KEMs, or each party generates public/private key pairs for a different subset of the n ingredient KEMs?
    - How to deliver the public key and ciphertext to the other party?
    - How to achieve authentication, forward or backward secrecy?

    Our solution is more suitable for two parties who will communicate with each other frequently.
4) We formally define forward/backward secrecy and compression entropy, to capture security requirements on key exchange protocols from different angles. We compare our solutions with existing key exchange protocols from various angles of security, including IND-CCA/CPA security, authentication, forward/backward secrecy, and compression entropy.
5) We propose series framework for hybrid key exchange protocols, in addition to widely used parallel framework. We also propose almost-blackbox [3] construction to integrate our proposed schemes with existing solutions (e.g. TLS/SSL, IKEv2).
6) We implement our proposed solution and run experiments to evaluate the practical performance of our solutions and confirm the n times speedup, which is mentioned previously in our second contribution.

We emphasize that, our study of randomness blender function and compression entropy, may be of independent interests.

1.2. Organization

The rest of this paper is organized in this way: The next Section 2 discussed the related works. Section 3 provides the important definitions to characterize the security features of key encapsulation mechanism (KEM) or key exchange protocol (KEP). Then we give the two generic constructions of robust combiners of key encapsulation mechanism in Section 4. Next, we construct key exchange protocols in Section 5 by adding more optimizations to the robust combiners of KEM proposed in previous section. We also discuss how to integrate our solution with existing solutions (e.g. TLS, IKEv2). We provide our experiment data in Section 7. At the end, Section 8 closes this paper.

2. For example, the 17 PQC key encapsulation mechanism schemes in the second round of NIST competition of post quantum cryptography

3. Our solution is a blackbox approach, except that we need read and overwrite the session key of existing solutions, e.g. TLS or IKEv2.

2. Related Works

2.1. Robust Combiner

As early as 1980's, Even and Goldreich [12] started the research work of combining multiple symmetric ciphers. [6], [9], [21] studied combining multiple public key crypto schemes. Bindel *et al.* [2] combined multiple signature scheme.

Recently, Giacon *et al.* [7] gave a generic parallel framework for robust combiner of key encapsulation mechanisms. [1], [8], [10] extends the work on parallel combination of multiple KEM schemes. Here we give a refined and more complete description of this parallel framework.

2.1.1. Parallel Framework of Robust Combiner for KEMs. Let Φ_i's, $i \in [0, n-1]$ denote the n ingredient KEM scheme; let Φ denote the resulting robust combiner for KEM. The public/private keys of Φ will be a collection of all public/private keys generated by each ingredient KEM scheme Φ_i. In the encapsulation algorithm of Φ, we invoke encapsulation algorithm from each ingredient KEM Φ_i to generate (c_i, s_i), and then aggregate all of s_i, c_i together to compute the secret $s \leftarrow W(\ldots, s_i, \ldots, c_i, \ldots)$ for the combiner KEM Φ. Optionally, we may also compute a message authentication code $t \leftarrow T(\ldots, s_i, \ldots, c_i, \ldots)$ in order to achieve security under chosen ciphertext attack (CCA). To design CPA-secure robust combiner of KEM, we may simply ignore function T by treating it as $T(\cdots) = \perp$. [7] calls such key derivation function W as *core* function, while we add the *tag* function T into this generic framework.

2.2. Integrate with Existing System

Bos *et al.* [3] proposed a post-quantum key exchange protocols based on Ring-LWE problem and integrated it with TLS/SSL. Crockett, Paquin and Stebila [4] developed prototypes to integrate a subset of 10 post quantum key exchange protocols submitted to NIST Post-Quantum Cryptography Standardization competition, with OpenSSL and OpenSSH. [4] indicated that they failed to integrate some candidate post-quantum key exchange protocol with OpenSSL/OpenSSH, since the size of keys or ciphertext is larger than the maximum size defined in the corresponding RFC of TLS/SSL or SSH. Both works ([3], [4]) adopted a white-box approach for integration: They carefully analyse the structure and information flow in the TLS/SSL protocol (e.g. `ClientHello`, `ClientKeyExchange`, `ServerKeyExchange` messages), and have to hack or modify the implementation (e.g. OpenSSL): (1) Original TLS/SSL protocol only allows to use a single scheme for one purpose (e.g. Key Exchange, or Encryption of payload, or Authentication), and does not support hybrid model which executes two or more schemes for the same purpose in a single session; (2) Add post-quantum key exchange protocols to the "ciphersuite" of TLS/SSL; (3) Some implementation of TLS/SSL may introduce a limitation on the length of certain fields for storage of keys or ciphertext, which is smaller than the protocol specification in RFC. But some post-quantum key exchange protocol may have key or ciphertext size within the scope specified in RFC, but larger than the max size allowed in the implementation software of TLS/SSL. Similarly, a very recent work [14] by Sikeridis, Kampanakis and Devetsikiotis, integrated post-quantum signature scheme with TLS 1.3 with a white-box approach.

Based on existing works (e.g. [1], [3], [6], [7], [9], [12], [21]), Stebila and Fluhrer and Gueron [15], [16] proposed a draft for support of hybrid key exchange in TLS 1.3. In addition, Tjhai *et al.* proposed a draft for Internet Key Exchange Protocol Version 2 (IKEv2) [17].

3. Formulation

In this section, we introduce the important definitions in this work.

3.1. Robust Combiner

Definition 1 (Robust Combiner [9] (Informal)). 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 $\mathcal{P}$.

3.2. Security Formulation of Key Exchange Protocol

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 algorithm running in quantum computer.

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-CAP and IND-CCA for KEM is well known and can be found in [7], [10]. Unger *et al.* [18] summarized security features for secure message communications, including forward/backward secrecy, and stated that, "the terms are controversial and vague in literature [5]". In this work, we give a formal

definition of forward secrecy and backward secrecy for key exchange protocols.

