CreTAKE

Credential-**T**yped **A**uthenticated **K**ey **E**xchange Suite

Authors: Xianhui Lu[1,2], luxianhui@iie.ac.cnn

Jingnan He[1,2], hejingnan@iie.ac.cn

Yao Cheng[1], chengyao@iie.ac.cn

Haiyang Xue[3], haiyangxue@smu.edu.sg

Yamin Liu[4], liuyamin3@huawei.com

Jiabo Wang[1,2], wangjiabo@iie.ac.cn

Organizations: 1. State Key Laboratory of Cyberspace Security Defense, Institute of Information Engineering, CAS

2. University of Chinese Academy of Sciences

3. Singapore Management University

4. Huawei Technologies

Postal Address: No. 19, Shucun Road, Beijing 100093, China

Contact: Jingnan He, hejingnan@iie.ac.cn

Contents

1 Introduction

Authenticated key exchange (AKE) establishes a shared session key while authenticating the communicating parties and is a fundamental component of secure-channel protocols. As post-quantum cryptographic algorithms move toward deployment, migration is unlikely to occur simultaneously across all endpoints and trust infrastructures [21, 6]. Protocol software may be updated relatively quickly, while certificate formats, credential issuance, trust stores, certification policies, and constrained devices often evolve on different schedules. Communicating parties may therefore hold different types of long-term credentials for an extended period. One endpoint may use a KEM-based credential while its peer continues to use a signature-based credential. This specification addresses this migration problem in the setting of two-message AKE, where mutual authentication and session-key establishment must be completed using one message in each direction.

Existing secure-channel protocols and AKE constructions generally adopt a fixed credential configuration. Signature-authenticated designs include SIGMA and its use in IKE [16, 15], as well as signature-authenticated TLS 1.3 [4]. Two-message implicit-authentication designs include MQV and HMQV under Diffie–Hellman assumptions [18, 17], as well as lattice-based counterparts under ideal-lattice assumptions [26]. KEM-based designs include the FSXY line of two-message AKE [7, 8, 9, 12] and application-level proposals such as KEMTLS and PQ-WireGuard [22, 5, 13]. These approaches provide important homogeneous or application-specific solutions, but they do not give one two-message AKE suite covering every ordered combination of KEM- and signature-based long-term credentials.

CreTAKE is designed to provide this complete coverage through two design layers. The *framework layer* specifies the AKE constructions and their security properties. The *instantiation layer* selects concrete post-quantum primitives and parameter sets to obtain deployable instances with different communication and computation profiles.

At the framework layer, let K denote a KEM-based long-term credential and S a signature-based long-term credential. The credential types of the initiator and responder define four two-message frameworks: CreTAKE-K2S, CreTAKE-S2K, CreTAKE-K2K, and CreTAKE-S2S. They cover the two heterogeneous credential orders and the two homogeneous configurations under a common $(\mathsf{KG}, \mathsf{Init}, \mathsf{Der}_{\mathsf{resp}}, \mathsf{Der}_{\mathsf{init}})$ interface.

The heterogeneous CreTAKE-K2S and CreTAKE-S2K frameworks originate from our earlier HetAKE work [19]. They support the two possible orders of KEM- and signature-based credentials and achieve the applicable IND-AA or IND-StAA security in the QROM using standard KEM and signature interfaces. Other mixed-credential constructions either require stronger primitive properties or additional long-term key material [14], or use an additional protocol round and consider a different exposure model [3].

The homogeneous CreTAKE-K2K framework builds on our earlier double-key KEM work [25]. That work introduced the 2KEM abstraction and showed two complementary realization routes. The generic route combines an IND-CCA-secure KEM with an IND-CPA-secure encapsulation and captures the separate-ciphertext structure of FSXY-style two-message AKE [7, 8]. The compact route exploits shared algebraic structure in the underlying primitive to combine ciphertext components. The CreTAKE-K2K framework uses 2KEM as a common interface for both routes,

so that generic and structure-aware compact constructions are represented within the same AKE framework.

The homogeneous CreTAKE-S2S framework provides a two-message construction for signature-based credentials. Standard mutually authenticated SIGMA uses three protocol messages, with the initiator completing authentication in the final flow [16]. In contrast, CreTAKE-S2S combines mutual signature authentication with an ephemeral wKEM exchange and derives the initiator's ephemeral key pair from its signing key and fresh session randomness. This structure permits the exchange to complete in two messages and supports the applicable session-state-reveal security analysis.

At the instantiation layer, the four frameworks are realized using PolarLAC, ZEN, and BiT at the 128-bit, 256-bit, and 512-bit security levels. These three primitive families are submitted separately as algorithm candidates and are used here as the concrete building blocks of the CreTAKE instance set. The two KEM families provide complementary public-key and ciphertext profiles, while BiT supplies the signature component in the signature-bearing frameworks. The CreTAKE-K2K instances include the generic double-key construction and, for the applicable 512-bit PolarLAC parameter sets, compact double-key constructions. These choices give 25 concrete CreTAKE instances, accompanied by reference and AVX2 implementations and performance evaluations.

The contributions of this specification are organized according to the two design layers.

- **Framework layer.** We specify a complete matrix of homogeneous and heterogeneous two-message AKE frameworks for KEM- and signature-based long-term credentials. The four frameworks share a common interface and achieve the applicable IND-AA or IND-StAA security in the QROM. This coverage allows a deployment to select an AKE framework according to the credentials currently provisioned at its two endpoints, making the suite suitable for asynchronous post-quantum migration as well as stable post-quantum deployments.

- **Instantiation layer.** We provide 25 concrete post-quantum instances by combining the four frameworks with PolarLAC, ZEN, and BiT at three security levels. The instance set includes complementary KEM communication profiles and both generic and compact CreTAKE-K2K realizations, providing alternatives for different bandwidth distributions, endpoint roles, and implementation environments. Reference and AVX2 implementations quantify their computation and communication costs.

Section 2 introduces the notation and notions used throughout the specification. Section 3 specifies the four frameworks and their concrete instances. Section 4 presents the framework- and instantiation-layer design rationale. Sections 5 and 6 give the security analyses and performance evaluation, respectively, and Section 7 summarizes the principal features of the suite.

2 Preliminaries

2.1 Two-Message AKE

Definition 1 (Two-Message AKE [12]). *A two-message AKE scheme* $\mathsf{AKE}=(\mathsf{KG}, \mathsf{Init}, \mathsf{Der}_{\mathsf{resp}}, \mathsf{Der}_{\mathsf{init}})$ *consists of the following four algorithms.*

- $\mathsf{KG}(i)$*: Taking a party identity* i *as input, the key generation algorithm outputs a key pair* (pk_i, sk_i)*.*
- $\mathsf{Init}(sk_i, pk_j)$*: Taking a secret key* sk_i *and a public key* pk_j *as input, the initialisation algorithm outputs a message* M *and a state* st*.*
- $\mathsf{Der}_{\mathsf{resp}}(sk_j, pk_i, M)$*: Taking a secret key* sk_j*, a public key* pk_i *and a message* M *as input, the responder derivation algorithm outputs a message* M' *and a session key* K'*.*
- $\mathsf{Der}_{\mathsf{init}}(sk_i, pk_j, M', st)$*: Taking a secret key* sk_i*, a public key* pk_j*, a message* M' *and a state* st *as input, the initiator derivation algorithm outputs a session key* K*.*

Definition 2 (IND-(St)AA security of AKE [12]). *We adopt the IND-(St)AA security experiment defined in [12]. The* IND-(St)AA *advantage function of an adversary* $\mathcal{A}$ *against AKE is defined as*

$$\mathrm{Adv}^{\mathsf{IND\text{-}(St)AA}}_{\mathsf{AKE}}(\mathcal{A}) := \left|\Pr\left[\mathsf{IND\text{-}(St)AA}^{\mathcal{A}} \Rightarrow 1\right] - \frac{1}{2}\right|.$$

2.2 PKE/KEM

Definition 3 (Key Encapsulation Mechanism (KEM)). *A KEM scheme* $\mathsf{KEM} = (\mathsf{Gen}, \mathsf{Encaps}, \mathsf{Decaps})$ *consists of a triple of polynomial-time algorithms.* Gen *is the key generation algorithm, which is probabilistic and outputs a public/secret key pair* (pk, sk)*. When the randomness is explicitly provided as input, the algorithm becomes deterministic.* Encaps *is the encapsulation algorithm, which is probabilistic and outputs a ciphertext* c *and a key* k *on the input* pk*.* Decaps *is the decapsulation algorithm, which is deterministic and outputs a key* k *on the inputs* sk, c*.*

Definition 4 (IND-CCA Security for KEM). *Let* $\mathsf{KEM} = (\mathsf{Gen}, \mathsf{Encaps}, \mathsf{Decaps})$ *be a KEM scheme. We define the* IND-CCA *advantage function of an adversary* $\mathcal{A}$ *against* KEM *as*

$$\begin{aligned}\mathrm{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathsf{KEM}}(\mathcal{A}) &:= \left|\Pr\left[\text{IND-CCA}^{\mathcal{A}}_{\mathsf{KEM}} \Rightarrow 1\right] - \frac{1}{2}\right| \\ &= \Bigg|\Pr\Big[b' = b : b' \leftarrow \mathcal{A}^{O_{\mathrm{Decaps}}}(pk, c^*, K^*_b), K^*_1 \xleftarrow{\$} \mathcal{K}, \\ &\qquad (c^*, K_0)^* \xleftarrow{\$} \mathsf{Encaps}(pk), b \xleftarrow{\$} \{0,1\}, (pk, sk) \xleftarrow{\$} \mathsf{Gen}\Big] - \frac{1}{2}\Bigg|\end{aligned}$$

where $O_{\mathrm{Decaps}}(c \neq c^*)$ *returns* $\mathrm{Decaps}(sk, c)$*.*

KEM *is* IND-CCA *secure if for any probabilistic polynomial time (PPT) adversary* $\mathcal{A}$*, the advantage* $\mathrm{Adv}^{\text{IND-CCA}}_{\mathsf{KEM}}(\mathcal{A})$ *is negligible.*

Specifically, KEM *is* IND-CPA *secure if the adversary* $\mathcal{A}$ *cannot access the decapsulation oracle* O_{Decaps}*.*

Definition 5 (Collision probability of key generation)**.** *We define it as*

$$\mu(\mathsf{Gen}) := \Pr[(pk, sk) \leftarrow \mathsf{Gen}, (pk', sk') \leftarrow \mathsf{Gen} : pk = pk']$$

Definition 6 (Collision probability of ciphertexts)**.** *We define*

$$\mu(\mathsf{Encaps}) := \Pr[(pk, sk) \leftarrow \mathsf{Gen}, (c, k) \leftarrow \mathsf{Encaps}(pk), (c', k') \leftarrow \mathsf{Encaps}(pk) : c = c']$$

Double-Key encryption and Double-Key key encapsulation mechanisms (KEMs) were proposed by Xue et al. [25] to better understand the underlying constructions of Authenticated Key Exchange (AKE). At a high level, these primitives involve encryption or encapsulation under two independent public keys. Notably, in some cases (where a public parameter may be shared between the two keys), the resulting construction can achieve a more compact structure than a naive combination of two separate ciphertexts.

Definition 7 (Double-Key KEM (2KEM) [25])**.** *A double-key KEM scheme 2KEM consists of a tuple of polynomial-time algorithms* $(\mathsf{Gen1}, \mathsf{Gen2}, \mathsf{2Encaps}, \mathsf{2Decaps})$.

- $\mathsf{Gen1}()$ *: output a pair of public-secret keys* (pk_1, sk_1)*. To show the randomness that is used, we denote key generation algorithm as* $\mathsf{Gen1}(r)$.
- $\mathsf{Gen2}()$ *: output a pair of public-secret keys* (pk_2, sk_2)*. To show the randomness that is used, we denote key generation algorithm as* $\mathsf{Gen2}(r)$.
- $\mathsf{2Encaps}(pk_1, pk_2)$ *: on input public keys* pk_1, pk_2*, output ciphertext* c *and encapsulated key* k *in key space* $\mathcal{K}$*. Sometimes, we explicitly add the randomness* r *and denote it as* $\mathsf{2Encaps}(pk_1, pk_2; r)$.
- $\mathsf{2Decaps}(sk_1, sk_2, c)$ *: on input secret keys* sk_1, sk_2*, output key* k *or* $\perp$.

Definition 8 ([IND-CCA, IND-CPA] security of 2KEM [25])**.** *Let* $\mathsf{2KEM} = (\mathsf{Gen1}, \mathsf{Gen2}, \mathsf{2Encaps}, \mathsf{2Decaps})$ *be a double-key KEM. We define the advantages of adversary* $\mathcal{A}$ *and* $\mathcal{B}$ *winning two games in the table.*

Game [IND-CCA, $\cdot$] on pk_1	**Game [$\cdot$, IND-CPA] on pk_2**
01 $(pk_1, sk_1) \leftarrow \mathsf{Gen1}()$	09 $(pk_2, sk_2) \leftarrow \mathsf{Gen2}()$
02 $L_2 = \{(-,-,-)\}$	10 $L_1 = \{(-,-,-)\}$
03 $(\text{state}, pk_2^*) \leftarrow \mathcal{A}_1^{\mathcal{O}_{\mathsf{cca}}, \mathcal{O}_{\mathsf{leak}_2}}(pk_1)$	11 $(\text{state}, pk_1^*) \leftarrow \mathcal{B}_1^{\mathcal{O}_{\mathsf{leak}_1}}(pk_2)$;
04 $b \leftarrow \{0,1\}$	12 $b \leftarrow \{0,1\}$
05 $(c^*, k_0^*) \leftarrow \mathsf{2Encaps}(pk_1, pk_2^*)$	13 $(c^*, k_0^*) \leftarrow \mathsf{2Encaps}(pk_1^*, pk_2)$
06 $k_1^* \leftarrow \mathcal{K}$	14 $k_1^* \leftarrow \mathcal{K}$
07 $b' \leftarrow \mathcal{A}_2^{\mathcal{O}_{\mathsf{cca}}, \mathcal{O}_{\mathsf{leak}_2}}(\text{state}, c^*, k_b^*)$	15 $b' \leftarrow \mathcal{B}_2^{\mathcal{O}_{\mathsf{leak}_1}}(\text{state}, c^*, k_b^*)$
08 return $b' \stackrel{?}{=} b$	16 return $b' \stackrel{?}{=} b$

- $\mathcal{O}_{\mathsf{leak}_2}$*: in i-th query, run* $(pk_2^i, sk_2^i) \leftarrow \mathsf{Gen2}(r_2^i)$*, return* (pk_2^i, sk_2^i, r_2^i) *and set* $L_2 = L_2 \cup \{(pk_2^i, sk_2^i, r_2^i)\}$.

- $\mathcal{O}_{\mathsf{leak}_1}$*: in i-th quiry, run* $(pk_1^i, sk_1^i) \leftarrow \mathsf{Gen1}(r_1^i)$*, return* (pk_1^i, sk_1^i, r_1^i) *and set* $L_1 = L_1 \cup \{(pk_1^i, sk_1^i, r_1^i)\}$.
- $\mathcal{O}_{\mathsf{cca}}(pk_2, c)$*: if* $\exists (pk_2, sk_2, \cdot) \in L_2$ *and* $pk_2 \neq pk_2^* \vee c \neq c^*$ *return* $\mathsf{2Decaps}(sk_1, sk_2, c)$*, otherwise* $\perp$.

The advantage of $\mathcal{A}$ *and* $\mathcal{B}$ *are defined as*

$$\mathrm{Adv}_{\mathsf{2KEM}}^{[\mathsf{IND\text{-}CCA},\cdot]}(\mathcal{A}) := \left|\Pr[[\text{IND-CCA}, \cdot]_{\mathsf{2KEM}}^{\mathcal{A}} \Rightarrow 1] - \frac{1}{2}\right|,$$

$$\mathrm{Adv}_{\mathsf{2KEM}}^{[\cdot,\mathsf{IND\text{-}CPA}]}(\mathcal{A}) := \left|\Pr[[\cdot, \text{IND-CPA}]_{\mathsf{2KEM}}^{\mathcal{B}} \Rightarrow 1] - \frac{1}{2}\right|.$$

We say that 2KEM *is [IND-CCA, IND-CPA] secure, if both advantages are negligible for both PPT adversary* $\mathcal{A}$ *and* $\mathcal{B}$.

Remark 1. As demonstrated in [25, 24], a [IND-CCA, IND-CPA] -secure 2KEM can be constructed by simply combining an IND-CCA-secure KEM and an IND-CPA-secure KEM. In this generic construction, the final key is derived by hashing both encapsulated keys along with the first public key in the (quantum) random oracle model. Specifically, if the IND-CCA-secure encapsulated key is k_1 under public key pk_1, and the IND-CPA-secure encapsulated key is k_2, the final key is $k = h(pk_1, k_1, k_2)$ where h is a random oracle.

