

Bridging the Cryptographic Transition: A Unified Framework for Post-Quantum Key Exchange

Benjamin Dowling
King's College London
London, United Kingdom
benjamin.dowling@kcl.ac.uk

Bhagya Wimalasiri
King's College London
London, United Kingdom
bhagya.wimalasiri@kcl.ac.uk

Abstract

In a widely publicized recent announcement, Google established a formal 2029 deadline for completing its post-quantum cryptography (PQC) migration—one that deliberately preempts the 2035 deadline set by NIST and the US federal government to complete the deprecation of classical public-key algorithms. Google cited faster-than-expected progress across three fronts: quantum computing hardware development; quantum error correction; and revised estimates of the resources required for quantum computers to break current cryptographic schemes; as motivation for setting an ambitious internal target ahead of the regulatory horizon. This tightened deadline places particular pressure on existing cryptographic protocols that rely on classical key exchange primitives for secure communication. Our work aims to ease this transition by constructing a unified, formally analyzed security framework for migrating classical key exchange constructions to their post-quantum counterparts.

CCS Concepts

• Security and privacy;

Keywords

Key Exchange, Post-Quantum, Formalization, Provable Security

1 Introduction

In the three decades since Shor first published his seminal work introducing a quantum algorithm for factoring large integers and computing discrete logarithms in polynomial time [27], the prospect of a quantum computer capable of executing it has progressed from remote theoretical possibility through distant horizons, to an increasingly present and imminent reality. The United States National Security Agency published its CNSA 2.0 framework in 2022, providing a detailed roadmap for complete quantum-safe infrastructure migration by 2035—encompassing defence contractors, federal agencies, and regulated industries within its compliance scope [21]. In the same year, NIST announced its first post-quantum cryptography algorithm selections [16], three of which were subsequently standardized in 2024 (ML-KEM, ML-DSA, and SLH-DSA) [18–20]. Following these developments, other bodies have followed suit: the European Union, the United Kingdom's National Cyber Security Centre, and regulatory authorities in Canada, Australia, and Japan have each published coordinated roadmaps converging on the same 2035 deadline for comprehensive quantum-safe migration [24].

However, two concurrent yet independent events have altered the trajectory of the 2035 quantum migration deadline. First, Google published a whitepaper [4] reporting a substantial reduction in the quantum resource estimates (in terms of qubit count and gate complexity) required to execute Shor's algorithm against the 256-bit elliptic curve discrete logarithm problem (ECDLP-256), which underlies widely deployed curves such as NIST P-256 [17] and secp256k1 [28]. Shortly thereafter, Oratomic independently published a new quantum error-correction architecture [7] capable of breaking P-256 in as few as 10 days and RSA-2048 within 100 days on a neutral atom quantum computer. Together, these developments have prompted major industry stakeholders, including Google and Cloudflare [1, 30], to advance their quantum migration deadlines to as early as 2029.

Against this backdrop, we turn our attention to the impact of the post-quantum transition on secure communication protocols whose security rests on classical key exchange primitives rendered vulnerable by Shor's algorithm, specifically Diffie-Hellman and its elliptic curve variants. We observe that there already exists a rich body of literature proposing post-quantum-secure alternatives to existing cryptographic protocols such as Transport Layer Security, or the Signal Protocol. Notably, Schwabe et al. [25, 26] investigate post-quantum variants of TLS 1.3, proposing a KEM-based construction termed KEMTLS. The authors formally analyze its security within a multi-stage authenticated key exchange (AKE) model, capturing properties analogous to those of TLS 1.3, including key indistinguishability, forward secrecy, and both implicit and explicit authentication of communicating entities. This work specifically focuses on round-trip and bandwidth efficiency in the post-quantum transition, at the cost of deferring explicit server authentication and full forward secrecy by one round-trip compared to TLS 1.3.

Angel et al. [2] propose a post-quantum variant of the Noise protocol framework [23], analyzing the security of their construction within an adapted fACCE model [11] capturing confidentiality and authentication guarantees. Their PQNoise gives a recipe for translating a Noise pattern into a PQNoise pattern that preserves confidentiality and authenticity, at the possible cost of additional roundtrips. Moreover, by abstracting keying material processing into a hash-object, they further prove security for every valid PQNoise pattern at once.

