A preliminary version of this paper appears in the proceedings of the International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT 2026).
This is the full version.

Tightly Secure and Modular Post-Quantum TLS Handshakes

Biming Zhou[1⋆] Jinrong Chen[2⋆] Haodong Jiang[3]
Rongmao Chen[2] Yunlei Zhao[1]

[1] Fudan University [2] National University of Defense Technology
[3] Information Engineering University
bmzhou22@m.fudan.edu.cn, jinrongchen@nudt.edu.cn, hdjiang13@163.com,
chromao@nudt.edu.cn, ylzhao@fudan.edu.cn

Abstract

In the post-quantum (PQ) migration of the TLS 1.3 handshake, the classical Diffie–Hellman key exchange is replaced with a post-quantum key encapsulation mechanism (KEM). Furthermore, KEMTLS builds upon PQ TLS 1.3 by replacing signature-based authentication with KEM-based authentication. However, all existing analyses of PQ TLS 1.3 and KEMTLS incur at least quadratic reduction losses in the number of sessions, and achieving a tightly secure construction remains an open problem.

To address this gap, we present the first unified, tight, and modular reduction framework for PQ TLS 1.3 in the standard model and KEMTLS in the random oracle model (ROM). Our framework tightly reduces the security of these protocols to the security of three core components: (i) the MUC-IND-1RCCA or MUC-IND-1CCA security of the underlying ephemeral KEM, (ii) the MU-(dual)-PRF security of HKDF (for PQ TLS 1.3), and (iii) the MU-EUF-CMA^c security of the signature scheme (for PQ TLS 1.3) or the MUC-OW-ChCCA* security of the static KEM (for KEMTLS). Furthermore, we establish the first concrete MU-(dual)-PRF bounds for HMAC and HKDF in the quantum-accessible random oracle model (QROM).

Instantiating this framework with tightly secure LWE-based KEM and signature schemes yields the first (almost) tightly secure constructions of PQ TLS 1.3 and KEMTLS under the LWE assumption. Moreover, our framework demonstrates that the weaker IND-1RCCA security notion for the ephemeral KEM already suffices to achieve security for both PQ TLS 1.3 and KEMTLS, thereby enabling simpler instantiations.

Keywords. PQ TLS 1.3 · KEMTLS · tight reduction

1 Introduction

The Transport Layer Security (TLS 1.3) protocol [Res18] is a cornerstone of modern secure communication, widely deployed in applications such as web browsing, email, and instant messaging. However, the ephemeral Diffie–Hellman key exchange used in classical TLS 1.3 [DH76] is vulnerable to quantum attacks due to Shor's algorithm [Sho94]. As the NIST standardization of post-quantum cryptography progresses [Nat24b, Nat24a, Nat24c, Nat25], transitioning TLS 1.3 to achieve post-quantum (PQ) security has become a critical research focus. To resist "harvest-now, decrypt-later" attacks, the Diffie–Hellman key exchange should be replaced with a PQ-secure key encapsulation mechanism (KEM). Furthermore, achieving full PQ security, including authentication, requires the adoption of a PQ-secure digital signature scheme (SIG).

⋆These authors contributed equally to this work.

The classical TLS 1.3 protocol [Res18, DFGS21], as well as other cryptographic protocols [BFG+22, ADH+22], relies on the PRF-ODH assumption [BFGJ17] to achieve security, typically incurring a quadratic loss of $\mathcal{O}(t_s^2)$[1] in the number of sessions. However, achieving a *tight security reduction* is essential for the practical deployment of cryptographic protocols, as it ensures that the theoretical guarantees of the underlying primitives yield meaningful concrete security in real-world implementations. Motivated by this, Diemert and Jager [DJ21] and, independently, Davis and Günther [DG21] established tight security proofs for the TLS 1.3 full 1-RTT handshake protocol. Their analyses build upon the Strong Diffie–Hellman (SDH) assumption [ABR01] and rely on the multi-user security of digital signatures [BHJ+15] under adaptive corruption, as well as on the multi-user security of pseudorandom functions (PRFs) [BCK96b] in the random oracle model (ROM) [BR93]. Subsequently, Davis et al. [DDGJ22] presented tight security proofs for the TLS 1.3 PSK handshake modes in the indifferentiability framework [BDG20] by introducing a new abstraction of the TLS 1.3 key schedule. In particular, their analysis of the PSK-(EC)DHE mode also relies on the SDH assumption [ABR01] in the ROM. However, there is no SDH-like assumption available in the post-quantum setting, and such tight classical reductions crucially exploit the random self-reducibility of the SDH problem, which does not hold for post-quantum assumptions such as lattice-based assumptions.

For post-quantum (PQ) TLS 1.3, several works [DFGS21, HV22, ZJZ24] have shown that PQ TLS 1.3 remains secure if the underlying KEM satisfies IND-1CCA security (with an EUF-CMA SIG and secure PRFs). Informally, IND-1CCA security for KEMs ensures that any polynomial-time adversary cannot distinguish between a legitimately generated key and a random key when allowed at most one decapsulation query. The resulting reductions [HV22, ZJZ24] from the multi-stage security of PQ TLS 1.3 to an IND-1CCA-secure KEM also incur a quadratic loss of $\mathcal{O}(t_s^2)$. Subsequent works [HV22, ZJZ24, CZC+26] refined this analysis by relying on intermediate security notions for KEMs (see Section 1.2 for a detailed discussion), showing that OW-CPA-secure KEMs suffice to establish the security of PQ TLS 1.3 in both the ROM and the quantum-accessible random oracle model (QROM) [BDF+11], and that the reduction loss for KEMs remains at least $\mathcal{O}(t_s^2)$. Note that all existing security reductions for PQ TLS [DFGS21, HV22, ZJZ24, CZC+26] incur a quadratic loss of $\mathcal{O}(t_s \cdot t_u)$ in the signature component as a function of the numbers of users and sessions.

Among the NIST-standardized PQ algorithms, signature schemes such as ML-DSA [Nat24a] typically have larger sizes and higher computational costs than KEMs such as ML-KEM [Nat24b] at the same security level. Motivated by this gap, Schwabe, Stebila, and Wiggers [SSW20] proposed KEMTLS, a TLS 1.3 variant that replaces signature-based authentication with KEM-based authentication. The security of KEMTLS [SSW20, SSW21] relies on IND-1CCA security for the ephemeral KEM and IND-CCA security for the static KEM, with reduction losses of $\mathcal{O}(t_s^2)$ and $\mathcal{O}(t_s \cdot t_u)$, respectively. In summary, all known reductions for PQ TLS 1.3 and KEMTLS incur a quadratic loss of $\mathcal{O}(t_s^2)$ in the ephemeral KEM component and an additional $\mathcal{O}(t_s \cdot t_u)$ loss in the signature or static KEM component. This motivates the central question of this work:

Is it possible to eliminate the quadratic loss and obtain a tight reduction for PQ TLS 1.3 and KEMTLS, particularly under the Learning with Errors (LWE) assumption?

1.1 Our Contributions

We answer this question by revisiting the security of the PQ TLS 1.3 and KEMTLS handshake protocols in a modular and tight manner. Our main contributions, illustrated in Fig. 1, are summarized as follows.

[1] For simplicity, we omit the contribution of the signature component here, as it introduces a smaller loss of $\mathcal{O}(t_s \cdot t_u)$ compared to that of the PRF-ODH term, where t_u and t_s denote the numbers of users and sessions, respectively.

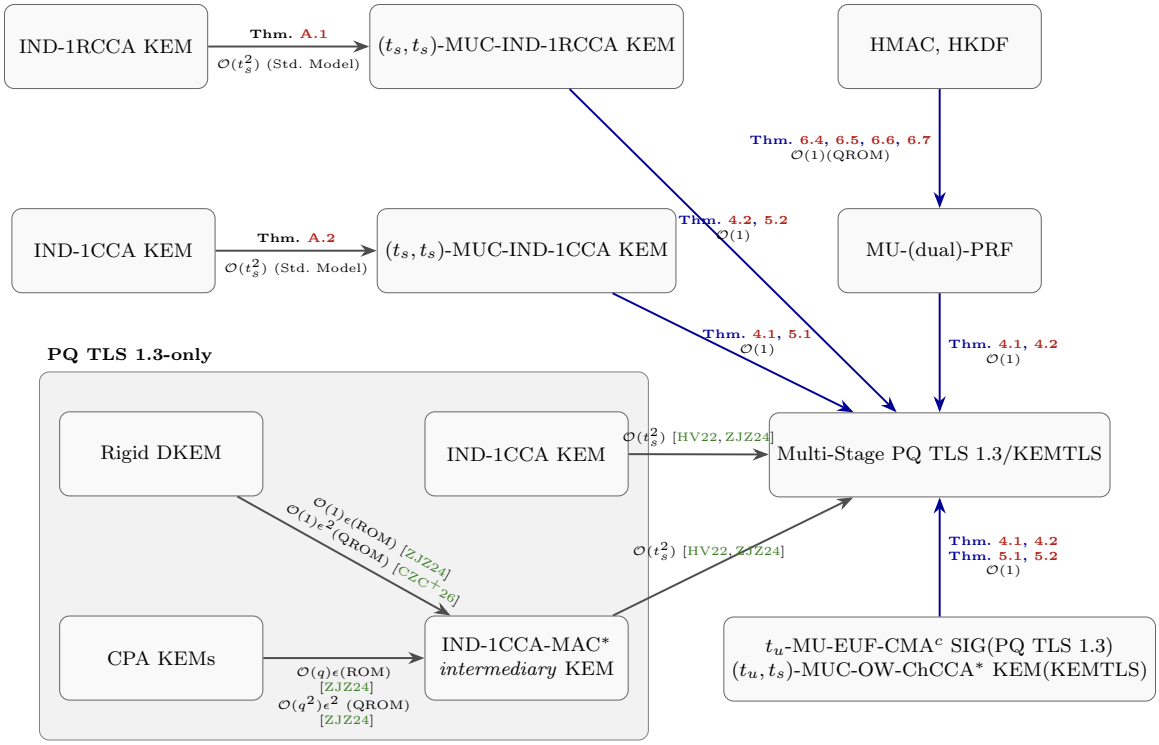


Figure 1: Overview of results from our framework for PQ TLS 1.3, KEMTLS, and related works. Blue arrows denote the tight reductions established in this paper, and the gray-shaded region highlights prior reductions specific to PQ TLS 1.3. In PQ TLS 1.3, security relies on a t_u-MU-EUF-CMA^c-secure signature scheme, whereas in KEMTLS, it relies on a (t_u, t_s)-MUC-OW-ChCCA*-secure KEM, as established in Thm. 4.1, 4.2, 5.1, and 5.2. Note that the security analysis of PQ TLS 1.3 is in the standard model, whereas that of KEMTLS is in the ROM.

1. We establish a unified, modular, and tight reduction framework for analyzing the multi-stage security of the full 1-RTT handshakes of PQ TLS 1.3 and KEMTLS. Our results demonstrate that:
   - the security of PQ TLS 1.3 can be tightly reduced to (i) the MUC-IND-1CCA[2] or MUC-IND-1RCCA[3] security of the underlying KEM, (ii) the MU-EUF-CMA^c security of the signature scheme, and (iii) the MU-(dual)-PRF security of HKDF in the standard model.
   - the security of KEMTLS can be tightly reduced to (i) the MUC-IND-1CCA or MUC-IND-1RCCA security of the ephemeral KEM, (ii) the MUC-OW-ChCCA*[4] security of the static KEM in the ROM, where the domain-separated HKDF and the HMAC are modeled as independent random oracles.

   By instantiating our framework with the tight MUC-IND-CCA-secure KEM from [PWZ23b, HLWG23], the MU-EUF-CMA^c signature scheme from [PW22], and the MUC-OW-ChCCA*-

[2]MUC and MU denote the multi-user, multi-challenge and multi-user settings, respectively. The superscript ^{c} indicates adaptive corruption. In this work, dual-PRF security refers to PRF security with the second argument serving as the secret key.

[3]MUC-IND-1RCCA is the multi-user, multi-challenge variant of IND-1RCCA. The IND-1RCCA notion is a relaxation of IND-1CCA, where the decapsulation of any ciphertext $c \neq c^*$ that produces one of the challenge keys $\{k_0, k_1\}$ returns a special symbol test.

[4]The MUC-OW-ChCCA* notion is a relaxed version of OW-ChCCA from [PWZ23a] obtained by removing the Reveal oracle available to the adversary.

secure KEM from [PWZ23a], we obtain the first PQ TLS 1.3 and KEMTLS protocols with a tight[5] reduction from the LWE assumption; see Table 1 for a detailed comparison.
2. We establish the first concrete MU-PRF bounds for HMAC and HKDF used in PQ TLS 1.3 and KEMTLS in the QROM. We also establish concrete MU-dual-PRF bounds for HMAC and HKDF.Extract in both the ROM and QROM.

Our security models largely follow prior Multi-Stage formulations for PQ TLS 1.3 [DFGS21, HV22] and KEMTLS [SSW20], except in their treatment of internal stage keys. Whereas prior formulations synchronize these keys in both Test branches, our model preserves the actual protocol state in the real branch and consistently propagates the random key only in the ideal branch. This prevents KEM decapsulation failures from being masked in the real experiment and more faithfully captures the actual protocol execution. See Section 3 for more details.

Table 1: Comparison of security reductions for (PQ) TLS 1.3 and KEMTLS protocols. The underlying KEM and signature (SIG) constructions are based on the LWE assumption; λ denotes the security parameter. The tight PQ TLS 1.3 construction in the standard model is obtained by instantiating the tight MUC-IND-CCA KEM and the MU-EUF-CMA^c signature scheme from [HLWG23]. In the ROM, tight constructions are obtained by instantiating the tight MUC-IND-CCA KEM from [PWZ23b] together with either the MU-EUF-CMA^c scheme from [PW22] for PQ TLS 1.3 or the OW-ChCCA scheme from [PWZ23a] for KEMTLS.

Protocol	**Security assumption**	**Security loss**	**Model**
TLS 1.3 [DG21, DJ21]	SDH	$\mathcal{O}(1)$	ROM
PQ TLS 1.3 [HV22, ZJZ24, CZC+26]	LWE	$\Omega(t_s^2)$	ROM/Std
PQ TLS 1.3 (this work)	LWE	$\mathcal{O}(\lambda^2)$	Std
PQ TLS 1.3 (this work)	LWE	$\mathcal{O}(\lambda)$	ROM
KEMTLS [SSW20]	LWE	$\Omega(t_s^2)$	ROM/Std
KEMTLS (this work)	LWE	$\mathcal{O}(\lambda)$	ROM

Remark 1.1. As shown in Table 1, we achieve tight security based on the LWE assumption for PQ TLS 1.3 in both the standard model and the ROM, and for KEMTLS in the ROM. We leave the construction of tightly secure PQ TLS 1.3 and KEMTLS protocols based on the LWE assumption in the QROM as an open question. The main obstacle lies in the difficulty of proving tight QROM security for the underlying schemes in [PW22, PWZ23a]. These constructions either rely on the sequential-query property of the ROM, where the adversary's queries to different inputs x_0 and x_1 are assumed to occur in a fixed order, whereas in the QROM the adversary can query both in superposition, or invoke the One-Way-to-Hiding Lemma [AHU19], which inherently incurs a square-root loss in the reduction.

Remark 1.2. Our framework demonstrates that IND-1RCCA-secure KEMs suffice to guarantee the security of PQ TLS 1.3 and KEMTLS. Since IND-1RCCA is a weaker security notion than IND-1CCA, this finding allows for simpler and more efficient constructions for PQ TLS 1.3 and KEMTLS. In particular, IND-1RCCA-secure KEMs can be efficiently constructed from OW-CPA-secure PKEs by deriving the shared key as $H(m)$, as demonstrated in [CZC+26]. In contrast, the IND-1CCA-secure KEM constructions of [HV22, JMZ23] hash both the message and the ciphertext, namely $H(m, c)$. This may incur additional overhead in practice, since ciphertexts in post-quantum KEMs such as ML-KEM [Nat24b] and HQC [MAB+25] are typically much larger than the underlying

[5]Note that the constructions in [HLWG23, PWZ23a, PWZ23b] achieve almost tight reductions, for which the loss is a small polynomial in the security parameter λ. We call them tight as well, while specifying the concrete loss in Table 1.

message m. Overall, our results offer both theoretical and practical advances toward modular and tightly secure proofs for PQ TLS 1.3 and KEMTLS, thereby strengthening confidence in their concrete post-quantum security.

1.2 Related Work

Security analysis of PQ TLS 1.3. Several works [DFGS21, HV22, ZJZ24] have established the security of PQ TLS 1.3 under IND-1CCA security of KEMs, EUF-CMA security of signatures, and PRF security of HKDF. Huguenin-Dumittan and Vaudenay [HV22] showed that OW-CPA-secure KEMs suffice for PQ TLS 1.3 in the ROM. Zhou et al. [ZJZ24] later improved these security bounds in the ROM and extended the analysis to the QROM. Both analyses rely on intermediate KEM security notions, incurring a loss of $\mathcal{O}(t_s^2)$ in the reductions from PQ TLS 1.3 security to these notions. Building on this line of work, Chen et al. [CZC+26] introduced a refined notion, IND-1CCA-1MAC, and demonstrated that the loss factors in [ZJZ24] can be further improved: $\mathcal{O}(q)$ in the ROM and $\mathcal{O}(q^2)$ in the QROM for OW-CPA-secure KEMs, and $\mathcal{O}(1)$ in both models for IND-CPA-secure KEMs by adding a short tag $H(K)$ in the KEM ciphertext. They further proved that without ciphertext expansion, the loss factors of $\mathcal{O}(q)$ (ROM) and $\mathcal{O}(q^2)$ (QROM) are unavoidable.

Tight security analysis for authenticated key exchange (AKE). Tightly secure AKE has recently emerged as an active research area, with several works proposing general constructions [BHJ+15, LLGW20, HJK+21, GJ18, JKRS21]. Pan et al. [PWZ23a] introduced the first lattice-based AKE protocol in the ROM that achieves (almost) tight security under the LWE assumption. Their construction builds on a KEM secure under the one-way checkable chosen-ciphertext attack (OW-ChCCA) notion. Note that our MUC-OW-ChCCA* notion is a weaker variant of OW-ChCCA that removes the adversary's access to the Reveal oracle. Subsequently, Pan et al. [PRZ24] proposed a lattice-based AKE protocol that achieves full forward secrecy (FS) in the ROM. They introduced one-wayness against key-verification attacks and weak forward secrecy (OW-VwFS) for AKEs, and proved that an OW-VwFS-secure AKE with key confirmation tightly implies full forward secrecy. Furthermore, they showed that OW-ChCCA-secure KEMs tightly imply OW-VwFS-secure AKEs, thereby circumventing the impossibility result of [GGJJ23]. All of the above tight AKE constructions use variants of the BR model [BR94], which differs from the multi-stage model [DJ21, SSW20] used in this work. In particular, the multi-stage model involves multiple sequential stages, including internal stages, with distinct security requirements and dependencies through the key schedule.

1.3 Technical Overview

Previous Techniques for PQ TLS 1.3 in [DFGS21, HV22, ZJZ24, CZC+26].

The existing security proofs [DFGS21, HV22, ZJZ24, CZC+26] for the multi-stage PQ TLS 1.3 handshake build upon the Multi-Stage model of Dowling et al. [DFGS21]. An overview of the protocol is given in Fig. 6. For the full 1-RTT handshake based on IND-1CCA KEMs, the proof first restricts the adversary to a single Test query via a hybrid argument, incurring a loss of $6t_s$ for the six stages and t_s sessions. If the tested session lacks a contributive partner at stage 1, the reduction guesses the peer's identity and reduces to the EUF-CMA security of the signature scheme, incurring an additional t_u loss, where t_u is the number of users. Otherwise, the reduction guesses the contributive partner among the t_s sessions and embeds the IND-1CCA challenge. The IND-1CCA decapsulation oracle enables consistent simulation when the tested session is a server session and its contributive partner receives a ciphertext different from the one sent by the server. In this case, the reduction queries the oracle to obtain the shared secret K and completes the simulation of the client session.

Overall, the resulting bounds are of the form $O(t_s^2\epsilon_{\mathcal{A}} + t_s t_u \epsilon_{\mathcal{B}}^{\mathrm{SIG}})$, where $\epsilon_{\mathcal{A}}$ and $\epsilon_{\mathcal{B}}^{\mathrm{SIG}}$ denote the advantages against the IND-1CCA security of the KEM and the EUF-CMA security of the signature scheme, respectively.

Previous Techniques for KEMTLS in [SSW20]. The security proof of KEMTLS similarly begins by restricting the adversary to a single Test query via a hybrid argument, incurring a loss of $6t_s$. If the tested session has a contributive partner at stage 1, the reduction guesses that partner among the t_s sessions and uses the IND-1CCA security of the ephemeral KEM to replace the shared ephemeral secret with a uniformly random value. If the tested session lacks a contributive partner, it must be a client session, and its intended peer must remain uncorrupted until the client accepts the session. In this case, the reduction guesses its intended peer V, incurring a t_u loss, and embeds an IND-CCA challenge to replace the static KEM key with a uniformly random value. It simulates the honest party V using its decapsulation oracle. In both cases, the stage keys and MAC keys are subsequently replaced with uniformly random values via the PRF security of HKDF. The adversary can then make the client accept maliciously only by forging a valid MAC, which establishes the explicit acceptance property of KEMTLS.

Tight Security Analysis for PQ TLS 1.3. We present a modular and tight security reduction for the multi-stage security of PQ TLS 1.3. Building on the multi-stage security framework of [DJ21], we prove that the multi-stage security of PQ TLS 1.3 tightly reduces to the (t_s, t_s)-MUC-IND-1CCA or (t_s, t_s)-MUC-IND-1RCCA security of the underlying KEM, together with the t_u-MU-EUF-CMA^c security of the SIG scheme, the t_s-MU-PRF security of HKDF, and the t_s-MU-dual-PRF security of $\mathsf{HKDF.Extract}$.

The high-level idea of the proof is as follows: We first rely on a t_u-MU-EUF-CMA^c-secure SIG to argue that all the fresh tested sessions must have a contributive partner at stage 1. Otherwise, if there exists a session with a tested stage that is fresh at the end of the experiment but without a contributive partner at stage 1, a t_u-MU-EUF-CMA^c adversary can embed the challenge public keys into the corresponding parties and use the oracles $\mathsf{O}^{\mathsf{SIGN}}(i, \cdot)$ and $\mathsf{O}^{\mathsf{Cor}}(i)$ to simulate the multi-stage security game for PQ TLS 1.3 and output a valid forgery corresponding to the Test session that lacks a contributive partner at stage 1 but nevertheless receives a valid signature.

Reduction to MUC-IND-1CCA. Since all fresh tested sessions have a contributive partner at stage 1, we rely on the (t_s, t_s)-MUC-IND-1CCA security of the underlying KEM to replace all shared secret keys K in the possible test sessions with uniformly random values, while ensuring consistency whenever the public key and KEM ciphertext coincide. Subsequently, we invoke these PRF properties to gradually replace all stage keys derived from the randomized KEM keys, including all possible tested stage keys, with uniformly random and independent values[6]. Leveraging the (t_s, t_s)-MUC-IND-1CCA game directly addresses the so-called *commitment problem*, wherein a tight reduction must be able to respond to both Test and Reveal queries for all sessions without relying on guessing. Specifically, when the reduction embeds a public key pk_i^* into the initial $\mathtt{ClientHello}$ message $\mathtt{CH} = (n_c, \mathsf{pk}_i^*)$ of session label_C, there may exist multiple server sessions label_S that receive the same public key pk_i^*. Let $\mathsf{label}_{S,j}$ denote the j-th corresponding server session. Consequently, the reduction embeds multiple ciphertext challenges $\mathsf{ct}_{i,j}$ into the corresponding $\mathtt{ServerHello}$ messages $\mathtt{SH}$ of the sessions $\mathsf{label}_{S,j}$. If the client in session label_C receives a challenge ciphertext, the reduction uses the corresponding challenge key consistently. Otherwise, the reduction invokes the one-time decapsulation oracle (available once per user) to compute the corresponding decapsulated secret K. This value enables the reduction to simulate the client session label_C, complete the handshake protocol, and answer the adversary's queries consistently. The reduction answers Reveal queries using the corresponding derived stage keys, thereby maintaining consistency with the Test oracle.

Reduction to MUC-IND-1RCCA. In PQ TLS 1.3, all stage key computations depend on the KEM ciphertext through the transcript hashes. This observation enables us to leverage the MUC-IND-1RCCA security of the underlying KEM to argue that all possible tested stage keys are uniformly random and independent across non-partnered sessions. The key point here is that we can invoke

[6] We actually show that all possible tested stage keys are uniformly random and independent, except for those that are partnered, but we omit the details here.

the (t_s, t_s)-MUC-IND-1RCCA security to replace all shared secret keys K in the test sessions with uniformly random values, while maintaining consistency for repeated ciphertexts and for replayable ciphertexts. Subsequently, we invoke these PRF properties to iteratively replace all possibly tested stage keys with uniformly random values. Since each stage key computation takes its corresponding transcript hash as input, after excluding hash collisions and applying the PRF replacements, the stage keys of a client that is not partnered with the server are independent of the server's stage keys. Specifically, if the client receives a different KEM ciphertext that decapsulates to the same shared key as the intended server's ciphertext, the stage keys for both parties are uniformly random and independent because the transcript hashes in the stage key derivations differ. The remaining cases are analogous to the MUC-IND-1CCA analysis. In particular, the reduction must also simulate the client session that receives a ciphertext outside the set of challenge ciphertexts for its public key. To do so, it queries its 1RCCA decapsulation oracle $\mathsf{O}^{\mathsf{Dec}'}(i, \mathsf{ct}')$.

Tight Security Analysis for KEMTLS in the ROM. We model the domain-separated HKDF functions and the HMAC function used for Finished messages as independent public random oracles. Note that a session in KEMTLS falls into one of two possible cases. Case 1: Let $\mathsf{Session}_1$ denote the set of sessions that have a contributive partner at stage 1. Case 2: Let $\mathsf{Session}_2$ denote the set of sessions without such a partner. In the second case, the fresh tested session label corresponds to a client session, and its intended peer V may be corrupted only after the client accepts the session. For Case 1, we invoke the (t_s, t_s)-MUC-IND-1CCA or (t_s, t_s)-MUC-IND-1RCCA security to replace the shared ephemeral KEM secrets with uniformly random values, as in the tight analysis of PQ TLS 1.3 described above. We then use the ROM to show that, except with negligible probability, the adversary never explicitly queries the corresponding key-derivation inputs. Hence, the fresh stage keys and MAC keys are indistinguishable from uniformly random values. The adversary can therefore cause the client to accept maliciously only by forging a valid MAC, which establishes explicit authentication for KEMTLS in this case.

Reduction to MUC-OW-ChCCA.* We analyze Case 2 using the MUC-OW-ChCCA* security of the static KEM, a weaker variant of OW-ChCCA [PWZ23a]. Intuitively, MUC-OW-ChCCA* captures one-wayness for KEMs in a multi-user, multi-challenge setting. The adversary is given multiple public keys and challenge ciphertexts, and its goal is to recover an encapsulated key for a challenge ciphertext under an uncorrupted user. The adversary can query a Check oracle to test candidate keys against ciphertexts, corrupt users to obtain their secret keys, and make decapsulation queries.

We first reformulate the game using the challenges and oracles of the MUC-OW-ChCCA* experiment, with $G_2 = \mathsf{HKDF.Extract}$ modeled as a random oracle. The reduction embeds the challenge public keys pk_i^* as the servers' long-term KEM public keys and obtains the ciphertext for every honest client encapsulation under such a public key from its Enc oracle. Server-side decapsulation queries on non-challenge ciphertexts and corruption queries are answered using its Dec and Corr oracles, respectively. For challenge ciphertexts, it must simulate the honest parties without the corresponding static KEM key K_S, unless the server has been corrupted. To overcome this, the reduction lazily simulates G_2 using a list $\mathcal{L}_{G_2}$. When it needs to compute $\mathsf{AHS} = G_2(\mathsf{dHS}, K_S)$ for an honest party using a challenge ciphertext, it uses the Check oracle to check whether candidate keys in entries of $\mathcal{L}_{G_2}$ with the same dHS match the corresponding ciphertext. If a matching entry exists, the reduction uses its stored output as AHS. Otherwise, it samples a random AHS for each new input and ensures that repeated inputs and subsequent random-oracle queries remain consistent via the Check oracle. Thus, this reformulation is perfect in the ROM, except with negligible probability.

Let QUERY denote the event that the adversary's explicit G_2 query history contains a registered challenge key under a currently uncorrupted server. To bound this event, we construct an adversary $\mathcal{B}_3$ against MUC-OW-ChCCA* security as follows. $\mathcal{B}_3$ simulates the game as above and tests candidate keys using $\mathsf{Check}(i, \mathsf{ct}, K)$, including earlier queries when a new challenge is registered. At the first successful check for an uncorrupted user, it outputs (i, ct, K) and halts. Excluding static-KEM correctness failures, this yields a valid MUC-OW-ChCCA* solution.

Now consider a client session in Case 2 whose intended server remains uncorrupted before the client accepts at stage 6. When QUERY does not occur, the $G_2(\mathsf{dHS}, K_S)$ output in this client session remains a private random-oracle value hidden from the adversary. By the standard ROM analysis, its fresh stage keys and MAC keys are also indistinguishable from uniformly random values to the adversary before this client session accepts at stage 6. Since no honest server computes the required Finished tag, such a client session accepts at stage 6 before its intended server is corrupted only with negligible probability. This establishes both key secrecy and explicit authentication for Case 2.

Concrete MU-(dual)-PRF bounds for HMAC and HKDF. The main challenge is to handle quantum queries to the secret-key-dependent random-oracle inputs used inside HMAC and its swapped variant $\mathsf{HMAC}^{\mathsf{swap}}(k, x) := \mathsf{HMAC}(x, k)$. We first carefully rule out key and relevant hash collisions and ensure domain separation, so that each fresh answered PRF query uses an outer-hash input not used by any previous PRF query in either its inner- or outer-hash evaluations. We then apply the OW2H lemma [AHU19] to decouple the adversary's access to the random oracle from the PRF oracle. After these steps, the real PRF responses can be simulated using an independent random function. The bounds for HKDF.Extract and HKDF.Expand follow by standard reductions to HMAC.

We only outline the proof at a high level here. Several technical subtleties are handled carefully in the full proof.