For lattice-based instantiations [25], the second components of both ciphertexts can be summed together. While this technique reduces the overall ciphertext size, it inherently increases the decryption failure rate (DFR). Consequently, this optimization is applicable when there is sufficient margin to tolerate a higher DFR. We present such a construction for Polar-LAC in Section 3.2.1.

The collision probability of key generation and ciphertext can be defined similarly.

Definition 9 (Collision probability of ciphertexts)**.** *We define*

$$\begin{aligned}\mu(\mathsf{2Encaps}) := \max\{&\Pr_{pk_2}[(pk_1, sk_1) \leftarrow \mathsf{Gen1}, (c, k) \leftarrow \mathsf{2Encaps}(pk_1, pk_2), (c', k') \leftarrow \mathsf{2Encaps}(pk_1, pk_2) : c = c'],\\ &\Pr_{pk_1}[(pk_2, sk_2) \leftarrow \mathsf{Gen2}, (c, k) \leftarrow \mathsf{2Encaps}(pk_1, pk_2), (c', k') \leftarrow \mathsf{2Encaps}(pk_1, pk_2) : c = c']\}\end{aligned}$$

2.3 Signature

Definition 10 (Signature (SIG))**.** *A signature scheme* $\mathsf{SIG} = (\mathsf{SGen}, \mathsf{Sign}, \mathsf{Ver})$ *consists of a triple of polynomial-time algorithms.* SGen *is the key generation algorithm, which is probabilistic and outputs a public/secret key pair* (pk, sk)*.* Sign *is the signing algorithm, which is probabilistic and outputs a signature* σ *on the inputs* sk, m*, where* m *is a message.* Ver *is the verification algorithm, which is deterministic and outputs* $0/1$ *on the inputs* pk, m, σ.

Definition 11 (EUF-CMA Security for SIG)**.** *Let* $\mathsf{SIG} = (\mathsf{SGen}, \mathsf{Sign}, \mathsf{Ver})$ *be a SIG scheme. We define the* EUF-CMA *advantage function of an adversary* $\mathcal{A}$ *against SIG as*

$$\mathrm{Adv}_{\mathsf{SIG}}^{\mathsf{EUF\text{-}CMA}}(\mathcal{A}) := \Pr[\mathsf{Ver}(pk, m, \sigma) = 1 \wedge m \notin M_{\mathsf{Sign}}, (m, \sigma) \leftarrow \mathcal{A}^{O_{\mathsf{Sign}}}(pk), (pk, sk) \leftarrow \mathsf{SGen}]$$

where $O_{\mathsf{Sign}}(m)$ *returns* $\mathsf{Sign}(sk, m)$ *in response to a query, and* M_{Sign} *denotes the set of messages that have been queried to* O_{Sign}.

SIG *is* EUF-CMA *secure if for any PPT adversary* $\mathcal{A}$*, the advantage* $\mathrm{Adv}^{\text{EUF-CMA}}_{\mathsf{SIG}}(\mathcal{A})$ *is negligible.*

3 Algorithm Description

3.1 CreTAKE Frameworks

Let $\mathsf{wKEM} = (\mathsf{wGen}, \mathsf{wEncaps}, \mathsf{wDecaps})$ be an IND-CPA secure KEM scheme with key space $\mathcal{K}$, $\mathsf{KEM} = (\mathsf{Gen}, \mathsf{Encaps}, \mathsf{Decaps})$ be an IND-CCA secure KEM scheme with key space $\mathcal{K}$, and let $\mathsf{2KEM} = (\mathsf{Gen1}, \mathsf{Gen2}, \mathsf{2Encaps}, \mathsf{2Decaps})$ be an [IND-CCA, IND-CPA] secure double-key KEM. Let $\mathsf{SIG} = (\mathsf{SGen}, \mathsf{Sign}, \mathsf{Ver})$ be an EUF-CMA secure signature scheme. Let $H : \{0,1\}^* \to \mathcal{Y}$ and $G : \{0,1\}^* \to \mathcal{R}$ be random oracles.

3.1.1 CreTAKE-K2S

The algorithms of the AKE scheme in the CreTAKE-K2S framework $\mathsf{AKE} = \mathsf{F}_{\mathsf{K2S}}[\mathsf{wKEM}, \mathsf{KEM}, \mathsf{SIG}, H] = (\mathsf{KG}, \mathsf{Init}, \mathsf{Der}_{\mathsf{resp}}, \mathsf{Der}_{\mathsf{init}})$ are defined in Figure 1.

$\mathsf{KG}(i)(i \in [2N])$:

1 : **if** $i \in [N]$ // for initiator
2 : $\quad \mathsf{KG}_{\mathsf{init}}(i) : (sk_i, pk_i) \leftarrow \mathsf{Gen}$
3 : **return** (sk_i, pk_i)
4 : **if** $i \in [N+1, 2N]$ // for responder
5 : $\quad \mathsf{KG}_{\mathsf{resp}}(i) : (sk_i, pk_i) \leftarrow \mathsf{SGen}$
6 : **return** (sk_i, pk_i)

$\mathsf{Init}(sk_i, pk_j)(i \in [N], j \in [N+1, 2N])$:

1 : $\widetilde{sk}, \widetilde{pk} \leftarrow \mathsf{wGen}$
2 : $M := \widetilde{pk}$
3 : $st := \widetilde{sk} \| \widetilde{pk}$
4 : **return** (M, st)

$\mathsf{Der}_{\mathsf{resp}}(sk_j, pk_i, M)(i \in [N], j \in [N+1, 2N])$:

1 : $\widetilde{pk} := M$
2 : $\widetilde{c}, \widetilde{k} = \mathsf{wEncaps}(\widetilde{pk})$
3 : $c_i, k_i = \mathsf{Encaps}(pk_i)$
4 : $\sigma := \mathsf{Sign}(sk_j, \widetilde{pk} \| \widetilde{c} \| c_i)$
5 : $M' := \widetilde{c} \| c_i \| \sigma$
6 : $K' = H(pk_i, pk_j, \widetilde{pk}, \widetilde{c}, c_i, k_i, \widetilde{k})$
7 : **return** (M', K')

$\mathsf{Der}_{\mathsf{init}}(sk_i, pk_j, M', st)(i \in [N], j \in [N+1, 2N])$:

1 : $\widetilde{c}, c_i, \sigma := M'$; $\quad \widetilde{sk}, \widetilde{pk} := st$
2 : **if** $\mathsf{Ver}(pk_j, \widetilde{pk} \| \widetilde{c} \| c_i, \sigma) = 0$
3 : $\quad$ **return** $\perp$
4 : $\widetilde{k} = \mathsf{wDecaps}(\widetilde{sk}, \widetilde{c})$
5 : $k_i = \mathsf{Decaps}(sk_i, c_i)$
6 : $K = H(pk_i, pk_j, \widetilde{pk}, \widetilde{c}, c_i, k_i, \widetilde{k})$
7 : **return** K

Figure 1: CreTAKE-K2S Framework

The CreTAKE-K2S framework considers N initiators, each equipped with a long-term public-private key pair based on a KEM scheme, and N responders, each equipped with a long-term public-private key pair based on a SIG scheme. These key pairs are generated by the key generation algorithms $\mathsf{KG}_{\mathsf{init}}$ and $\mathsf{KG}_{\mathsf{resp}}$, respectively, as part of the overall KG procedure. The process of negotiation of a session key between initiator i and responder j is as follows. The

initiator generates the ephemeral key pair of wKEM, stores the ephemeral secret key in its state, and sends the ephemeral public key $\widetilde{pk}$ to the intended responder. The responder then runs wEncaps on the ephemeral public key $\widetilde{pk}$, obtaining a ciphertext $\widetilde{c}$ and a key $\widetilde{k}$. Next, it executes Encaps on the initiator's public key pk_i, generating a ciphertext c_i and a key k_i. The responder signs the concatenation $\widetilde{pk}\|\widetilde{c}\|c_i$ and transmits $\widetilde{c}$, c_i, and the signature σ to the initiator. It then computes the shared key K using the hash function H with common parameters $pk_i, pk_j, \widetilde{pk}, \widetilde{c}, c_i$ and the keys $k_i, \widetilde{k}$. Upon receiving the message, the initiator verifies the signature and runs wDecaps and Decaps to retrieve the keys k_i and $\widetilde{k}$. Finally, it derives the shared key using the hash function H, obtaining the same value K.

3.1.2 CreTAKE-S2K

The algorithms of the AKE scheme in the CreTAKE-S2K framework $\mathsf{AKE} = \mathsf{F}_{\mathsf{S2K}}$ [wKEM, KEM, SIG, H, G] = (KG, Init, $\mathsf{Der}_{\mathsf{resp}}$, $\mathsf{Der}_{\mathsf{init}}$) are defined in Figure 2.

$\mathsf{KG}(i)(i \in [2N])$:

1 : **if** $i \in [N]$//for initiator
2 : $\quad \mathsf{KG}_{\mathsf{init}}(i) : (sk_i, pk_i) \leftarrow \mathsf{SGen}$
3 : **return** (sk_i, pk_i)
4 : **if** $i \in [N+1, 2N]$//for responder
5 : $\quad \mathsf{KG}_{\mathsf{resp}}(i) : (sk_i, pk_i) \leftarrow \mathsf{Gen}$
6 : **return** (sk_i, pk_i)

$\mathsf{Init}(sk_i, pk_j)(i \in [N], j \in [N+1, 2N])$:

1 : $\widetilde{r} \xleftarrow{\$} \mathcal{R}$
2 : $\widetilde{sk}, \widetilde{pk} := \mathsf{wGen}(G(sk_i, \widetilde{r}))$
3 : $c_j, k_j = \mathsf{Encaps}(pk_j)$
4 : $\sigma := \mathsf{Sign}(sk_i, \widetilde{pk}\|c_j)$
5 : $M := \widetilde{pk}\|c_j\|\sigma$
6 : $st := \widetilde{r}\|c_j\|k_j$
7 : **return** (M, st)

$\mathsf{Der}_{\mathsf{resp}}(sk_j, pk_i, M)(i \in [N], j \in [N+1, 2N])$:

1 : $\widetilde{pk}, c_j, \sigma := M$
2 : **if** $\mathsf{Ver}(pk_i, \widetilde{pk}\|c_j, \sigma) = 0$
3 : $\quad$ **return** $\perp$
4 : $\widetilde{c}, \widetilde{k} = \mathsf{wEncaps}(\widetilde{pk})$
5 : $k_j = \mathsf{Decaps}(sk_j, c_j)$
6 : $M' := \widetilde{c}$
7 : $K' = H(pk_i, pk_j, \widetilde{pk}, \widetilde{c}, c_j, k_j, \widetilde{k})$
8 : **return** (M', K')

$\mathsf{Der}_{\mathsf{init}}(sk_i, pk_j, M', st)(i \in [N], j \in [N+1, 2N])$:

1 : $\widetilde{c} := M'$; $\quad \widetilde{r}, c_j, k_j := st$
2 : $\widetilde{sk}, \widetilde{pk} := \mathsf{wGen}(G(sk_i, \widetilde{r}))$
3 : $\widetilde{k} = \mathsf{wDecaps}(\widetilde{sk}, \widetilde{c})$
4 : $K = H(pk_i, pk_j, \widetilde{pk}, \widetilde{c}, c_j, k_j, \widetilde{k})$
5 : **return** K

Figure 2: CreTAKE-S2K Framework

The CreTAKE-S2K framework considers N initiators, each equipped with a long-term public-private key pair based on a SIG scheme, and N responders, each equipped with a long-term public-private key pair based on a KEM scheme. These key pairs are generated by the key generation algorithms $\mathsf{KG}_{\mathsf{init}}$ and $\mathsf{KG}_{\mathsf{resp}}$, respectively, as part of the overall KG procedure. The process of negotiation of a session key between initiator i and responder j is as follows. The initiator samples a random number $\widetilde{r}$ and generates the ephemeral key pair of wKEM using $G(sk_i, \widetilde{r})$. It then executes Encaps on the intended responder's public key pk_j and stores $\widetilde{r}\|c_j\|k_j$ in its state. The initiator signs the concatenation $\widetilde{pk}\|c_j$ and sends $\widetilde{pk}, c_j, \sigma$ to the intended responder. Upon receiving the message, the responder verifies the signature. It then runs wEncaps on the ephemeral public key $\widetilde{pk}$, obtaining a ciphertext $\widetilde{c}$ and a key $\widetilde{k}$. Next, it executes Decaps to retrieve k_j. The responder then sends $\widetilde{c}$ to the initiator and computes the shared key

K using the hash function H with the common parameters $pk_i, pk_j, \widetilde{pk}, \widetilde{c}, c_j$ and the keys $k_j, \widetilde{k}$. Upon receiving the message, the initiator runs $\mathsf{wDecaps}$ to obtain $\widetilde{k}$ and retrieves k_j from its state. Finally, it derives the shared key using the hash function H, obtaining the same value K.

Remark 2. In practice, the wGen algorithm within the $\mathsf{Der}_{\mathsf{init}}$ algorithm can be optimized by calling only the component responsible for generating $\widetilde{sk}$, thereby improving efficiency. This optimization is achievable by storing $\widetilde{pk}$ directly in the state (st), removing the need for redundant computations.

3.1.3 CreTAKE-K2K

The algorithms of the AKE scheme in the $\mathsf{CreTAKE\text{-}K2K}$ framework $\mathsf{AKE} = \mathsf{F}_{\mathsf{K2K}}[\mathsf{2KEM}, \mathsf{KEM}, H, G] = (\mathsf{KG}, \mathsf{Init}, \mathsf{Der}_{\mathsf{resp}}, \mathsf{Der}_{\mathsf{init}})$ are defined in Figure 3.

The $\mathsf{CreTAKE\text{-}K2K}$ framework considers N initiators, each equipped with a long-term public-private key pair based on a KEM scheme, and N responders, each equipped with a long-term public-private key pair based on a KEM scheme. These key pairs are generated by the key generation algorithms $\mathsf{KG}_{\mathsf{init}}$ and $\mathsf{KG}_{\mathsf{resp}}$, respectively, as part of the overall $\mathsf{Gen}/\mathsf{Gen1}$ procedure. The process of negotiation of a session key between initiator i and responder j is as follows. The initiator generates the ephemeral/second key pair of $\mathsf{2KEM}$ via $\mathsf{Gen2}$, and then runs Encaps on the public key of the responder pk_j, obtaining a ciphertext c_j and a key k_j . It stores $r\|k_j\|c_j$ in its state. The initiator sends $\widetilde{pk}, c_j$ to the intended responder. The responder runs $\mathsf{2Encaps}$ on the public key of the initiator pk_i and the second ephemeral key $\widetilde{pk}$, obtaining a ciphertext c_i and a key k_i, and sends c_i to the initiator. Next, it runs Decaps to retrieve the key k_j. It then computes the shared key K using the hash function H with common parameters $pk_i, pk_j, \widetilde{pk}, c_i, c_j$ and the keys k_i, k_j. Upon receiving the message, the initiator runs $\mathsf{2Decaps}$ to retrieve the key k_i. Finally, it derives the shared key using the hash function H, obtaining the same value K.

$\mathsf{KG}(i)(i \in [2N])$:

1 : **if** $i \in [N]$//for initiator
2 : $\quad \mathsf{KG}_{\mathsf{init}}(i) : (sk_i, pk_i) \leftarrow \mathsf{Gen1}$
3 : **return** (sk_i, pk_i)
4 : **if** $i \in [N+1, 2N]$//for responder
5 : $\quad \mathsf{KG}_{\mathsf{resp}}(i) : (sk_i, pk_i) \leftarrow \mathsf{Gen}$
6 : **return** (sk_i, pk_i)

$\mathsf{Init}(sk_i, pk_j)(i \in [N], j \in [N+1, 2N])$:

1 : $r \xleftarrow{\$} \mathcal{R}$
2 : $\widetilde{sk}, \widetilde{pk} := \mathsf{Gen2}(G(r, sk_i))$
3 : $c_j, k_j = \mathsf{Encaps}(pk_j)$
4 : $M := \widetilde{pk}\|c_j$
5 : $st := r\|k_j\|c_j$
6 : **return** (M, st)

$\mathsf{Der}_{\mathsf{resp}}(sk_j, pk_i, M)(i \in [N], j \in [N+1, 2N])$:

1 : $\widetilde{pk}, c_j := M$
2 : $c_i, k_i = \mathsf{2Encaps}(pk_i, \widetilde{pk})$
3 : $k_j := \mathsf{Decaps}(sk_j, c_j)$
4 : $M' := c_i$
5 : $K' = H(pk_i, pk_j, \widetilde{pk}, M', c_j, k_i, k_j)$
6 : **return** (M', K')

$\mathsf{Der}_{\mathsf{init}}(sk_i, pk_j, M', st)(i \in [N], j \in [N+1, 2N])$:

1 : $c_i := M'; \quad r, k_j, c_j := st$
2 : $\widetilde{sk}, \widetilde{pk} \leftarrow \mathsf{Gen2}(G(r, sk_i))$
3 : $k_i := \mathsf{2Decaps}(sk_i, \widetilde{sk}, c_i)$
4 : $K = H(pk_i, pk_j, \widetilde{pk}, M', c_j, k_i, k_j)$
5 : **return** K

Figure 3: $\mathsf{CreTAKE\text{-}K2K}$ Framework

3.1.4 CreTAKE-S2S

The algorithms of the AKE scheme in the CreTAKE-S2S framework $\mathsf{AKE} = \mathsf{F}_{\mathsf{S2S}}[\mathsf{wKEM}, \mathsf{SIG}, H, G] = (\mathsf{KG}, \mathsf{Init}, \mathsf{Der}_{\mathsf{resp}}, \mathsf{Der}_{\mathsf{init}})$ are defined in Figure 4.