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 fully 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 $\beta(N) = 1$.

3.3. Compression Entropy

Inspired by Yao's Entropy, and Xu and Zhou [19], we propose a notion of "Compression Entropy", to capture the security strength of leakage resilient cryptography against side-channel attack or covert channel attack.

Definition 4 (Compression Entropy or Extended Yao's Entropy). Let function $f : \{0,1\}^{m_0} \times \{0,1\}^{m_1} \to \{0,1\}^{n_0} \times \{0,1\}^{n_1}$ 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 $\Upsilon^{\mathcal{D}}_{\epsilon,t}(f_{m_0,n_0}) \geq k$, if for any t-time compression algorithm $\mathbb{C}$ and t-time decompression algorithm $\mathbb{D}$ with $\mathbb{C}(\cdots) \in \{0,1\}^l$

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

  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 $\Upsilon^{\mathcal{D}}_{\epsilon,t}(f) \leq l$, if there exist some t-time compression algorithm $\mathbb{C}$ and t-time decompression algorithm $\mathbb{D}$ with $\mathbb{C}(\cdots) \in \{0,1\}^l$,

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

  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

$$\upsilon^{\mathcal{D}}_{\epsilon,t} = \frac{\Upsilon^{\epsilon,t}_{\mathcal{D}}(f_{m_0,n_0})}{n_0} \qquad (3)$$

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

$$\Upsilon^{\mathcal{D}}_{\epsilon,t}(f) \leq \Upsilon^{\mathcal{D}}_{\epsilon+\Delta,t}(f). \qquad (4)$$

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

$$\Upsilon(I) = \mathbf{H}^{\mathtt{Yao}}(x) \qquad (5)$$

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

$$\mathbf{H}^{\mathtt{Yao}}_{\epsilon,t}(y) \geq \Upsilon^{\mathcal{D}}_{\epsilon,t}(f) \geq \mathbf{H}_{\infty}(y). \qquad (6)$$

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

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

$$\mathbf{H}^{\mathtt{Yao}}_{\epsilon,t}(y|a) = \Upsilon^{\mathcal{D}}_{\epsilon,t}(f). \qquad (7)$$

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

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

$$\mathbf{H}^{\mathtt{Yao}}(y) \geq poly(\lambda) \qquad (9)$$

3.4. Randomness Blender Function

We define *randomness blender function*, which (informally) converts an input string with k bits entropy *somewhere* to an output string with k bits entropy *everywhere*.

Definition 5 (Randomness Blender). Let $(a_0, \cdots, a_{n-1}) \in \{0,1\}^{\rho \cdot n}$ be a sequence of n independent random variables $a_i \in \{0,1\}^{\rho}$ and $(b_0, \cdots, b_{n-1}) = f(a_0, \cdots, a_{n-1}; r)$ with each $b_i \in \{0,1\}^{\rho'}$ $(\rho' \geq \rho)$. We say function f is (ϵ, δ)-Randomness Blender w.r.t. block size ρ, if the following conditions hold

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

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

$$\mathsf{F}(\vec{a}; r) = (\mathsf{P}_{\vec{a}}(r), \mathsf{P}_{\vec{a}}(r+1), \cdots, \mathsf{P}_{\vec{a}}(r+n-1)) \tag{10}$$

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

The above lemma can be proved from results in [13].

4. Key Encapsulation Mechanisms

Key Encapsulation Mechanism is a public key cryptography primitive, closely related to public key encryption. We can construct a key encapsulation mechanism by applying public key encryption on a (uniformly) random plaintext. In the other direction, we can also construct a public key encryption by simply combining key encapsulation mechanism and one-time pad cipher. Combination of the above two ideas is a generic way to construct CCA-secure PKE from CPA-secure PKE. Key encapsulation mechanism is also widely used as key exchange protocol. In Jan 2019, 17 proposals of KEM/PKE schemes are selected to enter the second round of NIST competition of post quantum cryptography.

Definition 6 (Key Encapsulation Mechanism). A KEM scheme consists of 3 algorithms $(\mathsf{Gen}, \mathsf{Encaps}, \mathsf{Decaps})$, described as below

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

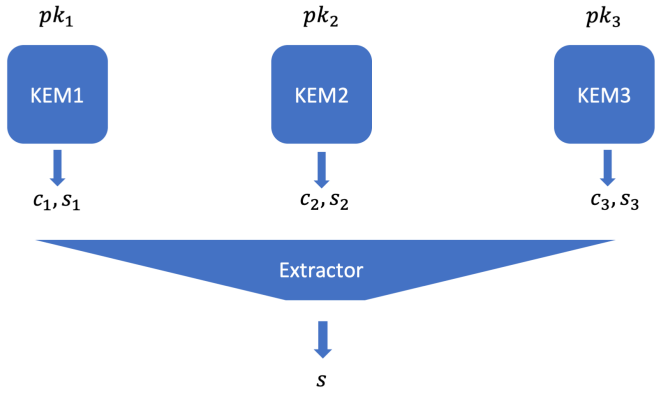
Given n number of ingredient KEM schemes Φ_i, $i = 0, 1, 2, \ldots, n-1$. We will propose two generic constructions, denoted as Φ and Ψ, of stateful KEM schemes, following the parallel framework with core [4] function (or key derivation function) F and G respectively, where functions F and G will be constructed and studied later in this paper. In fact, our proposed KEM schemes are robust combiners for KEM. Without loss of generality, we assume: (1) each KEM scheme Φ_i will output ℓ bits shared secret key (i.e. s) in every invocation; (2) the two communication parties, say Alice and Bob, 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. We aim to achieve good atomized complexity for two frequently contacted parties.