2 Preliminaries

This section introduces the notation and cryptographic primitives used in our security analyses.

2.1 Notation and Multi-User KEM Security

The security parameter is denoted by λ. The set $\{0, \ldots, q-1\}$ is denoted by $[q]$. QPT refers to quantum polynomial-time algorithms. The symbols $\mathcal{K}$, $\mathcal{M}$, and $\mathcal{C}$ denote the key space, message space, and ciphertext space, respectively. For a finite set $\mathcal{X}$, the notation $x \leftarrow\!\$\ \mathcal{X}$ indicates uniform sampling from $\mathcal{X}$; sampling from a distribution $\mathcal{D}$ is written as $x \leftarrow\!\$\ \mathcal{D}$. The notation $\Pr[P : G]$ denotes the probability that predicate P holds when its variables are instantiated by program G. For a randomized algorithm A, the notation $y \leftarrow\!\$\ A(x)$ is used; for deterministic A, the notation $y \leftarrow A(x)$ is used. The expression $x =?y$ evaluates to 1 if $x = y$, and 0 otherwise. The predicate 1_{C} is 1 when the condition C is satisfied, and 0 otherwise. The size of a set X is denoted by $|X|$. $A^{|H\rangle}$ indicates that algorithm A is given quantum access to the oracle H.

Definition 2.1 (Key Encapsulation Mechanism). *A key encapsulation mechanism (KEM) over $\mathcal{K}$ is a tuple of algorithms* $(\mathsf{Gen}, \mathsf{Encaps}, \mathsf{Decaps})$. *(1)* $(\mathsf{pk}, \mathsf{sk}) \leftarrow\!\$\ \mathsf{Gen}(1^\lambda)$*: the key-generation algorithm takes as input the security parameter λ and outputs a key pair* $(\mathsf{pk}, \mathsf{sk})$. *Usually, we will omit the input of* Gen *for brevity. (2)* $(\mathsf{ct}, K) \leftarrow\!\$\ \mathsf{Encaps}(\mathsf{pk})$*: the encapsulation algorithm takes as input a public key* pk *and outputs a pair* (ct, K)*, where $K \in \mathcal{K}$ and $\mathsf{ct} \in \mathcal{C}$. (3) $K' \leftarrow \mathsf{Decaps}(\mathsf{sk}, \mathsf{ct})$: the decapsulation algorithm takes as input a secret key* sk *and a ciphertext* ct*, and outputs either a key $K' \in \mathcal{K}$ or the rejection symbol $\perp$ (if explicit rejection is allowed).*

Correctness. A KEM scheme is said to be δ-correct if $\Pr\left[\mathsf{Decaps}(\mathsf{sk}, \mathsf{ct}) \neq K \;\middle|\; (\mathsf{pk}, \mathsf{sk}) \leftarrow\!\$\ \mathsf{Gen};\ (\mathsf{ct}, K) \leftarrow\!\$\ \mathsf{Encaps}(\mathsf{pk})\right] \leq \delta$, where the probability is taken over the randomness of Gen and Encaps.

Min-Entropy of Public Keys. A KEM is said to have public-key min-entropy at least γ_{pk} if the public key pk generated by $(\mathsf{pk}, \mathsf{sk}) \leftarrow\!\$\ \mathsf{Gen}()$ satisfies $H_\infty(\mathsf{pk}) = -\log_2\big(\max_p \Pr[\mathsf{pk} = p]\big) \geq \gamma_{\mathsf{pk}}$, where the probability is taken over the randomness of Gen.

Min-Entropy of Encapsulated Keys. A KEM is said to have encapsulated-key min-entropy at least γ if, for every honestly generated public key pk, the key K generated by $(\mathsf{ct}, K) \leftarrow\!\$\ \mathsf{Encaps}(\mathsf{pk})$ satisfies $H_\infty(K) = -\log_2\big(\max_k \Pr[K = k]\big) \geq \gamma$, where the probability is taken over the randomness of Encaps under a fixed pk.

Game (N, S)-MUC-IND-1(R)CCA$_{\mathsf{KEM}}(\mathcal{A})$

$b \leftarrow\!\$\ \{0,1\}$
for $i \in [N]$:
  $\mathcal{Q}_i \leftarrow 0$
  $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow\!\$\ \mathsf{Gen}()$
  $\mathbf{pk}[i] \leftarrow \mathsf{pk}_i$
  for $j \in [S]$:
    $(\mathsf{ct}_{i,j}, K^0_{i,j}) \leftarrow\!\$\ \mathsf{Encaps}(\mathsf{pk}_i)$
    $K^1_{i,j} \leftarrow\!\$\ \mathcal{K}$, $\mathbf{ct}[i,j] \leftarrow \mathsf{ct}_{i,j}$
    $\mathbf{K}_0[i,j] \leftarrow K^0_{i,j}$, $\mathbf{K}_1[i,j] \leftarrow K^1_{i,j}$
$b' \leftarrow \mathcal{A}^{\mathsf{Dec}}(\mathbf{pk}, \mathbf{ct}, \mathbf{K}_b)$ // for MUC-IND-1CCA
$b' \leftarrow \mathcal{A}^{\mathsf{Dec}'}(\mathbf{pk}, \mathbf{ct}, \mathbf{K}_b)$ // for MUC-IND-1RCCA
return $\mathbf{1}_{[b'=b]}$

Oracle $\mathsf{Dec}(i, \mathsf{ct})$

$\mathcal{Q}_i \leftarrow \mathcal{Q}_i + 1$
if $\mathcal{Q}_i \geq 2$: **return** $\perp$
for $j \in [S]$:
  if $\mathsf{ct} = \mathbf{ct}[i,j]$: **return** $\perp$
$K \leftarrow \mathsf{Decaps}(\mathsf{sk}_i, \mathsf{ct})$
return K

Oracle $\mathsf{Dec}'(i, \mathsf{ct})$

$\mathcal{Q}_i \leftarrow \mathcal{Q}_i + 1$
if $\mathcal{Q}_i \geq 2$: **return** $\perp$
for $j \in [S]$:
  if $\mathsf{ct} = \mathbf{ct}[i,j]$: **return** $\perp$
$K \leftarrow \mathsf{Decaps}(\mathsf{sk}_i, \mathsf{ct})$
if $\exists j \in [S]$ s.t. $K = K^0_{i,j}$:
  $j^* \leftarrow \min\{j \in [S] : K = K^0_{i,j}\}$
  return Test-j^*
if $\exists j \in [S]$ s.t. $K = K^1_{i,j}$:
  $j^* \leftarrow \min\{j \in [S] : K = K^1_{i,j}\}$
  return Test-j^*
return K

Figure 2: The (N, S)-MUC-IND-1CCA and (N, S)-MUC-IND-1RCCA games for a KEM scheme $\mathsf{KEM} = (\mathsf{Gen}, \mathsf{Encaps}, \mathsf{Decaps})$.

We first recall the MUC-IND-1RCCA and MUC-IND-1CCA security notions for KEMs considered in this paper, as shown in Fig. 2. These are multi-user, multi-challenge variants of the IND-1RCCA and IND-1CCA notions, respectively. In particular, in MUC-IND-1RCCA, for a non-challenge ciphertext, the Dec' oracle returns the smallest index matching a real challenge key, if one exists; otherwise, it returns the smallest index matching a random challenge key, if one exists. If neither match exists, it returns the decapsulation result.

Definition 2.2 (MUC-IND-1CCA Secure KEM). *Let* $\mathsf{KEM} = (\mathsf{Gen}, \mathsf{Encaps}, \mathsf{Decaps})$ *be a key encapsulation mechanism, and let* $N, S \in \mathbb{N}$. *Consider the game* (N, S)-MUC-IND-1CCA *described in Fig. 2. The KEM is said to be* (N, S)-MUC-IND-1CCA*-secure if, for every polynomial-time adversary* $\mathcal{A}$*, the following advantage is negligible in the security parameter* λ*:*

$$\mathsf{Adv}^{\mathsf{MUC\text{-}IND\text{-}1CCA}}_{\mathsf{KEM},N,S}(\mathcal{A}) := \left|\Pr\left[(N,S)\text{-MUC-IND-1CCA}_{\mathsf{KEM}}(\mathcal{A}) \Rightarrow 1\right] - \tfrac{1}{2}\right|.$$

Definition 2.3 (MUC-IND-1RCCA Secure KEM). *Let* $\mathsf{KEM} = (\mathsf{Gen}, \mathsf{Encaps}, \mathsf{Decaps})$ *be a key encapsulation mechanism, and let* $N, S \in \mathbb{N}$. *Consider the game* (N, S)-MUC-IND-1RCCA *described in Fig. 2. The KEM is said to be* (N, S)-MUC-IND-1RCCA*-secure if, for every polynomial-time adversary* $\mathcal{A}$*, the following advantage is negligible in the security parameter* λ*:*

$$\mathsf{Adv}^{\mathsf{MUC\text{-}IND\text{-}1RCCA}}_{\mathsf{KEM},N,S}(\mathcal{A}) := \left|\Pr\left[(N,S)\text{-MUC-IND-1RCCA}_{\mathsf{KEM}}(\mathcal{A}) \Rightarrow 1\right] - \tfrac{1}{2}\right|.$$

We then define the MUC-OW-ChCCA* notion for KEMs, which is a weaker variant of OW-ChCCA from [PWZ23a].

Game (N,S)-MUC-OW-ChCCA$^*_{\mathsf{KEM}}(\mathcal{A})$

$\mathcal{L}_{\mathsf{Enc}}, \mathcal{L}_{\mathsf{Corr}} \leftarrow \emptyset$
for $i \in [N]$:
  $\mathcal{Q}_i \leftarrow 0$
  $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow\!\!\$\ \mathsf{Gen}()$
  $\mathbf{pk}[i] \leftarrow \mathsf{pk}_i$
$O := (\mathsf{Enc}, \mathsf{Dec}, \mathsf{Corr}, \mathsf{Check})$
$(i^*, \mathsf{ct}^*, K^*) \leftarrow \mathcal{A}^O(\mathbf{pk})$
if $(i^*, \mathsf{ct}^*, K^*) \notin \mathcal{L}_{\mathsf{Enc}}$: **return** 0
if $i^* \in \mathcal{L}_{\mathsf{Corr}}$: **return** 0
return 1

Oracle $\mathsf{Corr}(i)$

$\mathcal{L}_{\mathsf{Corr}} \leftarrow \mathcal{L}_{\mathsf{Corr}} \cup \{i\}$
return sk_i

Oracle $\mathsf{Enc}(i)$

$\mathcal{Q}_i \leftarrow \mathcal{Q}_i + 1$
if $\mathcal{Q}_i \geq S + 1$: **return** $\perp$
$(\mathsf{ct}, K) \leftarrow\!\!\$\ \mathsf{Encaps}(\mathsf{pk}_i)$
$\mathcal{L}_{\mathsf{Enc}} \leftarrow \mathcal{L}_{\mathsf{Enc}} \cup \{(i, \mathsf{ct}, K)\}$
return ct

Oracle $\mathsf{Dec}(i, \mathsf{ct}')$

if $\exists K'$ such that $(i, \mathsf{ct}', K') \in \mathcal{L}_{\mathsf{Enc}}$:
  return $\perp$
$K' \leftarrow \mathsf{Decaps}(\mathsf{sk}_i, \mathsf{ct}')$
return K'

Oracle $\mathsf{Check}(i, \mathsf{ct}, K)$

if $\mathsf{Decaps}(\mathsf{sk}_i, \mathsf{ct}) = K$: **return** 1
return 0

Figure 3: The (N,S)-MUC-OW-ChCCA* security game for a key encapsulation mechanism $\mathsf{KEM} = (\mathsf{Gen}, \mathsf{Encaps}, \mathsf{Decaps})$. Here, N denotes the number of users and S bounds the number of encapsulation queries per user.

Definition 2.4 ((N,S)-MUC-OW-ChCCA* Secure KEM). *Let* KEM *be a key encapsulation mechanism and $N, S \in \mathbb{N}$. We say that* KEM *is (N,S)-MUC-OW-ChCCA*-secure if, for every polynomial-time adversary $\mathcal{A}$, the following advantage in the game of Fig. 3 is negligible in λ:*

$$\mathsf{Adv}^{\text{MUC-OW-ChCCA}^*}_{\mathsf{KEM},N,S}(\mathcal{A}) := \Pr[(N,S)\text{-MUC-OW-ChCCA}^*_{\mathsf{KEM}}(\mathcal{A}) \Rightarrow 1]\,.$$

2.2 Signatures and Pseudorandom Functions

We recall the signature and PRF security notions used in this paper.

Definition 2.5 (Signature). *A signature scheme over message space $\mathcal{M}$ is a tuple of three algorithms:* Gen, Sign, *and* Vrfy. *(1) $(\mathsf{pk}, \mathsf{sk}) \leftarrow\!\!\$\ \mathsf{Gen}(1^\lambda)$: The key generation algorithm takes as input the security parameter 1^λ and outputs a verification key* pk *and a signing key* sk. *Usually, we omit the input of* Gen *for brevity. (2) $\sigma \leftarrow\!\!\$\ \mathsf{Sign}(\mathsf{sk}, m)$: The signing algorithm takes as input a signing key* sk *and a message $m \in \mathcal{M}$ and outputs a signature σ. (3) $b \leftarrow \mathsf{Vrfy}(\mathsf{pk}, m, \sigma)$: The verification algorithm takes as input a verification key* pk*, a message $m \in \mathcal{M}$, and a signature σ and outputs a bit $b \in \{0,1\}$ indicating whether the signature is valid.*

Correctness. A signature scheme is correct if, for all messages $m \in \mathcal{M}$, $\Pr\left[\mathsf{Vrfy}(\mathsf{pk}, m, \sigma) = 1 : (\mathsf{pk}, \mathsf{sk}) \leftarrow\!\!\$\ \mathsf{Gen}(1^\lambda), \sigma \leftarrow\!\!\$\ \mathsf{Sign}(\mathsf{sk}, m)\right] \geq 1 - \mathsf{negl}(\lambda)$, where the probability is taken over the randomness of the key-generation and signing algorithms.

Definition 2.6 (MU-EUF-CMA^c Security for Signature Schemes). *Consider a signature scheme $\mathsf{SIG} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$, and let $N \in \mathbb{N}$ be a natural number. Consider the game N-MU-EUF-CMA^c defined in Fig. 4. The scheme is said to be N-MU-EUF-CMA^c-secure if, for every polynomial-time adversary $\mathcal{A}$, the following advantage is negligible in λ:*

$$\mathsf{Adv}^{\text{MU-EUF-CMA}^c}_{\mathsf{SIG},N}(\mathcal{A}) := \Pr[N\text{-MU-EUF-CMA}^c_{\mathsf{SIG}}(\mathcal{A}) \Rightarrow 1]\,.$$

Game N-MU-EUF-CMA$^{\mathsf{c}}_{\mathsf{SIG}}(\mathcal{A})$	$\mathsf{O}^{\mathsf{Cor}}(i)$
1 : **for** $i \in [N]$: $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow\!\!\$\ \mathsf{Gen}()$	1 : $\mathcal{L}_{\mathsf{Cor}} := \mathcal{L}_{\mathsf{Cor}} \cup \{i\}$
2 : Initialize $\mathcal{L}_{\mathsf{SIGN}} := \emptyset$ and $\mathcal{L}_{\mathsf{Cor}} := \emptyset$	2 : **return** sk_i
3 : $\mathsf{O} := (\mathsf{O}^{\mathsf{SIGN}}, \mathsf{O}^{\mathsf{Cor}})$	$\mathsf{O}^{\mathsf{SIGN}}(i, m)$
4 : $(i^*, m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{O}}((\mathsf{pk}_i)_{i\in[N]})$	1 : $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}_i, m)$
5 : **if** $i^* \in \mathcal{L}_{\mathsf{Cor}}$: **return** 0	2 : $\mathcal{L}_{\mathsf{SIGN}} := \mathcal{L}_{\mathsf{SIGN}} \cup \{(i, m)\}$
6 : **if** $(i^*, m^*) \in \mathcal{L}_{\mathsf{SIGN}}$: **return** 0	3 : **return** σ
7 : **return** $\mathsf{Vrfy}(\mathsf{pk}_{i^*}, m^*, \sigma^*)$	

Figure 4: The N-MU-EUF-CMA^{c} game for $\mathsf{SIG} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$.

Definition 2.7 (MU-PRF Security for Pseudorandom Functions). *Let* PRF *be a deterministic keyed function* $\mathsf{PRF} : \mathcal{K}_{\mathsf{PRF}} \times D \to R$*, where* $\mathcal{K}_{\mathsf{PRF}}$ *is a finite key space,* D *is the domain, and* R *is a finite range. Let* $N \in \mathbb{N}$*. Consider the game* N-MU-PRF *defined in Fig. 5. We say that* PRF *is* N-MU-PRF*-secure if, for every polynomial-time adversary* $\mathcal{A}$*, the following advantage is negligible in* λ*:*

$$\mathsf{Adv}^{\mathsf{MU\text{-}PRF}}_{\mathsf{PRF},N}(\mathcal{A}) := \left|\Pr[N\text{-MU-PRF}_{\mathsf{PRF}}(\mathcal{A}) \Rightarrow 1] - \tfrac{1}{2}\right|.$$

Moreover, we define the N-MU-dual-PRF *security of* PRF *as the* N-MU-PRF *security of the swapped function* $\mathsf{PRF}^{\mathsf{swap}}(k, x) := \mathsf{PRF}(x, k)$*. Note that the keys of the swapped function are sampled independently and uniformly from a fixed finite key space* $\mathcal{K}_{\mathsf{swap}} \subseteq D$*. Formally, we define the corresponding advantage as* $\mathsf{Adv}^{\mathsf{MU\text{-}dual\text{-}PRF}}_{\mathsf{PRF},N}(\mathcal{A}) := \mathsf{Adv}^{\mathsf{MU\text{-}PRF}}_{\mathsf{PRF}^{\mathsf{swap}},N}(\mathcal{A})$*.*

Game N-MU-PRF$_{\mathsf{PRF}}(\mathcal{A})$	$\mathsf{O}_b(i, x)$
1 : $b \leftarrow\!\!\$\ \{0, 1\}$	1 : **if** $b = 0$: **return** $\mathsf{PRF}(k_i, x)$
2 : **for** $i \in [N]$:	2 : **else**: **return** $f_i(x)$
3 : $\quad k_i \leftarrow\!\!\$\ \mathcal{K}_{\mathsf{PRF}}$	
4 : $\quad f_i \leftarrow\!\!\$\ \{f \mid f : D \to R\}$	
5 : $b' \leftarrow \mathcal{A}^{\mathsf{O}_b}()$	
6 : **return** $1_{b'=b}$	

Figure 5: The game N-MU-PRF for a pseudorandom function PRF.

2.3 HMAC and HKDF

We next recall the HMAC and HKDF constructions used in PQ TLS 1.3 and KEMTLS. HMAC [BCK96a, KBC97] is a prominent message authentication code (MAC) that can be modeled as a pseudorandom function (PRF). It is constructed from a cryptographic hash function as follows.

Definition 2.8 (HMAC). *Let* $H\colon \{0,1\}^* \to \{0,1\}^\mu$ *be a cryptographic hash function with output length* μ *and block size* B *bytes, and let* κ *be the key length. The* HMAC *message authentication code scheme is defined as follows:*

- MAC.Gen: *Choose a key* $k \leftarrow\!\!\$\ \{0,1\}^\kappa$ *and return* k*.*
- MAC.Tag(k, m): *Return* $t \leftarrow H\big((k \oplus \mathit{opad}) \parallel H((k \oplus \mathit{ipad}) \parallel m)\big)$*.*

– $\mathsf{MAC.Vrfy}(k, m, t)$: *Return* 1 *if and only if* $t = \mathsf{MAC.Tag}(k, m)$.

Here, the constants `opad` *and* `ipad` *are defined as the byte values* `0x5c` *and* `0x36`*, respectively, each repeated* B *times, according to* RFC 2104 *[KBC97]. The key* k *is padded with zeros to match the block size* B*. If the byte length of* k *exceeds* B*, it is first hashed using* H*, and the output is then padded with zeros to form a* B*-byte string.*

The key derivation function (KDF) HKDF [Kra10], standardized in RFC 5869 [KE10], forms the basis of key derivation in TLS 1.3 [HV22,DFGS21]. HKDF follows the extract-then-expand paradigm using HMAC and consists of two subroutines: HKDF.Extract and HKDF.Expand. The HKDF.Extract function acts as a randomness extractor, while HKDF.Expand serves as a variable-length PRF.

Definition 2.9 (HKDF). *The key derivation function* HKDF*, built from HMAC, consists of two subroutines:*

– $\mathsf{HKDF.Extract}(\mathsf{xts}, \mathsf{skm})$*: Given a salt* $\mathsf{xts} \in \{0,1\}^*$ *and source key material* $\mathsf{skm} \in \{0,1\}^*$*, output the pseudorandom key* $\mathsf{prk} \leftarrow \mathsf{HMAC}(\mathsf{xts}, \mathsf{skm})$.
– $\mathsf{HKDF.Expand}(\mathsf{prk}, \mathsf{ctx}, L)$*: Given a pseudorandom key* $\mathsf{prk} \in \{0,1\}^\mu$*, optional context information* $\mathsf{ctx} \in \{0,1\}^*$*, and a desired output length* L*, compute* $\omega \leftarrow \lceil L/\mu \rceil$ *and define:*
  - $\mathsf{K}(1) := \mathsf{HMAC}(\mathsf{prk}, \mathsf{ctx} \parallel 0)$*;*
  - $\mathsf{K}(i{+}1) := \mathsf{HMAC}(\mathsf{prk}, \mathsf{K}(i) \parallel \mathsf{ctx} \parallel i)$*, for* $1 \leq i < \omega$.

  Output the keying material $\mathsf{km} \leftarrow \mathsf{K}(1) \parallel \cdots \parallel \mathsf{K}(\omega)$*, truncated to the first* L *bits.*

The HKDF *function is defined as the sequential composition of* HKDF.Extract *and* HKDF.Expand*. More precisely, given inputs* xts, skm, ctx*, and* L*, it computes* $\mathsf{prk} \leftarrow \mathsf{HKDF.Extract}(\mathsf{xts}, \mathsf{skm})$ *and outputs* $\mathsf{km} \leftarrow \mathsf{HKDF.Expand}(\mathsf{prk}, \mathsf{ctx}, L)$*. For simplicity, we omit the third parameter* L *of* HKDF.Expand *in both protocols whenever* $L = \mu$.

3 PQ TLS 1.3 and KEMTLS: Protocols and Security Models

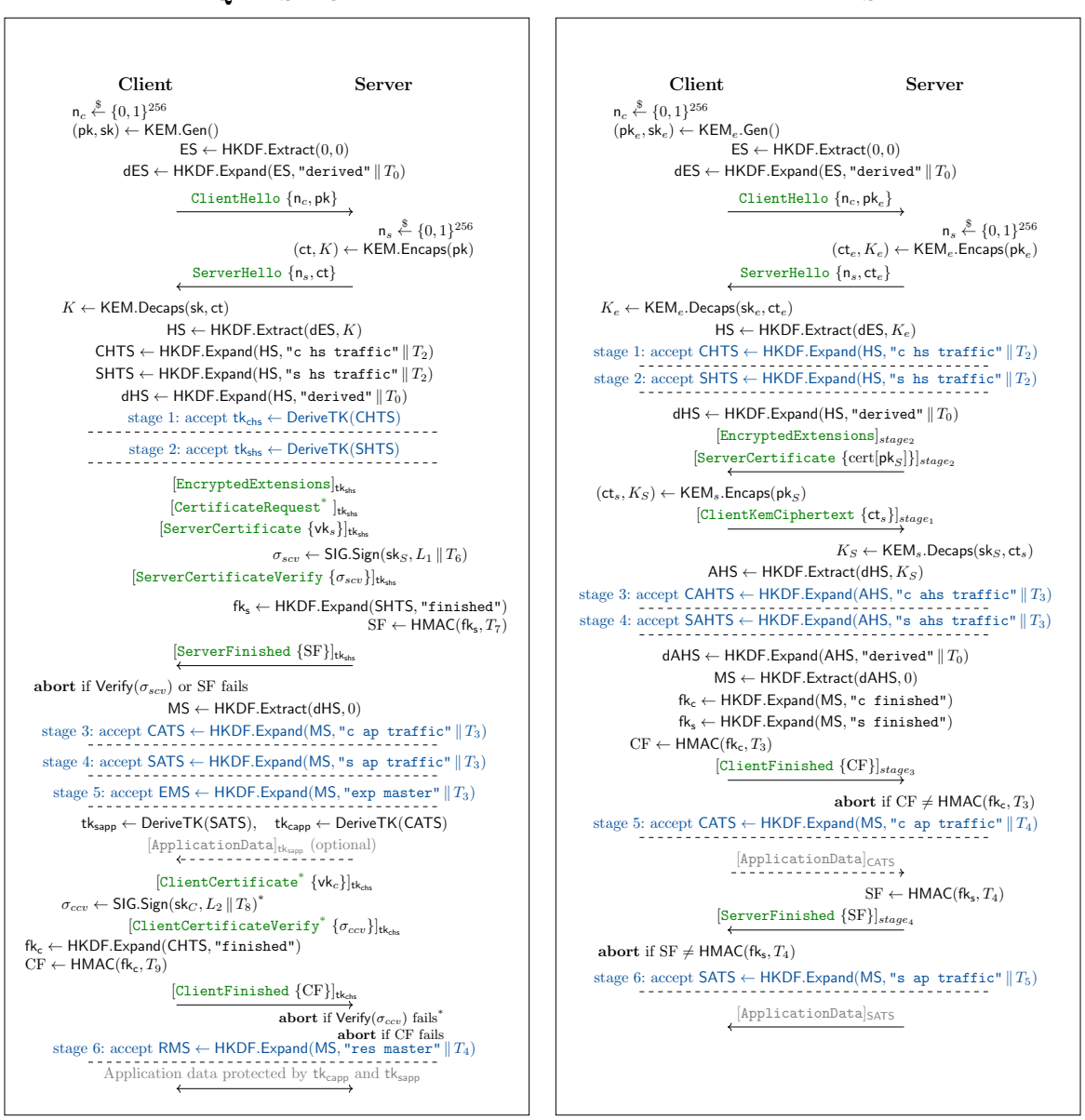
This section recalls the PQ TLS 1.3 1-RTT [DFGS21] and KEMTLS [SSW20] handshakes, together with their multi-stage security models. Building on the models in [DFGS21, SSW20], we provide a refined and unified security model for both protocols, with a more explicit treatment of internal keys. In particular, the model uses a common syntax and adversarial interface, with freshness and authentication requirements specified separately for each protocol.

Match security for PQ TLS 1.3 and KEMTLS captures correctness properties of session partnering, including agreement on stage keys between partnered sessions. As we focus on tight reductions for Multi-Stage (key-indistinguishability) security, we omit the formal Match-security definitions and refer the reader to [DFGS21, SSW20].

As the primary and most widely deployed mode of TLS 1.3, the 1-RTT[7] handshake is the focus of this paper. Throughout this paper, PQ TLS 1.3 denotes the natural post-quantum instantiation of the TLS 1.3 full 1-RTT handshake, in which the ephemeral Diffie–Hellman key exchange is replaced by a post-quantum KEM and post-quantum signatures are used for authentication. KEMTLS further replaces signature-based handshake authentication with KEM-based authentication. The certified long-term public key used for authentication is a signature verification key in PQ TLS 1.3 and a KEM public key in KEMTLS.

Figure 6 illustrates the PQ TLS 1.3 1-RTT handshake [DFGS21] and the KEMTLS handshake [SSW20]. We refer the reader to [DFGS21, SSW20] for a more comprehensive treatment of both protocols. Here, we omit TCP connection establishment and encoding details from the protocol descriptions.

[7] In a 1-RTT handshake, the client can send its first application message after one round-trip time.



Legend

$MSG\{Y\}$	Message `MSG` sent in the clear, carrying Y.
$[MSG]_{key}$	Message AEAD-encrypted under the corresponding traffic key.
$(\cdot)^*$	Present only when client certificate authentication is used (PQ TLS 1.3).
L_1, L_2	`"TLS 1.3, server CertificateVerify"` and `"TLS 1.3, client CertificateVerify"`.
T_i $(i \geq 1)$	Hash of all handshake messages up to the corresponding protocol point, in plaintext, excluding the message being generated or verified; indices are local to each panel.
$T_0 = H(\varepsilon)$	Hash of the empty transcript.

Figure 6: Comparison of PQ TLS 1.3 and KEMTLS handshakes.

3.1 The PQ TLS 1.3 1-RTT Handshake

Ephemeral key exchange. In PQ TLS 1.3, the client first generates an ephemeral KEM key pair $(\mathsf{pk}, \mathsf{sk}) \leftarrow\$ \mathsf{KEM.Gen}()$ and sends the public key pk together with a fresh nonce n_c in `ClientHello`. Upon receiving `ClientHello`, the server samples a fresh nonce n_s, computes $(\mathsf{ct}, K) \leftarrow\$ \mathsf{KEM.Encaps}(\mathsf{pk})$, and returns n_s and ct in `ServerHello`. The client decapsulates ct using sk to obtain K. For simplicity, we use `CH` and `SH`[8] to denote the `ClientHello` and `ServerHello` messages, including their respective key-share extensions. The resulting unauthenticated ephemeral shared secret K is used in the TLS 1.3 key schedule to derive the handshake secret HS, the client and server handshake traffic secrets CHTS and SHTS, and the derived handshake secret dHS. The handshake traffic keys used to encrypt subsequent handshake messages are derived as $\mathsf{tk}_{\mathsf{chs}} \leftarrow \mathsf{DeriveTK}(\mathsf{CHTS})$ and $\mathsf{tk}_{\mathsf{shs}} \leftarrow \mathsf{DeriveTK}(\mathsf{SHTS})$, completing stages 1 and 2, respectively. Here, $\mathsf{DeriveTK}$ derives a traffic key and an initialization vector (IV) as $\mathsf{DeriveTK}(\mathsf{ss}) := \big(\mathsf{HKDF.Expand}(\mathsf{ss}, \texttt{"key"}, \ell_k), \mathsf{HKDF.Expand}(\mathsf{ss}, \texttt{"iv"}, \ell_{\mathsf{iv}})\big)$, where ℓ_k and ℓ_{iv} denote the bit lengths of the traffic key and IV, respectively. In the security model, the server accepts stage 1 after constructing `SH` but before sending it, so that execution can pause at key acceptance as specified in Section 3.4.