Note that KEM constructions are inherently unable to derive shared secrets from static-static key combinations. In comparison, in the Diffie-Hellman setting, a static-static token yields a shared secret determined entirely by the two long-term keys, which survives compromise of either party's randomness. For a KEM, encapsulating under a static key affords no such guarantee, since the encapsulated secret depends on the sender's randomness. PQNoise



This work is licensed under a Creative Commons Attribution 4.0 International License.

uses *randomness hardening* as a means of circumventing this flaw, where static key material is mixed into the randomness source itself. This approach effectively lifts the cryptographic secrets into the entropy source itself, thereby exposing the cryptographic randomness underpinning the security of the entire construction to protocol implementers. We argue that this introduces unmitigated risks and represents a design choice that should be avoided.

In terms of secure messaging protocols, a significant number of works propose various post-quantum modifications to the Signal Protocol. Brendel et al. [6] introduce a novel split-KEM notion to adapt Signal’s X3DH handshake to the post-quantum setting, capturing indistinguishability of encapsulated keys; Dodis et al. [9] introduce Triple Ratchet, a hybrid post-quantum variant of the Signal Protocol, and analyze the security of their construction within a novel CKA model capturing key indistinguishability. Apple updated their iMessage secure messaging protocol to provide hybrid post-quantum security [3], with Stebila [29] analyzing the security of the PQ3 protocol within a multi-stage AKE model, capturing key indistinguishability. Finally, Benčina et al. [5] analyze the security of post-quantum SSH within an ACCE model, capturing notions of authentication and channel security. We note that the ACCE model presented in this work relies on a non-standard monolithic pseudo-random function assumption to underpin its security guarantees.

Although we acknowledge the significance of these works in facilitating the quantum migration, we observe that their protocol-by-protocol treatment introduces unnecessary fragmentation, wherein each construction independently retrofits quantum-secure KEMs in place of classical Diffie-Hellman exchanges and is analyzed within its own bespoke security model. We argue that this fragmentation is only compounded by the accelerated migration timeline, motivating the need for a unified, protocol-agnostic framework for quantum-secure protocol construction and analysis.

We observe that achieving a faithful one-to-one mapping between classical non-interactive DH-based primitives—which underpin a vast majority of deployed secure protocols—and their post-quantum KEM counterparts is, in practice, deeply non-trivial and that doing so without careful consideration of the associated security implications risks introducing vulnerabilities that should not be overlooked. For instance, Kret and Schmidt [13] propose a post-quantum variant of the X3DH key agreement protocol that incorporates an additional KEM key component as input to the key derivation function alongside the classical DH components. However, this approach treats post-quantum KEMs as interchangeable with their DH counterparts, overlooking fundamental structural distinctions that preclude a drop-in replacement. Specifically, the interactive nature of KEM primitives which consume fresh randomness per encapsulation, obviates the establishment of shared secrets from static key combinations in the way DH permits. This is precisely the limitation that Angel et al. [2] attempts to circumvent via *randomness hardening*, a design choice we argue introduces unmitigated security risks as discussed in the preceding section. Furthermore, should Shor’s algorithm be realized and the classical security of DH completely broken, the security of the derived PQXDH key would rest solely on the single KEM component. As we demonstrate in Section 4.3, KEM constructions exhibit strictly weaker resilience than their DH counterparts against certain classes of attacks—most notably, the inability to derive shared secrets from

static-static key combinations, which as a direct consequence precludes resilience against simultaneous randomness corruption of both parties in a two-party key exchange.

In light of these limitations, and with the urgency of the quantum migration increasing, this work aims to formally characterize the structural gap between DH and KEM primitives—providing a rigorous account of the security implications, tradeoffs, and constraints that arise in such a transition, and offering a systematic framework for reasoning about them in practice. With these goals in mind, we propose a generic and unifying framework to facilitate the post-quantum migration of key exchange constructions—herein referred to as KEX. The broader vision of our unified KEX framework is to equip practitioners with a general toolkit for streamlined and systematic transition to post-quantum key exchange constructions, offering the following capabilities:

- identification of the applicable DH-KEX pattern, its functionality, and its formal security guarantees;
- identification of design priorities for the target use case—resource constraints favoring lightweight constructions versus resource availability (bandwidth, memory, and computation) permitting stronger security guarantees;
- mapping the DH-KEX patterns to an appropriate KEM-KEX pattern satisfying the ideal constraints, and;
- formal security analysis within a unified formal security framework.

Our KEX framework thereby consolidates and offsets the effort of transitioning to post-quantum key exchange, whether by retrofitting existing constructions or designing novel quantum-resistant protocols, without compromising the rigor of the security analysis. We illustrate this vision in Figure 1.

Next we discuss the main contributions of our work.