4.1. The First Generic Construction Φ with Key Derivation Function F

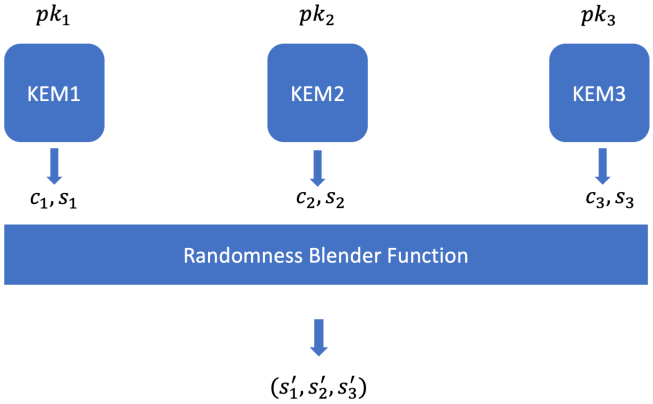
We denote this construction as $\Phi[F, n, m]$, which takes a function $F : \left(\{0,1\}^{\ell_0}\right)^n \times \{0,1\}^{\ell_0} \to \left(\{0,1\}^{\ell_1}\right)^m$, and n number of KEM schemes Φ_i's, $i \in [0, n-1]$, as building blocks, and generate and cache m number of bit-strings in $\{0,1\}^{\ell_1}$ at once. We remark that, in the below construction, optionally, a message authentication code will be generated for every ciphertext, in order to achieve security under chosen ciphertext attack (CCA) model.

4. Recall that the concept of parallel framework of robust combiner and core function are briefed in Section 2.

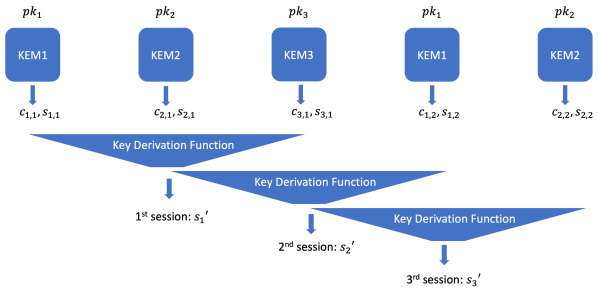
Figure 1. Illustration of Constructions of Robust Combiner from 3 Ingredient KEMs. Typically, the key derivation function is much more efficient than algorithm Gen, Encaps, Decaps in a KEM scheme.



(a) Robust combiner in existing works [7], [8], [10] adopts XOR function as an implicit randomness extractor



(b) Our first construction: This work proposes randomness blender function as key derivation function in robust combiner for KEMs



(c) Our second construction:This work modifies the parallel framework of robust combiner

We define the three algorithms $(\mathsf{Gen}, \mathsf{Encaps}, \mathsf{Decaps})$ as below. We also illustrate this construction in Figure 2(b).

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

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

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

- $\forall i \in [0, n-1]$, $(c_i, s_i) \leftarrow \Phi_i.\mathsf{Encaps}(pk_i)$.
- Compute $\bar{r} \leftarrow h_0(r, c_0, \ldots, c_{n-1})$.
- Compute $(x_0, \ldots, x_{m-1}) \leftarrow F(s_0, \ldots, s_{n-1}, \bar{r})$.
- Compute[5] $\sigma \leftarrow \mathsf{MAC}_{s_0,\ldots,s_{n-1}}(r, c_0, \ldots, c_{n-1})$.
- Compute $s \leftarrow (\ h_1(x_0), \ldots, h_1(x_{m-1})\ )$.
- Compute $c \leftarrow (\sigma, r, c_0, \ldots, c_{n-1})$.
- Output (s, c).

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

- $(\sigma, r, c_0, \ldots, c_{n-1}) \leftarrow c$.
- $\forall i \in [0, n-1]$, $s_i \leftarrow \Phi_i.\mathsf{Decaps}(sk_i, c_i)$.
- Compute $\bar{r} \leftarrow h_0(r, c_0, \ldots, c_{n-1})$.
- Compute $(x_0, \ldots, x_{m-1}) \leftarrow F(s_0, \ldots, s_{n-1}, \bar{r})$.
- If σ is a valid MAC for message $(r, c_0, \ldots, c_{n-1})$ w.r.t. key $(s_0, \ldots, s_{n-1})$, compute and output $s \leftarrow (h_1(x_0), \ldots, h_1(x_{m-1}))$. Otherwise, output $\perp$.

4.2. The Second Generic Construction Ψ with Key Derivation Function G

We denote this construction as $\Psi[G, n]$, which takes a function $G : \left(\{0,1\}^{\ell_0}\right)^n \times \{0,1\}^{\ell_0} \to \{0,1\}^{\ell_1}$ and n number of KEM schemes Φ_i's, $i \in [0, n-1]$, as building blocks.

We define the three algorithms $(\mathsf{Gen}, \mathsf{Encaps}, \mathsf{Decaps})$ as below. We also illustrate this construction in Figure 2(c).

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

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

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

- 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:
  - $\forall i \in [0, n-1]$, $(c_i, s_i) \leftarrow \Phi_i.\mathsf{Encaps}(pk_i)$.
  - Initialize $\text{CACHE} = (s_0, \ldots, s_{n-1})$.
  - Compute $s = h_1(G(\text{CACHE}, r))$.
  - Increment COUNT by 1 and output $c = (0, r, c_0, \ldots, c_{n-1})$ and s.
- If $\text{COUNT} \geq 1$:
  - Let $i = \text{COUNT} \pmod n$, and compute $(c'_i, s'_i) \leftarrow \Phi_i.\mathsf{Encaps}(pk_i)$.
  - Update the item s_i in CACHE: $s_i \leftarrow s'_i$.
  - Compute $s = h_1(G(\text{CACHE}; r))$.
  - Output $c = (\text{COUNT}, c'_i, r)$ and s. Increment COUNT by 1.