The CreTAKE-S2S framework considers N initiators, each equipped with a long-term public-private key pair based on a SIG scheme, and N responders, each equipped with a long-term public-private key pair based on a SIG scheme. These key pairs are generated by the key generation algorithms $\mathsf{KG}_{\mathsf{init}}$ and $\mathsf{KG}_{\mathsf{resp}}$, respectively, as part of the overall KG procedure. The process of negotiation of a session key between initiator i and responder j is as follows. The initiator samples a random number $\widetilde{r}$ and generates the ephemeral key pair of wKEM using $G(sk_i, \widetilde{r})$. It stores $\widetilde{r}$ in its state. The initiator signs $\widetilde{pk}$ and sends $\widetilde{pk}, \sigma_i$ to the intended responder. The responder verifies the signature and then runs wEncaps on the ephemeral public key $\widetilde{pk}$, obtaining a ciphertext $\widetilde{c}$ and a key $\widetilde{k}$. Next, it signs the concatenation $\widetilde{pk}\|\widetilde{c}$, and sends $\widetilde{c}\|\sigma_j$ to the initiator. It then computes the shared key K using the hash function H with common parameters $pk_i, pk_j, \widetilde{pk}, \widetilde{c}$ and the key $\widetilde{k}$. Upon receiving the message, the initiator verifies the signature and runs wDecaps to retrieve the key $\widetilde{k}$. Finally, it derives the shared key using the hash function H, obtaining the same value K.

$\mathsf{KG}(i)(i \in [2N])$:

1 : **if** $i \in [N]$//for initiator
2 : $\quad \mathsf{KG}_{\mathsf{init}}(i) : (sk_i, pk_i) \leftarrow \mathsf{SGen}$
3 : **return** (sk_i, pk_i)
4 : **if** $i \in [N+1, 2N]$//for responder
5 : $\quad \mathsf{KG}_{\mathsf{resp}}(i) : (sk_i, pk_i) \leftarrow \mathsf{SGen}$
6 : **return** (sk_i, pk_i)

$\mathsf{Init}(sk_i, pk_j)(i \in [N], j \in [N+1, 2N])$:

1 : $\widetilde{r} \xleftarrow{\$} \mathcal{R}$
2 : $\widetilde{sk}, \widetilde{pk} := \mathsf{wGen}(G(sk_i, \widetilde{r}))$
3 : $\sigma_i := \mathsf{Sign}(sk_i, \widetilde{pk})$
4 : $M := \widetilde{pk}\|\sigma_i$
5 : $st := \widetilde{r}$
6 : **return** (M, st)

$\mathsf{Der}_{\mathsf{resp}}(sk_j, pk_i, M)(i \in [N], j \in [N+1, 2N])$:

1 : $\widetilde{pk}, \sigma_i := M$
2 : **if** $\mathsf{Ver}(pk_i, \widetilde{pk}, \sigma_i) = 0$
3 : $\quad$ **return** $\perp$
4 : $\widetilde{c}, \widetilde{k} = \mathsf{wEncaps}(\widetilde{pk})$
5 : $\sigma_j := \mathsf{Sign}(sk_j, \widetilde{pk}\|\widetilde{c})$
6 : $M' := \widetilde{c}\|\sigma_j$
7 : $K' = H(pk_i, pk_j, \widetilde{pk}, \widetilde{c}, \widetilde{k})$
8 : **return** (M', K')

$\mathsf{Der}_{\mathsf{init}}(sk_i, pk_j, M', st)(i \in [N], j \in [N+1, 2N])$:

1 : $\widetilde{c}, \sigma_j := M'; \quad \widetilde{r} := st$
2 : $\widetilde{sk}, \widetilde{pk} := \mathsf{wGen}(G(sk_i, \widetilde{r}))$
3 : **if** $\mathsf{Ver}(pk_j, \widetilde{pk}\|\widetilde{c}, \sigma_j) = 0$
4 : $\quad$ **return** $\perp$
5 : $\widetilde{k} = \mathsf{wDecaps}(\widetilde{sk}, \widetilde{c})$
6 : $K = H(pk_i, pk_j, \widetilde{pk}, \widetilde{c}, \widetilde{k})$
7 : **return** K

Figure 4: CreTAKE-S2S Framework

3.2 Instantiations of the CreTAKE Frameworks

3.2.1 Underlying KEM Schemes

Here we present some concrete examples of admissible KEMs which can be employed by the framework. For computational efficiency and bandwidth, we choose PolarLAC and ZEN, two

lightweight structured-lattice-based KEMs, both with an IND-CPA form as wKEM and an IND-CCA version as KEM. Since these algorithms are used in a black-box way, we only give their framework here and omit the internal details, as illustrated in the following figures 5, 6.

PolarLAC. PolarLAC (Polar Codes Enhanced Lattice Based Cryptosystem) comprises two public key cryptographic primitives based on the module learning with errors (MLWE) assumption:

- PolarLAC.PKE.CPA: an IND-CPA secure public key encryption scheme; given a key derivation function, it can be transformed into a IND-CPA secure Key encapsulation mechanism trivially in a black-box way.
- PolarLAC.KEM.CCA: an IND-CCA secure key encapsulation mechanism, which is obtained by applying a variant of the FO transformation [10] to PolarLAC.PKE.CPA.

PKE.CPA.KG()

1 : $\mathsf{seed}_{\mathbf{a}}, \mathsf{seed} \xleftarrow{\$} \mathcal{S}$
2 : $\mathbf{A} \leftarrow \mathsf{RejSampU}(R_q; \mathsf{seed}_{\mathbf{a}}) \in R_q^{k \times k}$
3 : $\mathbf{s}, \mathbf{e} \leftarrow \mathsf{RejSamp}(T, \Psi_\sigma; \mathsf{seed}) \in R_q^{k \times 1}$
4 : $\mathbf{b} \leftarrow \mathbf{As} + \mathbf{e} \in R_q^{k \times 1}$
5 : **return** $(pk := (\mathsf{seed}_{\mathbf{a}}, \mathbf{b}), sk := \mathbf{s})$

PKE.CPA.Dec$(sk = \mathbf{s}, \mathbf{c} = (\mathbf{c}_1, \mathbf{c}_2))$

1 : $\mathbf{u} \leftarrow \mathbf{s}^T \mathbf{c}_1 \in R_q$
2 : $\mathbf{c}_2'' \leftarrow \mathsf{DeCompr}_d(\mathbf{c}_2) \in R_q$
3 : $\mathbf{c_m} \leftarrow \mathbf{c}_2'' - \mathbf{u} \in R_q$
4 : $\mathbf{m} \leftarrow \mathsf{PolarDec}(\mathbf{c_m})$
5 : **return** $\mathbf{m}$

PKE.CPA.Enc$(pk = (\mathsf{seed}_{\mathbf{a}}, \mathbf{b}), \mathbf{m} \in \mathcal{M}; \mathsf{seed} \in \mathcal{S})$

1 : $\mathbf{A} \leftarrow \mathsf{RejSampU}(R_q; \mathsf{seed}_{\mathbf{a}}) \in R_q^{k \times k}$
2 : $\hat{\mathbf{m}} \leftarrow \mathsf{PolarEnc}(\mathbf{m}) \in \{0,1\}^n$
3 : $\mathbf{r}, \mathbf{e}_1 \leftarrow \mathsf{RejSamp}(T, \Psi_\sigma; \mathsf{seed}) \in R_q^{k \times 1}$
4 : $\mathbf{e}_2 \leftarrow \mathsf{Samp}(\Psi_\sigma; \mathsf{seed}) \in R_q$
5 : $\mathbf{c}_1 \leftarrow \mathbf{A}^T \mathbf{r} + \mathbf{e}_1 \in R_q^{k \times 1}$
6 : $\mathbf{c}_2' \leftarrow \mathbf{b}^T \mathbf{r} + \mathbf{e}_2 + \bar{q} \cdot \hat{\mathbf{m}} \in R_q$
7 : $\mathbf{c}_2 \leftarrow \mathsf{Compr}_d(\mathbf{c}_2') \in \mathbb{Z}_{2^d}^n$
8 : **return** $\mathbf{c} := (\mathbf{c}_1, \mathbf{c}_2) \in R_q^{k \times 1} \times \mathbb{Z}_{2^d}^n$

Figure 5: Description of PolarLAC.CPA

KEM.CCA.KG()

1 : $(pk, \hat{sk}) \leftarrow$ **PKE.CPA.KG**
2 : $s \xleftarrow{\$} \mathcal{S}$
3 : $sk := (\hat{sk}, pk, s)$
4 : **return** (pk, sk)

KEM.CCA.Enc$(pk; \mathsf{seed}_{\mathbf{m}})$

1 : $\mathbf{m} \leftarrow \mathsf{Samp}(U(\mathcal{M}); \mathsf{seed}_{\mathbf{m}}) \in \mathcal{M}$
2 : $(\mathsf{seed}, K) \leftarrow G(\mathbf{m} \parallel pk)$
3 : $\mathbf{c} \leftarrow$ **PKE.CPA.Enc**$(pk, \mathbf{m}; \mathsf{seed})$
4 : **return** $(\mathbf{c}, K)$

KEM.CCA.Dec$(sk, \mathbf{c})$

1 : parse $sk \rightarrow (\hat{sk}, pk, s)$
2 : $\mathbf{m} \leftarrow$ **PKE.CPA.Dec**$(\hat{sk}, \mathbf{c})$
3 : $(K, \mathsf{seed}) \leftarrow G(\mathbf{m} \parallel pk)$
4 : $K' \leftarrow H(s \parallel \mathbf{c})$
5 : $\mathbf{c}' \leftarrow$ **PKE.CPA.Enc**$(pk, \mathbf{m}; \mathsf{seed})$
6 : if $\mathbf{c}' \neq \mathbf{c}$, then $K \leftarrow K'$
7 : **return** K

Figure 6: Description of PolarLAC.CCA

Double-key PolarLAC. We present two double-key variants of PolarLAC, which can be seamlessly integrated into the CreTAKE-K2K framework. The first one is the direct combination of KEM and PKE, with the result AKE similar to FSXY [7] and HKSU [11]. The second one gives a compact structure, which works under specific parameter settings, PolarLAC-512 and PolarLAC-512* as given in Section 5.2.

- PolarLAC.2KEM.[CCA,CPA]-general: an [IND-CCA, IND-CPA] secure double-key KEM, which is obtained by applying a strong combiner of underlying PolarLAC.KEM.CCA and PolarLAC.PKE.CPA [25, 24], as illustrated in Figure 7. In the key generation, Gen1 and Gen2 generate two pairs of $(pk, \hat{sk})$ and $(pk', \hat{sk}')$ as in PolarLAC.KEM.CCA and PolarLAC.PKE.CPA respectively. Sample $s \xleftarrow{\$} \mathcal{S}$, and let $(pk, sk = (\hat{sk}, pk, s))$ and $(pk', sk' = (\hat{sk}', pk'))$ be the two key pairs.
- PolarLAC.2PKE.CPA: A basic PKE for building PolarLAC.2KEM.[CCA,CPA]-compact as illustrated in Figure 8. Gen1 and Gen2 share the generation algorithm with PolarLAC.PKE.CPA, such that the two generated public keys share a common $\mathsf{seed_a}$, i.e., $(pk = (\mathsf{seed_a}, \mathbf{b}), sk = \mathbf{s}) \leftarrow \mathbf{PKE.CPA.KG}()$ and $(pk' = (\mathsf{seed_a}, \mathbf{b}'), sk = \mathbf{s}') \leftarrow \mathbf{PKE.CPA.KG}()$.
- PolarLAC.2KEM.[CCA,CPA]-compact: an [IND-CCA, IND-CPA] secure double-key KEM, which is obtained by applying a variant of the revisited FO transformation [25, 24] to PolarLAC.2PKE.CPA, as illustrated in Figure 9. In the key generation, Gen1 and Gen2 generate two pairs of $(pk, \hat{sk})$ and $(pk', \hat{sk}')$, respectively, with shared $\mathsf{seed_a}$ as in PolarLAC.2PKE.CPA. Sample $s \xleftarrow{\$} \mathcal{S}$, and let $(pk, sk = (\hat{sk}, pk, s))$ be the output of Gen1, and $(pk', sk' = (\hat{sk}', pk'))$ be that of Gen2.

$\mathbf{2KEM.CCA.Enc}(pk, pk'; \mathsf{seed_m}, \mathsf{seed'_m}, \mathsf{seed'})$	$\mathbf{2KEM.CCA.Dec}(sk, sk', \mathbf{c})$
1 : $\mathbf{m}' \leftarrow \mathsf{Samp}(U(\mathcal{M}); \mathsf{seed'}_m)$	1 : parse $sk \rightarrow (\hat{sk}, pk)$
2 : $\mathbf{c}_1, K_1 \leftarrow \mathbf{KEM.CCA.Enc}(pk; \mathsf{seed}_m)$	2 : parse $sk' \rightarrow (\hat{sk}', pk')$
3 : $\mathbf{c}_2 \leftarrow \mathbf{PKE.CPA.Enc}(pk'; \mathbf{m}', \mathsf{seed}')$	3 : parse $\mathbf{c} \rightarrow (\mathbf{c}_1, \mathbf{c}_2)$
4 : $\mathbf{c} = (\mathbf{c}_1, \mathbf{c}_2)$	4 : $K_1 \leftarrow \mathbf{KEM.CCA.Dec}(sk, \mathbf{c}_1)$
5 : $K = h(pk, K_1, \mathbf{m}')$	5 : $\mathbf{m}' \leftarrow \mathbf{PKE.CPA.Dec}(\hat{sk}', \mathbf{c}_2)$
6 : **return** $(\mathbf{c}, K)$	6 : $K \leftarrow h(pk, K_1, \mathbf{m}')$
	7 : **return** K

Figure 7: PolarLAC.2KEM.[CCA,CPA] generic construction

ZEN. ZEN is NTRU-based, thus is even more compact than PolarLAC, and also comprises two primitives, as illustrated in the following figures 10 and 11:

- ZEN.PKE: an IND-CPA secure public key encryption scheme;
- ZEN.KEM: an IND-CCA secure key encapsulation mechanism obtained from ZEN.PKE through an implicit-rejection Fujisaki-Okamoto transformation.