Server authentication. The server then sends `EncryptedExtensions`, an optional `CertificateRequest`, its `Certificate`, and `CertificateVerify` containing its signature on the preceding transcript hash. The server then sends `ServerFinished`, which authenticates the transcript using the server Finished key fk_{s} derived from SHTS. The client verifies the server certificate, the signature, and the `ServerFinished` message, and aborts if any check fails. Both parties derive the master secret MS from dHS and use it to derive the client application traffic secret CATS, the server application traffic secret SATS, and the exporter master secret EMS, corresponding to stages 3, 4, and 5, respectively. The application traffic keys are derived as $\mathsf{tk}_{\mathsf{capp}} \leftarrow \mathsf{DeriveTK}(\mathsf{CATS})$ and $\mathsf{tk}_{\mathsf{sapp}} \leftarrow \mathsf{DeriveTK}(\mathsf{SATS})$. The server may send protected application data immediately after its `ServerFinished` message.

Client Finished and optional client authentication. The client then sends its `Certificate` and `CertificateVerify` messages (only if client authentication is requested), followed by `ClientFinished`, which authenticates the transcript using the client Finished key fk_{c} derived from CHTS. The server verifies these messages and aborts if any check fails. Finally, both parties derive the resumption master secret RMS, completing stage 6 of the handshake.

3.2 The KEMTLS Handshake

Following [SSW20], we focus on KEMTLS with server-only authentication, reflecting common practice in TLS deployments.

Ephemeral key exchange. As shown in Fig. 6, the initial key-exchange phase of KEMTLS is similar to that of PQ TLS 1.3, with stages 1 and 2 corresponding to CHTS and SHTS, respectively. The stage 1 and 2 keys are used to encrypt subsequent handshake messages before stage 3.

KEM-based server authentication. The main difference lies in the authentication phase. Instead of authenticating the server using a handshake signature, KEMTLS uses a certified static KEM public key pk_S carried in the server certificate. After verifying the certificate, the client encapsulates under pk_S to obtain K_S and sends the resulting ciphertext ct_s in the `ClientKemCiphertext` message. The server decapsulates ct_s using its long-term secret key sk_S to obtain K_S. This implicitly authenticated shared secret is combined with dHS to derive the authenticated handshake secret $\mathsf{AHS} = \mathsf{HKDF.Extract}(\mathsf{dHS}, K_S)$. From AHS, both parties derive the client and server authenticated handshake traffic secrets CAHTS and SAHTS, corresponding to stages 3 and 4. They then derive the

[8] These correspond to `CH` + `CKS` and `SH` + `SKS` in [DFGS21], respectively.

master secret MS and the corresponding Finished keys $\mathsf{fk_c}$ and $\mathsf{fk_s}$. The stage 3 and 4 keys are used to encrypt subsequent handshake messages.

Client Finished and handshake completion. The client then sends `ClientFinished`, which authenticates the transcript using the client Finished key $\mathsf{fk_c}$, and accepts the client application traffic secret CATS. The server accepts CATS after verifying `ClientFinished`, completing stage 5 for both parties. The server then sends `ServerFinished` and accepts the server application traffic secret SATS. The client accepts SATS after verifying `ServerFinished`, completing stage 6 for both parties and providing explicit server authentication to the client.

3.3 Model Syntax

We now formalize the multi-stage security model for both protocols, beginning with its syntax. Section 3.4 specifies the adversary's interaction, while Sections 3.5 and 3.6 provide the protocol-specific parameters and security definitions.

We denote the set of identities (or users) by $\mathcal{U}$ and represent each session by a label $\mathsf{label} = (U, V, n) \in \mathsf{LABELS} = \mathcal{U} \times (\mathcal{U} \cup \{*\}) \times \mathcal{N}$. This label identifies the n-th session associated with user U and intended partner V. Each identity U requiring authentication is equipped with a certified long-term public key pk_U and a corresponding secret key sk_U. These long-term key pairs are signature key pairs in PQ TLS 1.3 and KEM key pairs in KEMTLS. Throughout this paper, KEMTLS uses server-only authentication. For a protocol with M stages, each session maintains a state that records the following information:

- $\mathsf{label} \in \mathsf{LABELS}$: the unique (administrative) session label.
- $\mathsf{id} \in \mathcal{U}$: the identity of the session owner.
- $\mathsf{pid} \in \mathcal{U} \cup \{*\}$: the identity of the intended partner, where the symbol $*$ denotes an identity that is currently unknown and may be set exactly once later in the protocol.[9]
- $\mathsf{role} \in \{\texttt{initiator}, \texttt{responder}\}$: the role of the session (e.g., client/server for PQ TLS).
- $\mathsf{auth}_{\mathsf{TLS}} \in \mathsf{AUTH}_{\mathsf{TLS}}$ (PQ TLS 1.3 only[10]): the intended authentication type (unilateral or mutual authentication), chosen at session creation. Different sessions may adopt different configurations.
- $\mathsf{st}_{\mathsf{exec}} \in \{\bot, \texttt{running}, \texttt{accepted}, \texttt{rejected}\}^M$: records the execution status of each stage and is initialized to $(\texttt{running}, \bot^{M-1})$.
- $\mathsf{stage} \in \{0, \ldots, M\}$: the current stage of the session, with 0 denoting the initial state before protocol execution.
- $\mathsf{sid} \in (\{0,1\}^* \cup \{\bot\})^M$: indicates the session identifier in each stage.
- $\mathsf{cid} \in (\{0,1\}^* \cup \{\bot\})^M$: indicates the contributive identifier in each stage.
- $\mathsf{key} \in (\{0,1\}^* \cup \{\bot\})^M$: indicates the established session key in each stage, set once upon acceptance in each stage.
- $\mathsf{st}_{\mathsf{key}} \in \{\mathsf{fresh}, \mathsf{revealed}\}^M$: indicates the state of a session key in each stage.
- $\mathsf{tested} \in \{\mathsf{true}, \mathsf{false}\}^M$: tested_i indicates whether key_i has been tested.

We write $\mathsf{label.x}$ to denote the value of field x in session label. Two distinct sessions label and label' are defined to be partnered in stage i if they have opposite roles and hold the same session identifier in that stage, i.e., $\mathsf{label.role} \neq \mathsf{label}'\mathsf{.role}$ and $\mathsf{label.sid}_i = \mathsf{label}'\mathsf{.sid}_i \neq \bot$. Similarly, they are contributive partners at stage i if they have opposite roles and $\mathsf{label.cid}_i = \mathsf{label}'\mathsf{.cid}_i \neq \bot$. We distinguish between *internal* stage keys, used within the handshake protocol, and *external* stage keys, exported for use by other protocols or applications. External stage keys may only be used outside the handshake protocol.

[9] Upon verifying the peer's certificate, the session sets pid to the certified identity if $\mathsf{pid} = *$; otherwise, it requires that this identity match the specified pid.

[10] This variable is defined only for PQ TLS 1.3, since we consider KEMTLS with fixed server-only authentication.

3.4 Adversary Interaction

We consider a polynomial-time adversary $\mathcal{A}$ that controls all communication and can intercept, inject, and drop messages. Accordingly, the adversary can initiate sessions and send and receive messages. Furthermore, it may reveal session keys and obtain long-term secrets through corruption. Finally, it can issue test queries, which return either a real or a random session key, and the adversary's goal is to distinguish between the two cases. More precisely, the adversary interacts with the following oracles:

- $\mathsf{NewSession}(U, V, \mathsf{role}[, \mathsf{auth}_{\mathsf{TLS}}])$: In PQ TLS 1.3, $\mathsf{NewSession}(U, V, \mathsf{role}, \mathsf{auth}_{\mathsf{TLS}})$ creates a new session label with owner U, role role, intended partner V, and authentication configuration $\mathsf{auth}_{\mathsf{TLS}}$. In KEMTLS, $\mathsf{NewSession}(U, V, \mathsf{role})$ creates a new session label with owner U, role role, and intended partner V. The intended partner may be left unspecified by setting $V = *$, capturing the post-specified peer (post-ID) setting [CK02][11].
- $\mathsf{Send}(\mathsf{label}, m)$: Sends message m to session label. If no such session has been created by $\mathsf{NewSession}$, return $\perp$. Otherwise, if $\mathsf{label.stage} = 0$, set $\mathsf{label.stage} \leftarrow 1$ before protocol execution. Run the protocol on behalf of $\mathsf{label.id}$, update its state, and return any response message and the current execution status $\mathsf{label.st}_{\mathsf{exec},\mathsf{label.stage}}$. If $\mathsf{label.st}_{\mathsf{exec},\mathsf{label.stage}} = \texttt{rejected}$, we terminate the execution of session label. To initiate a session, the adversary may submit the special message $m = \texttt{init}$ if $\mathsf{label.role} = \texttt{initiator}$.
  Handling stage acceptance. An internal key can only be tested before it is used, to avoid trivial attacks. To allow this, Send suspends execution at each stage acceptance and returns $\texttt{accepted}$ to $\mathcal{A}$. The adversary may then issue a Test query. Moreover, the adversary can resume the suspended session by querying $\mathsf{Send}(\mathsf{label}, \texttt{continue})$. If $\mathsf{label.st}_{\mathsf{exec},\mathsf{label.stage}} \neq \texttt{accepted}$, return $\perp$. Otherwise, before execution resumes, if $\mathsf{label.stage} < M$, set $\mathsf{label.stage} \leftarrow \mathsf{label.stage} + 1$ and $\mathsf{label.st}_{\mathsf{exec},\mathsf{label.stage}} \leftarrow \texttt{running}$. The protocol then continues and returns the next protocol message and $\mathsf{label.st}_{\mathsf{exec},\mathsf{label.stage}}$.
  On key acceptance at any stage i, if there exists a partnered session label' such that $\mathsf{label}'.\mathsf{tested}_i = \mathsf{true}$, set $\mathsf{label.tested}_i \leftarrow \mathsf{true}$. In this case, if $b = 0$ (where b is the global challenge bit used by the Test oracle) and the accepted stage key is internal, we furthermore set $\mathsf{label.key}_i \leftarrow \mathsf{label}'.\mathsf{key}_i$.
- $\mathsf{Reveal}(\mathsf{label}, i)$: If the stage key does not exist, return $\perp$. Otherwise, set $\mathsf{label.st}_{\mathsf{key},i} \leftarrow \mathsf{revealed}$ and return $\mathsf{label.key}_i$.
- $\mathsf{Corrupt}(U)$: Reveals the long-term secret key of user U, marks U as corrupted, and records the time of U's first corruption.
- $\mathsf{Test}(\mathsf{label}, i)$: Tests the stage-$i$ key of session label using a global hidden bit $b \in \{0, 1\}$. If $\mathsf{label.st}_{\mathsf{exec},i} \neq \texttt{accepted}$, or $\mathsf{label.tested}_i = \mathsf{true}$, or there exists a partner label' of label at stage i such that $\mathsf{label}'.\mathsf{tested}_i = \mathsf{true}$, then return $\perp$. If the stage-i key is internal and either $\mathsf{label.st}_{\mathsf{exec},i+1} \neq \perp$ or there exists a partnered session label' at stage i such that $\mathsf{label}'.\mathsf{st}_{\mathsf{exec},i+1} \neq \perp$, then return $\perp$. This ensures that neither the tested session nor any partnered session has already used the stage-i internal key.
  Set $\mathsf{label.tested}_i \leftarrow \mathsf{true}$ and choose a key K, where $K \leftarrow\!\!\$\ \mathcal{D}$[12] if $b = 0$ and K is equal to the real key $\mathsf{label.key}_i$ if $b = 1$. If the stage-i key is internal and $b = 0$, we replace $\mathsf{label.key}_i$ with K, so that the random key K is used consistently in all subsequent protocol computations involving $\mathsf{label.key}_i$. If there exists a partnered session label' that has accepted at stage i, then set $\mathsf{label}'.\mathsf{tested}_i \leftarrow \mathsf{true}$. Moreover, if $b = 0$ and the stage-i key is internal, we further set $\mathsf{label}'.\mathsf{key}_i \leftarrow \mathsf{label.key}_i$.
  Return K to the adversary.

[11] In the post-specified peer setting, a party may learn its peer's identity and long-term public key only during protocol execution.

[12] Here, $\mathcal{D}$ denotes the key distribution appropriate to the tested stage, with the corresponding key format and length.

Refinement of Internal-Key Handling. In the multi-stage models used in previous analyses of PQ TLS 1.3 and KEMTLS [SSW20, DFGS21, HV22, ZJZ24, CZC+26], after the Test oracle returns the challenge key K_b (a random key if $b = 0$ and the real key if $b = 1$) for an internal stage, the experiment replaces the corresponding internal stage keys of the tested session and its partner at that stage with K_b, which is then used in their subsequent protocol execution. In the ideal world ($b = 0$), this replacement is necessary because subsequent protocol messages depend on internal stage keys and would otherwise allow a trivial distinguishing attack. For an external stage key, the oracle simply returns K_b without modifying the protocol state.

This treatment is suitable for classical TLS 1.3, where the Diffie–Hellman key exchange is perfectly correct. However, for PQ TLS 1.3 and KEMTLS based on PQ KEMs that may incur decapsulation failures, the traditional handling of internal stage keys in the real world ($b = 1$) does not faithfully represent the actual protocol execution: the key replacement operation may artificially synchronize partnered sessions and hide the observable consequences of decapsulation failures. For example, when a KEM decapsulation error occurs, two partnered sessions may derive different internal stage keys with high probability. Since the traditional formulation overwrites the partner's internal stage key even in the real world, such disagreements are artificially eliminated at the internal stage and may therefore be masked in the key-indistinguishability game.

In this paper, we refine the Test oracle for internal stages as follows. In the real world ($b = 1$), the oracle returns the real key without modifying the protocol state. In the ideal world ($b = 0$), we retain the original treatment: the experiment replaces the corresponding internal stage keys of the tested session and its partner at that stage with the random key K_0, which is then used in their subsequent protocol execution. This ensures faithful protocol execution in the real world while maintaining the consistency required to simulate subsequent protocol messages in the ideal world.

3.5 Multi-Stage Security Model for PQ TLS 1.3

We now specify the parameters of PQ TLS 1.3 relevant to our proof in the multi-stage model, and then define the corresponding freshness condition and multi-stage security game.

Specifics of PQ TLS 1.3 in the Model

The protocol-specific properties of PQ TLS 1.3 are captured by the vector $(M, \mathsf{AUTH}_{\mathsf{TLS}}, \mathsf{FS}_{\mathsf{TLS}}, \mathsf{USE}_{\mathsf{TLS}}, \mathsf{REPLAY}_{\mathsf{TLS}})$, which specifies the number of protocol stages (M), the set of allowed authentication configurations ($\mathsf{AUTH}_{\mathsf{TLS}}$), the first stage from which forward secrecy is required ($\mathsf{FS}_{\mathsf{TLS}}$), the intended usage of each stage key (internal or external) ($\mathsf{USE}_{\mathsf{TLS}}$), and the replayability of each stage ($\mathsf{REPLAY}_{\mathsf{TLS}}$). Note that each session selects $\mathsf{auth}_{\mathsf{TLS}} \in \mathsf{AUTH}_{\mathsf{TLS}}$ upon creation.

- $M = 6$: PQ TLS 1.3 comprises six stages, which derive the following keys in order: the client and server handshake traffic keys $\mathsf{tk}_{\mathsf{chs}}, \mathsf{tk}_{\mathsf{shs}}$; the client and server application traffic secrets $\mathsf{CATS}, \mathsf{SATS}$; the exporter master secret EMS; and the resumption master secret RMS.
- $\mathsf{AUTH}_{\mathsf{TLS}} = \{((3, m), (3, m), (3, m), (4, m), (5, m), (6, m)) \mid m \in \{6, \infty\}\}$: For an $\mathsf{auth}_{\mathsf{TLS}} \in \mathsf{AUTH}_{\mathsf{TLS}}$, its i-th pair $(\mathsf{auth}_{\mathsf{TLS},i,1}, \mathsf{auth}_{\mathsf{TLS},i,2})$ specifies the intended stages of unilateral and mutual authentication for the stage-i key, respectively, with ∞ indicating that the corresponding authentication is not provided. The handshake traffic keys are initially unauthenticated and become unilaterally authenticated at stage 3; keys derived at later stages are unilaterally authenticated upon acceptance. If client authentication is performed, all keys additionally become mutually authenticated at stage $m = 6$; otherwise, $m = \infty$.
- $\mathsf{FS}_{\mathsf{TLS}} = 1$: PQ TLS 1.3 provides forward secrecy for all stage keys, subject to Freshness-TLS.
- $\mathsf{USE}_{\mathsf{TLS}} = (\texttt{internal} : \{1, 2\}, \texttt{external} : \{3, 4, 5, 6\})$: The handshake traffic keys are internal (used to protect subsequent handshake messages), while all other keys are external.
- $\mathsf{REPLAY}_{\mathsf{TLS}} = (\texttt{nonreplayable} : \{1, 2, 3, 4, 5, 6\})$: All stage keys are non-replayable in PQ TLS 1.3.

Session and contributive identifiers In PQ TLS 1.3, session identifiers are set upon acceptance of each stage and comprise a stage label together with all handshake messages generated or received up to the derivation of the corresponding key:

$$\begin{aligned}
\mathsf{sid}_1 &= (\texttt{"CHTS"}, \texttt{CH}, \texttt{SH}),\\
\mathsf{sid}_2 &= (\texttt{"SHTS"}, \texttt{CH}, \texttt{SH}),\\
\mathsf{sid}_3 &= (\texttt{"CATS"}, \texttt{CH}, \texttt{SH}, \texttt{EE}, \texttt{CR}^*, \texttt{SCRT}, \texttt{SCV}, \texttt{SF}),\\
\mathsf{sid}_4 &= (\texttt{"SATS"}, \texttt{CH}, \texttt{SH}, \texttt{EE}, \texttt{CR}^*, \texttt{SCRT}, \texttt{SCV}, \texttt{SF}),\\
\mathsf{sid}_5 &= (\texttt{"EMS"}, \texttt{CH}, \texttt{SH}, \texttt{EE}, \texttt{CR}^*, \texttt{SCRT}, \texttt{SCV}, \texttt{SF}),\\
\mathsf{sid}_6 &= (\texttt{"RMS"}, \texttt{CH}, \texttt{SH}, \texttt{EE}, \texttt{CR}^*, \texttt{SCRT}, \texttt{SCV}, \texttt{SF}, \texttt{CCRT}^*, \texttt{CCV}^*, \texttt{CF}).
\end{aligned}$$

Starred components (*) appear only in the mutual authentication setting. For the contributive identifiers in stages 1 and 2, the client (resp. server), upon sending (resp. receiving) `ClientHello`, sets $\mathsf{cid}_1 = (\texttt{"CHTS"}, \texttt{CH})$ and $\mathsf{cid}_2 = (\texttt{"SHTS"}, \texttt{CH})$. Subsequently, upon receiving (resp. sending) `ServerHello`, these identifiers are extended to $\mathsf{cid}_1 = (\texttt{"CHTS"}, \texttt{CH}, \texttt{SH})$ and $\mathsf{cid}_2 = (\texttt{"SHTS"}, \texttt{CH}, \texttt{SH})$. For stages $i \in \{3, 4, 5, 6\}$, we define $\mathsf{cid}_i = \mathsf{sid}_i$ once the corresponding session identifier is established.

In a client session, the first-stage key is accepted only upon receiving the `SH` message. Hence, a contributive partner of a tested client session must satisfy $\mathsf{cid}_1 = \mathsf{sid}_1$, ensuring that both parties have identical first-stage messages. Conversely, in the model, a server session accepts the first-stage key after receiving `CH` and constructing `SH`, but before sending `SH`. The Send oracle pauses at this acceptance point, when sid_1 includes `SH` but cid_1 contains only `CH`. Thus, at this point, the existence of a contributive partner guarantees agreement on the client messages, but not necessarily on the server messages.

Multi-Stage Security for PQ TLS 1.3

We now formalize the game defining *Multi-Stage security for PQ TLS 1.3*, following [DFGS21]. We first define Freshness-TLS.

Definition 3.1 (Freshness-TLS). *Let $i \in \{1, \ldots, M\}$ be a stage index. Stage i of a session* label *is said to satisfy* Freshness-TLS *if Conditions 1–3 below hold:*

1. *The stage key has not been revealed, i.e.,* $\mathsf{label}.\mathsf{st}_{\mathsf{key},i} = \mathsf{fresh}$.
2. *The stage key of any partnered session at stage i has not been revealed; namely, for every session* label' *partnered with* label *at stage i, we have* $\mathsf{label}'.\mathsf{st}_{\mathsf{key},i} = \mathsf{fresh}$.
3. *Either (a)* label *has a contributive partner at stage i, or (b) it has accepted at stage $j \neq \infty$ and no* $\mathsf{Corrupt}(\mathsf{label.pid})$ *query was issued before stage j of* label *accepted. Here, $j = \mathsf{label}.\mathsf{auth}_{\mathsf{TLS},i,1}$ if* $\mathsf{label.role} = \textit{initiator}$*, and $j = \mathsf{label}.\mathsf{auth}_{\mathsf{TLS},i,2}$ if* $\mathsf{label.role} = \textit{responder}$.

Compared with the original model in [DFGS21], we simplify the freshness conditions and relax the restrictions on long-term-key corruption.

Definition 3.2 (Multi-Stage Security Game for PQ TLS 1.3). *Let* KE *be a key-exchange protocol with properties* $(M, \mathsf{AUTH}_{\mathsf{TLS}}, \mathsf{FS}_{\mathsf{TLS}}, \mathsf{USE}_{\mathsf{TLS}}, \mathsf{REPLAY}_{\mathsf{TLS}})$ *and session-key distribution $\mathcal{D}$. For any polynomial-time adversary $\mathcal{A}$, the game* $\mathsf{G}^{\mathsf{MULTI\text{-}STAGE}}_{\mathsf{KE},\mathcal{A}}$ *is defined as follows:*

Setup: *A bit $b \leftarrow\!\!\$\ \{0,1\}$ is sampled uniformly. In the public-key setting, long-term key pairs $(\mathsf{pk}_U, \mathsf{sk}_U)$ are generated for all $U \in \mathcal{U}$.*

Query: *The adversary $\mathcal{A}$ receives all public keys and may issue queries to* $\mathsf{NewSession}$, Send, Reveal, $\mathsf{Corrupt}$, *and* Test.

Guess: *The adversary outputs a guess $b' \in \{0,1\}$.*

Finalize: *If there exists a session* label' *and a stage* $j \in \{1, \ldots, M\}$ *such that* $\mathsf{label}'.\mathsf{tested}_j = \mathsf{true}$ *but stage* j *of* label' *does not satisfy Freshness-TLS, then the game* $\mathsf{G}^{\text{MULTI-STAGE}}_{\mathsf{KE},\mathcal{A}}$ *outputs an independent uniformly random bit. Otherwise, it outputs* 1 *if and only if* $b' = b$.

The Multi-Stage advantage *of* $\mathcal{A}$ *is defined as*

$$\mathsf{Adv}^{\text{MULTI-STAGE}}_{\mathsf{KE}}(\mathcal{A}) = \left| \Pr\left[\mathsf{G}^{\text{MULTI-STAGE}}_{\mathsf{KE},\mathcal{A}} \Rightarrow 1\right] - \frac{1}{2} \right|.$$

3.6 Multi-Stage Security Model for KEMTLS

We next specify the parameters for KEMTLS relevant to our proof in the multi-stage model and retain the freshness and explicit-authentication conditions of [SSW20].

Specifics of KEMTLS in the Model

The protocol-specific properties of KEMTLS are specified by $(M, \mathsf{AUTH}_{\mathsf{KEMTLS}}, \mathsf{FS}_{\mathsf{KEMTLS}}, \mathsf{USE}_{\mathsf{KEMTLS}}, \mathsf{REPLAY}_{\mathsf{KEMTLS}})$, defined below. The parameters $\mathsf{AUTH}_{\mathsf{KEMTLS}}$ and $\mathsf{FS}_{\mathsf{KEMTLS}}$ are fixed for each role and interpreted according to the role of the session under consideration. They specify explicit-authentication and forward-secrecy requirements respectively, and differ in meaning from the corresponding parameters for PQ TLS 1.3.

- $M = 6$: KEMTLS comprises six stages, which derive the following keys in order: the client and server handshake traffic secrets $\mathsf{CHTS}, \mathsf{SHTS}$; the client and server authenticated handshake traffic secrets $\mathsf{CAHTS}, \mathsf{SAHTS}$; and the client and server application traffic secrets $\mathsf{CATS}, \mathsf{SATS}$.
- $\mathsf{AUTH}_{\mathsf{KEMTLS}} = (6, 6, 6, 6, 6, 6)$ for client sessions and $\mathsf{AUTH}_{\mathsf{KEMTLS}} = (\infty^{\times 6})$ for server sessions: The entry $\mathsf{AUTH}_{\mathsf{KEMTLS},i} = j$ specifies that the stage-i key is required to be explicitly authenticated upon acceptance at stage j, with ∞ indicating that explicit authentication is not required. In client sessions, keys derived at stages 1–5 become retroactively authenticated at stage 6, while the stage-6 key is authenticated upon acceptance.
- $\mathsf{FS}_{\mathsf{KEMTLS}} \in \{\mathsf{wfs}_1, \mathsf{wfs}_2, \mathsf{fs}\}^{M \times M}$: For $j \geq i$, the entry $\mathsf{FS}_{\mathsf{KEMTLS},i,j}$ specifies the forward-secrecy level required of the stage-i key once stage j has accepted in the session, subject to Freshness-KEMTLS. For client sessions,

$$\mathsf{FS}_{\mathsf{KEMTLS}} = \begin{pmatrix} \mathsf{wfs}_1 & \mathsf{wfs}_1 & \mathsf{wfs}_1 & \mathsf{wfs}_1 & \mathsf{wfs}_1 & \mathsf{fs} \\ & \mathsf{wfs}_1 & \mathsf{wfs}_1 & \mathsf{wfs}_1 & \mathsf{wfs}_1 & \mathsf{fs} \\ & & \mathsf{wfs}_2 & \mathsf{wfs}_2 & \mathsf{wfs}_2 & \mathsf{fs} \\ & & & \mathsf{wfs}_2 & \mathsf{wfs}_2 & \mathsf{fs} \\ & & & & \mathsf{wfs}_2 & \mathsf{fs} \\ & & & & & \mathsf{fs} \end{pmatrix}.$$

  For server sessions, $\mathsf{FS}_{\mathsf{KEMTLS},i,j} = \mathsf{wfs}_1$ for all $1 \leq i \leq j \leq M$.
- $\mathsf{USE}_{\mathsf{KEMTLS}} \in \{\texttt{internal}, \texttt{external}\}^M$: the per-stage key usage, where $\mathsf{USE}_{\mathsf{KEMTLS},i}$ indicates whether the stage-i key is internal, i.e., used within the handshake, or external, i.e., exported to applications. Both client and server sessions use $\mathsf{USE}_{\mathsf{KEMTLS}} = (\texttt{internal}^{\times 4}, \texttt{external}^{\times 2})$.
- $\mathsf{REPLAY}_{\mathsf{KEMTLS}} = (\texttt{nonreplayable} : \{1, 2, 3, 4, 5, 6\})$: replay protection is required for all stages.

Session and contributive identifiers The session identifier for each stage is set upon acceptance and consists of a stage label and the handshake transcript up to the derivation of the corresponding

key:

$$\begin{aligned}
\mathsf{sid}_1 &= (\texttt{"CHTS"}, \texttt{CH}, \texttt{SH}),\\
\mathsf{sid}_2 &= (\texttt{"SHTS"}, \texttt{CH}, \texttt{SH}),\\
\mathsf{sid}_3 &= (\texttt{"CAHTS"}, \texttt{CH}, \texttt{SH}, \texttt{EE}, \texttt{SC}, \texttt{CKC}),\\
\mathsf{sid}_4 &= (\texttt{"SAHTS"}, \texttt{CH}, \texttt{SH}, \texttt{EE}, \texttt{SC}, \texttt{CKC}),\\
\mathsf{sid}_5 &= (\texttt{"CATS"}, \texttt{CH}, \texttt{SH}, \texttt{EE}, \texttt{SC}, \texttt{CKC}, \texttt{CF}),\\
\mathsf{sid}_6 &= (\texttt{"SATS"}, \texttt{CH}, \texttt{SH}, \texttt{EE}, \texttt{SC}, \texttt{CKC}, \texttt{CF}, \texttt{SF}).
\end{aligned}$$

For stage 1, the client sets $\mathsf{cid}_1 = (\texttt{"CHTS"}, \texttt{CH})$ upon sending `ClientHello`; the server sets this identifier upon receiving this message. Upon sending `ServerHello`, the server updates the identifier to $\mathsf{cid}_1 = (\texttt{"CHTS"}, \texttt{CH}, \texttt{SH})$; the client makes the same update upon receiving this message. For each stage $i \in \{2, 3, 4, 5, 6\}$, we set $\mathsf{cid}_i = \mathsf{sid}_i$ when sid_i is set.

Multi-Stage Security for KEMTLS

We now formalize the game defining *Multi-Stage security for KEMTLS*, following [SSW20]. We first recall *Freshness-KEMTLS* and *malicious acceptance*.