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

- This algorithm maintains a state variable CACHE which is a list of exactly n elements from $\{0,1\}^{\ell_1}$.
- If $c = (0, r, c_0, \ldots, c_{n-1})$:
  - $\forall i \in [0, n-1]$, $s_i \leftarrow \Phi_i.\mathsf{Decaps}(sk_i, c_i)$.
  - Initialize the state variable $\text{CACHE} \leftarrow (\ldots s_i \ldots)_{i\in[0,n-1]}$.
  - Compute $s = h_1(G(\text{CACHE}, r))$ and output s.
- If c is (COUNT, c'_i, r):
  - Let $i = \text{COUNT} \pmod n$ and compute $s'_i = \Phi_i.\mathsf{Decaps}(sk_i, c'_i)$.
  - Update the item s_i in CACHE: $s_i \leftarrow s'_i$.
  - Compute and output $s = h_1(G(\text{CACHE}, r))$.

4.3. Instantiation of Function F and G

We recommend SHA256, SHA512, or SHA3 function to take the role of hash function h_0, h_1 in the above constructions.

4.3.1. Randomness Extractor and Pseudorandom Function. We observe that, the collection of all secretes 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 random; otherwise, the value s_i is considered as a constant in the view of the adversary. Existing works [7], [8], [10] proposed to apply exclusive-or (XOR) operator to aggregate all secret values s_i's. In fact, XOR function is an extremely efficient randomness extractor for block-fixing distribution. Thus, these existing works (e.g. [7], [8], [10]) can be considered as a special case of our first construction $\Phi[\mathsf{Ext}, n, 1]$, where Ext denotes a randomness extractor.

In our second construction, one possible choice of function $G(\text{CACHE}, r)$ could be a randomness extractor function Ext with input random variable CACHE and random seed r.

$$G_0(\ldots, s_i, \ldots, r) = \mathsf{Ext}(\ldots, s_i, \ldots; r) \tag{11}$$

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

5. Optional step.

One possible choice for function F in our first construction could be a combination of randomness extractor and cryptographically secure pseudorandom function Prf.

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

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

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

4.3.2. Randomness Blender Function. An alternative choice of F and G could be randomness blender function, defined in this paper.

$$F_1(\ldots, s_i, \ldots, r) = \mathsf{Blender}(\ldots, s_i, \ldots, r) \quad (13)$$
$$G_1(\ldots, s_i, \ldots, r) = \mathsf{Blender}(\ldots, s_i, \ldots, r)[0] \quad (14)$$

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

We construct G_1 and F_1 as below.

$$G_1(\ldots, s_i, \ldots, r) = \sum_{i=0}^{n-1} s_i \cdot r^{i+1} \quad (15)$$
$$F_1(\ldots, s_i, \ldots, r) = (\ldots G_1(\ldots, s_i, \ldots, r+j)\ldots)_{j\in[0,t-1]}$$
$$= (\ldots, s_i, \ldots) \times \mathbf{M}_r \quad (16)$$

where the above computation is over a finite filed (e.g. $GF(p)$ with prime order p) and $\mathbf{M}_r$ is a *Generalized Vandermonde Matrix*. Note that, F_1 is just the randomness blender function defined in Lemma 2.

In our prototype implementation, we will 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.

4.4. Security

Theorem 3. Assume h_0 is a cryptographically secure hash function and h_1 be an identity function. Then $\Phi[\mathsf{Ext}, n, 1]$ 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.

Theorem 4. Assume h_0 is a cryptographically secure hash function and h_1 is a random oracle. Then $\Phi[\mathsf{Blender}, n, n]$ 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 $\mathsf{Blender}$.

Theorem 5. Assume h_0 is a cryptographically secure hash function and h_1 is a random oracle. Then $\Phi[\mathsf{Ext} + \mathsf{Prf}, n, n]$ 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 $\Psi[\mathsf{Blender}[0], n]$ 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 $\mathsf{Blender}[0]$ in the output vector of function $\mathsf{Blender}$.

Theorem 7. Let $f(sk; c) = (\Phi[\mathsf{Blender}, n, n].\mathsf{Decaps}(sk, c), \perp)$. The compression entropy $\Upsilon_{\epsilon,t}(f) \geq m \times \log(1/\epsilon)$ 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).

Theorem 8. Let $f(sk; c) = (\Phi[\mathsf{Ext} + \mathsf{Prf}, n, n].\mathsf{Decaps}(sk, c), \perp)$. The compression entropy $\Upsilon_{\epsilon,t}(f) \leq \ell_1$ 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).

The above two theorems shows 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.

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

5. Public Key Exchange Protocol

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).

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: Mutual authentication between the two parties and forward secrecy. According to Wikipedia [6], "TLS 1.3 leaves ephemeral Diffie–Hellman as the only key exchange mechanism to provide forward secrecy" [11] and "OpenSSL supports forward secrecy using elliptic curve Diffie–Hellman since version 1.0, with a computational overhead of approximately 15% for the initial handshake [7]".

Typically, in almost all real world applications, 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 [8] is an example of stateful

6. https://en.wikipedia.org/wiki/Forward_secrecy#Protocols

7. https://security.googleblog.com/2011/11/protecting-data-for-long-term-with.html

8. https://tools.ietf.org/html/rfc5077

TABLE 1. COMPARISON OF CONSTRUCTION OF KEM COMBINERS IN EXISTING WORKS.