2PKE.CPA.Enc$(pk, pk', \mathbf{m} \in \mathcal{M}; \mathsf{seed}, \mathsf{seed}' \in \mathcal{S})$

1 : $\mathbf{c}_1, \widetilde{\mathbf{c}}_1 \leftarrow$ **PKE.CPA.Enc**$(pk; \mathbf{m}, \mathsf{seed})$
2 : $\mathbf{c}_2, \widetilde{\mathbf{c}}_2 \leftarrow$ **PKE.CPA.Enc**$(pk'; 0, \mathsf{seed}')$
3 : $\overline{\mathbf{c}} \leftarrow \mathsf{DeCompr}_d(\widetilde{\mathbf{c}}_1) + \mathsf{DeCompr}_d(\widetilde{\mathbf{c}}_2) \in R_q$
4 : $\mathbf{c}_3 \leftarrow \mathsf{Compr}_d(\overline{\mathbf{c}}) \in \mathbb{Z}_{2^d}^n$
5 : **return** $\mathbf{c} := (\mathbf{c}_1, \mathbf{c}_2, \mathbf{c}_3)$

2PKE.CPA.Dec$(sk, sk', \mathbf{c} = (\mathbf{c}_1, \mathbf{c}_2, \mathbf{c}_3))$

1 : parse $sk \rightarrow \mathbf{s}, sk' \rightarrow \mathbf{s}'$
2 : $\mathbf{u} \leftarrow \mathbf{s}^T \mathbf{c}_1 \in R_q$
3 : $\mathbf{u}' \leftarrow \mathbf{s}'^T \mathbf{c}_2 \in R_q$
4 : $\mathbf{v} \leftarrow \mathsf{DeCompr}_d(\mathbf{c}_3) \in R_q$
5 : $\mathbf{c_m} \leftarrow \mathbf{v} - \mathbf{u} - \mathbf{u}' \in R_q$
6 : $\mathbf{m} \leftarrow \mathsf{PolarDec}(\mathbf{c_m})$
7 : **return** $\mathbf{m}$

Figure 8: PolarLAC.2PKE.CPA

2KEM.CCA.Enc$(pk, pk'; \mathsf{seed}_\mathbf{m})$

1 : $\mathbf{m} \leftarrow \mathsf{Samp}(U(\mathcal{M}); \mathsf{seed}_\mathbf{m}) \in \mathcal{M}$
2 : $(\mathsf{seed}, \mathsf{seed}', K) \leftarrow G(\mathbf{m} \,\|\, H(pk, pk'))$
3 : $\mathbf{c} \leftarrow$ **2PKE.CPA.Enc**$(pk, pk', \mathbf{m}; \mathsf{seed}, \mathsf{seed}')$
4 : **return** $(\mathbf{c}, K)$

2KEM.CCA.Dec$(sk, sk', \mathbf{c})$

1 : parse $sk \rightarrow (\hat{sk}, pk, s), sk' \rightarrow (\hat{sk}', pk')$
2 : $\mathbf{m} \leftarrow$ **2PKE.CPA.Dec**$(\hat{sk}, \hat{sk}', \mathbf{c})$
3 : $(\mathsf{seed}, \mathsf{seed}', K) \leftarrow G(\mathbf{m} \,\|\, H(pk, pk'))$
4 : $K' \leftarrow H(s \,\|\, \mathbf{c})$
5 : $\mathbf{c}' \leftarrow$ **2PKE.CPA.Enc**$(pk, pk', \mathbf{m}; \mathsf{seed}, \mathsf{seed}')$
6 : if $\mathbf{c}' \neq \mathbf{c}$, then $K \leftarrow K'$
7 : **return** K

Figure 9: PolarLAC.2KEM.[CCA,CPA] compact construction

ZEN.PKE.KeyGen(ρ_g, ρ_f)

1 : **do**$\{\boldsymbol{g} \leftarrow \mathrm{Sample}(\mathcal{D}_g, \rho_g)\}$
2 : **until** $\boldsymbol{g}$ *invertible in* $\mathrm{R}_{n,q}$
3 : **do**$\{\boldsymbol{f} \leftarrow \mathrm{Sample}(\mathcal{D}_f, \rho_f), \boldsymbol{f}_{\mathrm{inv}} \leftarrow$ **FastInversion**$(\boldsymbol{f}, n/2)\}$
4 : **until** $\boldsymbol{f}$ *invertible in* $\mathrm{R}_{n,q}$ and $\boldsymbol{f}_{\mathrm{inv}} \neq \perp$
5 : $\widehat{\boldsymbol{g}} \leftarrow \mathrm{NTT}(\boldsymbol{g})$
6 : $\widehat{\boldsymbol{f}} \leftarrow \mathrm{NTT}(\boldsymbol{f})$
7 : $\widehat{\boldsymbol{h}} \leftarrow \widehat{\boldsymbol{g}} \odot \widehat{\boldsymbol{f}}^{-1} \pmod{\langle q, x^n + 1\rangle}$
8 : $pk \leftarrow \mathrm{Pack}_q(\widehat{\boldsymbol{h}})$
9 : $sk_{\mathrm{PKE}} \leftarrow (\widehat{\boldsymbol{f}}, \boldsymbol{f}_{\mathrm{inv}})$
10 : **return** (pk, sk_{PKE})

ZEN.PKE.Encrypt$(pk, \boldsymbol{m}, \rho)$

1 : $\widehat{\boldsymbol{h}} \leftarrow \mathrm{Unpack}_q(pk)$
2 : $\boldsymbol{s} \leftarrow \mathrm{Sample}(\mathcal{D}_s, \mathrm{PRF}(\rho, 0))$
3 : $\boldsymbol{e} \leftarrow \mathrm{Sample}(\mathcal{D}_e, \mathrm{PRF}(\rho, 1))$
4 : $\widehat{\boldsymbol{s}} \leftarrow \mathrm{NTT}(\boldsymbol{s})$
5 : $\boldsymbol{u} \leftarrow \mathrm{InvNTT}(\widehat{\boldsymbol{h}} \odot \widehat{\boldsymbol{s}}) + \boldsymbol{e} \pmod{\langle q, x^n + 1\rangle}$
6 : $\boldsymbol{c} \leftarrow \boldsymbol{u} + \frac{q+1}{2}(1 + x^{n/4} + x^{n/2} + x^{3n/4})\boldsymbol{m} \pmod{\langle q, x^n + 1\rangle}$
7 : $ct \leftarrow \mathrm{Pack}_{M_c}\big(\mathrm{Compress}_{q,M_c}(\boldsymbol{c})\big)$
8 : **return** ct

ZEN.PKE.Decrypt(sk_{PKE}, ct)

1 : Parse sk_{PKE} as $(\widehat{\boldsymbol{f}}, \boldsymbol{f}_{\mathrm{inv}})$
2 : $\widetilde{\boldsymbol{c}} \leftarrow \mathrm{Unpack}_{M_c}(ct)$
3 : $\boldsymbol{c}^\dagger \leftarrow \mathrm{Decompress}_{q,M_c}(\widetilde{\boldsymbol{c}})$
4 : $\widehat{\boldsymbol{c}} \leftarrow \mathrm{NTT}(\boldsymbol{c}^\dagger)$
5 : $\boldsymbol{a} \leftarrow (x^{n/2} + 1)\,\mathrm{InvNTT}(\widehat{\boldsymbol{c}} \odot \widehat{\boldsymbol{f}}) \pmod{\langle q, x^n + 1\rangle}$
6 : $\boldsymbol{m}' \leftarrow \boldsymbol{a}\boldsymbol{f}_{\mathrm{inv}} \pmod{\langle 2, x^{n/2} + 1\rangle}$
7 : $\boldsymbol{m} \leftarrow$ **SimpleDecoding**$(\boldsymbol{a}, \boldsymbol{m}', \boldsymbol{f}_{\mathrm{inv}})$
8 : **return** $\boldsymbol{m}$

Figure 10: Description of ZEN.PKE

3.2.2 Underlying Signature Scheme

For the exemplary signature which can be employed by the framework, we choose BiT (Bimodal Triangular distribution based lattice signature), a compact MLWE based signature following the

ZEN.KEM.KeyGen(ρ_g, ρ_f)

1 : $(pk, sk_{\mathrm{PKE}}) \leftarrow$ **ZEN.PKE.KeyGen**(ρ_g, ρ_f)
2 : $\kappa' \leftarrow \mathrm{Sample}(\mathcal{U}(\{0,1\}^{n/4}))$
3 : $h_{\mathrm{pk}} \leftarrow \mathsf{H}_{pk}(pk)$
4 : $sk_{\mathrm{KEM}} \leftarrow (sk_{\mathrm{PKE}}, pk, h_{\mathrm{pk}}, \kappa')$
5 : **return** (pk, sk_{KEM})

ZEN.KEM.Encaps(pk)

1 : $\boldsymbol{m} \leftarrow \mathrm{Sample}(\mathcal{U}(\{0,1\}^{n/4}))$
2 : $(K, \rho) \leftarrow \mathsf{H}_m(\boldsymbol{m}, \mathsf{H}_{pk}(pk))$
3 : $ct \leftarrow$ **ZEN.PKE.Encrypt**$(pk, \boldsymbol{m}, \rho)$
4 : **return** (ct, K)

ZEN.KEM.Decaps(sk_{KEM}, ct)

1 : $(sk_{\mathrm{PKE}}, pk, h_{\mathrm{pk}}, \kappa') \leftarrow sk_{\mathrm{KEM}}$
2 : $\boldsymbol{m} \leftarrow$ **ZEN.PKE.Decrypt**(sk_{PKE}, ct)
3 : $(K, \rho) \leftarrow \mathsf{H}_m(\boldsymbol{m}, h_{\mathrm{pk}})$
4 : $K' \leftarrow \mathsf{H}_K(\kappa', ct)$
5 : $ct' \leftarrow$ **ZEN.PKE.Encrypt**$(pk, \boldsymbol{m}, \rho)$
6 : **if** $ct' = ct$ **then return** K
7 : **else return** K'

Figure 11: Description of ZEN.KEM

Fiat-Shamir with abort framework, as illustrated in the following figure 12.

KGen(1^κ)

1 : $\mathbf{s}_0, \mathbf{e} \xleftarrow{\$} S_\eta^m \times S_\eta^n \subset \mathcal{R}_q^m \times \mathcal{R}_q^n$
2 : $\mathbf{A}_0 \xleftarrow{\$} \mathcal{R}_q^{n \times m}$
3 : $\mathbf{b} = \mathbf{A}_0\mathbf{s}_0 + \mathbf{e}$
4 : $\mathbf{A} = [\hat{q}\mathbf{j}_n - \mathbf{b}, \mathbf{A}_0, \mathbf{I}_n]$
5 : $\mathbf{s} := (1, \mathbf{s}_0^T, \mathbf{e}^T)^T$
6 : $\mathsf{pk} := \mathbf{A}; \mathsf{sk} := \mathbf{s}$

Verify$(\mathsf{pk}, \sigma, \mu)$

1 : if $||\mathbf{z}||_\infty > B_\infty$, then reject
2 : $\hat{\mathbf{w}}' = \mathbf{A}\mathbf{z} + \hat{q}\mathbf{j}_n c$
3 : $\mathbf{w}' = \mathsf{HighBits}(\hat{\mathbf{w}}', \gamma_2)$
4 : Accept if $c = \mathsf{H}(\mathbf{w}', \mu)$

Sign$(\mathsf{pk}, \mathsf{sk}, \mu)$

1 : $\mathbf{y} \xleftarrow{\$} T_{\gamma_1}^{m+n+1}$
2 : $\mathbf{w} := \mathbf{A}\mathbf{y}$
3 : $c \in \mathcal{R}_q := \mathsf{H}(\mathsf{HighBits}(\mathbf{w}, \gamma_2), \mu)$
4 : $b \xleftarrow{\$} \{0,1\}$
5 : $\mathbf{z} := \mathbf{y} + (-1)^b \mathbf{s}c$
6 : $\mathbf{z} := \mathsf{RejectSample}(\mathbf{z}, \gamma_1, \mathbf{s}c, \beta)$
7 : if $\mathbf{z} = \perp$, go into step 1
8 : if $\mathsf{HighBits}(\mathbf{w}, \gamma_2) \neq \mathsf{HighBits}(\mathbf{w} + (-1)^b \mathbf{j}_n c, \gamma_2)$, go into step 1
9 : if $||\mathbf{z}||_\infty > B_\infty$, then restart
10 : **return** $\sigma := (\mathbf{z}, c)$

Figure 12: Description of BiT

3.2.3 CreTAKE Instance Set

The concrete CreTAKE instances are formed by combining each of the four frameworks with the applicable PolarLAC, ZEN, and BiT parameter sets at the 128-bit, 256-bit, and 512-bit security levels. Each instance name begins with the corresponding framework tag, K2S, S2K, K2K, or S2S, followed by the primitive assignment and the security level $\lambda \in \{128, 256, 512\}$. All constituent primitives of an instance are selected from the same security category.

The names in Table 1 follow a uniform rule. In the heterogeneous cases, the primitive names

Table 1: CreTAKE instance names at security level $\lambda \in \{128, 256, 512\}$.

Framework	Primitive assignment	Instance name
CreTAKE-K2S	initiator PolarLAC, responder BiT	K2S-PolarLAC-BiT-λ
	initiator ZEN, responder BiT	K2S-ZEN-BiT-λ
CreTAKE-S2K	initiator BiT, responder PolarLAC	S2K-BiT-PolarLAC-λ
	initiator BiT, responder ZEN	S2K-BiT-ZEN-λ
CreTAKE-K2K	both credentials PolarLAC	K2K-PolarLAC-λ
	both credentials ZEN	K2K-ZEN-λ
	alternative compact PolarLAC 2KEM at 512-bit security	K2K-PolarLAC-512*
CreTAKE-S2S	both credentials BiT, ephemeral wKEM PolarLAC	S2S-BiT-ePolarLAC-λ
	both credentials BiT, ephemeral wKEM ZEN	S2S-BiT-eZEN-λ

appear in initiator-responder order. For example, K2S-PolarLAC-BiT-λ uses a PolarLAC KEM credential for the initiator and a BiT signature credential for the responder. In the S2S instances, the prefix "e" identifies the KEM family used as the ephemeral wKEM, while both long-term credentials use BiT. In the K2K instances, the KEM-family name identifies the family used for both long-term credentials and the associated double-key KEM construction.

For each security level, Table 1 specifies two instances for each framework, one using PolarLAC and one using ZEN. These eight regular patterns across the three security levels give 24 instances. The additional instance K2K-PolarLAC-512* gives the second PolarLAC-based K2K realization at the 512-bit security level, bringing the total to 25 instances.

The K2K-PolarLAC-128 and K2K-PolarLAC-256 instances use the generic double-key construction with separate ciphertext components. At the 512-bit security level, K2K-PolarLAC-512 and K2K-PolarLAC-512* use the two compact double-key constructions specified in Figure 9. All ZEN-based K2K instances use the generic double-key construction.

Correctness. For all instances except the two compact K2K constructions, the decryption-failure probability is inherited from the corresponding concrete KEM instantiation. The compact K2K-PolarLAC-512 and K2K-PolarLAC-512* instances combine the failure probabilities of the underlying CCA-secure KEM and CPA-secure PKE components. The resulting decryption-failure probabilities are bounded by 2^{-175} and 2^{-328}, respectively, according to the DFR analysis specified for PolarLAC.

4 Design Rationale

4.1 Design Goals and Two-Layer Structure

The primary design goal of CreTAKE is to provide a general AKE *framework layer* solution for the full range of KEM- and signature-based credential configurations that may arise during and after post-quantum migration. An initiator and a responder may hold either credential type, yielding two heterogeneous configurations during asynchronous migration and two homogeneous configurations in deployments that have converged on a common credential type. The design therefore begins with a primitive-independent framework layer that covers the complete ordered credential-type space.

Denoting a KEM-based credential by K and a signature-based credential by S, the ordered

credential types of the two parties give four cases: CreTAKE-K2S, CreTAKE-S2K, CreTAKE-K2K, and CreTAKE-S2S. These frameworks form a complete credential-type matrix. The heterogeneous frameworks address the two directions in which credential migration may become asynchronous, while the homogeneous frameworks cover deployments in which both endpoints use the same credential type. The framework layer consequently remains applicable throughout the migration process and after credential deployment has reached a stable state.

This framework-first design takes the ordered credential types of the two parties as the structural input to the AKE protocol. Each framework specifies how mutual authentication and fresh shared-secret establishment are realized, which primitive objects are carried in the two messages, and how the identities, transcript, and secret contributions are bound into the session key. All four frameworks expose the common $(\mathsf{KG}, \mathsf{Init}, \mathsf{Der}_{\mathsf{resp}}, \mathsf{Der}_{\mathsf{init}})$ interface and target the applicable IND-AA or IND-StAA security notion. Their definitions rely on primitive interfaces and security properties, including IND-CPA or IND-CCA KEM security, double-key KEM security, and EUF-CMA signature security. They do not depend on the internal structure of any particular algorithm. The same framework can therefore be instantiated with primitives selected for the security, correctness, performance, and deployment requirements of a specific application.

The *instantiation layer* demonstrates that the general frameworks admit concrete and efficient post-quantum realizations. In this submission, the KEM components are instantiated with PolarLAC and ZEN, and the signature components with BiT. Their CPA-secure, CCA-secure, and double-key variants are assigned according to the interfaces required by the four frameworks. Parameter sets are selected at the 128-bit, 256-bit, and 512-bit security levels according to primitive security, correctness, decryption-failure probability, communication size, and implementation efficiency. These choices provide specific realizations of the framework layer without restricting the frameworks to these three primitive families.

The 25 submitted instances arise systematically from this two-layer design. At the 128-bit and 256-bit security levels, each of the four frameworks has one PolarLAC-based and one ZEN-based realization. The same pattern applies at the 512-bit level, with an additional CreTAKE-K2K instance arising from the alternative double-key PolarLAC construction. The suite therefore consists of four reusable AKE structures combined with concrete primitive and parameter choices, not 25 independently designed protocols.