Definition 3.3 (Freshness-KEMTLS). *Let $i \in \{1, \ldots, M\}$ be a stage index. Stage i of session* label *satisfies* Freshness-KEMTLS *if both Conditions 1 and 2 hold and at least one of Conditions 3–5 holds:*

1. *The stage-i key has not been revealed, i.e.,* $\mathsf{label.st}_{\mathsf{key},i} = \mathsf{fresh}$.
2. *For every session* label′ *partnered with* label *at stage i,* $\mathsf{label'.st}_{\mathsf{key},i} = \mathsf{fresh}$.
3. *(Weak forward secrecy 1) There exists $j \geq i$ such that* $\mathsf{FS}_{\mathsf{KEMTLS},i,j} = \mathsf{wfs}_1$ *and* $\mathsf{label.st}_{\mathsf{exec},j} = \texttt{accepted}$, *and* label *has a contributive partner at stage i.*
4. *(Weak forward secrecy 2) There exists $j \geq i$ such that* $\mathsf{FS}_{\mathsf{KEMTLS},i,j} = \mathsf{wfs}_2$ *and* $\mathsf{label.st}_{\mathsf{exec},j} = \texttt{accepted}$, *and either (a)* label *has a contributive partner at stage i or (b) no* Corrupt(label.pid) *query has been issued.*
5. *(Forward secrecy) There exists $j \geq i$ such that* $\mathsf{FS}_{\mathsf{KEMTLS},i,j} = \mathsf{fs}$ *and* $\mathsf{label.st}_{\mathsf{exec},j} = \texttt{accepted}$, *and either (a)* label *has a contributive partner at stage i or (b) no* Corrupt(label.pid) *query was issued before stage j of* label *accepted.*

Definition 3.4 (Malicious acceptance). *Let $i \in \{1, \ldots, M\}$ be a stage index and let $j = \mathsf{AUTH}_{\mathsf{KEMTLS},i}$. Stage i of a session* label *is said to* maliciously accept *if all of the following conditions hold:*

1. *$j \neq \infty$ and* $\mathsf{label.st}_{\mathsf{exec},j} = \texttt{accepted}$;
2. *there does not exist a unique partner of* label *at stage i; and*
3. *no* Corrupt(label.pid) *query was issued before stage j of* label *accepted.*

Definition 3.5 (Multi-Stage Security Game for KEMTLS). *Let* KE *be an M-stage key-exchange protocol with properties $(M, \mathsf{AUTH}_{\mathsf{KEMTLS}}, \mathsf{FS}_{\mathsf{KEMTLS}}, \mathsf{USE}_{\mathsf{KEMTLS}}, \mathsf{REPLAY}_{\mathsf{KEMTLS}})$ and session-key distribution $\mathcal{D}$, and let $\mathcal{A}$ be a polynomial-time adversary interacting with* KE *via the oracles* NewSession, Send, Reveal, Corrupt, *and* Test. *The experiment $\mathsf{G}^{\text{MULTI-STAGE}}_{\mathsf{KE},\mathcal{A}}$ proceeds as follows:*

Setup: *The challenger generates a long-term key pair $(\mathsf{pk}_U, \mathsf{sk}_U)$ for each $U \in \mathcal{U}$ and samples a bit $b \leftarrow\!\!\$\ \{0, 1\}$ shared across all* Test *queries. It provides all public keys to $\mathcal{A}$.*
Query: *The adversary may issue queries to* NewSession, Send, Reveal, Corrupt, *and* Test.
Guess: *The adversary halts and outputs a bit b' as its guess for b.*
Finalize: *The experiment determines its output as follows:*

1. *If any stage has maliciously accepted, output 1.*
2. *If no stage has maliciously accepted:*

– *If all tested stages satisfy Freshness-KEMTLS, output* 1 *if* $b' = b$*, and* 0 *otherwise.*
– *If at least one tested stage does not satisfy Freshness-KEMTLS, output an independent uniformly random bit. Here, stage* j *of a session* label' *is tested if* $\mathsf{label}'.\mathsf{tested}_j = \mathsf{true}$.

The Multi-Stage advantage *of* $\mathcal{A}$ *is defined as*

$$\mathsf{Adv}^{\mathsf{MULTI\text{-}STAGE}}_{\mathsf{KE}}(\mathcal{A}) = \left|\Pr\left[\mathsf{G}^{\mathsf{MULTI\text{-}STAGE}}_{\mathsf{KE},\mathcal{A}} \Rightarrow 1\right] - \frac{1}{2}\right|.$$

Remark 3.6 (Output convention). In the original Multi-Stage game (Definition B.5) of [SSW20], Finalize outputs a uniformly random bit when no stage maliciously accepts, all tested stages are fresh, and $b' \neq b$. This gives the adversary additional advantage even when its guess is incorrect. For example, an adversary that makes only fresh tests and causes no malicious acceptance obtains advantage 1/4 by guessing a bit independently and uniformly at random. In this paper, we correct this inconsistency by outputting 0 when no stage maliciously accepts, all tested stages are fresh, and $b' \neq b$.

4 Tight Multi-Stage Security of the PQ TLS 1.3 Handshake

In this section, we establish tight Multi-Stage security bounds for PQ TLS 1.3 in the standard model. Specifically, we show that the multi-stage security of PQ TLS 1.3 tightly reduces to the MUC-IND-1CCA or MUC-IND-1RCCA security of the ephemeral KEMs, the MU-EUF-CMA^c security of the signature scheme, and the MU-(dual-)PRF security of HKDF.

4.1 Security Analysis from MUC-IND-1CCA KEMs

Theorem 4.1 (Multi-Stage Security of PQ TLS 1.3 from IND-1CCA KEMs). *Let* ℓ_{nonce} *denote the nonce length. Suppose* KEM *is* δ_e*-correct and has public-key min-entropy at least* γ_{pk}*. For any polynomial-time adversary* $\mathcal{A}$ *in the Multi-Stage security game for* PQ TLS 1.3 *(Definition 3.2) with* t_u *users and at most* t_s *sessions, there exist polynomial-time adversaries* $\mathcal{B}_1, \ldots, \mathcal{B}_8$ *such that*

$$\begin{aligned}\mathsf{Adv}^{\mathsf{MULTI\text{-}STAGE}}_{\mathsf{PQTLS1.3\text{-}1RTT}}(\mathcal{A}) \leq{}& \frac{{t_s}^2}{2^{\gamma_{\mathsf{pk}}}} + \frac{t_s^2}{2^{\ell_{\mathsf{nonce}}}} + t_s^2\delta_e \\ &+ \mathsf{Adv}^{\mathsf{COLL}}_{H}(\mathcal{B}_1) + \mathsf{Adv}^{\mathsf{MU\text{-}EUF\text{-}CMA}^c}_{\mathsf{SIG},\,t_u}(\mathcal{B}_2) \\ &+ 2\,\mathsf{Adv}^{\mathsf{MUC\text{-}IND\text{-}1CCA}}_{\mathsf{KEM},\,t_s,\,t_s}(\mathcal{B}_3) + 2\,\mathsf{Adv}^{\mathsf{MU\text{-}dual\text{-}PRF}}_{\mathsf{HKDF.Extract},\,t_s}(\mathcal{B}_4) \\ &+ 2\,\mathsf{Adv}^{\mathsf{MU\text{-}PRF}}_{\mathsf{HKDF.Expand},\,t_s}(\mathcal{B}_5) + 2\,\mathsf{Adv}^{\mathsf{MU\text{-}PRF}}_{\mathsf{HKDF.Expand},\,2t_s}(\mathcal{B}_6) \\ &+ 2\,\mathsf{Adv}^{\mathsf{MU\text{-}PRF}}_{\mathsf{HKDF.Extract},\,t_s}(\mathcal{B}_7) + 2\,\mathsf{Adv}^{\mathsf{MU\text{-}PRF}}_{\mathsf{HKDF.Expand},\,t_s}(\mathcal{B}_8),\end{aligned}$$

where each $\mathcal{B}_i$*, for* $i \in \{1, \ldots, 8\}$*, runs in time comparable to that of* $\mathcal{A}$*.*

Proof. Let $\mathcal{A}$ be an adversary against the Multi-Stage security of PQ TLS 1.3. We assume that all tested sessions satisfy Freshness-TLS at the end of the experiment, as otherwise the adversary loses the game[13]. Consider the following sequence of games.

Game G_0**.** This is the original multi-stage security game for PQ TLS 1.3 (defined in Definition 3.2), with the following equivalent presampling of ephemeral KEM computations. At the beginning of the game, we independently generate $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow\!\$\ \mathsf{KEM.Gen}()$ for each $i \in [t_s]$. Moreover, we abort if two pre-sampled public keys coincide, incurring a loss of at most ${t_s}^2/2^{\gamma_{\mathsf{pk}}}$, since they are independently generated and have min-entropy at least γ_{pk}. For each $i, j \in [t_s]$, we additionally sample

[13] In this case, the game actually outputs an independent uniformly random bit, so such executions contribute zero to the adversary's advantage.

$(\mathsf{ct}_{i,j}, K^0_{i,j}) \leftarrow\!\!\$\ \mathsf{KEM.Encaps}(\mathsf{pk}_i)$. For the $(i+1)$-th honest client session label^i_C, the challenger uses $(\mathsf{pk}_i, \mathsf{sk}_i)$ as its ephemeral key pair. Let $\mathsf{label}^{i,j}_S$ (where $i, j \in [t_s]$) denote the $(j+1)$-th honest server session to receive pk_i. We now use $(\mathsf{ct}_{i,j}, K^0_{i,j})$ as the encapsulation values in $\mathsf{label}^{i,j}_S$. The rest of the game remains unchanged.

For later game analysis, we use the following notation: $\mathcal{C}_i := \{\mathsf{ct}_{i,j} \mid j \in [t_s]\}$. Note that we may have $|\mathcal{C}_i| \leq t_s$, since some ciphertexts may collide. For each $c \in \mathcal{C}_i$, let $j_i(c) := \min\{j \in [t_s] \mid \mathsf{ct}_{i,j} = c\}$ be its representative index. Note that presampling these KEM values does not change the distribution of the original game. Therefore,

$$\mathsf{Adv}^{\mathsf{MULTI\text{-}STAGE}}_{\mathsf{PQTLS1.3\text{-}1RTT}}(\mathcal{A}) \leq \left|\Pr[G^{\mathcal{A}}_0 \Rightarrow 1] - \tfrac{1}{2}\right| + \frac{{t_s}^2}{2^{\gamma_{\mathsf{pk}}}}.$$

Game G_1. In this game, we ensure that no two distinct honest sessions sample the same client nonce n_c or server nonce n_s. Specifically, if a session label samples a nonce that has already been sampled by another distinct honest session $\mathsf{label}' \neq \mathsf{label}$, the game aborts. By the birthday bound, the probability of such a collision among t_s sessions is at most $\frac{t_s^2}{2^{\ell_{\mathsf{nonce}}}}$. We additionally abort if an honestly presampled KEM ciphertext $\mathsf{ct}_{i,j}$ results in a decapsulation error, that is, $\mathsf{Decaps}(\mathsf{sk}_i, \mathsf{ct}_{i,j}) \neq K^0_{i,j}$. By the δ_e-correctness of the KEM and a union bound over at most t_s^2 ciphertexts, this event occurs with probability at most $t_s^2\delta_e$. Hence,

$$\left|\Pr[G^{\mathcal{A}}_0 \Rightarrow 1] - \Pr[G^{\mathcal{A}}_1 \Rightarrow 1]\right| \leq \frac{t_s^2}{2^{\ell_{\mathsf{nonce}}}} + t_s^2\delta_e.$$

Game G_2. In this game, we abort if honest sessions compute the same hash value under H on different inputs. That is, if there exist distinct inputs $x \neq x'$ evaluated by honest sessions such that $H(x) = H(x')$, the game aborts. This abort event yields a reduction to an adversary $\mathcal{B}_1$ that breaks the collision resistance of H. Therefore,

$$\left|\Pr[G^{\mathcal{A}}_1 \Rightarrow 1] - \Pr[G^{\mathcal{A}}_2 \Rightarrow 1]\right| \leq \mathsf{Adv}^{\mathsf{COLL}}_H(\mathcal{B}_1).$$

Game G_3 (SIG). In this game, we ensure that every tested session has an honest contributive partner at stage 1 at the end of the experiment. Specifically, at the end of the experiment, the game aborts if there exists a tested session without an honest contributive partner at stage 1. We denote this abort event by $\mathbf{Abort}_{\mathbf{SIG}}$.

The scenario in which the tested session label lacks a contributive partner at stage 1 implies that it has no contributive partner at any stage. By Condition 3 of Freshness-TLS (Definition 3.1), such a session must have accepted at the relevant authentication stage, with its intended peer uncorrupted before that acceptance. This implies that, for such a tested client session, the session has accepted at one of stages 3–6 (since stages 1 and 2 are initially unauthenticated), and its intended server peer must not have been corrupted before the session accepted at stage 3. Similarly, for such a server session, the session has accepted at stage 6, and its intended client peer must not have been corrupted before the session accepted at stage 6. Consequently, $\mathbf{Abort}_{\mathbf{SIG}}$ in Game G_3 implies that a client (resp. server) session without a contributive partner at stage 1 has successfully verified a `ServerCertificateVerify` (resp. `ClientCertificateVerify`) message and accepted at stage 3 (resp. stage 6) while its intended peer was still uncorrupted at that acceptance. We denote this acceptance event by BAD.

We now bound the probability of the event BAD by the advantage of an adversary $\mathcal{B}_2$ against the security of the signature scheme SIG under multi-user chosen-message attacks with corruption, denoted $\mathsf{Adv}^{\mathsf{MU\text{-}EUF\text{-}CMA}^c}_{\mathsf{SIG},t_u}(\mathcal{B}_2)$. In the game $\mathsf{G}^{\mathsf{MU\text{-}EUF\text{-}CMA}^c}_{\mathsf{SIG},t_u}(\mathcal{B}_2)$, $\mathcal{B}_2$ receives t_u challenge public keys $(\mathsf{pk}_U)_{U\in[t_u]}$ as input and can query a signing oracle $\mathsf{O}^{\mathsf{SIGN}}(i,\cdot)$ and a corruption oracle $\mathsf{O}^{\mathsf{Cor}}(i)$. Finally, $\mathcal{B}_2$ must output a forgery (i^*, m^*, σ^*) that constitutes a valid signature, where party i^* has not been corrupted ($\mathcal{B}_2$ does not query $\mathsf{O}^{\mathsf{Cor}}(i^*)$) and $\mathcal{B}_2$ has not previously queried $\mathsf{O}^{\mathsf{SIGN}}(i^*, m^*)$. The forger

$\mathcal{B}_2$ simulates Game G_3 for $\mathcal{A}$ as follows: it uses the challenge public keys to instantiate the signature keys of all parties and leverages the oracle $\mathsf{O}^{\mathsf{SIGN}}(i,\cdot)$ to generate signatures on behalf of party i. Upon receiving a corruption query $\mathsf{Corrupt}(j)$ from $\mathcal{A}$, the forger $\mathcal{B}_2$ invokes its oracle $\mathsf{O}^{\mathsf{Cor}}(j)$ and forwards the response to $\mathcal{A}$. Once the BAD event occurs in Game G_3, the forger $\mathcal{B}_2$ immediately halts the game and extracts a forgery. Let σ^* denote the signature received in the `ServerCertificateVerify` (resp. `ClientCertificateVerify`) message by the session label triggering BAD, which has no contributive partner at stage 1. Let U be the party such that σ^* is a valid signature under pk_U on message m^*. The forger $\mathcal{B}_2$ then outputs the tuple (U, m^*, σ^*).

We now argue that the tuple (U, m^*, σ^*) constitutes a successful forgery for $\mathcal{B}_2$. Recall that the expected signature is computed over $L_1 \parallel T_6$ for `ServerCertificateVerify` and $L_2 \parallel T_8$ for `ClientCertificateVerify`. In particular, the transcripts hashed in T_6 and T_8 both contain the first-stage messages $(\mathtt{CH}, \mathtt{SH})$ of the session label. However, the session label lacks a contributive partner in the first stage; thus it has no session partner at any stage. Since hash collisions have been excluded in Game G_2, no honest session has previously signed m^*. Hence, $\mathcal{B}_2$ has not previously queried $\mathsf{O}^{\mathsf{SIGN}}(U, m^*)$. Furthermore, by the definition of BAD, the corresponding peer U remains uncorrupted at this point, and hence $\mathcal{B}_2$ has not queried $\mathsf{O}^{\mathsf{Cor}}(U)$ to obtain the secret key sk_U. Therefore, the tuple (U, m^*, σ^*) satisfies the conditions of a valid forgery for $\mathcal{B}_2$. Since $\mathcal{B}_2$ perfectly simulates Game G_3 up to the first occurrence of BAD through access to its signing and corruption oracles, it succeeds whenever BAD occurs. As the $\mathbf{Abort_{SIG}}$ event implies a prior occurrence of the BAD event, we obtain

$$\left|\Pr[G_2^{\mathcal{A}} \Rightarrow 1] - \Pr[G_3^{\mathcal{A}} \Rightarrow 1]\right| \leq \Pr[\mathbf{Abort_{SIG}}] \leq \Pr[\mathsf{BAD}] \leq \mathsf{Adv}^{\mathsf{MU\text{-}EUF\text{-}CMA}^c}_{\mathsf{SIG},t_u}(\mathcal{B}_2).$$

Next, we show that, for all potentially tested sessions having an honest contributive partner at stage 1, the real stage keys are indistinguishable from independent uniformly random values (except that sessions partnered at a given stage share the same stage key).

Game G_4 (ephemeral KEM). In this game, the shared KEM secrets associated with the pre-sampled table are replaced with uniformly random values. Specifically, for each $i \in [t_s]$ and each distinct $c \in \mathcal{C}_i$, we first sample an independent $\widehat{K}_{i,c} \leftarrow\!\!\$\, \mathcal{K}$. Note that we may have $|\mathcal{C}_i| < t_s$ if some pre-sampled ciphertexts coincide. Then, for every server session $\mathsf{label}^{i,j}_S$, its KEM secret is replaced with $\widehat{K}_{i,\mathsf{ct}_{i,j}}$. If the client session label^{i}_C receives any ciphertext $c \in \mathcal{C}_i$, its KEM secret is also replaced with $\widehat{K}_{i,c}$. Note that this includes ciphertexts from table entries that have not been used by any honest server session. For any session in which K is replaced, all subsequent secret values that would have been derived from K are instead derived from its replacement value $\hat{K}$. The rest of the game remains unchanged.

Any adversary $\mathcal{A}$ that can distinguish between Game G_3 and Game G_4 can be used to construct an adversary $\mathcal{B}_3$ against the $\mathsf{MUC\text{-}IND\text{-}1CCA}$ security of the underlying KEM. The adversary $\mathcal{B}_3$ receives a challenge tuple $(\mathbf{pk}^*, \mathbf{ct}^*, \mathbf{K}^*_{\hat{b}})$ from its challenger. It defines $\mathcal{C}_i$ and $j_i(c)$ using its challenge ciphertext matrix. For each client session label^{i}_C, $\mathcal{B}_3$ embeds pk^*_i as its ephemeral public key. For each server session $\mathsf{label}^{i,j}_S$, it embeds $\mathsf{ct}^*_{i,j}$ into the corresponding $\mathtt{SH}$ message and sets the shared KEM secret to $K^*_{i,j_i(\mathsf{ct}^*_{i,j}),\hat{b}}$. If label^{i}_C later receives any $c \in \mathcal{C}_i$, $\mathcal{B}_3$ sets its KEM secret to $K^*_{i,j_i(c),\hat{b}}$, regardless of whether c has been sent by an honest server. If $\mathcal{A}$ instead delivers $c \notin \mathcal{C}_i$ to label^{i}_C, then $\mathcal{B}_3$ queries its IND-1CCA decapsulation oracle $\mathsf{O}^{\mathsf{Dec}}(i,c)$ to simulate label^{i}_C. Each client session performs at most one decapsulation, so $\mathcal{B}_3$ makes at most one decapsulation query per challenge public key. The adversary $\mathcal{B}_3$ simulates all other parts of the game exactly as in Game G_3.

By Game G_1, there are no decapsulation errors for the pre-sampled ciphertexts on non-aborting executions. Consequently, all pre-sampled entries with the same public key and ciphertext have the same real KEM key, which also equals the key obtained by the corresponding client session through decapsulation in Game G_3. Hence when $\hat{b} = 0$, the challenge keys correspond to real KEM keys, and $\mathcal{B}_3$ perfectly simulates Game G_3. When $\hat{b} = 1$, the challenge keys correspond to uniformly random

values, and $\mathcal{B}_3$ perfectly simulates Game G_4. Therefore,

$$\left|\Pr[G_3^{\mathcal{A}} \Rightarrow 1] - \Pr[G_4^{\mathcal{A}} \Rightarrow 1]\right| \leq 2\,\mathsf{Adv}^{\mathsf{MUC\text{-}IND\text{-}1CCA}}_{\mathsf{KEM},\,t_s,\,t_s}(\mathcal{B}_3).$$

Game G_5 (random HS). In this game, for each session using the randomized $\widehat{K}_{i,c}$, its HS is replaced with a uniformly random value. Specifically, the handshake secret $\mathsf{HS} = \mathsf{HKDF.Extract}(\mathsf{dES}, \widehat{K}_{i,c})$ of all sessions associated with the same pair (i, c) (that is, sessions using the corresponding pk_i and some $c \in \mathcal{C}_i$ that use the same randomized $\widehat{K}_{i,c}$ in Game G_4), where $i \in [t_s]$ and $c \in \mathcal{C}_i$, is replaced with the same uniformly random value $\widehat{\mathsf{HS}}_{i,c}$. For distinct pairs (i, c), the replacement values are independent. For any session in which HS is replaced, all subsequent secret values derived from HS are instead derived from $\widehat{\mathsf{HS}}$. This transition is indistinguishable if $\mathsf{HKDF.Extract}$ is a secure $\mathsf{MU\text{-}dual\text{-}PRF}$ when its second argument is the PRF secret key.

We construct an adversary $\mathcal{B}_4$ against the $\mathsf{MU\text{-}dual\text{-}PRF}$ security of $\mathsf{HKDF.Extract}$ as follows. The PRF reduction $\mathcal{B}_4$ is given access to $\mathsf{O}^{\mathsf{PRF}}_{\hat{b}}$ and simulates the game for $\mathcal{A}$ as follows. It assigns a distinct PRF user $j \in [t_s]$ to each pair (i, c) when $\widehat{K}_{i,c}$ is first used by a session. Note that there are at most t_s sessions in the game. For each session using $\widehat{K}_{i,c}$, it queries the corresponding oracle $\mathsf{O}^{\mathsf{PRF}}_{\hat{b}}(j, \mathsf{dES})$ and uses the response as its handshake secret HS. All other parts of the game are simulated as in Game G_4. If $\hat{b} = 0$, the response is $\mathsf{HKDF.Extract}(\mathsf{dES}, \widehat{K}_{i,c})$, which perfectly simulates Game G_4. If $\hat{b} = 1$, the responses are independent uniform values across distinct pairs, which perfectly simulates Game G_5. Therefore,

$$\left|\Pr[G_4^{\mathcal{A}} \Rightarrow 1] - \Pr[G_5^{\mathcal{A}} \Rightarrow 1]\right| \leq 2\,\mathsf{Adv}^{\mathsf{MU\text{-}dual\text{-}PRF}}_{\mathsf{HKDF.Extract},\,t_s}(\mathcal{B}_4).$$

Game G_6 (random CHTS, SHTS, and dHS). In this game, for each session using the randomized $\widehat{\mathsf{HS}}_{i,c}$, its CHTS, SHTS, and dHS are replaced with uniformly random values. Note that sessions associated with the same pair (i, c) share the same $\widehat{\mathsf{dHS}}_{i,c}$. They further use the same $\widehat{\mathsf{CHTS}}$ and $\widehat{\mathsf{SHTS}}$ if their T_2 values are also the same, and independent replacement values otherwise. For sessions associated with different pairs (i, c), all replacement values are independent. All subsequent secret values derived from these values are instead derived from their replacements. This transition is indistinguishable if $\mathsf{HKDF.Expand}$ behaves as a secure $\mathsf{MU\text{-}PRF}$ for t_s users when keyed with the random values $\widehat{\mathsf{HS}}_{i,c}$. Therefore,

$$\left|\Pr[G_5^{\mathcal{A}} \Rightarrow 1] - \Pr[G_6^{\mathcal{A}} \Rightarrow 1]\right| \leq 2\,\mathsf{Adv}^{\mathsf{MU\text{-}PRF}}_{\mathsf{HKDF.Expand},\,t_s}(\mathcal{B}_5).$$

Game G_7 (random $\mathsf{tk_{chs}}$, $\mathsf{tk_{shs}}$, $\mathsf{fk_c}$, and $\mathsf{fk_s}$). In this game, for each session using the randomized $\widehat{\mathsf{CHTS}}$ and $\widehat{\mathsf{SHTS}}$, its corresponding traffic keys $\mathsf{tk_{chs}}$, $\mathsf{tk_{shs}}$ and Finished keys $\mathsf{fk_c}$, $\mathsf{fk_s}$ are replaced with independent uniformly random values $\widehat{\mathsf{tk}}_{\mathsf{chs}}, \widehat{\mathsf{tk}}_{\mathsf{shs}}, \widehat{\mathsf{fk}}_{\mathsf{c}}, \widehat{\mathsf{fk}}_{\mathsf{s}}$. Note that sessions associated with the same pair (i, c) use the same corresponding replacement values if their T_2 values are also the same, and independent replacement values otherwise. For sessions associated with different pairs (i, c), the replacement values are independent. For any session in which these keys are replaced, all subsequent computations use the corresponding randomized values. This transition is indistinguishable if $\mathsf{HKDF.Expand}$ is a secure $\mathsf{MU\text{-}PRF}$ for $2t_s$ users when keyed with the random values $\widehat{\mathsf{CHTS}}$ and $\widehat{\mathsf{SHTS}}$. Therefore,

$$\left|\Pr[G_6^{\mathcal{A}} \Rightarrow 1] - \Pr[G_7^{\mathcal{A}} \Rightarrow 1]\right| \leq 2\,\mathsf{Adv}^{\mathsf{MU\text{-}PRF}}_{\mathsf{HKDF.Expand},\,2t_s}(\mathcal{B}_6).$$

Game G_8 (random MS). In this game, for each session using the randomized $\widehat{\mathsf{dHS}}_{i,c}$, its master secret $\mathsf{MS} = \mathsf{HKDF.Extract}(\widehat{\mathsf{dHS}}_{i,c}, 0)$ is replaced with a uniformly random value $\widehat{\mathsf{MS}}_{i,c}$. Note that sessions associated with the same pair (i, c) share the same $\widehat{\mathsf{MS}}_{i,c}$. For sessions associated with different pairs (i, c), the replacement values are independent. For any session in which MS is replaced,

all subsequent secret values that would have been derived from MS are instead computed using $\widehat{\mathsf{MS}}$. This transition is indistinguishable if $\mathsf{HKDF.Extract}$ behaves as a secure $\mathsf{MU\text{-}PRF}$ for t_s users when keyed with the random value $\widehat{\mathsf{dHS}}$. Thus,

$$\left|\Pr[G_7^{\mathcal{A}} \Rightarrow 1] - \Pr[G_8^{\mathcal{A}} \Rightarrow 1]\right| \leq 2\,\mathsf{Adv}^{\mathsf{MU\text{-}PRF}}_{\mathsf{HKDF.Extract},\,t_s}(\mathcal{B}_7).$$

Game G_9 (random CATS, SATS, EMS, and RMS). In this game, for each session using the randomized $\widehat{\mathsf{MS}}_{i,c}$, its CATS, SATS, EMS, and RMS are replaced with uniformly random values. Note that sessions associated with the same pair (i, c) use the same $\widehat{\mathsf{CATS}}$, $\widehat{\mathsf{SATS}}$, and $\widehat{\mathsf{EMS}}$ if their T_3 values are also the same, and independent replacement values otherwise. They further use the same $\widehat{\mathsf{RMS}}$ if their T_4 values are the same, and independent replacement values otherwise. For sessions associated with different pairs (i, c), the replacement values are independent. All subsequent secret values derived from these values are instead derived from their replacements. This transition is indistinguishable if $\mathsf{HKDF.Expand}$ behaves as a secure $\mathsf{MU\text{-}PRF}$ for t_s users when keyed with the random values $\widehat{\mathsf{MS}}_{i,c}$. Therefore,

$$\left|\Pr[G_8^{\mathcal{A}} \Rightarrow 1] - \Pr[G_9^{\mathcal{A}} \Rightarrow 1]\right| \leq 2\,\mathsf{Adv}^{\mathsf{MU\text{-}PRF}}_{\mathsf{HKDF.Expand},\,t_s}(\mathcal{B}_8).$$

At this point, all stage keys in every potentially tested session have been replaced with uniformly random values that are independent across sessions that are not partnered at the corresponding stage (since, by Game G_2, only sessions partnered at the corresponding stage have the same transcript hash used in that stage's key derivation). Therefore, the challenge bit is independent of the adversary's view, and we have

$$\Pr[G_9^{\mathcal{A}} \Rightarrow 1] = \tfrac{1}{2}.$$

Combining the above bounds completes the proof. □

4.2 Security Analysis from MUC-IND-1RCCA KEMs

Theorem 4.2 (Multi-Stage Security of PQ TLS 1.3 from MUC-IND-1RCCA KEMs). *Let ℓ_{nonce} denote the nonce length. Let $\mathcal{K}$ denote the key space of KEM. Suppose KEM is δ_e-correct and has public-key min-entropy at least γ_{pk}. For any polynomial-time adversary $\mathcal{A}$ in the Multi-Stage security game for* PQ TLS 1.3 *(Definition 3.2) with t_u users and at most t_s sessions, there exist polynomial-time adversaries $\mathcal{B}_1, \ldots, \mathcal{B}_8$ such that*

$$\begin{aligned}
\mathsf{Adv}^{\mathsf{MULTI\text{-}STAGE}}_{\mathsf{PQTLS1.3\text{-}1RTT}}(\mathcal{A}) \leq\ & \frac{{t_s}^2}{2^{\gamma_{\mathsf{pk}}}} + \frac{t_s^2}{2^{\ell_{\mathsf{nonce}}}} + t_s^2\delta_e + \frac{t_s^2}{|\mathcal{K}|} \\
& + \mathsf{Adv}^{\mathsf{COLL}}_{H}(\mathcal{B}_1) + \mathsf{Adv}^{\mathsf{MU\text{-}EUF\text{-}CMA}^c}_{\mathsf{SIG},\,t_u}(\mathcal{B}_2) \\
& + 2\,\mathsf{Adv}^{\mathsf{MUC\text{-}IND\text{-}1RCCA}}_{\mathsf{KEM},\,t_s,\,t_s}(\mathcal{B}_3) + 2\,\mathsf{Adv}^{\mathsf{MU\text{-}dual\text{-}PRF}}_{\mathsf{HKDF.Extract},\,t_s}(\mathcal{B}_4) \\
& + 2\,\mathsf{Adv}^{\mathsf{MU\text{-}PRF}}_{\mathsf{HKDF.Expand},\,t_s}(\mathcal{B}_5) + 2\,\mathsf{Adv}^{\mathsf{MU\text{-}PRF}}_{\mathsf{HKDF.Expand},\,2t_s}(\mathcal{B}_6) \\
& + 2\,\mathsf{Adv}^{\mathsf{MU\text{-}PRF}}_{\mathsf{HKDF.Extract},\,t_s}(\mathcal{B}_7) + 2\,\mathsf{Adv}^{\mathsf{MU\text{-}PRF}}_{\mathsf{HKDF.Expand},\,t_s}(\mathcal{B}_8),
\end{aligned}$$

where each $\mathcal{B}_i$, for $i \in \{1, \ldots, 8\}$, runs in time comparable to that of $\mathcal{A}$.