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

Note: [1] only combines two ingredient KEM schemes. $\mathsf{dualPrf}(s_0, s_1)$ is random if either s_0 or s_1 is random. ROM: Random Oracle Model; SM: Standard Model; ICM: Ideal Cipher Mode; Prf: Pseudorandom Function; Kdf: Key Derivation Function

TABLE 2. PUBLIC KEY EXCHANGE BETWEEN ALICE AND BOB USING KEY ENCAPSULATION MECHANISM

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

key exchange protocol, but is not widely adopted yet. In this work, we are interested in key exchange between two parties who want to communicate securely and *frequently*, and will propose stateful hybrid key exchange protocol with good *atomized complexity* by combining existing key encapsulation mechanisms/public key exchange protocols. Our proposals may have one or more rounds of communication.

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.

The construction of key exchange protocol from key encapsulation mechanism in Table 2 looks very simple and straightforward. For hybrid key exchange protocol, we will explore details in Table 2 and attempt to refine and optimize them, in order to achieve better performance and/or security.

5.1. Who Generates the Private Keys?

Typically and possibly implicitly, in exiting KEM combiners [1], [7], [10], one party (e.g. Alice) will generate 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 $(c, s) \leftarrow \mathsf{Encaps}(pk)$ and passes c to Alice; in the second step, Alice will recover the same value $s \leftarrow \mathsf{Dec}(sk, c)$. This naive solution has some drawbacks:

- from security point of view, Alice didn't 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.

Our solution is to distribute 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 $s_i \leftarrow \Phi_i.\mathsf{KeyGen}(1^\lambda)$; for each $i \in \bar{S}$, Bob generates the private key $s_i \leftarrow \Phi_i.\mathsf{KeyGen}(1^\lambda)$. To generate a shared key, Alice and Bob can compute in parallel as below

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

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.

5.2. How to Deliver the Public Keys and Ciphertext?

In typical application of public key cryptography, the public key is simply made available to everyone, as the name

"public key" suggests. So is the ciphertext.

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. Therefore, we intend to exchange public keys between Alice and Bob using a secure channel, established using another secure public key exchange protocol, to achieve two layer of defence (See Table 3). This combination method can be treated as a generic *series KEP combiner framework* compared to the parallel KEM combiner framework [7], [10].

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.

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.

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 could not get the public keys or ciphertext for the second KEP scheme Φ_1, from sniffed information, even with unlimited computation power, not mention 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 for once to 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 could 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.

5.3. How to Construct Authenticated Key Exchange Protocol?

It is well known that, the diffie-hellman key exchange protocol, is not authenticated and thus suffering 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).

5.4. How to Derive a Session Key?

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 will derive the session key k_{i+1} as below

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

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.

The naive method to compute the above 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.

Our solution is to maintain only a constant size of aggregation value of all previous history session keys, from which we can derive the new session key as defined in the above equation (17). The key idea is stated in the below claim.

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

$$h(x) = f_1(f_0(x)) \tag{18}$$
$$f_0(x \| y) = f_2(f_0(x), y) \tag{19}$$

The above claim 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.

5.5. How to Choose Ingredient KEM or KEP Schemes?

Typically, there may be two reasons to combine multiple KEM/KEP schemes.

5.5.1. Construct a potentially more Secure KEM/KEP Scheme. Since the hard problems behind post quantum cryptography are quite young, compared to factorization of large integer and discrete log problem, it is a good idea 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 ongoing NIST competition, but also solid works (e.g. [3], [20]) in post quantum KEM/KEP, which are not submitted to NIST competition (e.g. proposed after the deadline of NIST competition).

5.5.2. Comply with the Future Standard. NIST standard may come out tentatively between 2022 to 2024. At the time of writing, the NIST competition of post quantum cryptography is in the second round with 17 short listed candidate KEM/KEP proposals. We can combine all of these KEM schemes together with some classical KEM (e.g. RSA, DH) to comply with both current and future standards. Note that, looking back the history of NIST competition of AES and SHA3, the winner candidate scheme is not allowed to make significant changes between its final standardized version and its submitted version, since necessary significant change is a hint of immature design.

5.6. How to Achieve Forward/Backward Secrecy

Just like ephemeral diffie-hellman key exchange protocol provides prefect forward secrecy, our KEM scheme can also achieve forward/backward secrecy by frequently refresh 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 perform can be found from Table 18 and Table 20.

5.7. Variant Version of Our Proposed Scheme

It is easy to see that, similar to RSA scheme, our first construction Φ in Section 4.1 and second construction Ψ in Section 4.2 cannot achieve forward or backward secrecy. To achieve forward/backward secrecy, we have to keep refreshing our public/private key pairs, just like Diffie-Hellman key exchange protocol. Let Ψ' denote the variant version of our second construction Ψ, such that, all refinement and optimization ideas in this Section 5 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 proposed scheme Ψ' with existing key exchange protocols from security aspect in Table 4 and from performance aspect in Table 5. Note that the workload of 17 candidate PQC KEMs are distributed to client and server as in Table 6.

5.7.1. 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

$$\alpha(N) = 1; \tag{20}$$

$$\beta(N) \geq 1 - (n-1)/N. \tag{21}$$

against polynomial time classical or quantum adversary.

6. Integrate with Existing Systems

How to integrate post-quantum cryptography (e.g. key exchange protocol) with existing widely deployed protocols, like TLS/SSL, IPsec, and SSH, is an interesting and important problem. It may have large impact on how quickly post-quantum cryptography can be adopted widely in real world applications, and benefit most users.

Recall that, in Section 2, we reviewed that existing works [3], [4], [15], [17] adopt a white-box strategy to integrate post quantum key exchange protocols with existing system like TLS/SSL and IKEv2 for IPsec. In this work, we suggest two simpler and almost blackbox approaches to integrate any new Key Exchange Protocol (e.g. our proposed hybrid key exchange protocol) with existing protocols, like TLS/SSL, SSH, and IPsec:

- One is following parallel combiner framework and
- the other is following series combiner framework.