Identifying fundamental DH-KEX and KEM-KEX patterns. We identify 29 fundamental patterns in total: 14 DH-KEX and 15 KEM-KEX. The patterns are underpinned by two axes: asymmetric key ownership and key lifespan: who “contributes” keys, and whether these keys are generated per session or reused across multiple sessions. From this perspective, the KEX framework constitutes a multi-user, multi-instance construction, where patterns emerge from the static key ownership across users and the ephemeral keys generated within independent instances. We construct our KEX patterns within such a multi-user multi-instance setting where two independent parties—an initiator *I* and a responder *R*—engage in a key exchange to establish a shared key(s).

Characterizing security guarantees and their mapping between DH-KEX to KEM-KEX. Of the 29 patterns identified, 12, 6 per KEX variant, yield distinct combinations of security guarantees across *key indistinguishability*, *implicit authentication*, and *forward secrecy*. Where a pattern additionally admits *anonymity*, we note this as a remark rather than a formally captured guarantee. This scoping decision reflects our aim to ground each fundamental KEX pattern in its minimal key exchange components, leaving practitioners free to augment their security with additional notions as required. In cases where patterns share identical security notions within a given KEX variant, we analyze one and note the others. Additionally, we

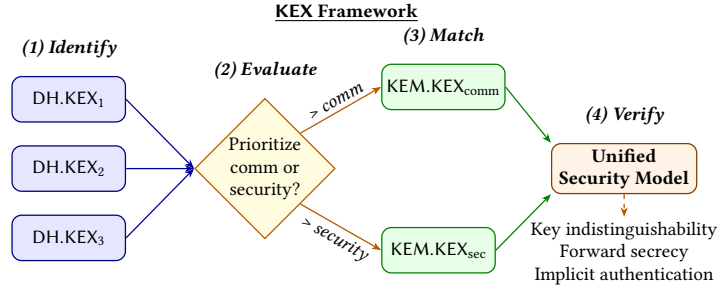


Figure 1: Overview of the KEX framework: practitioners (1) identify a DH-KEX pattern, (2) identify design priority, (3) match with an appropriate KEM-KEX pattern, and (4) verify formal security guarantees within unified security model.

collate the communication and security trade-offs between each DH-KEX pattern and its best-fit KEM-KEX migration target.

Formal security analysis of KEX within a unified security model. We provide formal security proofs for each unique KEX pattern within a unified security model. The model and its accompanying analysis are designed to facilitate practitioners seeking to replace DH-KEX patterns in existing constructions with appropriate KEM-KEX counterparts, with assurance of formal security guarantees and a systematized post-quantum migration process. We further note that our model and analysis support not only secure migration to post-quantum key exchange but also the design of novel post-quantum secure key exchange constructions.

Application of Framework to Existing Constructions. Using TLS 1.3 and Signal Double Ratchet as exemplar case studies, we demonstrate how our framework can be applied to systematically migrate existing DH-based protocols to their KEM-based counterparts, updating the associated security analysis to match the KEM primitive. We further provide a general overview of the communication and security implications arising from a DH-to-KEM transition across identified fundamental protocol patterns.

Organization. The remainder of this paper is organized as follows. Section 2 introduces the fundamental KEX patterns. Section 3 formalizes the KEX framework and its associated algorithms. Section 4 presents our security model, followed by the security analysis in Section 5. Section 6 concludes the paper with a concrete instantiation of our framework, using TLS 1.3 and Signal Double Ratchet as case studies, followed by a concluding discussion.

2 Fundamental KEX Patterns

In this section we describe the fundamental KEX patterns that underpin our framework. The notation for our KEX patterns draws from that of the Noise Protocol Framework [23], adapted to the specifics of our framework as described below. We emphasize that our KEX framework borrows only the notational components from Noise, not its design. We discuss in detail the distinctions between the Noise protocol framework and our work in Remark 1.

We broadly describe KEX as a primitive abstraction that models DH and KEM key exchange primitives uniformly and its KIND guarantees hold independently of any surrounding protocol. Additionally, compared to PQNoise that uses *randomness hardening* as

mitigation, KEX treats the absence of a static-static contribution in the post-quantum KEM setting as a property to be reflected rather than repaired.

When our KEX primitive is applied to an existing protocol, we are able to re-use any existing analysis of that protocol: our assumption establishes secrets exactly where the analysis established the indistinguishable keys from the key exchange primitive being replaced. It leaves the surrounding argument—its model, its adversarial interface, its remaining hops—untouched. This allows the KEX-based analysis to be re-used for any key exchange primitive that realizes the necessary KEX guarantees which, in particular, enables us to readily switch to post-quantum guarantees. We illustrate this versatility of KEX via its application to existing security analyses of TLS 1.3 and Signal Double Ratchet in Section 6.