The remainder of this section follows the same separation. Section 4.2 explains the common design principles and the rationale for the four credential-type frameworks. Section 4.3 then explains the primitive requirements, the selection of PolarLAC, ZEN, and BiT, the mapping from their parameter sets to the submitted instances, and the effect of primitive-level failure probabilities on AKE correctness.

4.2 Framework-Layer Rationale

The four frameworks are shaped jointly by the capabilities of KEM- and signature-based credentials and by the security requirements of two-message AKE. The applicable IND-(St)AA security model [12] requires mutual authentication and session-key indistinguishability against an adaptive active adversary, while permitting the long-term-key and session-state exposures

allowed by the corresponding freshness condition. It also captures weak forward secrecy for completed honest sessions. Each framework must therefore preserve at least one hidden secret contribution under every permitted reveal pattern and bind that contribution to the identities and complete public transcript.

Across the suite, signatures authenticate the transcript components generated by an S-credential holder, while successful recovery of a secret encapsulated to a long-term KEM key authenticates the corresponding K-credential holder. The session key is derived from the identities, public transcript, and all KEM-derived secrets used in the execution. This common structure couples authentication with key secrecy and prevents values from different sessions from being combined into an accepted exchange. The individual frameworks differ in how they realize these principles for the four ordered credential combinations.

4.2.1 Heterogeneous Credential Frameworks

The heterogeneous cases require an AKE construction in which authentication and session-key secrecy are supplied by credentials of different types. A signature credential authenticates public protocol values but does not itself produce hidden key material, whereas a KEM credential can contribute a secret through encapsulation and decapsulation. Both heterogeneous frameworks therefore combine an ephemeral wKEM contribution, which provides fresh session-specific key material, with a contribution associated with the long-term KEM credential. The identities, complete transcript, and both KEM-derived secrets are included in the final key derivation.

In CreTAKE-K2S, the initiator holds the long-term KEM credential and the responder holds the signature credential. The initiator sends an ephemeral wKEM public key, and the responder encapsulates both to this key and to the initiator's long-term KEM public key. The responder then signs the ephemeral public key and both ciphertexts. The signature authenticates the responder and prevents an active adversary from replacing any component of the second message, while the two encapsulated values provide fresh and long-term-credential-dependent secret contributions.

This structure is designed for the reveal patterns captured by IND-AA. In an honest matched execution, disclosure of the responder's signing key does not reveal the session key, because the adversary may obtain either the initiator's long-term KEM secret or its ephemeral session state, but not both. At least one of the two KEM-derived values therefore remains hidden. In a tampered execution, an initiator accepts only a response authenticated by the responder's signature. These mechanisms allow CreTAKE-K2S to achieve IND-AA security using standard IND-CPA wKEM, IND-CCA KEM, and EUF-CMA signature interfaces.

In CreTAKE-S2K, the credential placement is reversed, but the construction cannot be obtained by merely exchanging the party roles. The initiator's signature credential cannot directly supply a decapsulation secret. The initiator therefore generates an ephemeral wKEM public key, encapsulates to the responder's long-term KEM credential, and signs both values. The responder verifies the signature, decapsulates the long-term-credential ciphertext, and returns an encapsulation to the ephemeral public key. The signature authenticates the initiator's first message, while the two KEM-derived secrets provide the entropy of the session key.

State exposure creates an additional difficulty in this direction. Storing an independently

generated ephemeral secret key would reveal the wKEM contribution directly. Instead, the ephemeral key pair is generated from $G(sk_i, r)$, where sk_i is the initiator's long-term signing key and r is fresh session randomness. Recovering the ephemeral decapsulation secret therefore requires both values. In a matched execution, revealing either the signing key or the session state alone does not expose the wKEM secret. This secret mixing, together with the responder's long-term KEM contribution and transcript-bound key derivation, provides the intended protection against long-term-key and state-reveal attacks.

The current CreTAKE-S2K construction achieves IND-StAA security. The same security notion is achieved by the widely studied FSXY-type KEM-only AKE in its QROM analysis [12]. IND-StAA therefore represents an established security level for practical two-message AKE. It retains an active adversary and the permitted long-term-key and state-reveal queries, while excluding the active state-attack case in which a tampered second message is combined with disclosure of the initiator's state. Achieving full IND-AA security in this direction would require different state handling or an additional primitive interface, such as deterministic encapsulation coins derived jointly from the signing key and fresh session randomness.

Previous heterogeneous AKE constructions either require stronger primitive properties, additional long-term keys, or additional protocol rounds [14, 3]. The CreTAKE-K2S and CreTAKE-S2K frameworks use KEM and signature interfaces and retain the common two-message structure of CreTAKE. Our earlier work [19] introduced these two constructions and established their IND-AA and IND-StAA security in the QROM.

The two frameworks consequently solve different ordered credential cases under one common principle. CreTAKE-K2S uses the responder's signature to authenticate two responder-generated encapsulations and attains IND-AA security. CreTAKE-S2K uses the initiator's signature together with long-term-key-bound ephemeral generation and attains IND-StAA security. Their different structures follow from the security capabilities of the credential held by the initiator, not merely from the direction of the protocol flow.

4.2.2 Homogeneous Credential Frameworks

In CreTAKE-K2K, both parties hold long-term KEM credentials. To achieve the target IND-(St)AA security, the session key must remain hidden under the permitted exposures of long-term keys and session state. The initiator encapsulates to the responder's long-term public key, while the responder must encapsulate jointly to the initiator's long-term and ephemeral public keys. Security of this responder-side contribution must be preserved when either key pair is exposed according to the corresponding freshness condition.

We express this requirement through the double-key KEM abstraction introduced in our earlier work [25]. A double-key KEM takes two public keys during encapsulation and requires both corresponding secret keys for decapsulation. Its two-sided security notions capture the distinct long-term key and session-state exposure cases arising in AKE. This gives CreTAKE-K2K a single primitive interface for the responder-side operation and separates the AKE construction from the internal form of the underlying KEM.

The abstraction includes the established FSXY construction as a generic case, but is not restricted to it. For arbitrary KEMs, a double-key KEM can be constructed by combining sep-

arate CCA- and CPA-secure encapsulations, yielding the parallel-ciphertext structure used by FSXY-style AKE [7, 8]. When the underlying primitive allows the two key pairs or ciphertexts to share algebraic components, the same interface also admits a joint and more compact construction. Our earlier work demonstrated this possibility with a Module-LWE construction [25], and subsequent work instantiated the approach in the supersingular-isogeny setting through SIAKE [23].

This generality is the reason that CreTAKE-K2K is formulated using a double-key KEM instead of fixing the framework to two independent encapsulations. The generic construction supports KEMs for which no useful combination is available, while structure-aware constructions can reduce communication by sharing suitable components. In the submitted suite, the PolarLAC-based CreTAKE-K2K instances at the 128-bit and 256-bit security levels use the generic FSXY-style double-key KEM construction, since the compact construction does not provide a sufficient decryption-failure rate for these parameter sets. At the 512-bit security level, the decryption-failure rate permits the compact double-key constructions, which combine ciphertext components within the same CreTAKE-K2K interface.

In CreTAKE-S2S, neither long-term credential directly contributes a shared secret. An ephemeral wKEM therefore establishes fresh session-key material, while signatures authenticate the ephemeral public key and the resulting ciphertext in the two protocol messages. The initiator's ephemeral key pair is derived from $G(sk_i, r)$ so that its decapsulation secret depends jointly on the long-term signing key and fresh session randomness. This design preserves session-key secrecy when either the long-term key or the session state is revealed under the applicable IND-AA freshness condition. The two signatures provide mutual authentication, and the transcript-bound key derivation binds that authentication to the established wKEM secret.

The two homogeneous frameworks therefore use different mechanisms to meet the same AKE objectives. CreTAKE-K2K obtains authentication and hidden key contributions through long-term and ephemeral KEM keys, with the double-key KEM providing a general interface for both parallel and compact instantiations. CreTAKE-S2S uses signatures for authentication and an ephemeral wKEM for fresh shared-secret establishment.

4.3 Instantiation-Layer Rationale

The purpose of the instantiation layer is to realize the four general frameworks as a coherent and efficient post-quantum suite using a small set of compatible primitives. The selection must account not only for the security interfaces required by the frameworks, but also for communication size, computational efficiency, correctness, and the different placement of public keys, ciphertexts, and signatures in the two protocol messages.

For the KEM components, PolarLAC and ZEN provide complementary communication profiles. The compact public keys of PolarLAC reduce protocol flights carrying an ephemeral KEM key, while the compact ciphertexts of ZEN reduce ciphertext-dominated messages and often the total transcript size. Including both families therefore allows an instantiation to be selected according to first-flight limits, responder bandwidth, total communication, and implementation cost.

The signature choice is particularly important in CreTAKE-K2S, CreTAKE-S2K, and Cre-

TAKE-S2S because a post-quantum signature may constitute a large part of the corresponding protocol message. Within the post-quantum design space considered for this submission, lattice-based signatures provide a favorable balance between signature size and computational efficiency. We therefore select BiT, which offers competitive signing and verification performance together with moderate public-key and signature sizes. Using the same signature family across the three signature-bearing frameworks also makes their communication and runtime differences primarily reflect the framework structure and KEM selection.

The selected primitives also provide the related interfaces needed throughout the suite. The CPA-secure PKEs of PolarLAC and ZEN realize the ephemeral wKEM functionality, while their CCA-secure KEMs protect encapsulations associated with long-term KEM credentials. For CreTAKE-K2K, a generic double-key construction preserves compatibility with ordinary KEMs, whereas the structure-aware PolarLAC construction combines suitable ciphertext components to reduce communication. The framework can therefore exploit primitive-specific structure without depending on it.

The structure of PolarLAC supports a compact double-key construction in which suitable ciphertext components are shared while preserving the required two-key security [25]. Its use at a concrete security level is then constrained by correctness. The shared structure increases the effective decryption noise, so the 128-bit and 256-bit CreTAKE-K2K instances use the generic construction with separate components, while the lower DFR at the 512-bit level permits the compact PolarLAC-512 and PolarLAC-512* constructions.

The three concrete primitive families used in the CreTAKE suite, PolarLAC, ZEN, and BiT, are submitted separately as candidate algorithms. Their joint use allows the instantiation layer to align security categories and implementation techniques while offering complementary public-key, ciphertext, and signature profiles. The resulting suite demonstrates that complete credential-type coverage can be achieved efficiently without sacrificing the modularity of the framework layer. The concrete instances are specified in Section 3, and their security and performance are evaluated in Sections 5 and 6.

5 Security Statements and Analyses

5.1 Theoretical Security

5.1.1 CreTAKE-K2S

Theorem 1 (wKEM IND-CPA+ KEM IND-CCA+ SIG EUF-CMA $\overset{QROM}{\Rightarrow}$ $\mathsf{F}_{\mathsf{K2S}}$ [wKEM, KEM, SIG, H] IND-AA)**.** *Let* wKEM *be an* IND-CPA *secure KEM scheme with key space* $\mathcal{K}$*,* KEM *be an* IND-CCA *secure KEM scheme with key space* $\mathcal{K}$*, and* SIG *be a* EUF-CMA *secure signature scheme. Let* $\mathcal{A}$ *be an* IND-AA *adversary against* AKE $= \mathsf{F}_{\mathsf{KEM\text{-}SIG}}[\mathsf{wKEM}, \mathsf{KEM}, \mathsf{SIG}, H]$ *establishing* S *sessions between* $2N$ *parties and making* q *quantum queries to the random oracle* H*. Then we can construct an* IND-CPA *adversary* $\mathcal{B}$ *against* wKEM*, an* IND-CCA *adversary* $\mathcal{C}$ *against* KEM*, and an*

EUF-CMA *adversary* $\mathcal{D}$ *against* SIG*, such that:*

$$\begin{aligned}\mathrm{Adv}^{\mathsf{IND\text{-}AA}}_{\mathsf{AKE}}(\mathcal{A}) \leq & \frac{(S+2N)S(S-1)}{|\mathcal{K}|} + \frac{2q \cdot S(S+2N)}{\sqrt{|\mathcal{K}|}} \\ & + 2S^2 \cdot \mathrm{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{wKEM}}(\mathcal{B}) + 4SN \cdot \mathrm{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathsf{KEM}}(\mathcal{C}) \\ & + \frac{N \cdot \mathrm{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{SIG}}(\mathcal{D})}{1-(S-1)\mu(\mathsf{wEncaps})} + S(S-1) \cdot \mu(\mathsf{wGen}) \\ & + max\{(S-1) \cdot \mu(\mathsf{Encaps}) \cdot \mu(\mathsf{wEncaps}), (S-1) \cdot \mu(\mathsf{wGen})\}\end{aligned},$$

where $\mu(\mathsf{wGen})$ *represents the collision probabilities of key generation for* wKEM*.* $\mu(\mathsf{wEncaps})$ *and* $\mu(\mathsf{Encaps})$ *denote the collision probabilities of ciphertexts for* wKEM *and* KEM*, respectively.*

Sketch of Proof. To prove IND-AA security of $\mathsf{AKE} = \mathsf{F}_{\mathsf{KEM\text{-}SIG}}$ [wKEM, KEM, SIG, H], we consider an adversary $\mathcal{A}$ with black-box access to the protocol's algorithms and oracles. These oracles provide access to session keys from completed sessions, internal states, and the long-term secret keys of the participating parties, as defined in the IND-AA game. Intuitively, the adversary $\mathcal{A}$ will either obtain at most one of the keys k_i or $\widetilde{k}$, ensuring the session key remains pseudo-random, or fail to generate a valid signature of ciphertexts generated by itself encapsulating k_i and $\widetilde{k}$, preventing the successful completion of the test session between P_i and P_j.

- Honest Execution of the Test Session: We first consider the case where $\mathcal{A}$ executes the test session honestly. Note that there is no state on the right-hand side of the protocol. We assume $\mathcal{A}$ has learned the secret key of party P_j and can therefore generate valid signatures. Furthermore, $\mathcal{A}$ can either learn the secret key of party P_i to compute k_i, or access the state on the left-hand side of the protocol (which includes $\widetilde{sk}$) to compute $\widetilde{k}$, but not both.
- Dishonest Execution of the Test Session: In the case where $\mathcal{A}$ does not execute the test session honestly, it is prohibited from obtaining the long-term secret key of the test session's peer. Since $\mathcal{A}$ modifies the exchanged messages, the test sessions side becomes decoupled from the other side. If the test session is on the right-hand side, $\mathcal{A}$ can generate valid signatures. However, it cannot obtain k_i because it is not allowed to learn the secret key of party P_i. If the test session is on the left-hand side, $\mathcal{A}$ can generate both k_i and $\widetilde{k}$ by itself. However, without the ability to produce a valid signature of the corresponding ciphertexts $c_i, \widetilde{c}$, the test session will not be completed.

Roughly, let $\mathcal{A}$ be an adversary against the IND-AA security of AKE, establishing S sessions and making at most q (quantum) queries to the random oracle H. We first consider the case where $\mathcal{A}$ executed the test session honestly (i.e., $\mathfrak{M}(\mathsf{sID}^*) \neq \emptyset$). In the second part, we analyze the scenario where $\mathcal{A}$ tampered with the test session (i.e., $\mathfrak{M}(\mathsf{sID}^*) = \emptyset$).

$$\begin{aligned}\mathrm{Adv}^{\mathsf{IND\text{-}AA}}_{\mathsf{AKE}}(\mathcal{A}) &= |\Pr[\mathsf{IND\text{-}AA}^{\mathcal{A}} \Rightarrow 1] - \frac{1}{2}| \\ &\leq |\Pr[\mathsf{IND\text{-}AA}^{\mathcal{A}} \Rightarrow 1 \wedge \mathfrak{M}(\mathsf{sID}^*) \neq \emptyset] - \frac{1}{2}| \\ &\quad + |\Pr[\mathsf{IND\text{-}AA}^{\mathcal{A}} \Rightarrow 1 \wedge \mathfrak{M}(\mathsf{sID}^*) = \emptyset] - \frac{1}{2}|.\end{aligned}$$

The following two lemmas fulfill the gap to Theorem 1.