Fist of all, we implement our proposed key exchange protocol independently from existing protocols (e.g. TLS/SSL, IPsec): We decide the actual 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, the implementation of our proposed key exchange protocol is not necessary to change adaptively with every new version of implementations of existing protocols.

6.1. Series Combination

The procedure of series combination of existing security protocol (e.g. TLS/SSL, SSH, IPsec) and our key exchange protocol is described as below and illustrated in Figure 3(a).

1) Alice and Bob run the key exchange part of existing protocol (denoted as "$\Pi.kep$" in Figure 3(a)) as usual to establish a secure channel with session key k_0 (denoted as "$\Pi.sec\_channel(k_0)$" in Figure 3(a)). In case that the existing protocol adopts RSA or Diffie-Hellman as key exchange protocol, this secure channel will not be quantum-resistant.
2) Within this secure channel with session key k_0, Alice and Bob run our implementation of new key exchange protocols (denoted as "KEP2" in Figure 3(a)) to establish another shared key k_1.
3) As a result, Alice and Bob establish another secure channel with session key k_2 derived [9] from both k_0 and k_1.
4) Alice and Bob deliver the payload over secure channel with session key k_2.

6.2. Parallel Combination

The procedure of parallel combination of existing security protocol (e.g. TLS/SSL, SSH, IPsec) and our key

9. We will combine the two keys k_0 and k_1 using some robust combiner.

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)

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

exchange protocol is described as below and illustrated in Figure 3(b).

1) Alice and Bob run the key exchange part of existing protocol (denoted as "$\Pi.kep$" in Figure 3(b)) as usual to establish a secure channel with session key k_0 (denoted as "$\Pi.sec\_channel(k_0)$" in Figure 3(b)).
2) Alice and Bob run our implementation of new key exchange protocols (denoted as "KEP2" in Figure 3(a)) *independently* and *concurrently* to establish another shared key k_1.
3) As a result, Alice and Bob establish another secure channel with session key k_2 derived [10] from both k_0 and k_1.
4) Alice and Bob deliver the payload over secure channel with session key k_2.

We remark that, in both series and parallel combinations, Phase 1 and Phase 4 are identical with the original protocols (TLS/SSL, SSH, IPsec, etc). We introduce extra Phase 2 and 3 in-between. These two combinations actually share the same Phase 3, and with only difference in Phase 2. In fact, our series combination and parallel combination described as above, treat the implementation of existing security protocol (denoted as Π in Figure 2) as a blackbox, except that we will read and overwrite the session key of existing security protocol.

10. We will combine the two keys k_0 and k_1 using some robust combiner.

7. Experimental Evaluations

In this section, we evaluate performance of the proposed hybrid KEMs/KEPs and compare them with related work [7], [10]. 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.

In our prototype of hybrid KEM/KEP, we choose all of 17 candidate KEM proposals in the second round of NIST competition of post-quantum cryptography [11] and a RSA KEM as building blocks. The details of the RSA KEM is shown in Table 7.

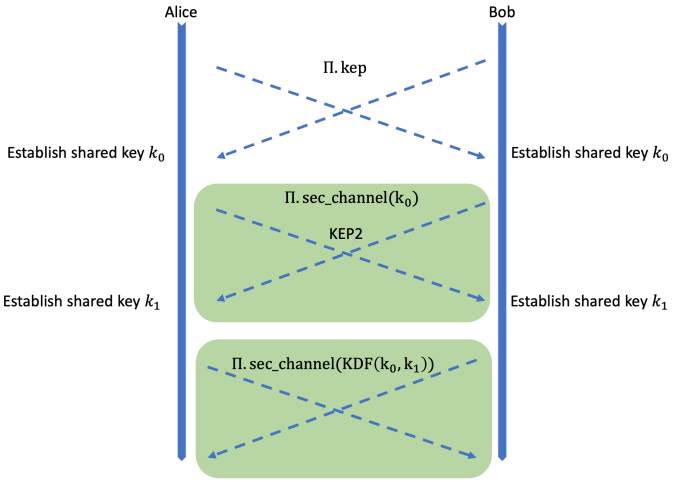
TABLE 7. RSA KEM CONSTRUCTION.

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

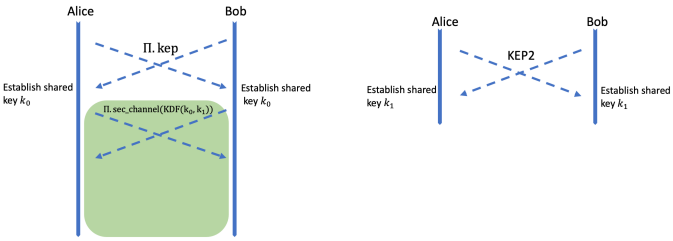
Each candidate KEM from NIST competition of PQC has several variants with different choices of parameters, as

11. https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions

Figure 2. Illustration of *almost-blackbox* combination of existing security protocol (e.g TLS, VPN, SSH, denoted as Π) and our key exchange protocol (denoted as "KEP2"). In the below pictures, a pair of dashed arrows between Alice and Bob indicates one or more rounds of communications. "KDF" stands for "key derivation function".



(a) Series Combination: Π.kep and KEP2 run in sequential order and KEP2 runs in the secure channel established by security protocol Π



(b) Parallel Combination: Π.kep and KEP2 run independently and concurrently

a result, our KEM combiner has many variants in practice. In our experiments, we evaluate 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.

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 [7], [10]. In addition, we also evaluate their performance in two modes, namely Φ and Ψ, as described in Section 4. So our experiments are performed essentially on 8 targets shown in Table 10.