Proof Sketch. The proof follows the same structure as that of Theorem 4.1. We first invoke the $\mathsf{MUC\text{-}IND\text{-}1RCCA}$ security of KEM to replace the relevant KEM keys K with uniformly random values, ensuring that sessions using the same challenge ciphertext or a replayable ciphertext associated with it receive the same random KEM key. Next, we rely on the $\mathsf{MU\text{-}dual\text{-}PRF}$ security of $\mathsf{HKDF.Extract}$ and the $\mathsf{MU\text{-}PRF}$ security of HKDF to gradually replace the stage keys with uniformly random values. Since each stage-key derivation incorporates a transcript hash that binds the KEM

ciphertext, a different ciphertext that decapsulates to the same KEM key yields a different derivation input after hash collisions are excluded. Consequently, the PRF replacements yield independent stage keys for sessions not partnered at the corresponding stage, while preserving matching keys for partnered sessions.

Proof. Let $\mathcal{A}$ be an adversary against the Multi-Stage security of PQ TLS 1.3. We assume that all tested sessions satisfy Freshness-TLS at the end of the experiment, and consider the following sequence of games.

Games G_0–G_3. These games are identical to Games G_0–G_3 in the proof of Theorem 4.1. We also reuse the notation defined in Game G_0 of Theorem 4.1.

Game G_4 (ephemeral KEM). In this game, the shared KEM secrets associated with the pre-sampled table are replaced with uniformly random values. This replacement also applies to client sessions receiving replayable ciphertexts outside the pre-sampled table.

Specifically, for each $i, j \in [t_s]$, we sample an independent $\widehat{K}_{i,j} \leftarrow\!\!\$\ \mathcal{K}$. For every server session $\mathsf{label}_S^{i,j}$, its KEM secret is replaced with $\widehat{K}_{i,j_i(\mathsf{ct}_{i,j})}$. If the client session label_C^i receives any $c \in \mathcal{C}_i$, its KEM secret is also replaced with $\widehat{K}_{i,j_i(c)}$. Note that this includes ciphertexts from table entries that have not been used by any honest server session. The main difference from Game G_4 in the proof of Theorem 4.1 is as follows. If label_C^i instead receives $c \notin \mathcal{C}_i$, we first compute $K' \leftarrow \mathsf{Decaps}(\mathsf{sk}_i, c)$. If $K' = K^0_{i,j}$ for some $j \in [t_s]$, we set $j^* := \min\{j \in [t_s] \mid K' = K^0_{i,j}\}$ and further replace the client's KEM secret with $\widehat{K}_{i,j^*}$. Otherwise, the client continues to use the decapsulated value K'. For any session in which K is replaced, all subsequent secret values derived from K are instead derived from its replacement $\widehat{K}$. The rest of the game remains unchanged.

Any adversary $\mathcal{A}$ that distinguishes Game G_3 from Game G_4 can be used to construct an adversary $\mathcal{B}_3$ against the MUC-IND-1RCCA security of KEM, as follows. The adversary $\mathcal{B}_3$ receives a challenge tuple $(\mathbf{pk}^*, \mathbf{ct}^*, \mathbf{K}^*_{\hat{b}})$ from its challenger. It defines $\mathcal{C}_i$ and $j_i(c)$ using its challenge ciphertext matrix. For each client session label_C^i, $\mathcal{B}_3$ embeds pk_i^* as its ephemeral public key. For each server session $\mathsf{label}_S^{i,j}$, it embeds $\mathsf{ct}^*_{i,j}$ into the corresponding SH message and sets the KEM secret to $K^*_{i,j_i(\mathsf{ct}^*_{i,j}),\hat{b}}$. If label_C^i later receives any $c \in \mathcal{C}_i$, $\mathcal{B}_3$ sets its KEM secret to $K^*_{i,j_i(c),\hat{b}}$, regardless of whether c has been sent by an honest server. If $\mathcal{A}$ instead delivers $c \notin \mathcal{C}_i$ to label_C^i, $\mathcal{B}_3$ queries $\mathsf{O}^{\mathsf{Dec}'}(i, c)$. If the oracle returns Test-j, it uses $K^*_{i,j,\hat{b}}$ as the KEM secret in label_C^i. Otherwise, it uses the returned decapsulation result to simulate label_C^i. Each client session performs at most one decapsulation, so $\mathcal{B}_3$ makes at most one decapsulation query per challenge public key. All other aspects of the game are simulated as in Game G_3.

By Game G_1, there are no decapsulation errors for the pre-sampled ciphertexts on non-aborting executions. Consequently, all entries in row i of $\mathbf{ct}$ containing the same ciphertext have the same real KEM key, which equals the corresponding decapsulation result in Game G_3. When $\hat{b} = 0$, the challenge keys $K^*_{i,j,0}$ correspond to real KEM keys, and $\mathcal{B}_3$ perfectly simulates Game G_3, except if a query to $\mathsf{O}^{\mathsf{Dec}'}(i, c)$ matches a hidden random key $K^*_{i,j,1}$ (i.e., if $\mathsf{Decaps}(\mathsf{sk}_i, c) = K^*_{i,j,1}$ for some $j \in [t_s]$) when the client session label_C^i receives $c \notin \mathcal{C}_i$. Since the hidden keys $K^*_{i,j,1}$ are uniformly random and independent of the adversary's view, a union bound over at most t_s decapsulation queries, each compared with at most t_s hidden keys, bounds the probability of this event by $t_s^2/|\mathcal{K}|$. Thus, when $\hat{b} = 0$, $\mathcal{B}_3$ simulates Game G_3 perfectly, except with probability at most $t_s^2/|\mathcal{K}|$.

When $\hat{b} = 1$, the challenge keys $K^*_{i,j,1}$ are uniformly random. For a client session label_C^i that receives $c \notin \mathcal{C}_i$, if $\mathsf{Decaps}(\mathsf{sk}_i, c)$ matches a real challenge key, the oracle $\mathsf{O}^{\mathsf{Dec}'}(i, c)$ returns Test-j for the smallest matching real-key index j. The reduction $\mathcal{B}_3$ then uses $K^*_{i,j,1}$ as the client's KEM secret, exactly as in Game G_4. If no real challenge key matches but a random challenge key $K^*_{i,j,1}$ matches, $\mathcal{B}_3$ also uses $K^*_{i,j,1}$, which is the ordinary decapsulation result in Game G_4. Otherwise, it uses the decapsulation result returned by $\mathsf{O}^{\mathsf{Dec}'}(i, c)$ to simulate label_C^i as in Game G_4. Therefore,

when $\hat{b} = 1$, $\mathcal{B}_3$ perfectly simulates Game G_4. Hence,

$$\left|\Pr[G_3^{\mathcal{A}} \Rightarrow 1] - \Pr[G_4^{\mathcal{A}} \Rightarrow 1]\right| \leq 2\,\mathsf{Adv}^{\mathsf{MUC\text{-}IND\text{-}1RCCA}}_{\mathsf{KEM},\,t_s,\,t_s}(\mathcal{B}_3) + \frac{t_s^2}{|\mathcal{K}|}.$$

In the following games, we say that a session is associated with a pair $(i, j) \in [t_s] \times [t_s]$ if the randomized KEM secret $\widehat{K}_{i,j}$ is assigned to it in Game G_4. In particular, sessions sharing the same pair (i, j) include those using the same public key and table ciphertext, as well as the corresponding client session receiving a non-table replayable ciphertext that maps to the same index j.

Game G_5 (random HS). In this game, for each session using the randomized $\widehat{K}_{i,j}$, its handshake secret $\mathsf{HS} = \mathsf{HKDF.Extract}(\mathsf{dES}, \widehat{K}_{i,j})$ is replaced with a uniformly random value $\widehat{\mathsf{HS}}_{i,j}$. Note that sessions associated with the same pair (i, j) share the same $\widehat{\mathsf{HS}}_{i,j}$. For sessions associated with different pairs (i, j), the replacement values are independent. For any session in which HS is replaced, all subsequent secret values derived from HS are instead derived from $\widehat{\mathsf{HS}}_{i,j}$. This transition is indistinguishable if $\mathsf{HKDF.Extract}$ is a secure $\mathsf{MU\text{-}dual\text{-}PRF}$ when its second argument is the secret key.

We construct an adversary $\mathcal{B}_4$ against the $\mathsf{MU\text{-}dual\text{-}PRF}$ security of $\mathsf{HKDF.Extract}$ as follows. The PRF reduction $\mathcal{B}_4$ is given access to $\mathsf{O}^{\mathsf{PRF}}_{\hat{b}}$ and simulates the game for $\mathcal{A}$. It generates the original KEM table locally and can therefore determine the index assigned to each session. It assigns a distinct PRF user $u \in [t_s]$ to each pair (i, j) when $\widehat{K}_{i,j}$ is first assigned to a session. Since there are at most t_s sessions and each session is associated with at most one such pair, at most t_s users are needed. For each session associated with (i, j), it queries the corresponding oracle $\mathsf{O}^{\mathsf{PRF}}_{\hat{b}}(u, \mathsf{dES})$ and uses the response as its handshake secret HS. All other parts of the game are simulated as in Game G_4. If $\hat{b} = 0$, the response is $\mathsf{HKDF.Extract}(\mathsf{dES}, \widehat{K}_{i,j})$, which perfectly simulates Game G_4. If $\hat{b} = 1$, the responses are independent uniform values across distinct pairs, which perfectly simulates Game G_5. Therefore,

$$\left|\Pr[G_4^{\mathcal{A}} \Rightarrow 1] - \Pr[G_5^{\mathcal{A}} \Rightarrow 1]\right| \leq 2\,\mathsf{Adv}^{\mathsf{MU\text{-}dual\text{-}PRF}}_{\mathsf{HKDF.Extract},\,t_s}(\mathcal{B}_4).$$

Game G_6 (random CHTS, SHTS, and dHS). In this game, for each session using the randomized $\widehat{\mathsf{HS}}_{i,j}$, its CHTS, SHTS, and dHS are replaced with uniformly random values. Note that sessions associated with the same pair (i, j) share the same $\widehat{\mathsf{dHS}}_{i,j}$. They use the same $\widehat{\mathsf{CHTS}}$ and $\widehat{\mathsf{SHTS}}$ if their T_2 values are also the same, and independent replacement values otherwise. For sessions associated with different pairs (i, j), the replacement values are independent. All subsequent secret values derived from these values are instead derived from their replacements. Notably, if a client session receives a different replayable ciphertext but shares the same $\widehat{K}_{i,j}$ with a server session, their T_2 values differ since hash collisions are excluded in Game G_2. Thus, they use independent randomized handshake traffic secrets $\widehat{\mathsf{CHTS}}$ and $\widehat{\mathsf{SHTS}}$.

This transition is indistinguishable if $\mathsf{HKDF.Expand}$ behaves as a secure $\mathsf{MU\text{-}PRF}$ for t_s users when keyed with the random values $\widehat{\mathsf{HS}}_{i,j}$. Therefore,

$$\left|\Pr[G_5^{\mathcal{A}} \Rightarrow 1] - \Pr[G_6^{\mathcal{A}} \Rightarrow 1]\right| \leq 2\,\mathsf{Adv}^{\mathsf{MU\text{-}PRF}}_{\mathsf{HKDF.Expand},\,t_s}(\mathcal{B}_5).$$

Game G_7 (random $\mathsf{tk_{chs}}$, $\mathsf{tk_{shs}}$, $\mathsf{fk_c}$, and $\mathsf{fk_s}$). In this game, for each session using the randomized $\widehat{\mathsf{CHTS}}$ and $\widehat{\mathsf{SHTS}}$, its corresponding traffic keys $\mathsf{tk_{chs}}$, $\mathsf{tk_{shs}}$ and Finished keys $\mathsf{fk_c}$, $\mathsf{fk_s}$ are replaced with independent uniformly random values $\widehat{\mathsf{tk}}_{\mathsf{chs}}, \widehat{\mathsf{tk}}_{\mathsf{shs}}, \widehat{\mathsf{fk}}_{\mathsf{c}}, \widehat{\mathsf{fk}}_{\mathsf{s}}$. Note that sessions associated with the same pair (i, j) use the same corresponding replacement values if their T_2 values are also the same, and independent replacement values otherwise. For sessions associated with different pairs (i, j), the replacement values are independent. For any session in which these keys are replaced, all

subsequent computations use the corresponding randomized values. This transition is indistinguishable if HKDF.Expand is a secure MU-PRF for $2t_s$ users when keyed with the random values $\widehat{\mathsf{CHTS}}$ and $\widehat{\mathsf{SHTS}}$. Therefore,

$$\left|\Pr[G_6^{\mathcal{A}} \Rightarrow 1] - \Pr[G_7^{\mathcal{A}} \Rightarrow 1]\right| \leq 2\,\mathsf{Adv}^{\mathsf{MU\text{-}PRF}}_{\mathsf{HKDF.Expand},\,2t_s}(\mathcal{B}_6).$$

Game G_8 (random MS). In this game, for each session using the randomized $\widehat{\mathsf{dHS}}_{i,j}$, its master secret $\mathsf{MS} = \mathsf{HKDF.Extract}(\widehat{\mathsf{dHS}}_{i,j}, 0)$ is replaced with a uniformly random value $\widehat{\mathsf{MS}}_{i,j}$. Note that sessions associated with the same pair (i,j) share the same $\widehat{\mathsf{MS}}_{i,j}$. For sessions associated with different pairs (i,j), the replacement values are independent. For any session in which MS is replaced, all subsequent secret values derived from MS are instead derived from $\widehat{\mathsf{MS}}_{i,j}$. This transition is indistinguishable if HKDF.Extract behaves as a secure MU-PRF for t_s users when keyed with the random values $\widehat{\mathsf{dHS}}_{i,j}$. Therefore,

$$\left|\Pr[G_7^{\mathcal{A}} \Rightarrow 1] - \Pr[G_8^{\mathcal{A}} \Rightarrow 1]\right| \leq 2\,\mathsf{Adv}^{\mathsf{MU\text{-}PRF}}_{\mathsf{HKDF.Extract},\,t_s}(\mathcal{B}_7).$$

Game G_9 (random CATS, SATS, EMS, and RMS). In this game, for each session using the randomized $\widehat{\mathsf{MS}}_{i,j}$, its CATS, SATS, EMS, and RMS are replaced with uniformly random values. Note that sessions associated with the same pair (i,j) use the same $\widehat{\mathsf{CATS}}$, $\widehat{\mathsf{SATS}}$, and $\widehat{\mathsf{EMS}}$ if their T_3 values are also the same, and independent replacement values otherwise. They use the same $\widehat{\mathsf{RMS}}$ if their T_4 values are the same, and independent replacement values otherwise. For sessions associated with different pairs (i,j), the replacement values are independent. All subsequent secret values derived from these values are instead derived from their replacements. This transition is indistinguishable if HKDF.Expand behaves as a secure MU-PRF for t_s users when keyed with the random values $\widehat{\mathsf{MS}}_{i,j}$. Therefore,

$$\left|\Pr[G_8^{\mathcal{A}} \Rightarrow 1] - \Pr[G_9^{\mathcal{A}} \Rightarrow 1]\right| \leq 2\,\mathsf{Adv}^{\mathsf{MU\text{-}PRF}}_{\mathsf{HKDF.Expand},\,t_s}(\mathcal{B}_8).$$

At this point, all stage keys in every potentially tested session have been replaced with uniformly random values that are independent across sessions that are not partnered at the corresponding stage (since, by Game G_2, only sessions partnered at the corresponding stage have the same transcript hash used in that stage's key derivation). Therefore, the challenge bit is independent of the adversary's view, and we have

$$\Pr[G_9^{\mathcal{A}} \Rightarrow 1] = \tfrac{1}{2}.$$

Combining the above bounds completes the proof. □

5 Tight Multi-Stage Security of the KEMTLS Handshake in the ROM

In this section, we show that the security of KEMTLS can be tightly reduced to the MUC-IND-1CCA/MUC-IND-1RCCA security of KEM_e and the MUC-OW-ChCCA* security of KEM_s in the ROM. The complete KEMTLS handshake flow is illustrated in Fig. 6.

We model the key domain-separated HKDF functions considered in our security analysis and the HMAC function used for the client Finished/server Finished messages as independent public random oracles. Specifically, for HKDF, we use random oracles G_i, $H_1^{(i)}$, and $H_2^{(i)}$ for $i \in \{1,2,3\}$, and $H_3^{(j)}$ for $j \in \{1,2,3,4\}$, each mapping $\{0,1\}^*$ to $\{0,1\}^n$. For HMAC, we use the random oracle $H_{\mathsf{MAC}}\colon \{0,1\}^n \times \{0,1\}^* \to \{0,1\}^\mu$, where μ is the HMAC output length.

We can then rewrite the key schedule of KEMTLS as follows. The handshake, authenticated handshake, and master secrets are computed as $\mathsf{HS} = G_1(K_e)$, $\mathsf{AHS} = G_2(\mathsf{dHS}, K_S)$, and $\mathsf{MS} = G_3(\mathsf{dAHS})$. The handshake traffic secrets and the derived handshake secret are computed as $\mathsf{CHTS} = H_1^{(1)}(\mathsf{HS}, T_2)$, $\mathsf{SHTS} = H_1^{(2)}(\mathsf{HS}, T_2)$, and $\mathsf{dHS} = H_1^{(3)}(\mathsf{HS})$. The corresponding authenticated handshake secrets are computed analogously: $\mathsf{CAHTS} = H_2^{(1)}(\mathsf{AHS}, T_3)$, $\mathsf{SAHTS} = H_2^{(2)}(\mathsf{AHS}, T_3)$, and $\mathsf{dAHS} = H_2^{(3)}(\mathsf{AHS})$. Moreover, the Finished keys and application traffic secrets are derived as $\mathsf{fk_c} = H_3^{(1)}(\mathsf{MS})$, $\mathsf{fk_s} = H_3^{(2)}(\mathsf{MS})$, $\mathsf{CATS} = H_3^{(3)}(\mathsf{MS}, T_4)$, and $\mathsf{SATS} = H_3^{(4)}(\mathsf{MS}, T_5)$. Finally, the Finished messages are computed as $\mathtt{CF} = H_{\mathsf{MAC}}(\mathsf{fk_c}, T_3)$ and $\mathtt{SF} = H_{\mathsf{MAC}}(\mathsf{fk_s}, T_4)$. Here, T_i denotes the hash of the cumulative public transcript up to the corresponding protocol step, computed using the hash function H.

5.1 Security Analysis for KEMTLS from MUC-IND-1CCA KEMs

Theorem 5.1 (Multi-Stage Security of KEMTLS from IND-1CCA KEMs). *Let ℓ_{nonce} and μ denote the nonce length and HMAC output length, respectively. Let $\mathcal{K}_e$ denote the shared-key space of KEM_e. Suppose KEM_e and KEM_s are δ_e-correct and δ_s-correct, respectively, the public keys generated by KEM_e have min-entropy at least γ_{pk}, and KEM_s has encapsulated-key min-entropy at least γ. Model the domain-separated HKDF functions as independent public random oracles with n-bit outputs, and the HMAC function used for Finished messages as an additional independent public random oracle $H_{\mathsf{MAC}} \colon \{0,1\}^n \times \{0,1\}^* \to \{0,1\}^\mu$.*

For any polynomial-time adversary $\mathcal{A}$ in the Multi-Stage security game for KEMTLS *(Definition 3.5) with t_u users and at most t_s sessions, making at most q_{G_i}, $q_{1,i}$, $q_{2,i}$, and $q_{3,j}$ explicit queries to G_i, $H_1^{(i)}$, $H_2^{(i)}$, and $H_3^{(j)}$, respectively, for $i \in \{1,2,3\}$ and $j \in \{1,2,3,4\}$, and at most q_{MAC} explicit queries to H_{MAC}, there exist polynomial-time adversaries $\mathcal{B}_1, \ldots, \mathcal{B}_3$ such that*

$$\begin{aligned}\mathsf{Adv}^{\mathsf{MULTI\text{-}STAGE}}_{\mathsf{KEMTLS}}(\mathcal{A}) \leq{}& \frac{t_s^2}{2^{\gamma_{\mathsf{pk}}}} + \frac{t_s^2}{2^{\ell_{\mathsf{nonce}}}} + \frac{t_s^2}{2^{\gamma}} + \frac{t_s}{2^{\mu}} + t_s^2\delta_e + t_u t_s \delta_s \\ &+ \mathsf{Adv}^{\mathsf{COLL}}_{H}(\mathcal{B}_1) + 2\,\mathsf{Adv}^{\mathsf{MUC\text{-}IND\text{-}1CCA}}_{\mathsf{KEM}_e, t_s, t_s}(\mathcal{B}_2) + \mathsf{Adv}^{\mathsf{MUC\text{-}OW\text{-}ChCCA}^*}_{\mathsf{KEM}_s, t_u, t_s}(\mathcal{B}_3) \\ &+ \frac{t_s(q_{G_1} + t_s)}{|\mathcal{K}_e|} + \frac{t_s}{2^n}\left(\sum_{i=2}^{3} q_{G_i} + \sum_{i=1}^{3}(q_{1,i} + q_{2,i}) + \sum_{j=1}^{4} q_{3,j} + q_{\mathsf{MAC}} + 7t_s\right),\end{aligned}$$

where each $\mathcal{B}_i$, for $i \in \{1,2,3\}$, runs in time comparable to that of $\mathcal{A}$.

Proof. Let $\mathcal{A}$ be an adversary against the Multi-Stage security of KEMTLS (Definition 3.5). In the proof, we distinguish two cases according to whether a session has a contributive partner at stage 1. Let Case A denote sessions for which such a partner exists, and Case B denote client sessions without such a partner. In Case A, we use the MUC-IND-1CCA security of the ephemeral KEM to randomize the shared ephemeral KEM secrets K_e. In Case B, we use the MUC-OW-ChCCA* security of the static KEM to randomize their authenticated handshake secrets AHS while their intended servers remain uncorrupted. In both cases, we then analyze the remaining key derivations and Finished messages in the ROM to establish key secrecy and explicit authentication.

Throughout the proof, we simulate the independent random oracles via lazy sampling, and consider the following sequence of games.

Game G_0. This is the original multi-stage security game for KEMTLS, with the following equivalent pre-sampling of ephemeral KEM computations, as in Game G_0 in the proof of Theorem 4.1. At the beginning of the game, we independently generate $(\mathsf{pk}_{e,i}, \mathsf{sk}_{e,i}) \leftarrow\!\!\$\ \mathsf{KEM}_e.\mathsf{Gen}()$ for each $i \in [t_s]$. Moreover, we abort if two pre-sampled public keys coincide, incurring a loss of at most $t_s^2/2^{\gamma_{\mathsf{pk}}}$, since they are independently generated and have min-entropy at least γ_{pk}. For each $i, j \in [t_s]$, we additionally sample $(\mathsf{ct}_{i,j}, K^0_{e,i,j}) \leftarrow\!\!\$\ \mathsf{KEM}_e.\mathsf{Encaps}(\mathsf{pk}_{e,i})$. For the $(i+1)$-th honest client session

label_C^i, we use $(\mathsf{pk}_{e,i}, \mathsf{sk}_{e,i})$ as its ephemeral key pair. Let $\mathsf{label}_S^{i,j}$ denote the $(j+1)$-th honest server session receiving $\mathsf{pk}_{e,i}$. We use $(\mathsf{ct}_{i,j}, K^0_{e,i,j})$ as the encapsulation values in $\mathsf{label}_S^{i,j}$. The rest of the game remains unchanged.

For the following games, let $\mathcal{C}_i := \{\mathsf{ct}_{i,j} \mid j \in [t_s]\}$. For each $c \in \mathcal{C}_i$, let $j_i(c) := \min\{j \in [t_s] \mid \mathsf{ct}_{i,j} = c\}$ be its representative index. Note that pre-sampling does not change the distribution of the original game. Therefore,

$$\mathsf{Adv}^{\mathsf{MULTI\text{-}STAGE}}_{\mathsf{KEMTLS}}(\mathcal{A}) \leq \left|\Pr[G_0^{\mathcal{A}} \Rightarrow 1] - \tfrac{1}{2}\right| + \frac{t_s^2}{2^{\gamma_{\mathsf{pk}}}}.$$

Game G_1. In this game, we abort if an honest session label samples a nonce that has already been sampled by another honest session $\mathsf{label}' \neq \mathsf{label}$. By the birthday bound, the probability of such a collision among at most t_s sessions is at most $\frac{t_s^2}{2^{\ell_{\mathsf{nonce}}}}$. Moreover, we abort if an honestly pre-sampled ephemeral KEM ciphertext $\mathsf{ct}_{i,j}$ or an honestly generated static KEM ciphertext results in a decapsulation error. By the δ_e-correctness of KEM_e and a union bound over t_s^2 ciphertexts, the probability of an ephemeral-KEM decapsulation error is at most $t_s^2\delta_e$. Similarly, by the δ_s-correctness of KEM_s and a union bound over t_u static public keys and at most t_s potential honest encapsulations per user, the probability of a static-KEM decapsulation error is at most $t_u t_s \delta_s$.

We additionally abort if two honest client sessions compute the same static KEM key K_S. Since the KEM has encapsulated-key min-entropy at least γ, this event occurs with probability at most $t_s^2/2^\gamma$. Therefore,

$$\left|\Pr[G_0^{\mathcal{A}} \Rightarrow 1] - \Pr[G_1^{\mathcal{A}} \Rightarrow 1]\right| \leq \frac{t_s^2}{2^{\ell_{\mathsf{nonce}}}} + \frac{t_s^2}{2^\gamma} + t_s^2\delta_e + t_u t_s \delta_s.$$

Game G_2. In this game, we abort if honest sessions compute the same hash value under H on different inputs. This abort event yields a reduction to an adversary $\mathcal{B}_1$ that breaks the collision resistance of H. Therefore,

$$\left|\Pr[G_1^{\mathcal{A}} \Rightarrow 1] - \Pr[G_2^{\mathcal{A}} \Rightarrow 1]\right| \leq \mathsf{Adv}^{\mathsf{COLL}}_H(\mathcal{B}_1).$$

Game G_3 (ephemeral KEM for Case A). In this game, we replace the shared ephemeral KEM secrets associated with the pre-sampled table with uniformly random values. This replacement covers all sessions in Case A. Specifically, for each $i \in [t_s]$ and each distinct $c \in \mathcal{C}_i$, we first sample an independent $\widehat{K}_{e,i,c} \leftarrow\!\!\$\ \mathcal{K}_e$. Then for every server session $\mathsf{label}_S^{i,j}$, its KEM secret is replaced with $\widehat{K}_{e,i,\mathsf{ct}_{i,j}}$. If client session label_C^i receives a ciphertext $c \in \mathcal{C}_i$, its KEM secret is also replaced with $\widehat{K}_{e,i,c}$. For any session in which K_e is replaced, all subsequent secret values that would have been derived from K_e are instead derived from its replacement. The rest of the game remains unchanged.

Any adversary $\mathcal{A}$ that can distinguish between Game G_2 and Game G_3 can be used to construct an adversary $\mathcal{B}_2$ against the $\mathsf{MUC\text{-}IND\text{-}1CCA}$ security of KEM_e as follows. The adversary $\mathcal{B}_2$ receives a challenge tuple $(\mathbf{pk}^*, \mathbf{ct}^*, \mathbf{K}^*_{\hat{b}})$ from its challenger. It defines $\mathcal{C}_i$ and $j_i(c)$ using its challenge ciphertext matrix. For each client session label_C^i, $\mathcal{B}_2$ embeds pk_i^* as its ephemeral public key. For each server session $\mathsf{label}_S^{i,j}$, it embeds $\mathsf{ct}_{i,j}^*$ into the corresponding $\mathtt{SH}$ message and sets the shared KEM secret to $K^*_{i,j_i(\mathsf{ct}^*_{i,j}),\hat{b}}$. If label_C^i later receives any $c \in \mathcal{C}_i$, $\mathcal{B}_2$ sets its KEM secret to $K^*_{i,j_i(c),\hat{b}}$, regardless of whether c has been sent by an honest server. If $\mathcal{A}$ instead delivers $c \notin \mathcal{C}_i$ to label_C^i, then $\mathcal{B}_2$ queries its IND-1CCA decapsulation oracle $\mathsf{O}^{\mathsf{Dec}}(i, c)$ to simulate label_C^i. Since each client session performs at most one decapsulation, $\mathcal{B}_2$ makes at most one decapsulation query for each challenge public key. Adversary $\mathcal{B}_2$ locally generates the static KEM key pair for each party and simulates the rest of the game as in Game G_2.

By Game G_1, there are no decapsulation errors for the pre-sampled ephemeral KEM ciphertexts on non-aborting executions. Consequently, all pre-sampled entries with the same public key and

ciphertext have the same real KEM key, which also equals the key obtained by the corresponding client session through decapsulation in Game G_2. Thus, when $\hat{b} = 0$, the challenge keys correspond to real KEM keys, and $\mathcal{B}_2$ perfectly simulates Game G_2. When $\hat{b} = 1$, the challenge keys are independent uniformly random values, and $\mathcal{B}_2$ also perfectly simulates Game G_3. Therefore,

$$\left|\Pr[G_2^{\mathcal{A}} \Rightarrow 1] - \Pr[G_3^{\mathcal{A}} \Rightarrow 1]\right| \leq 2\,\mathsf{Adv}^{\mathsf{MUC\text{-}IND\text{-}1CCA}}_{\mathsf{KEM}_e, t_s, t_s}(\mathcal{B}_2).$$

Next, we analyze all sessions that have no honest contributive partner at stage 1 at the end of the experiment (Case B). In this case, a session satisfying Freshness-KEMTLS or malicious acceptance must be a client session whose intended peer remains uncorrupted before it accepts at stage 6. We first use the MUC-OW-ChCCA* security of the static KEM in the ROM to show that their authenticated handshake secrets AHS remain private random-oracle values while their intended servers remain uncorrupted.