Lemma 1. *There exists an adversary $\mathcal{B}$ and an adversary $\mathcal{C}$ such that*

$$\begin{aligned}
&|\Pr[\mathsf{IND\text{-}AA}^{\mathcal{A}} \Rightarrow 1 \wedge \mathfrak{M}(\mathsf{sID}^*) \neq \emptyset] - \frac{1}{2}| \\
&\quad \leq \frac{(S+N)S(S-1)}{|\mathcal{K}|} + \frac{2q \cdot S(S+N)}{\sqrt{|\mathcal{K}|}} + S(S-1) \cdot \mu(\mathsf{wGen}) \\
&\quad + max\{(S-1) \cdot \mu(\mathsf{Encaps}) \cdot \mu(\mathsf{wEncaps}), (S-1) \cdot \mu(\mathsf{wGen})\} \\
&\quad + 2S^2 \cdot \mathrm{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{wKEM}}(\mathcal{B}) + 2SN \cdot \mathrm{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathsf{KEM}}(\mathcal{C})
\end{aligned}$$

Refer to appendix D of our accepted work [19] for the proof of Lemma 1.

Lemma 2. *There exists an adversary $\mathcal{C}$ and an adversary $\mathcal{D}$ such that*

$$\begin{aligned}
&|\Pr[\mathsf{IND\text{-}AA}^{\mathcal{A}} \Rightarrow 1 \wedge \mathfrak{M}(\mathsf{sID}^*) = \emptyset] - \frac{1}{2}| \\
&\quad \leq \frac{SN(S-1)}{|\mathcal{K}|} + \frac{2SN \cdot q}{\sqrt{|\mathcal{K}|}} + 2SN \cdot \mathrm{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathsf{KEM}}(\mathcal{C}) \\
&\quad + \frac{N \cdot \mathrm{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{SIG}}(\mathcal{D})}{1-(S-1)\mu(\mathsf{wEncaps})}
\end{aligned}$$

Refer to appendix E of our accepted work [19] for the proof of Lemma 2.

5.1.2 CreTAKE-S2K

Theorem 2 (wKEM IND-CPA+ KEM IND-CCA+ SIG EUF-CMA $\overset{QROM}{\Rightarrow}$ $\mathsf{F}_{\mathsf{S2K}}$ [wKEM, KEM, SIG, H, G] IND-StAA)**.** *Let* wKEM *be an* IND-CPA *secure KEM with key space $\mathcal{K}$,* KEM *be an* IND-CCA *secure KEM with key space $\mathcal{K}$, and* SIG *be a* EUF-CMA *secure signature. Let $\mathcal{A}$ be an* IND-StAA *adversary against* $\mathsf{AKE} = \mathsf{F}_{\mathsf{SIG\text{-}KEM}}[\mathsf{wKEM}, \mathsf{KEM}, \mathsf{SIG}, H, G]$ *establishing S sessions between $2N$ parties and making q quantum queries to the random oracle H, and G. Then we can construct an* IND-CPA *adversary $\mathcal{B}$ against* wKEM*, an* IND-CCA *adversary $\mathcal{C}$ against* KEM*, and an* EUF-CMA *adversary $\mathcal{D}$ against* SIG*, such that:*

$$\begin{aligned}
\mathrm{Adv}^{\mathsf{IND\text{-}StAA}}_{\mathsf{AKE}}(\mathcal{A}) \leq &\frac{(2+N)S^2(S-1)}{|\mathcal{K}|} + \frac{2qSN(S+1)}{\sqrt{|\mathcal{K}|}} \\
&+ \frac{2qS(2S+1)}{\sqrt{|\mathcal{K}|}} + 2S^2(2+N) \cdot \mathrm{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{wKEM}}(\mathcal{B}) \\
&+ 2SN \cdot \mathrm{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathsf{KEM}}(\mathcal{C}) + N \cdot \mathrm{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{SIG}}(\mathcal{D}) \\
&+ 2q(S+N) \cdot \sqrt{\mathrm{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{SIG}}(\mathcal{D})} + SN(S-1)\mu(\mathsf{wEncaps}) \\
&+ (2+SN+N)S(S-1) \cdot \mu(\mathsf{wGen}) + \frac{SN(S-1)^2}{|\mathcal{K}|} \cdot \mu(\mathsf{Encaps}) \\
&+ max\{(S-1) \cdot \mu(\mathsf{wEncaps}), (S-1) \cdot \mu(\mathsf{wGen}) \cdot \mu(\mathsf{Encaps})\}
\end{aligned}$$

where μ(wGen) represents the collision probabilities of key generation for wKEM*. μ(Encaps) denotes the collision probabilities of ciphertexts for* KEM*.*

Sketch of Proof. To prove IND-StAA security of $\mathsf{AKE} = \mathsf{F}_{\mathsf{S2K}}$ [wKEM, KEM, SIG, H, G], we consider an adversary $\mathcal{A}$ with black-box access to the protocol's algorithms and oracles.

These oracles reveal keys from completed sessions, internal states, and long-term secret keys of the participating parties, as specified in the IND-StAA game. Intuitively, the adversary $\mathcal{A}$ either cannot guess the keys k_j and $\widetilde{k}$ at the same time, ensuring the pseudo-randomness of the session key, or cannot generate a valid signature of $\widetilde{pk}$ and c_j generated by itself, leading to the incompleteness of the test session.

- Honest Execution of the Test Session: We first consider the case where $\mathcal{A}$ executes the test session honestly. Note that the responder does not need to retain any state after sending its response. We assume $\mathcal{A}$ has learned the secret key of party P_j and can therefore compute k_j. Additionally, $\mathcal{A}$ can either obtain the secret key sk_i of party P_i or access the state of party P_i (which includes $\widetilde{r}$), but not both. Since $\widetilde{sk}$ is generated using the function $G(sk_i, \widetilde{r})$, $\mathcal{A}$ cannot compute $\widetilde{sk}$ without both inputs, thus, it cannot derive $\widetilde{k}$.
- Dishonest Execution of the Test Session: In the case where $\mathcal{A}$ does not execute the test session honestly, it is prohibited from obtaining the long-term secret key of the test session's peer. Since $\mathcal{A}$ modifies the exchanged messages, the test-session side decouples from the other side. If the test session is on the right-hand side, $\mathcal{A}$ can obtain k_j. The adversary may either produce a valid signature for $\widetilde{pk}\|c_j$ that appears in other sessions, or fail to generate a valid signature for an ephemeral public key it generated itself. In the former case, since $\mathcal{A}$ cannot obtain sk_i, it cannot compute $\widetilde{sk}$ then derive $\widetilde{k}$. In the latter case, the test session is not completed, falling into the trivial case. If the test session is on the left-hand side, $\mathcal{A}$ can obtain $\widetilde{k}$, but it cannot access k_j due to lack of sk_j.

Roughly, let $\mathcal{A}$ be an adversary against the IND-StAA security of AKE, establishing S sessions and issuing at most q (quantum) queries to random oracles H, and G. We will first examine the case that $\mathcal{A}$ executed the test session honestly (i.e., the case that $\mathfrak{M}(\mathsf{sID}^*) \neq \emptyset$), and examine the case that $\mathcal{A}$ tampered with the test session (i.e., the case that $\mathfrak{M}(\mathsf{sID}^*) = \emptyset$) in the second part.

$$\begin{aligned}
\mathrm{Adv}^{\mathsf{IND\text{-}StAA}}_{\mathsf{AKE}}(\mathcal{A}) &= |\Pr[\mathsf{IND\text{-}StAA}^{\mathcal{A}} \Rightarrow 1] - \frac{1}{2}| \\
&\leq |\Pr[\mathsf{IND\text{-}StAA}^{\mathcal{A}} \Rightarrow 1 \wedge \mathfrak{M}(\mathsf{sID}^*) \neq \emptyset] - \frac{1}{2}| \\
&\quad + |\Pr[\mathsf{IND\text{-}StAA}^{\mathcal{A}} \Rightarrow 1 \wedge \mathfrak{M}(\mathsf{sID}^*) = \emptyset] - \frac{1}{2}|.
\end{aligned}$$

The following two lemmas fulfill the gap to Theorem 2.

Lemma 3. *There exists an adversary $\mathcal{B}$ and an adversary $\mathcal{D}$ such that*

$$\begin{aligned}
&|\Pr[\mathsf{IND\text{-}StAA}^{\mathcal{A}} \Rightarrow 1 \wedge \mathfrak{M}(\mathsf{sID}^*) \neq \emptyset] - \frac{1}{2}| \\
\leq &\frac{2S^2(S-1)}{|\mathcal{K}|} + \frac{2qS(2S+1)}{\sqrt{|\mathcal{K}|}} + 2S(S-1)\cdot\mu(\mathsf{wGen}) + 4S^2\cdot \mathrm{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{wKEM}}(\mathcal{B}) \\
&+ 2qS\sqrt{\mathrm{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{SIG}}(\mathcal{D})} + max\{(S-1)\cdot\mu(\mathsf{wEncaps}), (S-1)\cdot\mu(\mathsf{wGen})\cdot\mu(\mathsf{Encaps})\}.
\end{aligned}$$

Refer to appendix F of our accepted work [19] for the proof of Lemma 3.

Lemma 4. *There exists an adversary $\mathcal{B}$, an adversary $\mathcal{C}$ and an adversary $\mathcal{D}$ such that*

$$\begin{aligned}
&|\Pr[\mathsf{IND\text{-}StAA}^{\mathcal{A}} \Rightarrow 1 \wedge \mathfrak{M}(\mathsf{sID}^*) = \emptyset] - \frac{1}{2}| \\
&\quad \leq \frac{2qSN(S+1)}{\sqrt{|\mathcal{K}|}} + \frac{NS^2(S-1)}{|\mathcal{K}|} + \frac{SN(S-1)^2}{|\mathcal{K}|} \cdot \mu(\mathsf{Encaps}) \\
&\quad\quad + (S+1)SN(S-1) \cdot \mu(\mathsf{wGen}) + SN(S-1)\mu(\mathsf{wEncaps}) \\
&\quad\quad + 2NS^2 \cdot \mathrm{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{wKEM}}(\mathcal{B}) + 2SN \cdot \mathrm{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathsf{KEM}}(\mathcal{C}) \\
&\quad\quad + N \cdot \mathrm{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{SIG}}(\mathcal{D}) + 2N \cdot q\sqrt{\mathrm{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{SIG}}(\mathcal{D})}.
\end{aligned}$$

Refer to appendix G of our accepted work [19] for the proof of Lemma 4.

5.1.3 CreTAKE-K2K

Theorem 3 ($\mathsf{IND\text{-}CPA} \overset{QROM}{\Rightarrow} \mathsf{F}_{\mathsf{K2K}}[2\mathsf{KEM}, \mathsf{KEM}, H, G]$ $\mathsf{IND\text{-}StAA}$)**.** *Let* $\mathsf{KEM} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *be a δ-correct key encapsulation mechanism with key space $\mathcal{K}$ and ciphertext-collision probability* $\mu(\mathsf{Encaps})$*. Assume that the decryption-failure rate of the double-key KEM (via simple combination of two ciphertexts or adding of the second part of two ciphertexts as in Remark 1 and shown in section 3.2.1) built on* KEM *is δ_1. Let G and H (and h) be modeled as quantum random oracles with output length n. For any* $\mathsf{IND\text{-}StAA}$ *adversary $\mathcal{A}$ against* $\mathsf{AKE} = \mathsf{F}_{\mathsf{K2K}}[2\mathsf{KEM}, \mathsf{KEM}, H, G]$ *establishing S sessions between $2N$ parties, issuing at most q_R classical queries to* Reveal*, at most q_G quantum queries to G, and at most q_H quantum queries to H, (and at most q_h quantum queries to h) there exists an adversary $\mathcal{B}$ against the* $\mathsf{IND\text{-}CPA}$ *security of the underlying* KEM*, such that*

$$\begin{aligned}
\mathrm{Adv}^{\mathsf{IND\text{-}StAA}}_{\mathsf{AKE}}(\mathcal{A}) &\leq 4N^2S(q_G + q_h + 2q_R)\sqrt{Adv^{\mathsf{IND\text{-}CPA}}_{\mathsf{KEM}}(\mathcal{B})} \\
&\quad + 2N^2S \cdot (2q_R + q_H + q_h + 4)2^{\frac{-n+1}{2}} + 2N \cdot (2q_h + S^2)2^{\frac{-n+1}{2}} \\
&\quad + 16N(q_G + 2q_R)^2\delta_1 + 16N(q_G + 2q_R)^2\delta + 2N^2S\mu(\mathsf{Encaps}),
\end{aligned}$$

Security Justification. The K2K framework is the KEM-only profile of CreTAKE. Its security follows the QROM analysis of the KEM-only AKE transformation in [24]. When the double-key KEM is instantiated by a simple combination of a CCA-secure KEM with a CPA-secure KEM, it is similar with the FSXY and revised version at by Hövelmanns, Kiltz, Schäge, and Unruh [11], which identifies IND-StAA as the security level achieved by the FSXY-type KEM-only construction and gives the above concrete bound for the corresponding two-message AKE transformation. Therefore, we do not repeat the full hybrid proof here.

The correspondence is direct: the long-term KEM credentials of the initiator and the responder play the roles of the two long-term public-key encryption keys, the initiator-generated ephemeral KEM key pair corresponds to $(\widetilde{pk}, \widetilde{sk})$, and the ciphertexts exchanged in K2K correspond to the three ciphertexts protected in the HKSU proof. In an honest execution of the test session, the adversary may learn long-term keys or session state subject to the IND-StAA freshness condition, but at least one of the hidden plaintexts remains unknown. In a tampered execution, the test session is decoupled from its peer session, and the IND-StAA restriction pre-

vents the adversary from simultaneously using the peer long-term key and the test-session state to recover all hidden values. The remaining hidden plaintext is protected by the IND-CPA security and disjoint simulatability of the underlying encryption scheme, and the final key derived by the random oracle is therefore pseudorandom. Consequently, K2K inherits the IND-StAA guarantee stated in Theorem 3.

5.1.4 CreTAKE-S2S

Theorem 4 (Security of CreTAKE-S2S)**.** *Let* wKEM *be an* IND-CPA *secure KEM with key space* $\mathcal{K}$*, and let* SIG *be a* EUF-CMA *secure signature scheme. Let* H *and* G *be modeled as quantum random oracles. Let* $\mathcal{A}$ *be an* IND-AA *adversary against* $\mathsf{AKE} = \mathsf{F}_{S2S}[\mathsf{wKEM}, \mathsf{SIG}, H, G]$ *establishing* S *sessions between* $2N$ *parties and making at most* q *quantum queries to the random oracles* H *and* G*. Then we can construct an* IND-CPA *adversary* $\mathcal{B}$ *against* wKEM *and* EUF-CMA *adversaries* $\mathcal{D}_I$ *and* $\mathcal{D}_R$ *against* SIG *such that:*

$$\begin{aligned}\mathrm{Adv}^{\mathsf{IND\text{-}AA}}_{\mathsf{AKE}}(\mathcal{A}) \leq & \frac{(2+N)S^2(S-1)}{|\mathcal{K}|} + \frac{2qS(2S+SN+1)}{\sqrt{|\mathcal{K}|}} + (2+N)S(S-1)\cdot\mu(\mathsf{wGen}) \\ & + 2S^2(2+N)\cdot \mathrm{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{wKEM}}(\mathcal{B}) + 2q(S+N)\sqrt{\mathrm{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{SIG}}(\mathcal{D}_I)} + N\cdot \mathrm{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{SIG}}(\mathcal{D}_I) \\ & + \frac{N\cdot \mathrm{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{SIG}}(\mathcal{D}_R)}{1-(S-1)\mu(\mathsf{wEncaps})} + \max\{(S-1)\cdot\mu(\mathsf{wEncaps}), (S-1)\cdot\mu(\mathsf{wGen})\}.\end{aligned}$$

Here $\mu(\mathsf{wGen})$ *represents the collision probability of key generation for* wKEM*, and* $\mu(\mathsf{wEncaps})$ *denotes the collision probability of ciphertext generation for* wKEM*.*

Sketch of Proof. The proof follows the proof strategy for the S2K framework. The only structural difference is that the responder-side KEM credential in S2K is replaced by a responder-side signature credential in S2S. Therefore, the parts of the S2K proof that rely on the IND-CCA security of the responder KEM are replaced by reductions to the EUF-CMA security of the responder signature scheme.

In an honest execution of the test session, the responder honestly encapsulates $\bar{k}$ under the initiator's ephemeral wKEM public key $\widetilde{pk}$ and signs the transcript component $\widetilde{pk}\|\bar{c}$. As in the S2K proof, the adversary may obtain either the initiator's long-term signing key or the initiator's session state, but the freshness condition prevents it from obtaining both. Since $\widetilde{sk}$ is derived as a function of both the long-term secret key and the stored randomness through G, the adversary cannot reconstruct $\widetilde{sk}$ and therefore cannot recover $\bar{k}$ except with the advantage of breaking the IND-CPA security of wKEM or guessing the hidden value through the random oracle.