We remark that, the related works [7], [10] 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 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 XOR $\bigoplus_i x_i$ with hash $h(x_0 \| x_1 \| \ldots)$ 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.

7.1. Experimental Set-ups

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

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.

7.2. Experimental Results

7.2.1. PQCHx-SHA256-KEM vs. PQCHx-M257pX-KEM. We evaluate the throughput (i.e. key generate rate) of the proposed 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.

7.2.2. PQCHx-M257pX-KEM vs. RSAx-SHA256-KEM. We also compare 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.

7.2.3. PQCHx-xxx-KEM-Ψ. 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. Table 17, 18, 19 and 20 show the performance of every round in this mode.

8. Conclusion

In this work, we gave the security formulation of key encapsulation mechanism or key exchange protocol, in-

TABLE 8. 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	ntruhps2048509	256	*Level 1*
NTRU Prime	ntrulpr653	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 $\times$ 256	

TABLE 9. 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	ntruhps4096821	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	R5N1_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 $\times$ 256	

TABLE 10. 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

cluding IND-CCA/CPA security, forward/backward security, authentication, leakage resilient. We then proposed generic constructions of robust combiners of KEM. Adding more refinement and optimization in many aspects of details in KEM, we proposed practical hybrid stateful key exchange protocols. We also gave suggestions how to integrate with existing solutions like TLS/SSL, IKEv2. We also implement our schemes and run various experiments, and record the experiment data.

Our hybrid KEM/KEP is optimal in performance, due to our first construction of randomness blender function. We also proposed the notion of compression entropy to measure the security strength of leakage resilient cryptography against side channel attack or covert channel attack. Our study of randomness blender function and compression entropy may have independent interest.

References

[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.

TABLE 11. 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%

TABLE 12. 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`

[2] N. Bindel, U. Herath, M. McKague, and D. Stebila, "Transitioning to a quantum-resistant public key infrastructure," in *Post-Quantum Cryptography — PQCrypto '17*, 2017, pp. 384–405.

[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.

[5] W. Diffie, P. C. V. Oorschot, and M. J. Wiener, "Authentication and authenticated key exchanges," *Designs, Codes and Cryptography*, vol. 2, no. 2, p. 107–125, 1992. [Online]. Available: https://doi.org/10.1007/BF00124891

[6] Y. Dodis and J. Katz, "Chosen-ciphertext security of multiple encryption," in *Proceedings of the Second International Conference on Theory of Cryptography*, ser. TCC'05, 2005, pp. 188–209.

[7] F. Giacon, F. Heuer, and B. Poettering, "Kem combiners," in *Public-Key Cryptography – PKC 2018*, 2018, pp. 190–218.

[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.

[9] D. Harnik, J. Kilian, M. Naor, O. Reingold, and A. Rosen, "On robust combiners for oblivious transfer and other primitives," in *Proceedings of the 24th Annual International Conference on Theory and Applications of Cryptographic Techniques*, ser. EUROCRYPT'05, 2005, pp. 96–113.

[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.

[11] E. Rescorla, " The Transport Layer Security (TLS) Protocol Version 1.3 (Internet Engineering Task Force)," August 2018. [Online]. Available: https://www.rfc-editor.org/rfc/pdfrfc/rfc8446.txt.pdf

[12] E. S. and G. O., "On the power of cascade ciphers," in *Advances in Cryptology — Crypto '84*, 1984, pp. 43–50. [Online]. Available: https://doi.org/10.1007/978-1-4684-4730-9_4

[13] I. Shparlinski, "On the singularity of generalised vandermonde matrices over finite fields," *Finite Fields and Their Applications*, vol. 11, no. number, pp. 193–199, April 2005.

[14] D. Sikeridis, P. Kampanakis, and M. Devetsikiotis, "Post-Quantum Authentication in TLS 1.3: A Performance Study," in *Network and Distributed Systems Security (NDSS) Symposium*, 2020. [Online]. Available: https://eprint.iacr.org/2020/071

[15] D. Stebila, S. Fluhrer, and S. Gueron, "Hybrid key exchange in TLS 1.3 (Work in Progress) ," Jan 2020. [Online]. Available: https://datatracker.ietf.org/doc/draft-stebila-tls-hybrid-design

[16] ——, "Design issues for hybrid key exchange in TLS 1.3 (Work in Progress) ," July 2019. [Online]. Available: https://tools.ietf.org/id/draft-stebila-tls-hybrid-design-01.html

[17] C. Tjhai, M. Tomlinson, G. Bartlett, S. Fluhrer, D. V. Geest, O. Garcia-Morchon, and V. Smyslov, "Framework to Integrate Post-quantum Key Exchanges into Internet Key Exchange Protocol Version 2 (IKEv2) (Work in Progress) ," July 2019. [Online]. Available: https://tools.ietf.org/html/draft-tjhai-ipsecme-hybrid-qske-ikev2-04

[18] N. Unger, S. Dechand, J. Bonneau, S. Fahl, H. Perl, I. Goldberg, and M. Smith, "Sok: Secure messaging," in *2015 IEEE Symposium on Security and Privacy*, 2015, pp. 232–249.

[19] J. Xu and J. Zhou, "Strong leakage-resilient encryption: enhancing data confidentiality by hiding partial ciphertext," 2020. [Online]. Available: https://doi.org/10.1007/s10207-020-00487-7

[20] Z. Yang, Y. Chen, and S. Luo, "Two-message key exchange with strong security from ideal lattices," Cryptology ePrint Archive, Report 2018/361, 2018, https://eprint.iacr.org/2018/361.