Game G_4. In this game, we reformulate the game using the challenge public keys **pk** and the oracles Enc, Dec, Corr, and Check of the (t_u, t_s)-MUC-OW-ChCCA* experiment, by modeling G_2 as a random oracle. Specifically, for each $i \in [t_u]$, the challenger embeds $\mathbf{pk}[i]$ as user i's long-term KEM public key and uses its Corr oracle to answer the adversary's Corrupt queries. Define $\mathcal{L}_{\mathsf{ENC}} = \emptyset$. Whenever an honest client session needs to perform a static encapsulation under user i's certified public key, the challenger queries $\mathsf{Enc}(i)$ to obtain a ciphertext c and includes c in the session's CKC message. We say that such a client session and any server session of user i receiving the ciphertext c returned by $\mathsf{Enc}(i)$ correspond to pair (i, c). We add (i, c) to the table $\mathcal{L}_{\mathsf{ENC}}$, i.e., $\mathcal{L}_{\mathsf{ENC}} \leftarrow \mathcal{L}_{\mathsf{ENC}} \cup \{(i, c)\}$. Note that the challenger makes at most t_s queries to Enc. If a server session of user i receives a static ciphertext c such that $(i, c) \notin \mathcal{L}_{\mathsf{ENC}}$, the challenger queries $\mathsf{Dec}(i, c)$ to obtain the decapsulation result K_S and simulates the session as in Game G_3.

For each session corresponding to $(i, c) \in \mathcal{L}_{\mathsf{ENC}}$, the challenger simulates its $\mathsf{AHS} = G_2(\mathsf{dHS}, K_S)$ without knowing the corresponding K_S, as follows. The challenger lazily simulates G_2 using an initially empty list $\mathcal{L}_{G_2}$ of tuples $(\mathsf{dHS}, K, \mathsf{AHS})$ and maintains an initially empty auxiliary list $\mathcal{L}_D$ of tuples $(i, c, \mathsf{dHS}, \mathsf{AHS})$ for $(i, c) \in \mathcal{L}_{\mathsf{ENC}}$. Consider an honest session corresponding to (i, c), and let dHS denote its derived handshake secret. To compute AHS, the challenger first checks whether there exists an entry $(i, c, \mathsf{dHS}, \mathsf{AHS}) \in \mathcal{L}_D$. If so, it uses the stored AHS. Otherwise, it checks whether there is an entry $(\mathsf{dHS}, K, \mathsf{AHS}) \in \mathcal{L}_{G_2}$ such that $\mathsf{Check}(i, c, K) = 1$. If such an entry exists, the challenger uses its stored output as AHS for that session. Otherwise, it samples a uniformly random $\widehat{\mathsf{AHS}}$, uses it as the session's AHS, and sets $\mathcal{L}_D \leftarrow \mathcal{L}_D \cup \{(i, c, \mathsf{dHS}, \widehat{\mathsf{AHS}})\}$.

To ensure consistency between the random oracle G_2 and $\mathcal{L}_D$, whenever G_2 is queried on (dHS, K_S) (including implicit queries during protocol computations), the challenger first checks whether $\mathcal{L}_{G_2}$ contains an entry for (dHS, K_S). If so, it returns the stored AHS. Otherwise, it checks whether there exists an entry $(i, c, \mathsf{dHS}, \mathsf{AHS}) \in \mathcal{L}_D$ such that $\mathsf{Check}(i, c, K_S) = 1$. If a matching entry exists, it uses the stored AHS. Otherwise, it samples a uniformly random AHS. It then updates $\mathcal{L}_{G_2} \leftarrow \mathcal{L}_{G_2} \cup \{(\mathsf{dHS}, K_S, \mathsf{AHS})\}$ and returns AHS. All remaining parts of the game are simulated as in Game G_3.

By Game G_1, all honestly generated static KEM ciphertexts decapsulate correctly, and distinct honest static encapsulations yield distinct KEM keys. Therefore, sessions corresponding to registered pairs (i, c) have the same input (dHS, K_S) to G_2 if and only if their (i, c, dHS) values are the same. The Check oracle ensures that entries in $\mathcal{L}_{G_2}$ and $\mathcal{L}_D$ corresponding to the same G_2 input use the same output. Moreover, sessions with the same (i, c, dHS) use the same AHS, while distinct G_2 inputs yield independent uniformly random AHS values. Hence, the above simulation preserves $\mathsf{AHS} = G_2(\mathsf{dHS}, K_S)$ and the simulated game has the same distribution as Game G_3. Thus,

$$\Pr[G_3^{\mathcal{A}} \Rightarrow 1] = \Pr[G_4^{\mathcal{A}} \Rightarrow 1].$$

Game G_5 (Abort on QUERY). In this game, we abort if the event QUERY defined below occurs. Let QUERY denote the event that the adversary has explicitly queried G_2 on (dHS, K_S) such that

$\mathsf{Decaps}(\mathsf{sk}_i, c) = K_S$ for some $(i, c) \in \mathcal{L}_{\mathsf{ENC}}$, while user i is still uncorrupted. Here, sk_i denotes user i's long-term KEM secret key corresponding to $\mathbf{pk}[i]$. Since Games G_4 and G_5 are the same unless QUERY occurs, by the difference lemma, we have $|\Pr[G_4^{\mathcal{A}} \Rightarrow 1] - \Pr[G_5^{\mathcal{A}} \Rightarrow 1]| \leq \Pr[\mathsf{QUERY}]$.

To bound $\Pr[\mathsf{QUERY}]$, we construct an adversary $\mathcal{B}_3$ against the $\mathsf{MUC\text{-}OW\text{-}ChCCA}^*$ security of KEM_s as follows. $\mathcal{B}_3$ simulates Game G_4 and records the adversary's explicit G_2 query inputs. Let $\mathcal{C}$ denote the set of currently corrupted users. Before answering each explicit query $G_2(\mathsf{dHS}, K_S)$ from the adversary, it evaluates $\mathsf{Check}(i, c, K_S)$ for every $(i, c) \in \mathcal{L}_{\mathsf{ENC}}$ with $i \notin \mathcal{C}$. Whenever a new challenge (i, c) with $i \notin \mathcal{C}$ is registered, it also evaluates $\mathsf{Check}(i, c, K_S)$ for every previously recorded explicit query $G_2(\mathsf{dHS}, K_S)$. At the first successful check (if any), i.e., when $\mathsf{Check}(i, c, K_S) = 1$, $\mathcal{B}_3$ obtains the corresponding tuple (i, c, K_S) and halts the game, returning (i, c, K_S) as its solution. If QUERY occurs, such a tuple exists. By Game G_1, K_S is the corresponding encapsulated key recorded by $\mathcal{B}_3$'s challenger, and user i is still uncorrupted. Therefore, (i, c, K_S) is a valid solution to the $\mathsf{MUC\text{-}OW\text{-}ChCCA}^*$ game. Thus, $\mathcal{B}_3$ succeeds whenever QUERY occurs, and we have

$$\left|\Pr[G_4^{\mathcal{A}} \Rightarrow 1] - \Pr[G_5^{\mathcal{A}} \Rightarrow 1]\right| \leq \Pr[\mathsf{QUERY}] \leq \mathsf{Adv}^{\mathsf{MUC\text{-}OW\text{-}ChCCA}^*}_{\mathsf{KEM}_s, t_u, t_s}(\mathcal{B}_3).$$

After Game G_5, we have that all sessions that have an honest contributive partner at stage 1 at the end of the experiment (Case A) have uniformly random ephemeral KEM secrets K_e. For client sessions that do not have such a partner (Case B), the AHS values corresponding to registered challenges in $\mathcal{L}_{\mathsf{ENC}}$ remain private random-oracle values while their intended servers remain uncorrupted. Next, we show that the corresponding fresh stage keys and MAC keys in these sessions are also derived at random-oracle inputs that the adversary has not explicitly queried. In Games G_6–G_{11}, earlier explicit queries are also checked whenever the relevant secret is derived or a session first meets the respective abort condition.

Game G_6 (abort on G_1 queries for Case A). In this game, we abort if the adversary has explicitly queried the random oracle G_1 on the randomized ephemeral KEM secret $\widehat{K}_{e,i,c}$ used by a session in Case A.

Note that neither the Reveal oracle nor the $\mathsf{Corrupt}$ oracle directly reveals these randomized secrets $\widehat{K}_{e,i,c}$. In particular, Reveal returns only stage keys derived from the corresponding handshake secrets via subsequent random oracles, whereas $\mathsf{Corrupt}$ returns only long-term secret keys. However, the adversary may control the ephemeral KEM secrets used by unreplaced sessions (i.e., sessions whose ephemeral KEM secrets were not replaced in Game G_3) and exploit the revealed stage keys and further random-oracle queries to test whether these secrets coincide with $\widehat{K}_{e,i,c}$, without explicitly querying G_1 on these randomized secrets.

Let $\mathsf{Bad}_{\mathsf{col}}$ denote the event that an ephemeral KEM secret used by an unreplaced session coincides with some randomized secret $\widehat{K}_{e,i,c}$ before the adversary explicitly queries G_1 on any such randomized secret. Since each $\widehat{K}_{e,i,c}$ is sampled independently and uniformly from $\mathcal{K}_e$, and remains hidden before the first matching explicit query, each such coincidence occurs with probability at most $1/|\mathcal{K}_e|$. By a union bound over at most t_s randomized secrets and at most t_s unreplaced sessions, we have $\Pr[\mathsf{Bad}_{\mathsf{col}}] \leq \frac{t_s^2}{|\mathcal{K}_e|}$. Assuming that $\mathsf{Bad}_{\mathsf{col}}$ does not occur, the implicit G_1 evaluations of unreplaced sessions are independent of each $G_1(\widehat{K}_{e,i,c})$ and do not help the adversary guess $\widehat{K}_{e,i,c}$. Hence, the probability that the adversary explicitly queries G_1 on some randomized secret $\widehat{K}_{e,i,c}$ is at most $t_s q_{G_1}/|\mathcal{K}_e|$. Combining the above analyses, we obtain

$$\left|\Pr[G_5^{\mathcal{A}} \Rightarrow 1] - \Pr[G_6^{\mathcal{A}} \Rightarrow 1]\right| \leq \frac{t_s(q_{G_1} + t_s)}{|\mathcal{K}_e|}.$$

Game G_7 (abort on $H_1^{(r)}$ queries for Case A). In this game, we abort if the adversary has explicitly queried some random oracle $H_1^{(r)}$, for $r \in \{1, 2, 3\}$, on an input whose first argument is the HS used by a session in Case A.

By Game G_6, the corresponding $\mathsf{HS} = G_1(\widehat{K}_{e,i,c})$ values have not been explicitly queried through their G_1 inputs and are therefore uniformly random n-bit values. As in the analysis of Game G_6, we consider collisions between a randomized $\mathsf{HS} = G_1(\widehat{K}_{e,i,c})$ value and an HS value used by another session and derived from a distinct G_1 input. The probability of such a collision occurring before any matching explicit $H_1^{(r)}$ query is at most $t_s^2/2^n$. Similarly, the probability of an explicit $H_1^{(r)}$ query on a randomized HS value before any such collision is at most $\frac{t_s}{2^n}\sum_{r=1}^{3} q_{1,r}$. Therefore, we have

$$\left|\Pr[G_6^{\mathcal{A}} \Rightarrow 1] - \Pr[G_7^{\mathcal{A}} \Rightarrow 1]\right| \leq \frac{t_s}{2^n}\left(\sum_{r=1}^{3} q_{1,r} + t_s\right).$$

Game G_8 (abort on G_2 queries for Case A). In this game, we abort if the adversary has explicitly queried the random oracle G_2 on an input whose first argument is the dHS used by a session in Case A. By Game G_7, the corresponding dHS values are outputs of $H_1^{(3)}$ at inputs that have not been explicitly queried by the adversary. By an argument analogous to that in Game G_7, with at most t_s session-derived values and q_{G_2} explicit queries, we have

$$\left|\Pr[G_7^{\mathcal{A}} \Rightarrow 1] - \Pr[G_8^{\mathcal{A}} \Rightarrow 1]\right| \leq \frac{t_s(q_{G_2} + t_s)}{2^n}.$$

Thus, after Game G_8, the corresponding AHS values for sessions in Case A and in Case B (as long as the intended server remains uncorrupted) remain private random-oracle outputs whose inputs have not been explicitly queried by the adversary. We next show that, except with negligible probability, the adversary also never queries the random oracles on the inputs used to derive the stage keys or MAC keys of these sessions.

Game G_9 (abort on $H_2^{(r)}$ queries for Cases A and B). In this game, we abort if the adversary has explicitly queried some random oracle $H_2^{(r)}$, for $r \in \{1,2,3\}$, on an input whose first argument is the AHS used by either a session in Case A or a client session in Case B whose intended server is currently uncorrupted. By Games G_5 and G_8, the corresponding G_2 inputs have not been explicitly queried by the adversary. By an argument analogous to that in Game G_7, with at most t_s sessions and $\sum_{r=1}^{3} q_{2,r}$ explicit queries, we obtain

$$\left|\Pr[G_8^{\mathcal{A}} \Rightarrow 1] - \Pr[G_9^{\mathcal{A}} \Rightarrow 1]\right| \leq \frac{t_s}{2^n}\left(\sum_{r=1}^{3} q_{2,r} + t_s\right).$$

Game G_{10} (abort on G_3 queries for Cases A and B). In this game, we abort if the adversary has explicitly queried the random oracle G_3 on the dAHS used by either a session in Case A or a client session in Case B whose intended server is currently uncorrupted. By Game G_9, the corresponding dAHS values are outputs of $H_2^{(3)}$ at inputs that have not been explicitly queried by the adversary. By an argument analogous to that in Game G_7, with at most t_s sessions and q_{G_3} explicit queries, we obtain

$$\left|\Pr[G_9^{\mathcal{A}} \Rightarrow 1] - \Pr[G_{10}^{\mathcal{A}} \Rightarrow 1]\right| \leq \frac{t_s(q_{G_3} + t_s)}{2^n}.$$

Game G_{11} (abort on $H_3^{(r)}$ queries for Cases A and B). In this game, we abort if the adversary has explicitly queried some random oracle $H_3^{(r)}$, for $r \in \{1,2,3,4\}$, on an input whose first argument is the MS used by either a session in Case A or a client session in Case B whose intended server is currently uncorrupted. By Game G_{10}, the corresponding MS values are outputs of G_3 at inputs that have not been explicitly queried by the adversary. By an argument analogous to that in Game G_7, with at most t_s sessions and $\sum_{r=1}^{4} q_{3,r}$ explicit queries, we obtain

$$\left|\Pr[G_{10}^{\mathcal{A}} \Rightarrow 1] - \Pr[G_{11}^{\mathcal{A}} \Rightarrow 1]\right| \leq \frac{t_s}{2^n}\left(\sum_{r=1}^{4} q_{3,r} + t_s\right).$$

Game G_{12} (explicit authentication). In this game, we abort if a client session accepts at stage 6 without a partner at that stage and its intended server has not been corrupted before that acceptance. By Game G_{11}, for either a session in Case A or a client session in Case B whose intended server is currently uncorrupted, the corresponding server Finished key $\mathsf{fk_s}$ remains a private, uniformly sampled random-oracle output. We also account for collisions between Finished keys derived from distinct inputs or by different Finished-key oracles. Each session derives at most two Finished keys, so there are at most $2t_s$ protocol-derived candidate keys. By an argument analogous to that in Game G_7, the probability of such a collision or an explicit H_{MAC} query using one of the at most t_s private server Finished keys as its first argument is at most $t_s(q_{\mathsf{MAC}} + 2t_s)/2^n$. Suppose neither event occurs.

Recall that honest servers compute their Finished tags as $H_{\mathsf{MAC}}(\mathsf{fk_s}, T_4)$. Since the client has no partner at stage 6 and nonce and hash collisions are excluded by Games G_1 and G_2, no honest server has computed the required tag. Client Finished tags use T_3 and thus correspond to different oracle inputs. Therefore, the required tag is a fresh uniformly random μ-bit value from the adversary's perspective. If the abort event occurs, then the adversary has successfully forged a valid tag, which occurs with probability at most $t_s/2^\mu$ by a union bound over at most t_s sessions. Hence,

$$\left|\Pr[G_{11}^{\mathcal{A}} \Rightarrow 1] - \Pr[G_{12}^{\mathcal{A}} \Rightarrow 1]\right| \leq \frac{t_s(q_{\mathsf{MAC}} + 2t_s)}{2^n} + \frac{t_s}{2^\mu}.$$

Analysis of Game G_{12}. In this game, consider a tested stage that satisfies Freshness-KEMTLS at the end of the experiment. If the session has a contributive partner at stage 1 (Case A), Game G_7 excludes explicit queries to the random-oracle inputs used to derive its keys at stages 1–2. These keys are therefore uniformly random. Moreover, by Games G_9–G_{11}, the adversary has not explicitly queried the random oracles on the inputs used to derive its keys at stages 3–6. These keys are therefore also uniformly random from the adversary's perspective.

If the session has no contributive partner at stage 1, Freshness-KEMTLS requires it to be a client session, and thus this session corresponds to Case B. Before its stage 6 acceptance, only stages 3–5 can satisfy Freshness-KEMTLS, and the intended server must remain uncorrupted. Similarly, by Games G_9–G_{11}, the adversary has not explicitly queried the random oracles on the inputs used to derive these stage keys, and thus these keys are also uniformly random from the adversary's perspective. If such a client session accepts at stage 6, Freshness-KEMTLS requires its intended server to remain uncorrupted until that acceptance. Since the session has no partner at that stage, Game G_{12} directly aborts.

In conclusion, all fresh tested stage keys are uniformly random from the adversary's perspective. By domain separation and Games G_1 and G_2, these keys are sampled independently and uniformly, except that sessions partnered at the corresponding stage share the same stage-key random-oracle input and hence use the same stage key. Freshness-KEMTLS also excludes revealing the tested key or any partner's corresponding key. Hence, the hidden challenge bit b is independent of the adversary's view. A client's partner at stage 6 is also its partner at every preceding stage, and these partners are unique by Game G_1. Moreover, in Game G_{12}, the game aborts if any client session maliciously accepts. Note that server sessions do not require explicit client authentication in this proof. Therefore, we have

$$\Pr[G_{12}^{\mathcal{A}} \Rightarrow 1] = \tfrac{1}{2}.$$

Combining the above bounds completes the proof. □

5.2 Security Analysis for KEMTLS from MUC-IND-1RCCA KEMs

Theorem 5.2 (Multi-Stage Security of KEMTLS from IND-1RCCA KEMs). *Let ℓ_{nonce} and μ denote the nonce length and HMAC output length, respectively. Let $\mathcal{K}_e$ denote the shared-key space of KEM_e. Suppose KEM_e and KEM_s are δ_e-correct and δ_s-correct, respectively, the public keys*

generated by KEM_e *have min-entropy at least* γ_{pk}*, and* KEM_s *has encapsulated-key min-entropy at least* γ*. Model the domain-separated HKDF functions as independent public random oracles with* n*-bit outputs, and the HMAC function used for Finished messages as an additional independent public random oracle* $H_{\mathsf{MAC}}\colon \{0,1\}^n \times \{0,1\}^* \to \{0,1\}^\mu$*.*

For any polynomial-time adversary $\mathcal{A}$ *in the Multi-Stage security game for* KEMTLS *(Definition 3.5) with* t_u *users and at most* t_s *sessions, making at most* q_{G_i}*,* $q_{1,i}$*,* $q_{2,i}$*, and* $q_{3,j}$ *explicit queries to* G_i*,* $H_1^{(i)}$*,* $H_2^{(i)}$*, and* $H_3^{(j)}$*, respectively, for* $i \in \{1,2,3\}$ *and* $j \in \{1,2,3,4\}$*, and at most* q_{MAC} *explicit queries to* H_{MAC}*, there exist polynomial-time adversaries* $\mathcal{B}_1, \ldots, \mathcal{B}_3$ *such that*

$$\begin{aligned}\mathsf{Adv}^{\mathsf{MULTI\text{-}STAGE}}_{\mathsf{KEMTLS}}(\mathcal{A}) \leq{}& \frac{t_s^2}{2^{\gamma_{\mathsf{pk}}}} + \frac{t_s^2}{2^{\ell_{\mathsf{nonce}}}} + \frac{t_s^2}{2^{\gamma}} + \frac{t_s}{2^{\mu}} + t_s^2\delta_e + t_u t_s \delta_s \\ &+ \mathsf{Adv}^{\mathsf{COLL}}_{H}(\mathcal{B}_1) + 2\,\mathsf{Adv}^{\mathsf{MUC\text{-}IND\text{-}1RCCA}}_{\mathsf{KEM}_e,\, t_s,\, t_s}(\mathcal{B}_2) + \mathsf{Adv}^{\mathsf{MUC\text{-}OW\text{-}ChCCA}^*}_{\mathsf{KEM}_s,\, t_u,\, t_s}(\mathcal{B}_3) \\ &+ \frac{t_s(q_{G_1} + 2t_s)}{|\mathcal{K}_e|} + \frac{t_s}{2^n}\left(\sum_{i=2}^{3} q_{G_i} + \sum_{i=1}^{3}(q_{1,i} + q_{2,i}) + \sum_{j=1}^{4} q_{3,j} + q_{\mathsf{MAC}} + 7t_s\right),\end{aligned}$$

where each $\mathcal{B}_i$*, for* $i \in \{1,2,3\}$*, runs in time comparable to that of* $\mathcal{A}$*.*

Proof. Let $\mathcal{A}$ be an adversary against the Multi-Stage security of KEMTLS (Definition 3.5). We follow the proof of Theorem 5.1, using the same case distinction. Let Case A denote sessions with a contributive partner at stage 1, and Case B denote client sessions without such a partner. All game transitions and their analyses remain the same, except for the transition from Game G_2 to Game G_3. In this transition, for Case A, we rely on the MUC-IND-1RCCA security of the ephemeral KEM to randomize the shared ephemeral KEM secrets K_e, as in the proof of Theorem 4.2.

Games G_0–G_2. These games are identical to Games G_0–G_2 in the proof of Theorem 5.1. Specifically, we pre-sample the ephemeral KEM table and exclude the relevant collisions and decapsulation errors in both the ephemeral and static KEMs. We also reuse the notation for the pre-sampled ephemeral KEM table, including $\mathcal{C}_i$ and $j_i(c)$.

Game G_3 (ephemeral KEM for Case A). In this game, we replace the shared ephemeral KEM secrets associated with the pre-sampled table with uniformly random values. This replacement covers all sessions in Case A. This replacement also applies to client sessions receiving replayable ciphertexts outside the pre-sampled table, as in Game G_4 in the proof of Theorem 4.2.

Specifically, for each $i, j \in [t_s]$, we sample an independent $\widehat{K}_{e,i,j} \leftarrow\!\!\$\, \mathcal{K}_e$. For every server session $\mathsf{label}_S^{i,j}$, its ephemeral KEM secret is replaced with $\widehat{K}_{e,i,j_i(\mathsf{ct}_{i,j})}$. If client session label_C^i receives any $c \in \mathcal{C}_i$, its ephemeral KEM secret is also replaced with $\widehat{K}_{e,i,j_i(c)}$. Note that this includes ciphertexts from table entries that have not been used by any honest server session. The main difference from Game G_3 in the proof of Theorem 5.1 is as follows. If label_C^i instead receives $c \notin \mathcal{C}_i$, we first compute $K' \leftarrow \mathsf{KEM}_e.\mathsf{Decaps}(\mathsf{sk}_{e,i}, c)$. If $K' = K^0_{e,i,j}$ for some $j \in [t_s]$, we set $j^* := \min\{j \in [t_s] \mid K' = K^0_{e,i,j}\}$ and further replace the client's ephemeral KEM secret with $\widehat{K}_{e,i,j^*}$. Otherwise, the client continues to use K'. For any session in which K_e is replaced, all subsequent secret values derived from K_e are instead derived from its replacement. The rest of the game remains unchanged.

Any adversary $\mathcal{A}$ that distinguishes Game G_2 from Game G_3 can be used to construct an adversary $\mathcal{B}_2$ against the $\mathsf{MUC\text{-}IND\text{-}1RCCA}$ security of KEM_e. The reduction follows the analysis of Game G_4 in the proof of Theorem 4.2, with the static KEM keys generated locally. Therefore,

$$\left|\Pr[G_2^{\mathcal{A}} \Rightarrow 1] - \Pr[G_3^{\mathcal{A}} \Rightarrow 1]\right| \leq 2\,\mathsf{Adv}^{\mathsf{MUC\text{-}IND\text{-}1RCCA}}_{\mathsf{KEM}_e,\, t_s,\, t_s}(\mathcal{B}_2) + \frac{t_s^2}{|\mathcal{K}_e|}.$$

Games G_4–G_{12}. These games make the same changes as Games G_4–G_{12} in the proof of Theorem 5.1. The same analysis and bounds apply. There are at most t_s randomized ephemeral secrets

used by sessions, including those assigned to non-table replayable ciphertexts. The index for such a ciphertext is selected using the original KEM keys, independently of an unused replacement. Specifically, Games G_4–G_5 handle the static KEM for Case B. Games G_6–G_8 exclude explicit queries to the hidden random-oracle inputs used to derive HS, CHTS, SHTS, dHS, and AHS for sessions in Case A. Games G_9–G_{11} exclude explicit queries to the hidden random-oracle inputs used in the remaining key derivations for sessions in Case A and client sessions in Case B whose intended servers remain uncorrupted. Game G_{12} then aborts if a client accepts at stage 6 without a partner at that stage and its intended server has not been corrupted before that acceptance.

Analysis of Game G_{12}. As in Game G_{12} in the proof of Theorem 5.1, all fresh tested stage keys are uniformly random from the adversary's perspective. By domain separation and Games G_1 and G_2, these keys are sampled independently and uniformly, except that sessions partnered at the corresponding stage share the same stage-key random-oracle input and hence use the same stage key. Freshness-KEMTLS excludes revealing the tested key or any partner's corresponding key. Hence, the hidden challenge bit b is independent of the adversary's view. Moreover, Game G_{12} aborts if any client session maliciously accepts. Therefore,

$$\Pr[G_{12}^{\mathcal{A}} \Rightarrow 1] = \tfrac{1}{2}.$$

Combining the above bounds completes the proof. □

6 Concrete MU-(dual)-PRF Security Analysis of HMAC and HKDF

Bellare [Bel15] established PRF-security results for HMAC under suitable assumptions on the compression function underlying the hash function H. Diemert and Jager [DJ21] provided a concrete multi-user (MU-PRF) security bound for HMAC in the ROM. Backendal et al. [BBGS23] analyzed its dual-PRF security in the standard model and established multi-user bounds under appropriate domain restrictions. In this section, we give concrete MU-PRF security bounds for HMAC, HKDF.Extract, and HKDF.Expand in the QROM. We also establish concrete bounds for the MU-dual-PRF security of HMAC and HKDF.Extract in the ROM and QROM.

6.1 (Quantum-Accessible) Random Oracle Model

We refer the reader to [NC00] for foundational concepts in quantum computation and quantum information. The Random Oracle Model (ROM) [BR93] is an idealized framework wherein a hash function is modeled as a publicly accessible uniformly random function. In the quantum setting, an adversary may evaluate the hash function on superpositions of inputs. Consequently, the Quantum-Accessible Random Oracle Model (QROM) extends the ROM by allowing quantum adversaries to query the random oracle using quantum states [BDF+11]. Establishing the post-quantum security of a cryptographic construction analyzed in the random oracle paradigm requires proving its security in the QROM [BDF+11].

Below, we introduce several technical lemmas that will be used throughout this paper.

Lemma 6.1 (Simulating the Random Oracle [Zha12, Theorem 6.1]). *Let H be sampled uniformly from a $2q$-wise independent family of functions. Then any quantum algorithm making at most q queries to H has zero advantage in distinguishing H from a truly random function with the same domain and range.*

Lemma 6.2 (Quantum Collision Finding [Zha19, Theorem 2 and Lemma 5]). *Let $H : \mathcal{X} \to \{0,1\}^{\mu}$ be modeled as a quantum-accessible random oracle. Let $\mathcal{A}$ be a quantum algorithm*

making at most q queries to H and outputting $(u, v, y) \in \mathcal{X}^2 \times \{0,1\}^\mu$. Then,

$$\Pr\Big[u \neq v \wedge H(u) = H(v) = y : \ (u, v, y) \leftarrow \mathcal{A}^{|H\rangle}\Big] \leq \frac{8(q+1)^3}{2^\mu}.$$

Lemma 6.3 (One-Way to Hiding (OW2H) [AHU19, Theorem 3]). *Let $S \subseteq X$ be a random subset. Let G and H be oracles such that for all $x \notin S$, $G(x) = H(x)$. Let z be a random bitstring (note that S, G, H, and z may be arbitrarily correlated). Let $\mathcal{A}$ be a quantum oracle algorithm making at most q (not necessarily unitary) queries. Define an oracle algorithm $\mathcal{B}^{|H\rangle}$ which, on input z, proceeds as follows: choose $i \in [q]$, run $\mathcal{A}^{|H\rangle}(z)$ up to (but not including) the $(i+1)$-th query, measure all query input registers in the computational basis, and output the resulting set T of queried inputs. Then:*

$$\Big|\Pr[1 \leftarrow \mathcal{A}^{|H\rangle}(z)] - \Pr[1 \leftarrow \mathcal{A}^{|G\rangle}(z)]\Big| \leq 2q \cdot \sqrt{\Pr[S \cap T \neq \varnothing : T \leftarrow \mathcal{B}^{|H\rangle}(z)]}.$$

The lemma also holds when $\mathcal{A}$ and $\mathcal{B}$ have access to auxiliary oracles jointly distributed with (S, G, H, z), provided that these oracles remain unchanged between the two executions [BHH+19, Section 2]. Queries to these auxiliary oracles are not counted in q.

6.2 MU-PRF Security of HMAC in the QROM

We first analyze the MU-PRF security of HMAC in the QROM.

Theorem 6.4. *Let $\kappa, \mu \in \mathbb{N}$ with $\kappa \leq 8B$[14], where B is the byte length of `ipad` and `opad`. Let H be modeled as a quantum-accessible random oracle with output length μ. Suppose $\mathcal{A}$ is a QPT adversary against the* MU-PRF *security of* HMAC *for N users, making at most q_H quantum queries to H and q_F classical queries to the PRF oracle. Then we have*

$$\mathsf{Adv}^{\mathsf{MU\text{-}PRF}}_{\mathsf{HMAC},N}(\mathcal{A}) \leq \frac{N^2}{2^\kappa} + \frac{8(q_H + 2q_F + 1)^3}{2^\mu} + 2q_H\sqrt{\frac{N}{2^\kappa}}.$$

Proof. Let $\mathcal{A}$ be a QPT adversary against the MU-PRF security game for HMAC, making at most q_H quantum queries to H and q_F classical queries to the PRF oracle. Throughout, keys are implicitly zero-padded to $8B$ bits before XOR with `ipad` or `opad`. Consider the following sequence of games.