In a tampered execution, the test session is no longer partnered with an honest peer session. If the adversary changes the initiator's first flow, then acceptance by the responder requires a valid initiator signature on the modified ephemeral public key; otherwise the responder rejects. If the adversary changes the responder's flow, then acceptance by the initiator requires a valid responder signature on $\widetilde{pk}\|\bar{c}$; otherwise the initiator rejects. Hence, every successful tampering event that is not already covered by the wKEM indistinguishability and collision terms yields an EUF-CMA forgery against one of the two signature credentials. This gives the advantage bound in Theorem 4.

5.2 Practical Security

As to concrete security, the overall security of the framework is inherited from its individual components, where the aggregate security strength is upper-bounded by the module with the minimum bit-security. For example, if a KEM with 128-bit security is jointly used with a signature scheme with 256-bit security, then the overall bit security should be 128-bit.

Here we present the available security categories of exemplary KEM and signature modules.

For PolarLAC, there are five different security categories, namely light, 128, 256, 512, and 512* bit-security levels, however, we only use the latter four categories. Here the security is estimated via the Lattice-Estimator tool [1], and is performed in two cost models: the core-SVP model [2] which is relatively conservative, and the refined BKZ model (sometimes denoted as MATZOV model) [20], which is considered as more accurate. The parameter sets and corresponding security levels of PolarLAC are illustrated in the following table 2. Note that the CPA version and CCA version differ only in secret key sizes. Since we employ the algorithms in a black-box way, the detailed choices of internal parameters are omitted.

Categories	k	n	q	**pk**	**sk**		**ct**	DFR	Bit-Security (C/Q)	
					CPA	CCA			core-SVP	refined BKZ
128	2	256	257	530	1024	1570	640	2^{-146}	132.5/ 122.6	154.5/ 148.1
256	2	512	257	1060	2048	3140	1280	2^{-265}	261.6/ 240.8	280.8/ 263.4
512	2	1024	257	2116	4096	6276	2560	2^{-324}	512.1/ 482.5	527.3 / 499.3
512*	2	1024	769	2522	4096	6682	2970	2^{-545}	527.1/ 484.7	540.1/ 500.3

pk public key size (bytes) **sk** secret key size (bytes)
DFR decryption failure rate **ct** ciphertext size (bytes)

Table 2: Parameter Sets of PolarLAC

For BiT, it provides six different security categories, targeting 100, 128, 192, 256, 384, 512-bit security strength respectively. To match the security categories of KEM, we only choose the 128, 256 and 512 security categories of BiT. Note that the hardness of BiT is evaluated in terms of both MLWE and MSIS problems. Besides, though BiT could achieve strong unforgeability (SUF), we only cite its security strength of existential unforgeability, which is a few bits higher than that of SUF.

Table 3: Parameter, Security, Size of BiT

Security Level	128	256	512
q [modulus]	26881	119297	520193
d [order of the polynomial]	256	512	1024
(n, m) [dimension of pk]	(3, 3)	(3, 3)	(3,3)
MLWE Hardness (Core-SVP)			
Classical	129	256	512
Quantum	118	234	465
MSIS Hardness (Core-SVP)			
Classical	157	301	597
Quantum	142	273	541
pk (bytes)	1048	2144	5056
sig (bytes)	1504	3456	6695
pk+sig (bytes)	2552	5600	11751

For ZEN, the security strength is evaluated with regard to both NTRU and RLWE problems, and six security categories are provided: a light version with a core-SVP strength below 128 bits, alongside 128, 256, and 512-bit versions, the latter two of which also include additional non-compression variants. We only choose the 128, 256 and 512 security categories, as illustrated in the following table 4.

Table 4: ZEN parameters

scheme	n	q	PK	CT	CoreSVP		DFR
					Classical	Quantum	
ZEN-128	512	769	615	512	128	116	2^{-128}
ZEN-256	1024	769	1229	1024	257	233	2^{-175}
ZEN-512	2048	769	2458	2048	528	490	2^{-179}

For details of the theoretical and concrete security analysis of the exemplary schemes, please refer to the respective documents.

6 Performance Evaluation

This section evaluates the performance of all submitted CreTAKE instances across the 128-bit, 256-bit, and 512-bit security levels. The evaluation covers the four CreTAKE frameworks in the order K2S, S2K, K2K, and S2S, with concrete instantiations based on PolarLAC, ZEN, and BiT. At the 128-bit and 256-bit security levels, each framework is instantiated with the corresponding PolarLAC- and ZEN-based variants. At the 512-bit security level, the K2K framework includes

three instances: one based on ZEN-512 and two based on PolarLAC-512 and PolarLAC-512*. The PolarLAC-512 and PolarLAC-512* K2K instances use the compressed double-key KEM constructions specified in Figures 8 and 9, whereas the remaining K2K instances use the generic double-key KEM construction in Figure 7. For every submitted instance, we report results for both the portable ISO C reference implementation and the AVX2-optimized implementation.

6.1 Experimental Setup and Results

We evaluate the portable ISO C reference implementation and the AVX2-optimized implementation of the submitted CreTAKE instances at the 128-bit, 256-bit, and 512-bit security levels. The evaluation covers the CreTAKE-K2S, CreTAKE-S2K, CreTAKE-K2K, and CreTAKE-S2S frameworks instantiated with PolarLAC, ZEN, and BiT. The two implementations follow the same protocol specifications, interfaces, message formats, and parameter sets. The reference implementation does not use platform-specific vector instructions, whereas the optimized implementation uses the AVX2-enabled implementations of the underlying primitives on supported 64-bit x86 processors.

The CreTAKE code implements the protocol layer. Details of the underlying primitive implementations and their primitive-level optimizations are given in the corresponding PolarLAC, ZEN, and BiT specifications. The results below therefore report the end-to-end computation and outgoing communication costs of the resulting CreTAKE instances.

The experiments were performed on the platform summarized in Table 5. All implementations were built using the Makefiles included in the submission package; primitive-specific compilation options are defined in the corresponding implementation directories.

Table 5: Benchmark platform and software configuration.

Item	Configuration
Processor	Intel Core i7-1360P
Clock frequency	2.2 GHz
Memory	16 GB RAM
Operating system	Ubuntu 24.04.1 under WSL 2, x86-64
Compiler	GCC 13.3.0
Build system	GNU Make 4.3
Reference implementation	Portable ISO C
Optimized implementation	x86-64 with AVX2

All timings are averaged over 10,000 executions and are reported in microseconds. **Setup** denotes long-term credential generation. **Init** denotes generation of the initiator's first protocol message. **Der** denotes all computation after receiving a peer message, including response generation where applicable and session-key derivation. **Online** denotes the total per-session computation excluding Setup; it equals Init plus Der for the initiator and Der for the responder. **BW** denotes the number of protocol-message bytes sent by the corresponding party. All submitted instances use two messages, i.e., one round trip. The responder has no separate Init phase, which is marked by "–".

Tables 6–8 report the results for the portable reference implementation, while Tables 9–11

report the corresponding results for the AVX2-optimized implementation.

Table 6: CreTAKE-128 Performance (Reference)

Instantiation	Framework	Party	Setup (μs)	Init (μs)	Der (μs)	Online (μs)	BW (bytes)
K2S-PolarLAC-BiT-128	K2S	I	40.22	39.27	226.91	266.18	530
		R	123.22	–	648.52	648.52	2784
K2S-ZEN-BiT-128	K2S	I	53.89	48.84	209.47	258.31	615
		R	121.46	–	605.27	605.27	2528
S2K-BiT-PolarLAC-128	S2K	I	122.40	616.64	69.73	686.37	2674
		R	40.55	–	254.84	254.84	640
S2K-BiT-ZEN-128	S2K	I	120.52	607.21	82.98	690.20	2631
		R	52.70	–	220.18	220.18	512
K2K-PolarLAC-128	K2K	I	38.19	90.67	130.37	221.04	1170
		R	37.97	–	173.18	173.18	1280
K2K-ZEN-128	K2K	I	50.85	83.66	127.75	211.40	1127
		R	50.76	–	127.91	127.91	1024
S2S-BiT-ePolarLAC-128	S2S	I	115.95	532.46	190.55	723.01	2034
		R	114.60	–	661.07	661.07	2144
S2S-BiT-eZEN-128	S2S	I	119.14	559.94	207.50	767.44	2119
		R	118.02	–	682.59	682.59	2016

Table 7: CreTAKE-256 Performance (Reference)

Instantiation	Framework	Party	Setup (μs)	Init (μs)	Der (μs)	Online (μs)	BW (bytes)
K2S-PolarLAC-BiT-256	K2S	I	67.49	66.14	569.47	635.61	1060
		R	299.04	–	891.53	891.53	6016
K2S-ZEN-BiT-256	K2S	I	85.65	76.93	538.81	615.74	1229
		R	305.83	–	808.85	808.85	5504
S2K-BiT-PolarLAC-256	S2K	I	297.52	798.30	162.62	960.92	5796
		R	66.70	–	613.15	613.15	1280
S2K-BiT-ZEN-256	S2K	I	308.85	784.57	188.02	972.59	5709
		R	86.13	–	541.71	541.71	1024
K2K-PolarLAC-256	K2K	I	75.36	183.91	318.82	502.73	2340
		R	74.54	–	407.02	407.02	2560
K2K-ZEN-256	K2K	I	94.29	142.29	296.58	438.86	2253
		R	92.57	–	260.23	260.23	2048
S2S-BiT-ePolarLAC-256	S2S	I	461.95	1101.11	785.41	1886.52	4516
		R	480.25	–	1735.42	1735.42	4736
S2S-BiT-eZEN-256	S2S	I	493.95	1125.19	852.13	1977.32	4685
		R	477.44	–	1722.60	1722.60	4480

Table 8: CreTAKE-512 Performance (Reference)

Instantiation	Framework	Party	Setup (μs)	Init (μs)	Der (μs)	Online (μs)	BW (bytes)
K2S-PolarLAC-BiT-512	K2S	I	229.88	226.99	1673.85	1900.84	2116
		R	871.35	–	3389.21	3389.21	11815
K2S-ZEN-BiT-512	K2S	I	198.13	170.22	1306.49	1476.71	2458
		R	760.03	–	2630.68	2630.68	10791
S2K-BiT-PolarLAC-512	S2K	I	861.24	3023.49	578.29	3601.78	11371
		R	230.23	–	1851.81	1851.81	2560
S2K-BiT-ZEN-512	S2K	I	857.94	2754.82	562.49	3317.31	11201
		R	222.18	–	1459.37	1459.37	2048
K2K-PolarLAC-512*	K2K	I	244.63	574.63	1183.59	1758.22	5492
		R	243.67	–	1261.30	1261.30	5428
K2K-PolarLAC-512	K2K	I	209.02	495.05	1020.68	1515.73	4676
		R	210.10	–	1086.68	1086.68	4608
K2K-ZEN-512	K2K	I	212.74	307.30	730.03	1037.33	4506
		R	211.97	–	663.72	663.72	4096
S2S-BiT-ePolarLAC-512	S2S	I	881.63	2732.41	1543.75	4276.16	8811
		R	880.42	–	4008.11	4008.11	9255
S2S-BiT-eZEN-512	S2S	I	853.55	2628.39	1482.02	4110.41	9153
		R	854.03	–	3692.00	3692.00	8743

Table 9: CreTAKE-128 Performance (AVX2)

Instantiation	Framework	Party	Setup (μs)	Init (μs)	Der (μs)	Online (μs)	BW (bytes)
K2S-PolarLAC-BiT-128	K2S	I	27.24	25.89	147.32	173.21	530
		R	87.34	–	436.05	436.05	2784
K2S-ZEN-BiT-128	K2S	I	17.94	13.77	123.70	137.47	615
		R	84.44	–	403.72	403.72	2528
S2K-BiT-PolarLAC-128	S2K	I	88.55	425.26	48.95	474.22	2674
		R	27.85	–	172.00	172.00	640
S2K-BiT-ZEN-128	S2K	I	84.46	388.84	33.58	422.42	2631
		R	17.85	–	136.00	136.00	512
K2K-PolarLAC-128	K2K	I	27.53	65.44	93.78	159.22	1170
		R	27.44	–	124.98	124.98	1280
K2K-ZEN-128	K2K	I	17.70	38.92	57.38	96.30	1127
		R	17.93	–	79.42	79.42	1024
S2S-BiT-ePolarLAC-128	S2S	I	88.00	390.47	133.44	523.91	2034
		R	88.73	–	486.87	486.87	2144
S2S-BiT-eZEN-128	S2S	I	96.32	406.59	127.59	534.18	2119
		R	94.90	–	517.00	517.00	2016

Table 10: CreTAKE-256 Performance (AVX2)

Instantiation	Framework	Party	Setup (μs)	Init (μs)	Der (μs)	Online (μs)	BW (bytes)
K2S-PolarLAC-BiT-256	K2S	I	48.27	47.04	459.93	506.96	1060
		R	261.54	–	731.23	731.23	6016
K2S-ZEN-BiT-256	K2S	I	36.67	28.82	363.94	392.76	1229
		R	239.82	–	593.57	593.57	5504
S2K-BiT-PolarLAC-256	S2K	I	261.01	647.15	125.91	773.06	5796
		R	48.52	–	484.41	484.41	1280
S2K-BiT-ZEN-256	S2K	I	243.99	559.77	100.32	660.09	5709
		R	36.27	–	372.86	372.86	1024
K2K-PolarLAC-256	K2K	I	46.92	115.47	207.41	322.88	2340
		R	46.71	–	265.02	265.02	2560
K2K-ZEN-256	K2K	I	37.75	60.68	135.06	195.75	2253
		R	37.31	–	136.76	136.76	2048
S2S-BiT-ePolarLAC-256	S2S	I	259.55	563.16	409.64	972.80	4516
		R	257.34	–	909.21	909.21	4736
S2S-BiT-eZEN-256	S2S	I	248.85	534.45	374.74	909.20	4685
		R	249.06	–	847.62	847.62	4480

Table 11: CreTAKE-512 Performance (AVX2)

Instantiation	Framework	Party	Setup (μs)	Init (μs)	Der (μs)	Online (μs)	BW (bytes)
K2S-PolarLAC-BiT-512	K2S	I	175.95	173.51	1347.10	1520.61	2116
		R	749.08	–	2827.26	2827.26	11815
K2S-ZEN-BiT-512	K2S	I	110.73	83.45	1109.66	1193.11	2458
		R	735.85	–	2470.05	2470.05	10791
S2K-BiT-PolarLAC-512	S2K	I	745.66	2546.80	465.58	3012.38	11371
		R	175.12	–	1494.80	1494.80	2560
S2K-BiT-ZEN-512	S2K	I	726.18	2218.15	360.56	2578.71	11201
		R	110.68	–	1113.56	1113.56	2048
K2K-PolarLAC-512*	K2K	I	170.51	409.54	877.14	1286.68	5492
		R	170.82	–	944.52	944.52	5428
K2K-PolarLAC-512	K2K	I	172.21	410.10	839.96	1250.07	4676
		R	172.97	–	906.84	906.84	4608
K2K-ZEN-512	K2K	I	109.23	158.97	440.32	599.30	4506
		R	108.11	–	448.60	448.60	4096
S2S-BiT-ePolarLAC-512	S2S	I	733.23	2223.51	1220.97	3444.48	8811
		R	729.99	–	3252.24	3252.24	9255
S2S-BiT-eZEN-512	S2S	I	744.00	2200.96	1141.35	3342.31	9153
		R	746.19	–	3166.83	3166.83	8743

6.2 Performance Analysis

The measurements reveal a consistent framework-level pattern. The credential configuration determines the cryptographic operations assigned to each party and the primitive objects carried in each protocol message, while the concrete KEM and signature instantiations determine the resulting computation and communication costs. With the submitted BiT-, PolarLAC-, and ZEN-based instantiations, CreTAKE-K2S places the larger measured online computation and outgoing communication cost on the responder, whereas CreTAKE-S2K places the corresponding costs on the initiator. CreTAKE-K2K gives the lowest measured computational profile, while CreTAKE-S2S incurs higher signature-related costs but distributes them comparatively evenly between the two roles. These measured relationships remain stable across the 128-bit, 256-bit, and 512-bit security levels and in both the reference and AVX2 implementations. The complete credential-type matrix therefore provides four distinct operation and message layouts that yield predictable performance profiles under the submitted instantiations.

At the instantiation level, all four frameworks admit efficient concrete realizations. At the 128-bit security level, the maximum per-party online time is 0.77 ms for the reference implementation and 0.54 ms for the AVX2 implementation. The corresponding maxima are 1.98 ms and 0.98 ms at the 256-bit level, and 4.28 ms and 3.45 ms at the 512-bit level. The largest complete exchanges at these three levels are 4,178, 9,252, and 18,066 bytes, respectively. The 512-bit instances define the upper end of the suite, while the 128-bit and 256-bit instances operate with substantially lower computation and communication costs. The measurements use the SM3-based hash/XOF backend supplied with the ICCS evaluation framework. An optimized FIPS 202 SHAKE backend is expected to reduce the computation times without changing the protocol messages.