[21] R. Zhang, G. Hanaoka, J. Shikata, and H. Imai, "On the security of multiple encryption or cca-security+cca-security=cca-security?" in *Public Key Cryptography — PKC '04*, 2004, pp. 360–374. [Online]. Available: https://doi.org/10.1007/978-3-540-24632-9_26

TABLE 13. 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 14. 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 15. 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 16. 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 17. 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...	ntruhps2048509 + SHA256	5,168,458	797,193	1,440,639
5,23,41...	ntrulpr653 + 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_1KEM_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 18. PERFORMANCE OF PQCH1-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...	ntruhps2048509 + M257pX	5,273,087	498,212	1,159,798
5,23,41...	ntrulpr653 + 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_1KEM_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

TABLE 19. PERFORMANCE OF PQCH5-SHA256-KEM-Ψ ON OUR TESTBED.

#Round	Running Components	Encaps (CPU cycles)	Encaps (bps)	Decaps (CPU cycles)	Decaps (bps)
0	All 18 KEMs + SHA256	6,968,514,597	110.22	6,645,548,254	115.58
1,19,37...	RSA15360-KEM + SHA256	3,301,550	232,655.56	535,529,020	1,434.32
2,20,38...	BIKE1-256-CPA + SHA256	1,140,260	673,639.34	14,342,802	53,554.66
3,21,39...	FrodoKEM-1344 + SHA256	32,656,377	23,521.40	32,693,296	23,494.84
4,22,40...	ntruhps4096821 + SHA256	2,119,103	362,476.00	3,388,926	226,657.05
5,23,41...	ntrulpr857 + SHA256	50,027,474	15,354.04	74,623,333	10,293.34
6,24,42...	Kyber1024 + SHA256	929,246	826,609.95	830,982	924,356.96
7,25,43...	FireSaber + SHA256	728,748	1,054,032.38	724,096	1,060,804.08
8,26,44...	NewHope1024-CPA + SHA256	724,502	1,060,209.62	543,171	1,414,147.65
9,27,45...	HQC-256-1 + SHA256	3,239,258	237,129.61	4,241,794	181,084.70
10,28,46...	RQC-III + SHA256	4,570,153	168,074.02	19,990,419	38,424.60
11,29,47...	ROLLO-I-256 + SHA256	1,347,022	570,238.64	4,505,466	170,487.13
12,30,48...	LAC-256 + SHA256	2,290,661	335,328.53	3,384,723	226,938.51
13,31,49...	R5N1_5KEM_0d + SHA256	2,277,450,037	337.27	16,532,252	46,462.15
14,32,50...	LEDAcrypt-256-1 + SHA256	8,195,665	93,723.20	37,049,544	20,732.34
15,33,51...	NTS-KEM(13,136) + SHA256	1,406,714	546,041.33	19,796,587	38,800.82
16,34,52...	PapaBear + SHA256	2,805,759	273,766.91	5,257,472	146,101.39
17,35,53...	mceliece6688128 + SHA256	1,054,506	728,420.69	199,379,413	3,852.57
18,36,54...	SIKEp751 + SHA256	4,598,385,145	167.04	5,691,061,690	134.97

#Round	Statistics	Encaps	Decaps
1 - 18	Mean of Latency	129 ms	123 ms
1 - 18	Overall Throughput	1,977.33 bps	2,074.80 bps

TABLE 20. PERFORMANCE OF PQCH5-M257PX-KEM-Ψ ON OUR TESTBED.

#Round	Running Components	Encaps (CPU cycles)	Encaps (bps)	Decaps (CPU cycles)	Decaps (bps)
0	All 18 KEMs + M257pX	7,005,677,859	2,412.02	6,687,728,812	2,526.69
1,19,37...	RSA15360-KEM + M257pX	2,948,609	5,730,790.56	535,479,696	31,556.49
2,20,38...	BIKE1-256-CPA + M257pX	880,399	19,193,411.88	14,200,375	1,189,958.76
3,21,39...	FrodoKEM-1344 + M257pX	32,403,328	521,485.34	32,124,572	526,010.45
4,22,40...	ntruhps4096821 + M257pX	1,848,040	9,143,666.06	3,094,942	5,459,831.11
5,23,41...	ntrulpr857 + M257pX	49,411,919	341,979.44	74,170,298	227,825.16
6,24,42...	Kyber1024 + M257pX	553,790	30,513,119.83	564,958	29,909,941.32
7,25,43...	FireSaber + M257pX	462,318	36,550,297.91	462,665	36,522,885.09
8,26,44...	NewHope1024-CPA + M257pX	459,328	36,788,222.43	277,653	60,859,636.42
9,27,45...	HQC-256-1 + M257pX	2,845,994	5,937,419.62	4,070,303	4,151,499.44
10,28,46...	RQC-III + M257pX	4,295,625	3,933,737.37	19,975,630	845,923.78
11,29,47...	ROLLO-I-256 + M257pX	1,025,666	16,475,012.95	4,411,242	3,830,635.59
12,30,48...	LAC-256 + M257pX	2,131,010	7,929,507.90	3,013,607	5,607,187.87
13,31,49...	R5N1_5KEM_0d + M257pX	2,277,571,176	7,419.24	16,169,260	1,045,060.85
14,32,50...	LEDAcrypt-256-1 + M257pX	7,938,871	2,128,496.68	36,600,341	461,685.87
15,33,51...	NTS-KEM(13,136) + M257pX	1,145,431	14,752,403.79	19,117,274	883,905.34
16,34,52...	PapaBear + M257pX	2,541,585	6,648,552.23	5,090,449	3,319,522.62
17,35,53...	mceliece6688128 + M257pX	787,325	21,462,370.21	199,393,210	84,746.41
18,36,54...	SIKEp751 + M257pX	4,619,492,024	3,657.94	5,712,771,328	2,957.90

#Round	Statistics	Encaps	Decaps
1 - 18	Mean of Latency	129 ms	123 ms
1 - 18	Overall Throughput	43,397.44 bps	45,526.42 bps