Game G_0. This is the original MU-PRF game for HMAC as defined in Definition 2.7. Therefore,

$$\Big|\Pr[G_0^{\mathcal{A}} \Rightarrow 1] - \tfrac{1}{2}\Big| = \mathsf{Adv}^{\mathsf{MU\text{-}PRF}}_{\mathsf{HMAC},N}(\mathcal{A}).$$

Game G_1. In this game, we ensure that users have distinct PRF secret keys and that no inner-hash prefix coincides with an outer-hash prefix. Specifically, the game aborts if there exist indices $i \neq j$ such that $k_i = k_j$ or $k_i \oplus \texttt{ipad} = k_j \oplus \texttt{opad}$. We denote this event by $\mathsf{abort}_{\mathsf{key}}$. For each pair of users $i < j$, the probability that at least one of the two conditions holds is at most $2/2^\kappa$. By a union bound, we have $\Pr[\mathsf{abort}_{\mathsf{key}}] \leq 2\binom{N}{2}/2^\kappa \leq N^2/2^\kappa$. Therefore,

$$\Big|\Pr\big[G_0^{\mathcal{A}} \Rightarrow 1\big] - \Pr\big[G_1^{\mathcal{A}} \Rightarrow 1\big]\Big| \leq \frac{N^2}{2^\kappa}.$$

Game G_2. In this game, we introduce an abort condition if there exist two distinct queries (i, x) and (i, x') to the PRF oracle with $x \neq x'$ such that $H((k_i \oplus \texttt{ipad}) \parallel x) = H((k_i \oplus \texttt{ipad}) \parallel x')$, which implies a collision in the random oracle H. Since $\mathcal{A}$ makes at most q_H explicit queries to H, and each of the at most q_F PRF queries induces at most two evaluations of H, the total number of

[14] This restriction covers both TLS 1.3 and KEMTLS instantiations: for SHA-256 and SHA-384, we have $\kappa = \mu = 256$ and $\kappa = \mu = 384$, respectively.

queries to H is at most $q_H + 2q_F$. By Lemma 6.2, the probability of such a collision is bounded by $\frac{8(q_H+2q_F+1)^3}{2^\mu}$. Hence,

$$\left|\Pr[G_1^{\mathcal{A}} \Rightarrow 1] - \Pr[G_2^{\mathcal{A}} \Rightarrow 1]\right| \leq \frac{8(q_H + 2q_F + 1)^3}{2^\mu}.$$

Notably, Games G_1 and G_2 ensure that each fresh query (i, x) to the real PRF oracle uses outer-hash inputs that have not been used by any previous PRF query (including its outer- and inner-hash inputs). For the inner-hash inputs, Game G_1 ensures that distinct users have distinct keys, so distinct pairs (i, x) give distinct inputs $(k_i \oplus \texttt{ipad}) \parallel x$. For the outer-hash inputs, different users have distinct prefixes $k_i \oplus \texttt{opad}$, while different inputs queried for the same user have distinct inner-hash values by Game G_2, and hence distinct suffixes. Game G_1 further excludes any overlap between the inner- and outer-hash input domains.

It remains to decouple the adversary's access to the random oracle from the PRF oracle. This is achieved in Game G_3.

Game G_3. In this game, we replace the random oracle H accessible to the adversary with a new oracle H' defined as follows. Let $S := \{(k_i \oplus \texttt{opad}) \parallel y \mid i \in [N], y \in \{0,1\}^\mu\}$. Let $H'' \colon \{0,1\}^{8B+\mu} \to \{0,1\}^\mu$ be an independent random oracle. Define $H'(u) = H''(u)$ for $u \in S$ and $H'(u) = H(u)$ otherwise. Notably, the PRF oracle O_0 remains defined as $\mathsf{O}_0(i, x) := H((k_i \oplus \texttt{opad}) \parallel H((k_i \oplus \texttt{ipad}) \parallel x))$ in both Games G_2 and G_3.

Define an algorithm $A^{|\mathsf{O}\rangle,\mathsf{O}_b}(b)$ (for $\mathsf{O} \in \{H, H'\}$) that runs the adversary $\mathcal{A}$, using O to answer random-oracle queries and O_b to answer its PRF queries, where O_b is defined as in Games G_2 and G_3 (recall that it uses H to compute its responses when $b = 0$). All other simulations are the same as in Games G_2 and G_3. Upon receiving $\mathcal{A}$'s output bit b', A returns $b' =? b$. If $\mathsf{O} = H$, then A perfectly simulates Game G_2; if $\mathsf{O} = H'$, then A perfectly simulates Game G_3. Therefore, $\Pr[G_2^{\mathcal{A}} \Rightarrow 1] = \Pr[1 \leftarrow A^{|H\rangle,\mathsf{O}_b}(b)]$ and $\Pr[G_3^{\mathcal{A}} \Rightarrow 1] = \Pr[1 \leftarrow A^{|H'\rangle,\mathsf{O}_b}(b)]$.

Let C be an algorithm that selects $j \in [q_H]$ uniformly at random, runs $A^{|H'\rangle,\mathsf{O}_b}(b)$ until just before the $(j+1)$-th H'-query, measures the query input register in the computational basis, and outputs 1 if and only if the measurement outcome belongs to S. Applying the OW2H Lemma (Lemma 6.3) with auxiliary oracle O_b, we obtain

$$\left|\Pr[1 \leftarrow A^{|H\rangle,\mathsf{O}_b}(b)] - \Pr[1 \leftarrow A^{|H'\rangle,\mathsf{O}_b}(b)]\right| \leq 2q_H\sqrt{\Pr[C \Rightarrow 1]}.$$

In C's execution, the adversary has access to a random oracle H' and a PRF oracle O_b. As discussed before, each fresh query to O_0 uses an outer-hash input $x_{out} \in S$ which is distinct from all prior hash inputs (including outer-hash inputs and inner-hash inputs) used by O_0 before this query. Thus, the responses of the PRF oracle O_0 can equivalently be simulated via lazy sampling of a random function $f : [N] \times \{0,1\}^* \to \{0,1\}^\mu$, sampled independently of the keys and H'. Since each k_i is random and independent of $\mathcal{A}$'s view, we have $\Pr[C \Rightarrow 1] \leq \frac{N}{2^\kappa}$. Therefore,

$$\left|\Pr[G_2^{\mathcal{A}} \Rightarrow 1] - \Pr[G_3^{\mathcal{A}} \Rightarrow 1]\right| \leq 2q_H\sqrt{\frac{N}{2^\kappa}}.$$

Finally, in G_3, the same argument shows that the responses of the PRF oracle O_0 can equivalently be simulated via lazy sampling of a random function $f : [N] \times \{0,1\}^* \to \{0,1\}^\mu$, sampled independently of the keys k_i and H'. Therefore, the adversary's views are identically distributed for both $b \in \{0,1\}$. Formally, we have

$$\Pr\left[G_3^{\mathcal{A}} \Rightarrow 1\right] = \frac{1}{2}.$$

Putting everything together completes the proof. □

6.3 MU-dual-PRF Security of HMAC in the ROM and QROM

We next analyze the swapped function $\mathsf{HMAC}^{\mathsf{swap}}(k, x) := \mathsf{HMAC}(x, k)$, which interchanges the roles of the key and message in HMAC. Here, k is the secret key and x is the public input. We restrict public inputs x to $\{0,1\}^\ell$ for a fixed $\ell \leq 8B - 8$[15]. The fixed length makes zero-padding injective, while the length bound ensures that the inner- and outer-hash prefix domains are disjoint. The following theorem gives concrete MU-PRF security bounds for $\mathsf{HMAC}^{\mathsf{swap}}$ in the ROM and QROM.

Theorem 6.5. *Let $\kappa, \ell, \mu \in \mathbb{N}$ with $\ell \leq 8B - 8$, where B is the byte length of* `ipad` *and* `opad`*. Let H be modeled as a random oracle with output length μ, and consider the function $\mathsf{HMAC}^{\mathsf{swap}} \colon \{0,1\}^\kappa \times \{0,1\}^\ell \to \{0,1\}^\mu$ defined by $\mathsf{HMAC}^{\mathsf{swap}}(k, x) := \mathsf{HMAC}(x, k)$. Suppose $\mathcal{A}$ is an adversary against its* MU-PRF *security for N users with independently and uniformly sampled κ-bit keys, making, in total, at most q_H explicit queries to H and q_F classical queries to the PRF oracle. If H is modeled as a classical random oracle, then*

$$\mathsf{Adv}^{\mathsf{MU\text{-}PRF}}_{\mathsf{HMAC}^{\mathsf{swap}},\,N}(\mathcal{A}) \leq \frac{N^2}{2^\kappa} + \frac{(q_F + q_H)^2}{2^\mu} + \frac{N q_H}{2^\kappa} + \frac{N q_H}{2^\mu}.$$

If H is modeled as a quantum-accessible random oracle and $\mathcal{A}$ makes at most q_H quantum queries to H, then

$$\mathsf{Adv}^{\mathsf{MU\text{-}PRF}}_{\mathsf{HMAC}^{\mathsf{swap}},\,N}(\mathcal{A}) \leq \frac{N^2}{2^\kappa} + \frac{8(q_H + 2q_F + 1)^3}{2^\mu} + 2q_H\sqrt{\frac{N}{2^\kappa}} + 2q_H\sqrt{\frac{N}{2^\mu}}.$$

Proof. Let $\mathcal{A}$ be an adversary against the MU-PRF security game for $\mathsf{HMAC}^{\mathsf{swap}}$. Notably, public inputs x are implicitly zero-padded to $8B$ bits before XOR with `ipad` or `opad`. The real PRF oracle is defined as $\mathsf{O}_0(i, x) := H((x \oplus \texttt{opad}) \parallel H((x \oplus \texttt{ipad}) \parallel k_i))$, whereas O_1 consists of independent random functions for each user. Since ℓ is fixed, zero-padding for input x is injective, and $\ell \leq 8B - 8$ ensures that the last byte of the inner-hash prefix $x \oplus \texttt{ipad}$ is `0x36`, whereas that of the outer-hash prefix $x \oplus \texttt{opad}$ is `0x5c`. Thus, the inner- and outer-hash input domains of O_0 are disjoint.

Security Analysis in the ROM. We first consider the analysis in the ROM and the following sequence of games.

Game G_0. This is the original MU-PRF game for $\mathsf{HMAC}^{\mathsf{swap}}$, as defined in Definition 2.7. Therefore,

$$\left|\Pr[G_0^{\mathcal{A}} \Rightarrow 1] - \tfrac{1}{2}\right| = \mathsf{Adv}^{\mathsf{MU\text{-}PRF}}_{\mathsf{HMAC}^{\mathsf{swap}},\,N}(\mathcal{A}).$$

Game G_1. In this game, we ensure that users have distinct keys. Specifically, the game aborts if there exist $i \neq j$ such that $k_i = k_j$. Since the user keys are sampled independently and uniformly from $\{0,1\}^\kappa$, a union bound over all pairs of users bounds the probability of this event by $\frac{N^2}{2^\kappa}$. Hence,

$$\left|\Pr[G_0^{\mathcal{A}} \Rightarrow 1] - \Pr[G_1^{\mathcal{A}} \Rightarrow 1]\right| \leq \frac{N^2}{2^\kappa}.$$

Game G_2. In this game, we ensure that PRF evaluations for distinct users on the same public input yield distinct inner-hash values. Specifically, if there exist two queries (i, x) and (j, x) to the PRF oracle with $i \neq j$ such that $H((x \oplus \texttt{ipad}) \parallel k_i) = H((x \oplus \texttt{ipad}) \parallel k_j)$, the game aborts. Since there are at most $q_F + q_H$ queries to H on the inner-hash domain, by the birthday bound, the probability of the abort event is at most $\frac{(q_F+q_H)^2}{2^\mu}$. Thus,

$$\left|\Pr[G_1^{\mathcal{A}} \Rightarrow 1] - \Pr[G_2^{\mathcal{A}} \Rightarrow 1]\right| \leq \frac{(q_F + q_H)^2}{2^\mu}.$$

[15] In our PQ TLS 1.3 analysis, $x = \mathsf{dES}$ and $\ell = \mu$. Thus, $(\ell, B) = (256, 64)$ for SHA-256 and $(\ell, B) = (384, 128)$ for SHA-384, both satisfying $\ell \leq 8B - 8$.

Right now, for each fresh PRF query (i, x) that has not been queried before, the corresponding outer-hash input has also not been used in any previous PRF query. Games G_1 and G_2 ensure that each fresh PRF query uses inner- and outer-hash inputs that have not been used by any previous PRF query. Indeed, distinct pairs (i, x) have distinct inner-hash inputs. For the outer-hash inputs, distinct public inputs give distinct prefixes, while Game G_2 excludes collisions between users queried on the same public input. Domain separation further ensures that the inner- and outer-hash input domains are disjoint.

It remains to decouple the random oracle H accessed by the adversary from the PRF oracle. This is achieved in the next two games.

Game G_3. In this game, we replace the random oracle H accessible to the adversary with a random oracle H' defined as follows. Specifically, let $S_1 := \{(x \oplus \texttt{ipad}) \parallel k_i \mid i \in [N],\, x \in \{0,1\}^\ell\}$, and sample an independent random oracle R with the same domain and range as H. Define $H'(u) = R(u)$ for $u \in S_1$, and $H'(u) = H(u)$ otherwise. Notably, the PRF oracle O_b remains unchanged.

Define $\mathsf{QUERY}_{\mathsf{in}}$ as the event that the adversary makes an explicit hash query on $x \in S_1$. Note that Games G_2 and G_3 are identical until $\mathsf{QUERY}_{\mathsf{in}}$ occurs. By the difference lemma, we have $\left|\Pr[G_2^{\mathcal{A}} \Rightarrow 1] - \Pr[G_3^{\mathcal{A}} \Rightarrow 1]\right| \leq \Pr[\mathsf{QUERY}_{\mathsf{in}} : G_3]$. Conditional on distinct user keys, the private inner-hash values in the PRF oracle when $b = 0$ for Game G_3 are uniform and independent of H' and completely hide the key values k_i. Since each k_i is uniformly distributed over $\{0,1\}^\kappa$, a union bound over the N users and the q_H explicit queries shows that the probability of such a query is at most $\frac{Nq_H}{2^\kappa}$. Therefore,

$$\left|\Pr[G_2^{\mathcal{A}} \Rightarrow 1] - \Pr[G_3^{\mathcal{A}} \Rightarrow 1]\right| \leq \frac{Nq_H}{2^\kappa}.$$

Game G_4. In this game, we further replace the random oracle H' accessible to the adversary with a new random oracle H''. Specifically, let $S_2 := \{\, (x \oplus \texttt{opad}) \parallel H((x \oplus \texttt{ipad}) \parallel k_i) \mid i \in [N],\, x \in \{0,1\}^\ell \,\}$. Define $H''(u) = R(u)$ for $u \in S_2$, and $H''(u) = H'(u)$ otherwise. Notably, the PRF oracle O_b remains unchanged, and S_1 and S_2 lie in disjoint input domains. Define $\mathsf{QUERY}_{\mathsf{out}}$ as the event that the adversary makes an explicit hash query on $u \in S_2$. It is easy to see that Games G_3 and G_4 are identical until $\mathsf{QUERY}_{\mathsf{out}}$ occurs. Thus, we have $\left|\Pr[G_3^{\mathcal{A}} \Rightarrow 1] - \Pr[G_4^{\mathcal{A}} \Rightarrow 1]\right| \leq \Pr[\mathsf{QUERY}_{\mathsf{out}} : G_4]$.

In G_4, for any hash query $H''(x \parallel y)$ made by the adversary, if this query triggers $\mathsf{QUERY}_{\mathsf{out}}$, then we have $x = x' \oplus \texttt{opad}$ and $y = H((x' \oplus \texttt{ipad}) \parallel k_i)$ for some fixed $x' \in \{0,1\}^\ell$ and $i \in [N]$. Since each $H((x' \oplus \texttt{ipad}) \parallel k_i)$ is uniformly random over $\{0,1\}^\mu$, and the PRF outputs hide its value even given H'', the probability that the query satisfies this condition is at most $N/2^\mu$. Since the adversary makes at most q_H random oracle queries, we have $\Pr[\mathsf{QUERY}_{\mathsf{out}} : G_4] \leq \frac{Nq_H}{2^\mu}$. Therefore,

$$\left|\Pr[G_3^{\mathcal{A}} \Rightarrow 1] - \Pr[G_4^{\mathcal{A}} \Rightarrow 1]\right| \leq \frac{Nq_H}{2^\mu}.$$

At this point, for each fresh answered PRF query (i, x), the corresponding outer-hash input has not been used in any previous PRF query. Moreover, the random oracle accessed by the adversary is independent of the PRF oracle. Therefore, both challenge branches can be simulated using independent random functions f_i, one for each user $i \in [N]$. Thus, the adversary's views are identically distributed in both branches. Therefore,

$$\Pr[G_4^{\mathcal{A}} \Rightarrow 1] = \frac{1}{2}.$$

Combining the bounds completes the ROM proof. □

Security Analysis in the QROM. We lift the ROM analysis to the QROM by considering the following sequence of games.

Game G_0. This is the original $\mathsf{MU\text{-}PRF}$ game for $\mathsf{HMAC}^{\mathsf{swap}}$, but the adversary now makes (quantum) queries to the random oracle H.

Game G_1. In this game, we ensure that users have distinct keys. The analysis is the same as in the ROM. Therefore,

$$\left|\Pr[G_0^{\mathcal{A}} \Rightarrow 1] - \Pr[G_1^{\mathcal{A}} \Rightarrow 1]\right| \leq \frac{N^2}{2^\kappa}.$$

Game G_2. In this game, we ensure that PRF evaluations for distinct users on the same public input yield distinct inner-hash values. Specifically, the game aborts if there exist two queries (i, x) and (j, x) to the PRF oracle with $i \neq j$ such that $H((x \oplus \texttt{ipad}) \parallel k_i) = H((x \oplus \texttt{ipad}) \parallel k_j)$. Since $k_i \neq k_j$ due to Game G_1, this is a collision for H. Since there are at most $q_H + 2q_F$ queries to H (including the implicit queries from the PRF oracle), Lemma 6.2 gives

$$\left|\Pr[G_1^{\mathcal{A}} \Rightarrow 1] - \Pr[G_2^{\mathcal{A}} \Rightarrow 1]\right| \leq \frac{8(q_H + 2q_F + 1)^3}{2^\mu}.$$

It remains to decouple the adversary's access to the random oracle from the PRF oracle. We analyze the next two games using the OW2H Lemma (Lemma 6.3).

Game G_3. In this game, we replace the quantum-accessible random oracle H accessible to the adversary with a new oracle H'. Specifically, let $S_1 := \{(x \oplus \texttt{ipad}) \parallel k_i \mid i \in [N], x \in \{0,1\}^\ell\}$, and sample an independent random oracle R with the same domain and range as H. Define $H'(u) = R(u)$ for $u \in S_1$, and $H'(u) = H(u)$ otherwise. The PRF oracle O_b remains unchanged, and all hash evaluations in O_b use the original H.

Define an algorithm $A_1^{|\mathsf{O}\rangle,\mathsf{O}_b}(b)$ for $\mathsf{O} \in \{H, H'\}$. It runs $\mathcal{A}$, uses O to answer its explicit hash queries, and uses O_b (evaluated using the original H when $b = 0$) to answer its PRF queries. All other simulations are the same as in Games G_2 and G_3. Upon receiving $\mathcal{A}$'s output b', it returns 1 if and only if $b' = b$. If $\mathsf{O} = H$, then A_1 perfectly simulates Game G_2; if $\mathsf{O} = H'$, it perfectly simulates Game G_3. Therefore, $\Pr[G_2^{\mathcal{A}} \Rightarrow 1] = \Pr[1 \leftarrow A_1^{|H\rangle,\mathsf{O}_b}(b)]$ and $\Pr[G_3^{\mathcal{A}} \Rightarrow 1] = \Pr[1 \leftarrow A_1^{|H'\rangle,\mathsf{O}_b}(b)]$.

Let C_1 select $j \in [q_H]$ uniformly at random, run $A_1^{|H'\rangle,\mathsf{O}_b}(b)$ until just before the $(j+1)$-th explicit hash query, and measure the query input register in the computational basis. It outputs 1 if and only if the measured input belongs to S_1. Applying the OW2H Lemma (Lemma 6.3) with auxiliary oracle O_b, we have $\left|\Pr[G_2^{\mathcal{A}} \Rightarrow 1] - \Pr[G_3^{\mathcal{A}} \Rightarrow 1]\right| \leq 2q_H\sqrt{\Pr[C_1 \Rightarrow 1]}$. In C_1's execution, the private inner-hash values used by the PRF oracle when $b = 0$ are uniform and independent of H' and completely hide the key values k_i. Since each k_i is uniformly distributed over $\{0,1\}^\kappa$ and independent of the measured query input of $\mathcal{A}$, a union bound over the N users gives $\Pr[C_1 \Rightarrow 1] \leq \frac{N}{2^\kappa}$. Therefore,

$$\left|\Pr[G_2^{\mathcal{A}} \Rightarrow 1] - \Pr[G_3^{\mathcal{A}} \Rightarrow 1]\right| \leq 2q_H\sqrt{\frac{N}{2^\kappa}}.$$

Game G_4. In this game, we further replace the quantum-accessible random oracle H' accessible to the adversary with a new random oracle H''. Specifically, let $S_2 := \{(x \oplus \texttt{opad}) \parallel H((x \oplus \texttt{ipad}) \parallel k_i) \mid i \in [N], x \in \{0,1\}^\ell\}$. Define $H''(u) = R(u)$ for $u \in S_2$, and $H''(u) = H'(u)$ otherwise. Since S_1 and S_2 lie in disjoint input domains, we use the same independent oracle R sampled in Game G_3. The PRF oracle O_b remains unchanged.

Define an algorithm $A_2^{|\mathsf{O}\rangle,\mathsf{O}_b}(b)$ for $\mathsf{O} \in \{H', H''\}$. It runs $\mathcal{A}$, uses O to answer its explicit hash queries, and uses O_b (evaluated using the original H when $b = 0$) to answer its PRF queries. All other simulations are the same as in Games G_3 and G_4. Upon receiving $\mathcal{A}$'s output b', it returns 1 if and only if $b' = b$. If $\mathsf{O} = H'$, then A_2 perfectly simulates Game G_3; if $\mathsf{O} = H''$, it perfectly simulates Game G_4. Therefore, $\Pr[G_3^{\mathcal{A}} \Rightarrow 1] = \Pr[1 \leftarrow A_2^{|H'\rangle,\mathsf{O}_b}(b)]$ and $\Pr[G_4^{\mathcal{A}} \Rightarrow 1] = \Pr[1 \leftarrow A_2^{|H''\rangle,\mathsf{O}_b}(b)]$.

Let C_2 select $j \in [q_H]$ uniformly at random, run $A_2^{|H''\rangle,\mathsf{O}_b}(b)$ until just before the $(j+1)$-th explicit hash query, and measure the query input register in the computational basis. It outputs 1 if and only if the measured input belongs to S_2. Applying the OW2H Lemma (Lemma 6.3) with auxiliary oracle O_b, we have $\left|\Pr[G_3^{\mathcal{A}} \Rightarrow 1] - \Pr[G_4^{\mathcal{A}} \Rightarrow 1]\right| \leq 2q_H\sqrt{\Pr[C_2 \Rightarrow 1]}$.

In C_2's execution and Game G_4, the real PRF oracle $\mathsf{O}_0(i,x)$ can be simulated by $f(i,x)$, where $f : [N] \times \{0,1\}^\ell \to \{0,1\}^\mu$ is a uniformly random function sampled independently of H'' and the private inner-hash values. This simulation is equivalent since Games G_1 and G_2 ensure that each fresh PRF query uses a fresh outer-hash input that has not been used in $\mathsf{O}_0(i,x)$ before. Therefore, each private inner-hash value $H((x \oplus \texttt{ipad}) \parallel k_i)$ is uniformly random and independent of the adversary's view. Note that the prefix of the adversary's measured query input determines at most one valid public input x. For the input to belong to S_2, its suffix must equal the uniformly random value $H((x \oplus \texttt{ipad}) \parallel k_i)$ for some $i \in [N]$. Thus, we have $\Pr[C_2 \Rightarrow 1] \leq \frac{N}{2^\mu}$. Therefore,

$$\left|\Pr[G_3^{\mathcal{A}} \Rightarrow 1] - \Pr[G_4^{\mathcal{A}} \Rightarrow 1]\right| \leq 2q_H\sqrt{\frac{N}{2^\mu}}.$$

Finally, as noted above, in Game G_4, the PRF oracle in either branch can be simulated using independent random functions f_i, one for each user $i \in [N]$. Thus, the adversary's view is independent of the challenge bit b. Therefore,

$$\Pr[G_4^{\mathcal{A}} \Rightarrow 1] = \frac{1}{2}.$$

Combining the bounds completes the QROM proof. □

6.4 Security of HKDF

Finally, we derive the corresponding security bounds for HKDF.Extract and HKDF.Expand.

Theorem 6.6. *Let* HMAC *and* HKDF.Extract *be defined as in Definitions 2.8 and 2.9, respectively. Let $\mathcal{A}$ be an adversary against the* MU-PRF *security of* HKDF.Extract *with N users. Then,*

$$\mathsf{Adv}^{\mathsf{MU\text{-}PRF}}_{\mathsf{HKDF.Extract},\,N}(\mathcal{A}) = \mathsf{Adv}^{\mathsf{MU\text{-}PRF}}_{\mathsf{HMAC},\,N}(\mathcal{A}).$$

Moreover, under the same input domains and key distributions, for any adversary $\mathcal{B}$, we have

$$\mathsf{Adv}^{\mathsf{MU\text{-}dual\text{-}PRF}}_{\mathsf{HKDF.Extract},\,N}(\mathcal{B}) = \mathsf{Adv}^{\mathsf{MU\text{-}dual\text{-}PRF}}_{\mathsf{HMAC},\,N}(\mathcal{B}).$$

Proof. This result follows immediately from the definition of HKDF.Extract, which is defined as a direct application of the HMAC function, also after swapping the two inputs. Therefore, both the MU-PRF and MU-dual-PRF security of HKDF.Extract are equivalent to those of HMAC. □

Theorem 6.7. *Let* HMAC *and* HKDF.Expand *(with fixed output length L) be defined as in Definitions 2.8 and 2.9, respectively. Let $\mathcal{A}$ be an adversary against the* MU-PRF *security of* HKDF.Expand *for N users, making at most q_E classical PRF queries and q_H explicit hash queries. Then there exists an adversary $\mathcal{B}$ such that*

$$\mathsf{Adv}^{\mathsf{MU\text{-}PRF}}_{\mathsf{HKDF.Expand},\,N}(\mathcal{A}) \leq \mathsf{Adv}^{\mathsf{MU\text{-}PRF}}_{\mathsf{HMAC},\,N}(\mathcal{B}),$$

where $\mathcal{B}$ runs in time comparable to that of $\mathcal{A}$ and makes at most ωq_E classical PRF queries and q_H explicit hash queries, with $\omega = \lceil L/\mu \rceil$.

Proof. The reduction is straightforward. The adversary $\mathcal{B}$ can perfectly simulate the experiment $\mathsf{Exp}^{\mathsf{MU\text{-}PRF}}_{\mathsf{HKDF.Expand},\,N}(\mathcal{A})$ by using its own oracle. Upon receiving a query from $\mathcal{A}$, $\mathcal{B}$ simulates the HKDF.Expand procedure by performing $\lceil L/\mu \rceil$ sequential invocations of its oracle. If the oracle provides a real HMAC function, then $\mathcal{B}$ perfectly simulates HKDF.Expand. If the oracle is a truly random function, each fresh HKDF.Expand query receives an independent uniform bitstring, while repeated queries receive the same response. Hence, whenever $\mathcal{A}$ wins its experiment, $\mathcal{B}$ also wins. □

Combining Theorems 6.4, 6.5, 6.6, and 6.7, we obtain concrete MU-PRF bounds for HKDF.Extract and HKDF.Expand in the QROM, as well as MU-dual-PRF bounds for HMAC and HKDF.Extract in the ROM and QROM.

Acknowledgments

We would like to thank the anonymous reviewers for their valuable comments. The authors used large language models solely for basic language assistance, including grammar and spelling checks, and take full responsibility for the correctness and integrity of the manuscript. This work was supported by the National Natural Science Foundation of China (Grant Nos. 62572482 and 62632012), the Young Scientists Fund (Class C) of the National Natural Science Foundation of China (Grant No. 62602122), the Shanghai Collaborative Innovation Fund (Grant No. XTCX-KJ-2023-54), the Special Fund for Key Technologies in Blockchain of the Shanghai Science and Technology Committee (Grant No. 23511100300), and the Science and Technology Innovation Program of Hunan Province (Grant No. 2025RC1040).

References

[ABR01] M. Abdalla, M. Bellare, and P. Rogaway. The oracle diffie-hellman assumptions and an analysis of DHIES. In D. Naccache, editor, *Topics in Cryptology — CT-RSA 2001*, volume 2020 of *Lecture Notes in Computer Science*, pages 143–158, Berlin, Heidelberg, 2001. Springer. doi:10.1007/3-540-45353-9_12.

[ADH+22] Yawning Angel, Benjamin Dowling, Andreas Hülsing, Peter Schwabe, and Fiona Johanna Weber. Post quantum noise. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022: 29th Conference on Computer and Communications Security*, pages 97–109, Los Angeles, CA, USA, November 7–11, 2022. ACM Press. doi:10.1145/3548606.3560577.

[AHU19] Andris Ambainis, Mike Hamburg, and Dominique Unruh. Quantum security proofs using semi-classical oracles. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019, Part II*, volume 11693 of *Lecture Notes in Computer Science*, pages 269–295, Santa Barbara, CA, USA, August 18–22, 2019. Springer, Cham, Switzerland. doi:10.1007/978-3-030-26951-7_10.

[BBGS23] Matilda Backendal, Mihir Bellare, Felix Günther, and Matteo Scarlata. When messages are keys: Is HMAC a dual-PRF? In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology – CRYPTO 2023, Part III*, volume 14083 of *Lecture Notes in Computer Science*, pages 661–693, Santa Barbara, CA, USA, August 20–24, 2023. Springer, Cham, Switzerland. doi:10.1007/978-3-031-38548-3_22.