Among the concrete families, the ZEN–BiT instantiations provide the strongest overall efficiency profile. In the heterogeneous frameworks, they reduce aggregate communication by approximately 5% relative to the corresponding PolarLAC–BiT instances. In CreTAKE-K2K, ZEN gives the lowest online times and the smallest complete exchange at every security level. The PolarLAC–BiT family provides a complementary KEM backend and several distinct local profiles, including smaller initiator messages in CreTAKE-K2S and the specialized 512-bit double-key KEM constructions. The detailed analysis below therefore first examines the framework-level distribution of cost and then evaluates how the choice of concrete primitives refines each profile.

6.2.1 Framework structure and role profiles

The four frameworks determine which cryptographic operations are assigned to the initiator and responder and which primitive outputs are carried in each protocol message. The resulting computation and communication costs depend on the concrete KEM and signature instantiations. In the submitted BiT-based instances, signature processing is more expensive than the accompanying KEM operations and signature-bearing messages are comparatively large. This produces the pronounced role asymmetries observed for CreTAKE-K2S and CreTAKE-S2K.

In CreTAKE-K2S, the responder generates the signature-bearing response and performs the associated response-side KEM operations. The current instantiations therefore place the larger measured online time and outgoing message on the responder. In CreTAKE-S2K, the initiator

generates the signature-bearing first flight and performs the corresponding encapsulation, so the larger measured costs occur on the initiator. These directions follow from the framework structure, while the magnitude of the differences follows from the relative costs and output sizes of BiT and the selected KEM.

For a fixed security level and KEM family, CreTAKE-K2S and CreTAKE-S2K transmit the same aggregate amount of protocol material and differ in its direction. At the 128-bit level, for example, the PolarLAC-based instances both transmit 3,314 bytes. The CreTAKE-K2S instance divides this total into 530 bytes from the initiator and 2,784 bytes from the responder, while the CreTAKE-S2K instance divides it into 2,674 and 640 bytes. The same relationship holds at the higher security levels and for the ZEN-based instances. The heterogeneous frameworks therefore provide opposite message-direction profiles for the submitted primitive combinations.

The homogeneous frameworks assign similar classes of operations to the two roles. CreTAKE-K2K uses KEM operations at both endpoints and records the lowest online times in the current evaluation. CreTAKE-S2S performs signature-related processing at both endpoints and consequently has higher measured costs, but those costs are distributed comparatively evenly between the two parties. These observations characterize the submitted instantiations; different relative KEM and signature costs could change the degree of computational asymmetry without changing the operation layout of the four frameworks.

6.2.2 Primitive-dependent efficiency

Within each framework, the primitive selection determines the absolute computation and communication costs. The ZEN–BiT instantiations provide the strongest overall measured efficiency. In the heterogeneous frameworks, they reduce aggregate communication by approximately 5% relative to the corresponding PolarLAC–BiT instantiations at all three security levels. In CreTAKE-K2K, ZEN reduces the complete exchange by approximately 12.2% at the 128-bit and 256-bit levels and by 7.3% at the 512-bit level. It also gives the lowest measured CreTAKE-K2K online times for both parties at every security level.

The PolarLAC-based instances retain a directional communication advantage in CreTAKE-K2S. Their initiator flights are approximately 14% smaller than the corresponding ZEN-based flights: 530 instead of 615 bytes at the 128-bit level, 1,060 instead of 1,229 bytes at the 256-bit level, and 2,116 instead of 2,458 bytes at the 512-bit level. The ZEN-based responder flights are sufficiently smaller to give the lower aggregate communication cost. The KEM choice therefore affects both the total bandwidth and the message direction in which the reduction occurs.

The KEM choice has less influence on CreTAKE-S2S communication because BiT-related material accounts for most of both messages. The ZEN-based instances reduce aggregate communication by only about 1% relative to the corresponding PolarLAC-based instances, and their timing advantage is not uniform across all roles and security levels.

The three 512-bit CreTAKE-K2K instances provide further concrete profiles. Their complete exchanges are 10,920 bytes for PolarLAC-512*, 9,284 bytes for PolarLAC-512, and 8,602 bytes for ZEN-512. The PolarLAC-512 instance is smaller and faster than PolarLAC-512* in both implementations, whereas PolarLAC-512* adopts the most conservative decryption-failure-rate setting. The ZEN-512 instance gives the lowest measured computation and communication of

the three.

6.2.3 Scaling and implementation effects

Both computation and communication increase with the security level. The growth rate differs across frameworks because their protocol paths contain different combinations of KEM, signature, and hash operations. The measured role profiles nevertheless remain stable: CreTAKE-K2S places the larger cost on the responder, CreTAKE-S2K places it on the initiator, CreTAKE-K2K gives the lowest computational profile, and CreTAKE-S2S distributes its higher signature-related cost comparatively evenly.

The AVX2 implementation reduces every reported online time, with measured speedups ranging from approximately 1.07× to 2.24×. The improvement varies by role and framework according to the mixture of primitive operations executed along each path. The reference and AVX2 implementations produce identical message sizes, so the optimized implementation changes local execution time without changing the protocol interface or wire format.

All submitted instances complete in two messages and one round trip. Overall, the framework fixes the operation and message layout, the concrete primitives determine the resulting cost profile, and the optimized implementation reduces that cost without changing the protocol behavior.

7 Feature Statements

CreTAKE is a credential-typed two-message authenticated key exchange suite. Its four frameworks cover every ordered combination of KEM-based and signature-based long-term credentials, while its concrete instance set provides multiple primitive and security-level choices. The principal features of the suite are summarized below.

Innovativeness. CreTAKE treats the credential types of the two endpoints as an explicit design dimension of two-message AKE. The framework layer contains CreTAKE-K2S, CreTAKE-S2K, CreTAKE-K2K, and CreTAKE-S2S, thereby covering the complete ordered credential-type matrix formed by KEM-based and signature-based credentials. This organization supports both homogeneous post-quantum deployments and heterogeneous migration stages in which the two parties have adopted different credential types.

The four frameworks also consolidate distinct two-message AKE techniques within one suite. The heterogeneous frameworks support the two possible KEM–signature credential orders. The CreTAKE-K2K framework uses the double-key KEM abstraction and accommodates both a generic separate-ciphertext construction and compact structure-aware constructions. The CreTAKE-S2S framework combines mutual signature authentication with an ephemeral wKEM exchange and binds the initiator's ephemeral key generation to its long-term signing key and fresh session randomness. Each framework is analyzed under the applicable IND-AA or IND-StAA security notion in the QROM.

Simplicity. All four frameworks use the same two-message structure and the same high-level interface

$$(\mathsf{KG}, \mathsf{Init}, \mathsf{Der}_{\mathsf{resp}}, \mathsf{Der}_{\mathsf{init}}).$$

The initiator generates the first message, the responder returns the second message together with its session key, and the initiator derives the matching session key from the response and its stored state. Every instance therefore completes in one round trip.

The constructions are expressed using KEM, signature, double-key KEM, and hashing interfaces. They do not require credential conversion, an additional online trusted party, or an extra confirmation message. The same framework descriptions are reused across security levels and primitive families, which reduces duplication in the specification, implementation, and evaluation.

Flexibility. The suite provides flexibility at both the framework and instantiation layers. At the framework layer, a deployment selects CreTAKE-K2S, CreTAKE-S2K, CreTAKE-K2K, or CreTAKE-S2S according to the long-term credential types already provisioned at the initiator and responder. The heterogeneous frameworks also provide different role profiles. CreTAKE-K2S places more of the communication and computation on the responder, while CreTAKE-S2K places more on the initiator. A deployment can therefore select the credential order that matches the capabilities and communication constraints of its endpoints.

At the instantiation layer, the frameworks are realized with PolarLAC, ZEN, and BiT at the 128-bit, 256-bit, and 512-bit security levels. The two KEM families provide complementary public-key, ciphertext, and performance profiles. The CreTAKE-K2K instances additionally offer generic and compact double-key constructions where the underlying parameters permit them. These choices produce 25 concrete instances covering different security, bandwidth, computation, and decryption-failure requirements.

Compatibility. CreTAKE uses KEM-based and signature-based credentials through their standard algorithmic interfaces. An endpoint needs only the credential type required by its selected framework and is not required to maintain both a KEM credential and a signature credential. This property supports staged migration in systems where credential issuance, certification, trust-store updates, and device replacement proceed at different rates.

The suite is specified at the AKE layer and does not prescribe the record format or auxiliary services of a particular application protocol. A secure-channel protocol can bind a selected CreTAKE instance to its own identity representation, transcript format, negotiation procedure, and record layer. This is protocol-level compatibility and does not imply direct wire compatibility with an existing handshake.

The concrete instances use PolarLAC, ZEN, and BiT, which are submitted separately as algorithm candidates. Their security categories and implementation interfaces are aligned with the corresponding CreTAKE instances, allowing the primitive implementations to be reused by the AKE implementation.

Extensibility. The separation between frameworks and instantiations allows the suite to be extended without redesigning its credential-type structure. A new KEM or signature family can instantiate an existing framework when it provides the required security interface, correctness bound, key space, and target security strength. Additional parameter sets can likewise be added while retaining the same framework algorithms and message flow.

The CreTAKE-K2K framework provides two extension routes. Any suitable KEM family can be incorporated through the generic double-key construction. A primitive with appropriate shared algebraic structure may additionally support a compact double-key construction, provided that its security and decryption-failure probability are established. This permits future instantiations to exploit primitive-specific structure without making the framework layer dependent on that structure.

Performance advantages on variant platforms. All submitted instances require two messages and one round trip, so their round complexity is independent of the selected framework, primitive family, security level, or implementation platform. The framework choice determines the distribution of operations between the two parties. CreTAKE-K2K has the lowest measured computational profile, CreTAKE-S2S provides the homogeneous signature-based profile, and the heterogeneous frameworks allow the larger cost to be placed on either the initiator or the responder.

The submission includes a portable ISO C reference implementation and an AVX2-optimized implementation for all instances. The AVX2 implementation reduces every reported online execution time, with measured speedups ranging from approximately 1.07× to 2.24×. Both implementations use the same interfaces, parameter sets, message formats, and communication sizes. Platform-specific optimization therefore improves local computation without changing protocol behavior or interoperability.

The AKE implementation primarily coordinates calls to the underlying primitive implementations. Optimizations of PolarLAC, ZEN, and BiT can consequently be reused by the corresponding CreTAKE instances. The same modular organization also permits future optimized primitive implementations for embedded processors or hardware platforms to be incorporated without modifying the framework definitions or protocol message formats.

References

[1] Albrecht, M.R.: Lattice Estimator. `https://github.com/malb/lattice-estimator`

[2] Alkim, E., Ducas, L., Pöppelmann, T., Schwabe, P.: Post-quantum key exchange - A new hope. In: 25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016. pp. 327–343 (2016), `https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/alkim`

[3] Boyd, C., de Kock, B., Millerjord, L.: Modular design of kem-based authenticated key exchange. In: Australasian Conference on Information Security and Privacy. pp. 553–579. Springer (2023)

[4] Dierks, T., Rescorla, E.: The transport layer security (tls) protocol version 1.3. RFC 8446 (2018), available: https://tools.ietf.org/html/rfc8446

[5] Donenfeld, J.A.: Wireguard: Next generation kernel network tunnel. In: NDSS. pp. 1–12 (2017)

[6] National Cybersecurity Center of Excellence, U.: Migration to post-quantum cryptography. https://www.nccoe.nist.gov/crypto-agility-considerations-migrating-post-quantum-cryptographic-algorithms

[7] Fujioka, A., Suzuki, K., Xagawa, K., Yoneyama, K.: Strongly secure authenticated key exchange from factoring, codes, and lattices. In: Fischlin, M., Buchmann, J., Manulis, M. (eds.) Public Key Cryptography - PKC 2012. Lecture Notes in Computer Science, vol. 7293, pp. 467–484. Springer (2012)

[8] Fujioka, A., Suzuki, K., Xagawa, K., Yoneyama, K.: Practical and post-quantum authenticated key exchange from one-way secure key encapsulation mechanism. In: Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security. pp. 83–94 (2013)

[9] Fujioka, A., Suzuki, K., Xagawa, K., Yoneyama, K.: Strongly secure authenticated key exchange from factoring, codes, and lattices. Designs, Codes and Cryptography **76**, 469–504 (2015)

[10] Hofheinz, D., Hövelmanns, K., Kiltz, E.: A modular analysis of the fujisaki-okamoto transformation. In: TCC (1). Lecture Notes in Computer Science, vol. 10677, pp. 341–371. Springer (2017)

[11] Hövelmanns, K., Kiltz, E., Schäge, S., Unruh, D.: Generic authenticated key exchange in the quantum random oracle model. IACR Cryptology ePrint Archive **2018**, 928 (2018)

[12] Hövelmanns, K., Kiltz, E., Schäge, S., Unruh, D.: Generic authenticated key exchange in the quantum random oracle model. In: Public Key Cryptography (2). Lecture Notes in Computer Science, vol. 12111, pp. 389–422. Springer (2020)

[13] Hülsing, A., Ning, K.C., Schwabe, P., Weber, F.J., Zimmermann, P.R.: Post-quantum wireguard. In: 2021 IEEE Symposium on Security and Privacy (SP). pp. 304–321. IEEE (2021)

[14] Jager, T., Kiltz, E., Riepel, D., Schäge, S.: Tightly-secure authenticated key exchange, revisited. In: Annual international conference on the theory and applications of cryptographic techniques. pp. 117–146. Springer (2021)

[15] Kaufman, C., Hoffman, P.E., Nir, Y., Eronen, P., Kivinen, T.: Internet Key Exchange Protocol Version 2 (IKEv2). RFC 7296 (Oct 2014). https://doi.org/10.17487/RFC7296, https://www.rfc-editor.org/info/rfc7296

[16] Krawczyk, H.: Sigma: The sign-and-mac approach to authenticated diffie-hellman and its use in the ike protocols. In: Annual international cryptology conference. pp. 400–425. Springer (2003)

[17] Krawczyk, H.: Hmqv: A high-performance secure diffie-hellman protocol. In: Annual international cryptology conference. pp. 546–566. Springer (2005)

[18] Law, L., Menezes, A., Qu, M., Solinas, J., Vanstone, S.: An efficient protocol for authenticated key agreement. Designs, Codes and Cryptography **28**, 119–134 (2003)

[19] Lu, X., Cheng, Y., He, J., Xue, H., Liu, Y.: Hetake: Heterogeneous authenticated key exchange for post-quantum migration. In: ACISP 2026 (2026), accepted for publication

[20] MATZOV: Report on the security of LWE: Improved dual lattice attack. Tech. rep., Zenodo (2022). https://doi.org/10.5281/zenodo.6493704, `https://doi.org/10.5281/zenodo.6493704`

[21] NIST: Post-quantum cryptography standardization. `https://csrc.nist.gov/projects/post-quantum-cryptography`

[22] Schwabe, P., Stebila, D., Wiggers, T.: Post-quantum tls without handshake signatures. In: Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security. pp. 1461–1480 (2020)

[23] Xu, X., Xue, H., Wang, K., Au, M.H., Tian, S.: Strongly secure authenticated key exchange from supersingular isogenies. In: Galbraith, S.D., Moriai, S. (eds.) Advances in Cryptology - ASIACRYPT 2019 - 25th International Conference on the Theory and Application of Cryptology and Information Security, Kobe, Japan, December 8-12, 2019, Proceedings, Part I. Lecture Notes in Computer Science, vol. 11921, pp. 278–308. Springer (2019). https://doi.org/10.1007/978-3-030-34578-5_11, `https://doi.org/10.1007/978-3-030-34578-5_11`

[24] Xue, H., Au, M.H., Yang, R., Liang, B., Jiang, H.: Compact authenticated key exchange in the quantum random oracle model. IACR Cryptol. ePrint Arch. **2020**, 1282 (2020)

[25] Xue, H., Lu, X., Li, B., Liang, B., He, J.: Understanding and constructing ake via double-key key encapsulation mechanism. In: Advances in Cryptology–ASIACRYPT 2018: 24th International Conference on the Theory and Application of Cryptology and Information Security, Brisbane, QLD, Australia, December 2–6, 2018, Proceedings, Part II 24. pp. 158–189. Springer (2018)

[26] Zhang, J., Zhang, Z., Ding, J., Snook, M., Dagdelen, Ö.: Authenticated key exchange from ideal lattices. In: Advances in Cryptology-EUROCRYPT 2015: 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II 34. pp. 719–751. Springer (2015)