[BBM00] Mihir Bellare, Alexandra Boldyreva, and Silvio Micali. Public-key encryption in a multi-user setting: Security proofs and improvements. In Bart Preneel, editor, *Advances in Cryptology – EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 259–274, Bruges, Belgium, May 14–18, 2000. Springer Berlin Heidelberg, Germany. doi:10.1007/3-540-45539-6_18.

[BCK96a] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Keying hash functions for message authentication. In Neal Koblitz, editor, *Advances in Cryptology – CRYPTO'96*, volume 1109 of *Lecture Notes in Computer Science*, pages 1–15, Santa Barbara, CA, USA, August 18–22, 1996. Springer Berlin Heidelberg, Germany. doi:10.1007/3-540-68697-5_1.

[BCK96b] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Pseudorandom functions revisited: The cascade construction and its concrete security. In *37th Annual Symposium on Foundations of Computer Science*, pages 514–523, Burlington, Vermont, October 14–16, 1996. IEEE Computer Society Press. doi:10.1109/SFCS.1996.548510.

[BDF+11] Dan Boneh, Özgür Dagdelen, Marc Fischlin, Anja Lehmann, Christian Schaffner, and Mark Zhandry. Random oracles in a quantum world. In Dong Hoon Lee and Xiaoyun Wang, editors, *Advances in Cryptology – ASIACRYPT 2011*, volume 7073 of *Lecture Notes in Computer Science*, pages 41–69, Seoul, South Korea, December 4–8, 2011. Springer Berlin Heidelberg, Germany. doi:10.1007/978-3-642-25385-0_3.

[BDG20] Mihir Bellare, Hannah Davis, and Felix Günther. Separate your domains: NIST PQC KEMs, oracle cloning and read-only indifferentiability. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020, Part II*, volume 12106 of *Lecture Notes in Computer Science*, pages 3–32, Zagreb, Croatia, May 10–14, 2020. Springer, Cham, Switzerland. doi:10.1007/978-3-030-45724-2_1.

[Bel15] Mihir Bellare. New proofs for NMAC and HMAC: Security without collision resistance. *Journal of Cryptology*, 28(4):844–878, October 2015. doi:10.1007/s00145-014-9185-x.

[BFG+22] Jacqueline Brendel, Rune Fiedler, Felix Günther, Christian Janson, and Douglas Stebila. Post-quantum asynchronous deniable key exchange and the Signal handshake. In Goichiro Hanaoka, Junji Shikata, and Yohei Watanabe, editors, *PKC 2022: 25th International Conference on Theory and Practice of Public Key Cryptography, Part II*, volume 13178 of *Lecture Notes in Computer Science*, pages 3–34, Virtual Event, March 8–11, 2022. Springer, Cham, Switzerland. doi:10.1007/978-3-030-97131-1_1.

[BFGJ17] Jacqueline Brendel, Marc Fischlin, Felix Günther, and Christian Janson. PRF-ODH: Relations, instantiations, and impossibility results. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017, Part III*, volume 10403 of *Lecture Notes in Computer Science*, pages 651–681, Santa Barbara, CA, USA, August 20–24, 2017. Springer, Cham, Switzerland. doi:10.1007/978-3-319-63697-9_22.

[BHH+19] Nina Bindel, Mike Hamburg, Kathrin Hövelmanns, Andreas Hülsing, and Edoardo Persichetti. Tighter proofs of CCA security in the quantum random oracle model. In Dennis Hofheinz and Alon Rosen, editors, *TCC 2019: 17th Theory of Cryptography Conference, Part II*, volume 11892 of *Lecture Notes in Computer Science*, pages 61–90, Nuremberg, Germany, December 1–5, 2019. Springer, Cham, Switzerland. doi:10.1007/978-3-030-36033-7_3.

[BHJ+15] Christoph Bader, Dennis Hofheinz, Tibor Jager, Eike Kiltz, and Yong Li. Tightly-secure authenticated key exchange. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015: 12th Theory of Cryptography Conference, Part I*, volume 9014 of *Lecture Notes in Computer Science*, pages 629–658, Warsaw, Poland, March 23–25, 2015. Springer Berlin Heidelberg, Germany. doi:10.1007/978-3-662-46494-6_26.

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby, editors, *ACM CCS 93: 1st Conference on Computer and Communications Security*, pages 62–73, Fairfax, Virginia, USA, November 3–5, 1993. ACM Press. doi:10.1145/168588.168596.

[BR94] Mihir Bellare and Phillip Rogaway. Entity authentication and key distribution. In Douglas R. Stinson, editor, *Advances in Cryptology – CRYPTO'93*, volume 773 of *Lecture Notes in Computer Science*, pages 232–249, Santa Barbara, CA, USA, August 22–26, 1994. Springer Berlin Heidelberg, Germany. doi:10.1007/3-540-48329-2_21.

[CK02] Ran Canetti and Hugo Krawczyk. Security analysis of IKE's signature-based key-exchange protocol. In Moti Yung, editor, *Advances in Cryptology – CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 143–161, Santa Barbara, CA, USA, August 18–22, 2002. Springer Berlin Heidelberg, Germany. URL: https://eprint.iacr.org/2002/120/, doi:10.1007/3-540-45708-9_10.

[CZC+26] Jinrong Chen, Biming Zhou, Rongmao Chen, Haodong Jiang, Yi Wang, Xinyi Huang, Yunlei Zhao, and Moti Yung. Post-quantum tls 1.3 handshake from cpa-secure kems with tighter reductions. In Joan Daemen and Emmanuel Thomé, editors, *Advances in Cryptology – EUROCRYPT 2026*, pages 499–529, Cham, 2026. Springer Nature Switzerland. doi:10.1007/978-3-032-25317-0_17.

[DDGJ22] Hannah Davis, Denis Diemert, Felix Günther, and Tibor Jager. On the concrete security of TLS 1.3 PSK mode. In Orr Dunkelman and Stefan Dziembowski, editors, *Advances in Cryptology – EUROCRYPT 2022, Part II*, volume 13276 of *Lecture Notes in Computer*

Science, pages 876–906, Trondheim, Norway, May 30 – June 3, 2022. Springer, Cham, Switzerland. doi:10.1007/978-3-031-07085-3_30.

[DFGS21] Benjamin Dowling, Marc Fischlin, Felix Günther, and Douglas Stebila. A cryptographic analysis of the TLS 1.3 handshake protocol. *Journal of Cryptology*, 34(4):37, October 2021. doi:10.1007/s00145-021-09384-1.

[DG21] H. Davis and F. Günther. Tighter proofs for the SIGMA and TLS 1.3 key exchange protocols. In K. Sako and N. O. Tippenhauer, editors, *Applied Cryptography and Network Security – ACNS 2021*, volume 12727 of *Lecture Notes in Computer Science*, pages 333–353, Cham, 2021. Springer. doi:10.1007/978-3-030-78375-4_18.

[DH76] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976. doi:10.1109/TIT.1976.1055638.

[DJ21] Denis Diemert and Tibor Jager. On the tight security of TLS 1.3: Theoretically sound cryptographic parameters for real-world deployments. *Journal of Cryptology*, 34(3):30, July 2021. doi:10.1007/s00145-021-09388-x.

[GGJJ23] Kai Gellert, Kristian Gjøsteen, Håkon Jacobsen, and Tibor Jager. On optimal tightness for key exchange with full forward secrecy via key confirmation. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology – CRYPTO 2023, Part IV*, volume 14084 of *Lecture Notes in Computer Science*, pages 297–329, Santa Barbara, CA, USA, August 20–24, 2023. Springer, Cham, Switzerland. doi:10.1007/978-3-031-38551-3_10.

[GJ18] Kristian Gjøsteen and Tibor Jager. Practical and tightly-secure digital signatures and authenticated key exchange. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part II*, volume 10992 of *Lecture Notes in Computer Science*, pages 95–125, Santa Barbara, CA, USA, August 19–23, 2018. Springer, Cham, Switzerland. doi:10.1007/978-3-319-96881-0_4.

[HJK+21] Shuai Han, Tibor Jager, Eike Kiltz, Shengli Liu, Jiaxin Pan, Doreen Riepel, and Sven Schäge. Authenticated key exchange and signatures with tight security in the standard model. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology – CRYPTO 2021, Part IV*, volume 12828 of *Lecture Notes in Computer Science*, pages 670–700, Virtual Event, August 16–20, 2021. Springer, Cham, Switzerland. doi:10.1007/978-3-030-84259-8_23.

[HLWG23] Shuai Han, Shengli Liu, Zhedong Wang, and Dawu Gu. Almost tight multi-user security under adaptive corruptions from LWE in the standard model. In *Advances in Cryptology – CRYPTO 2023*, volume 14085 of *Lecture Notes in Computer Science*, pages 682–715. Springer, Cham, 2023. doi:10.1007/978-3-031-38554-4_22.

[HV22] Loïs Huguenin-Dumittan and Serge Vaudenay. On IND-qCCA security in the ROM and its applications - CPA security is sufficient for TLS 1.3. In Orr Dunkelman and Stefan Dziembowski, editors, *Advances in Cryptology – EUROCRYPT 2022, Part III*, volume 13277 of *Lecture Notes in Computer Science*, pages 613–642, Trondheim, Norway, May 30 – June 3, 2022. Springer, Cham, Switzerland. doi:10.1007/978-3-031-07082-2_22.

[JKRS21] Tibor Jager, Eike Kiltz, Doreen Riepel, and Sven Schäge. Tightly-secure authenticated key exchange, revisited. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology – EUROCRYPT 2021, Part I*, volume 12696 of *Lecture Notes in Computer Science*, pages 117–146, Zagreb, Croatia, October 17–21, 2021. Springer, Cham, Switzerland. doi:10.1007/978-3-030-77870-5_5.

[JMZ23] Haodong Jiang, Zhi Ma, and Zhenfeng Zhang. Post-quantum security of key encapsulation mechanism against CCA attacks with a single decapsulation query. In Jian Guo and Ron Steinfeld, editors, *Advances in Cryptology – ASIACRYPT 2023, Part IV*, volume 14441 of *Lecture Notes in Computer Science*, pages 434–468, Guangzhou, China, December 4–8, 2023. Springer, Singapore, Singapore. doi:10.1007/978-981-99-8730-6_14.

[KBC97] Hugo Krawczyk, Mihir Bellare, and Ran Canetti. HMAC: Keyed-Hashing for Message Authentication. Informational RFC 2104, RFC Editor, February 1997. Updated by RFC 6151. URL: https://www.rfc-editor.org/rfc/rfc2104.txt.

[KE10] Hugo Krawczyk and Pekka Eronen. HMAC-based Extract-and-Expand Key Derivation Function (HKDF). Informational RFC 5869, RFC Editor, May 2010. URL: https://www.rfc-editor.org/rfc/rfc5869.txt.

[Kra10] Hugo Krawczyk. Cryptographic extraction and key derivation: The HKDF scheme. In Tal Rabin, editor, *Advances in Cryptology – CRYPTO 2010*, volume 6223 of *Lecture Notes in Computer Science*, pages 631–648, Santa Barbara, CA, USA, August 15–19, 2010. Springer Berlin Heidelberg, Germany. doi:10.1007/978-3-642-14623-7_34.

[LLGW20] Xiangyu Liu, Shengli Liu, Dawu Gu, and Jian Weng. Two-pass authenticated key exchange with explicit authentication and tight security. In Shiho Moriai and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT 2020, Part II*, volume 12492 of *Lecture Notes in Computer Science*, pages 785–814, Daejeon, South Korea, December 7–11, 2020. Springer, Cham, Switzerland. doi:10.1007/978-3-030-64834-3_27.

[MAB+25] Carlos Aguilar Melchor, Nicolas Aragon, Slim Bettaieb, Loïc Bidoux, Olivier Blazy, Jurjen Bos, Jean-Christophe Deneuville, Arnaud Dion, Philippe Gaborit, Jérôme Lacan, Edoardo Persichetti, Jean-Marc Robert, Pascal Véron, and Gilles Zémor. HQC: Hamming Quasi-Cyclic, 2025. Selected by NIST for standardization on Mar. 11, 2025. https://pqc-hqc.org/.

[Nat24a] National Institute of Standards and Technology. Module-lattice-based digital signature standard (FIPS 204). Federal Information Processing Standards Publication 204, U.S. Department of Commerce, National Institute of Standards and Technology, Gaithersburg, MD, USA, August 2024. doi:10.6028/NIST.FIPS.204.

[Nat24b] National Institute of Standards and Technology. Module-lattice-based key-encapsulation mechanism standard (FIPS 203). Federal Information Processing Standards Publication 203, U.S. Department of Commerce, National Institute of Standards and Technology, Gaithersburg, MD, USA, August 2024. doi:10.6028/NIST.FIPS.203.

[Nat24c] National Institute of Standards and Technology. Stateless hash-based digital signature standard (FIPS 205). Federal Information Processing Standards Publication 205, U.S. Department of Commerce, National Institute of Standards and Technology, Gaithersburg, MD, USA, August 2024. doi:10.6028/NIST.FIPS.205.

[Nat25] National Institute of Standards and Technology. Post-quantum cryptography standardization. https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization, 2025. Accessed: 2025-10-01.

[NC00] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2000.

[PRZ24] Jiaxin Pan, Doreen Riepel, and Runzhi Zeng. Key exchange with tight (full) forward secrecy via key confirmation. In Marc Joye and Gregor Leander, editors, *Advances in Cryptology – EUROCRYPT 2024, Part VII*, volume 14657 of *Lecture Notes in Computer Science*, pages 59–89, Zurich, Switzerland, May 26–30, 2024. Springer, Cham, Switzerland. doi:10.1007/978-3-031-58754-2_3.

[PW22] Jiaxin Pan and Benedikt Wagner. Lattice-based signatures with tight adaptive corruptions and more. In Goichiro Hanaoka, Junji Shikata, and Yohei Watanabe, editors, *PKC 2022: 25th International Conference on Theory and Practice of Public Key Cryptography, Part II*, volume 13178 of *Lecture Notes in Computer Science*, pages 347–378, Virtual Event, March 8–11, 2022. Springer, Cham, Switzerland. doi:10.1007/978-3-030-97131-1_12.

[PWZ23a] Jiaxin Pan, Benedikt Wagner, and Runzhi Zeng. Lattice-based authenticated key exchange with tight security. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology – CRYPTO 2023, Part V*, volume 14085 of *Lecture Notes in Computer Science*, pages 616–647, Santa Barbara, CA, USA, August 20–24, 2023. Springer, Cham, Switzerland. doi:10.1007/978-3-031-38554-4_20.

[PWZ23b] Jiaxin Pan, Benedikt Wagner, and Runzhi Zeng. Tighter security for generic authenticated key exchange in the QROM. In Jian Guo and Ron Steinfeld, editors, *Advances in*

Cryptology – ASIACRYPT 2023, Part IV, volume 14441 of *Lecture Notes in Computer Science*, pages 401–433, Guangzhou, China, December 4–8, 2023. Springer, Singapore, Singapore. doi:10.1007/978-981-99-8730-6_13.

[Res18] Eric Rescorla. The transport layer security (TLS) protocol version 1.3, August 2018. doi:10.17487/RFC8446.

[Sho94] Peter W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *35th Annual Symposium on Foundations of Computer Science*, pages 124–134, Santa Fe, NM, USA, November 20–22, 1994. IEEE Computer Society Press. doi:10.1109/SFCS.1994.365700.

[SSW20] Peter Schwabe, Douglas Stebila, and Thom Wiggers. Post-quantum TLS without handshake signatures. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020: 27th Conference on Computer and Communications Security*, pages 1461–1480, Virtual Event, USA, November 9–13, 2020. ACM Press. doi:10.1145/3372297.3423350.

[SSW21] Peter Schwabe, Douglas Stebila, and Thom Wiggers. More efficient post-quantum KEMTLS with pre-distributed public keys. In Elisa Bertino, Haya Shulman, and Michael Waidner, editors, *ESORICS 2021: 26th European Symposium on Research in Computer Security, Part I*, volume 12972 of *Lecture Notes in Computer Science*, pages 3–22, Darmstadt, Germany, October 4–8, 2021. Springer, Cham, Switzerland. doi:10.1007/978-3-030-88418-5_1.

[Zha12] Mark Zhandry. Secure identity-based encryption in the quantum random oracle model. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 758–775, Santa Barbara, CA, USA, August 19–23, 2012. Springer Berlin Heidelberg, Germany. doi:10.1007/978-3-642-32009-5_44.

[Zha19] Mark Zhandry. How to record quantum queries, and applications to quantum indifferentiability. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019, Part II*, volume 11693 of *Lecture Notes in Computer Science*, pages 239–268, Santa Barbara, CA, USA, August 18–22, 2019. Springer, Cham, Switzerland. doi:10.1007/978-3-030-26951-7_9.

[ZJZ24] Biming Zhou, Haodong Jiang, and Yunlei Zhao. CPA-secure KEMs are also sufficient for post-quantum TLS 1.3. In Kai-Min Chung and Yu Sasaki, editors, *Advances in Cryptology – ASIACRYPT 2024, Part III*, volume 15486 of *Lecture Notes in Computer Science*, pages 433–464, Kolkata, India, December 9–13, 2024. Springer, Singapore, Singapore. doi:10.1007/978-981-96-0891-1_14.

A Reductions from MUC Security to Standard Security

In this section, we present reductions from MUC-IND-1RCCA to IND-1RCCA and from MUC-IND-1CCA to IND-1CCA, following the standard hybrid argument of [BBM00].

Theorem A.1 (MUC-IND-1RCCA from IND-1RCCA in the Standard Model). *Let* $\mathsf{KEM} = (\mathsf{Gen}, \mathsf{Encaps}, \mathsf{Decaps})$ *be a KEM that is* IND-1RCCA*-secure. Suppose its encapsulated keys have min-entropy at least* γ *for each honestly generated public key. Then, for any polynomial-time adversary* $\mathcal{A}$ *against the* (N, S)-MUC-IND-1RCCA *security of the scheme, there exists a polynomial-time adversary* $\mathcal{B}$ *such that*

$$\mathsf{Adv}^{\mathsf{MUC\text{-}IND\text{-}1RCCA}}_{\mathsf{KEM},N,S}(\mathcal{A}) \leq NS \cdot \mathsf{Adv}^{\mathsf{IND\text{-}1RCCA}}_{\mathsf{KEM}}(\mathcal{B}) + NS(S-1)\left(2^{-\gamma} + \frac{3}{|\mathcal{K}|}\right),$$

where $\mathcal{B}$ *runs in time comparable to that of* $\mathcal{A}$*.*

Proof. Let $\mathcal{A}$ be a polynomial-time adversary against the (N, S)-MUC-IND-1RCCA security of KEM. Define the following game sequence.

Game G_0. This is the original (N, S)-MUC-IND-1RCCA game with the challenge bit fixed to $b = 0$. Each game below outputs the adversary's bit b'.
Game $G_{i,j}$ for $i \in [N]$ and $j \in [S+1]$. In this game, $\mathcal{A}$ receives $\mathbf{K}_1[i', j']$ at indices satisfying $i' < i$ or ($i' = i$ and $j' < j$), and $\mathbf{K}_0[i', j']$ otherwise. The original matrices $\mathbf{K}_0$ and $\mathbf{K}_1$ remain unchanged in all replay checks. Note that $G_{0,0} = G_0$, $G_{i-1,S} = G_{i,0}$ for all $i \in \{1, \ldots, N-1\}$, and $G_{N-1,S}$ corresponds to the game in which all keys shown to $\mathcal{A}$ are taken from $\mathbf{K}_1$.

Let us bound the distinguishing advantage between two consecutive games $G_{i,j}$ and $G_{i,j+1}$. For each $(i, j) \in [N] \times [S]$, we construct an adversary $\mathcal{B}_{i,j}$ against the IND-1RCCA security of the KEM as follows. The adversary $\mathcal{B}_{i,j}$ receives a challenge tuple $(\mathsf{pk}^*, \mathsf{ct}^*, K^*_{\bar{b}})$ and embeds it into index (i, j) by setting $\mathbf{pk}[i] = \mathsf{pk}^*$ and $\mathbf{ct}[i, j] = \mathsf{ct}^*$, using $K^*_{\bar{b}}$ as the key shown to $\mathcal{A}$ at that index. The real and random keys used in replay checks at this index are K^*_0 and K^*_1 held by the single-challenge challenger, whose decapsulation oracle checks both keys. The adversary generates the other users' key pairs and all remaining table entries locally, using pk^* for user i, and follows the replacements prescribed by $G_{i,j}$.

All queries are subject to the original per-user query limit, including rejected queries. For $\mathcal{A}$'s query ct to $\mathsf{O}^{\mathsf{Dec}'}(i, \cdot)$, $\mathcal{B}_{i,j}$ returns $\perp$ if $\mathsf{ct} = \mathbf{ct}[i, j']$ for some $j' \in [S]$; otherwise, it forwards the query to its single-challenge oracle. It returns $\perp$ for a response $\perp$ and Test-j for a response Test. For a concrete returned key K, it checks the local entries with $j' \neq j$, returning the marker for the smallest matching real-key index, or, if none exists, the smallest matching random-key index. If neither matches, it returns K. For other users, it simulates the oracle locally using their secret keys and key matrices. Upon receiving b' from $\mathcal{A}$, $\mathcal{B}_{i,j}$ outputs $\bar{b}' := b'$.

Let $\mathsf{Bad}_{i,j}$ denote the event that either $\mathbf{K}_0[i, j]$ or $\mathbf{K}_1[i, j]$ coincides with a real or random key at another index in the same row. If $\mathsf{Bad}_{i,j}$ does not occur, a replay marker returned by the single-challenge oracle identifies index j uniquely. Hence, when $\bar{b} = 0$, the simulation corresponds to $G_{i,j}$; when $\bar{b} = 1$, it corresponds to $G_{i,j+1}$. Conditional on the public key, the real encapsulated keys are independent and have min-entropy at least γ. For each $j' \neq j$, the real–real equality has probability at most $2^{-\gamma}$, and the other three equalities each have probability $1/|\mathcal{K}|$. Thus, $\Pr[\mathsf{Bad}_{i,j}] \leq (S-1)\left(2^{-\gamma} + \frac{3}{|\mathcal{K}|}\right) =: \varepsilon_{\mathsf{coll}}$. Therefore, we have

$$\left|\Pr[G^{\mathcal{A}}_{i,j} \Rightarrow 1] - \Pr[G^{\mathcal{A}}_{i,j+1} \Rightarrow 1]\right| \leq 2 \cdot \mathsf{Adv}^{\mathsf{IND\text{-}1RCCA}}_{\mathsf{KEM}}(\mathcal{B}_{i,j}) + 2\varepsilon_{\mathsf{coll}}.$$

Game G_2. This game is identical to Game $G_{N-1,S}$. Since $\mathcal{A}$ receives $\mathbf{K}_1$ and the replay oracle remains unchanged, its view has the same distribution as in the original game with $b = 1$. Therefore,

$$\mathsf{Adv}^{\mathsf{MUC\text{-}IND\text{-}1RCCA}}_{\mathsf{KEM},N,S}(\mathcal{A}) = \frac{1}{2}\left|\Pr[G^{\mathcal{A}}_0 \Rightarrow 1] - \Pr[G^{\mathcal{A}}_2 \Rightarrow 1]\right|.$$

Let $\mathcal{B}$ choose $(i, j) \leftarrow\!\!\$\ [N] \times [S]$ and run $\mathcal{B}_{i,j}$. Averaging the signed differences between consecutive games and accounting for the simulation error in each branch gives

$$\mathsf{Adv}^{\mathsf{MUC\text{-}IND\text{-}1RCCA}}_{\mathsf{KEM},N,S}(\mathcal{A}) \leq NS \cdot \mathsf{Adv}^{\mathsf{IND\text{-}1RCCA}}_{\mathsf{KEM}}(\mathcal{B}) + NS\varepsilon_{\mathsf{coll}}.$$

Substituting the bound on $\varepsilon_{\mathsf{coll}}$ completes the proof. □

Theorem A.2 (MUC-IND-1CCA from IND-1CCA in the Standard Model). *Let* $\mathsf{KEM} = (\mathsf{Gen}, \mathsf{Encaps}, \mathsf{Decaps})$ *be a KEM that is* IND-1CCA*-secure. Then, for any polynomial-time adversary $\mathcal{A}$ against the (N, S)-*MUC-IND-1CCA *security of the scheme, there exists a polynomial-time adversary $\mathcal{B}$ such that*

$$\mathsf{Adv}^{\mathsf{MUC\text{-}IND\text{-}1CCA}}_{\mathsf{KEM},N,S}(\mathcal{A}) \leq NS \cdot \mathsf{Adv}^{\mathsf{IND\text{-}1CCA}}_{\mathsf{KEM}}(\mathcal{B}),$$

where $\mathcal{B}$ runs in time comparable to that of $\mathcal{A}$.

Proof. Let $\mathcal{A}$ be a polynomial-time adversary against the (N, S)-MUC-IND-1CCA security of KEM. Define the following game sequence.

Game G_0. This is the original (N, S)-MUC-IND-1CCA game with the challenge bit fixed to $b = 0$. Each game below outputs the adversary's bit b'.

Game $G_{i,j}$ for $i \in [N]$ and $j \in [S+1]$. In this game, we independently replace $\mathbf{K}_0[i', j']$ with a uniformly random key for all index pairs (i', j') such that $i' < i$ or $(i' = i$ and $j' < j)$, where $i' \in [N]$ and $j' \in [S]$. Note that $G_{0,0} = G_0$, $G_{i-1,S} = G_{i,0}$ for all $i \in \{1, \ldots, N-1\}$, and $G_{N-1,S}$ corresponds to the game in which all values $\mathbf{K}_0[i, j]$ are replaced with random keys.

Let us bound the distinguishing advantage between two consecutive games $G_{i,j}$ and $G_{i,j+1}$. For each $(i, j) \in [N] \times [S]$, we construct an adversary $\mathcal{B}_{i,j}$ against the IND-1CCA security of the KEM as follows. The adversary $\mathcal{B}_{i,j}$ receives a challenge tuple $(\mathsf{pk}^*, \mathsf{ct}^*, K_{\overline{b}})$ and embeds it into index (i, j) by setting $\mathbf{pk}[i] = \mathsf{pk}^*$, $\mathbf{ct}[i, j] = \mathsf{ct}^*$, and $\mathbf{K}_0[i, j] = K_{\overline{b}}$. The adversary then samples $(\mathsf{pk}_{i'}, \mathsf{sk}_{i'}) \leftarrow\!\!\$\ \mathsf{Gen}()$ for other users $i' \neq i$ and simulates the remainder of the game exactly as in $G_{i,j}$. In particular, it generates the other ciphertexts for user i using $\mathsf{pk}_i = \mathsf{pk}^*$.

All queries are subject to the original per-user query limit. For $\mathcal{A}$'s query ct to $\mathsf{O}^{\mathsf{Dec}}(i, \cdot)$, if $\mathsf{ct} = \mathbf{ct}[i, j']$ for some $j' \in [S]$, $\mathcal{B}_{i,j}$ directly returns $\perp$; otherwise, it forwards the query to its IND-1CCA decapsulation oracle. For other users $i' \neq i$, it uses the corresponding $\mathsf{sk}_{i'}$ to respond directly according to the same oracle rules. Upon receiving b' from $\mathcal{A}$, $\mathcal{B}_{i,j}$ outputs $\overline{b}' := b'$. If $\overline{b} = 0$, the embedded key is real, and the simulation corresponds to $G_{i,j}$. If $\overline{b} = 1$, the key is random, and the simulation corresponds to $G_{i,j+1}$. Hence,

$$\left|\Pr[G^{\mathcal{A}}_{i,j} \Rightarrow 1] - \Pr[G^{\mathcal{A}}_{i,j+1} \Rightarrow 1]\right| \leq 2 \cdot \mathsf{Adv}^{\mathsf{IND\text{-}1CCA}}_{\mathsf{KEM}}(\mathcal{B}_{i,j}).$$

Game G_2. This game is identical to Game $G_{N-1,S}$, in which all session keys $\mathbf{K}_0[i, j]$ for $(i, j) \in [N] \times [S]$ are uniformly random and independent. The adversary's view thus has the same distribution as in the original game with the challenge bit fixed to $b = 1$. Therefore,

$$\mathsf{Adv}^{\mathsf{MUC\text{-}IND\text{-}1CCA}}_{\mathsf{KEM},N,S}(\mathcal{A}) = \frac{1}{2}\left|\Pr[G^{\mathcal{A}}_0 \Rightarrow 1] - \Pr[G^{\mathcal{A}}_2 \Rightarrow 1]\right|.$$

Combining all the bounds together completes the proof. □

Table of Contents

Tightly Secure and Modular Post-Quantum TLS Handshakes 1
Biming Zhou⋆*, Jinrong Chen, Haodong Jiang, Rongmao Chen, and Yunlei Zhao*
1 Introduction 1
1.1 Our Contributions 2
1.2 Related Work 5
1.3 Technical Overview 5
2 Preliminaries 8
2.1 Notation and Multi-User KEM Security 8
2.2 Signatures and Pseudorandom Functions 10
2.3 HMAC and HKDF 11
3 PQ TLS 1.3 and KEMTLS: Protocols and Security Models 12
3.1 The PQ TLS 1.3 1-RTT Handshake 14
3.2 The KEMTLS Handshake 14
3.3 Model Syntax 15
3.4 Adversary Interaction 16
3.5 Multi-Stage Security Model for PQ TLS 1.3 17
3.6 Multi-Stage Security Model for KEMTLS 19
4 Tight Multi-Stage Security of the PQ TLS 1.3 Handshake 21
4.1 Security Analysis from MUC-IND-1CCA KEMs 21
4.2 Security Analysis from MUC-IND-1RCCA KEMs 25
5 Tight Multi-Stage Security of the KEMTLS Handshake in the ROM 28
5.1 Security Analysis for KEMTLS from MUC-IND-1CCA KEMs 29
5.2 Security Analysis for KEMTLS from MUC-IND-1RCCA KEMs 34
6 Concrete MU-(dual)-PRF Security Analysis of HMAC and HKDF 36
6.1 (Quantum-Accessible) Random Oracle Model 36
6.2 MU-PRF Security of HMAC in the QROM 37
6.3 MU-dual-PRF Security of HMAC in the ROM and QROM 39
6.4 Security of HKDF 42
A Reductions from MUC Security to Standard Security 47