For novel protocol designs, KEX supplies the key exchange component ready-made, with the security guarantees of the chosen pattern already established. Furthermore, beyond the KIND guarantees of the underlying key exchange primitive and a random oracle for DH key derivation, the concrete proofs for all KEX patterns depend on no further assumption.

Next, we describe our fundamental KEX patterns. The basic DH-KEX pattern XXX encodes static-key (*spk*) and ephemeral-key (*epk*) ownership by the initiator I and responder R. Positions are read left to right: the first two denote *spk* ownership by I and R, respectively, and the last two denote *epk* ownership likewise. For static keys, *K* indicates a preshared (or otherwise known) *spk*, *T* a transmitted *spk*, and *X* the absence of a static key. For ephemeral keys, *E* indicates presence and *X* absence.

The basic KEM-KEX pattern XXX encodes *spk* ownership by both parties and *epk* presence or absence by a single party. In our fundamental patterns we assign *epk* exclusively to I: since a single *epk* suffices to guarantee forward secrecy in KEM constructions, and that guarantee is independent of which party generates it, the distinction between I and R holding *epk* need not be represented. The first two positions of XXX encode *spk* ownership identically to DH-KEX; the third encodes *epk* presence (*E*) or absence (*X*). We omit KT patterns from our fundamental set for simplicity, as they arise straightforwardly from combining KK and TT patterns. We present our fundamental KEX patterns in Figure 2 along with their respective communication flows.

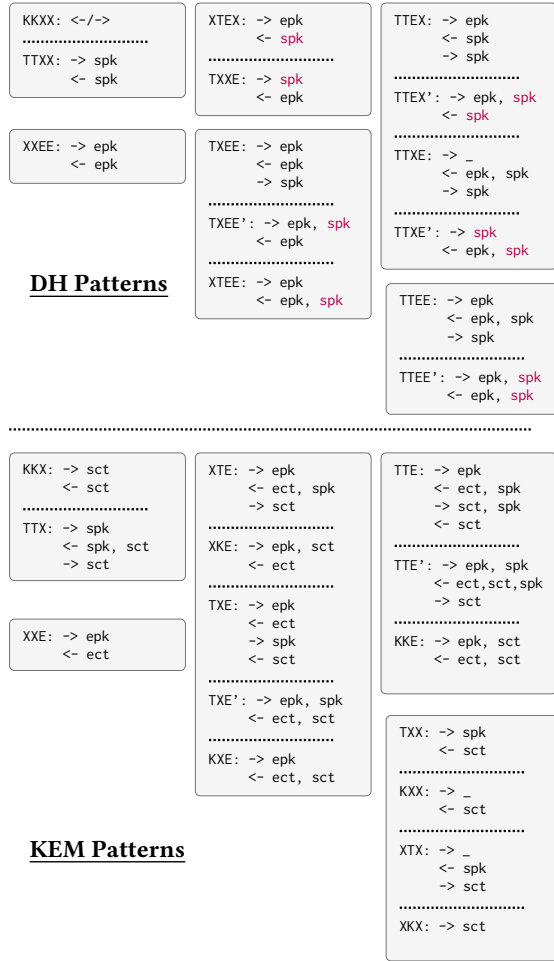


Figure 2: The fundamental KEX patterns, grouped by key contributions: static *spk* contributed by neither/one/both parties, and ephemeral *epk* contributed by neither/one/both parties. For KEM-KEX patterns, at most three keys are considered: *spk* for neither/one/both parties, and *epk* for I. Here *sct* and *ect* denote the ciphertexts produced by static and ephemeral key encapsulation, respectively. Prime KEX patterns preserve the key ownership of their non-prime counterparts while completing the key agreement in fewer round-trips. In the DH-KEX patterns, highlighted *spk* values indicate patterns for which a K-variant is obtained by simply presharing the highlighted *spk* without any further modification. For KEM-KEX, all K-variants are listed explicitly, since the interactive nature of KEM constructions alter the communication flow when *spk* is preshared.

REMARK 1 (COMPARISON TO THE NOISE PROTOCOL FRAMEWORK). *The Noise protocol [11, 23] framework (and its post-quantum counterpart PQNoise) is a suite of lightweight secure-channel constructions that provide practitioners with a set of ‘protocol patterns’ from which a developer may choose a protocol befitting the specific security needs*

of their application. In contrast, KEX is a primitive abstraction framework that recognizes different ‘patterns of interaction’ between two parties who communicate to establish a shared key through a cryptographic key exchange primitive. In this sense, Noise and KEX are markedly distinct in their functionality. Noise is a ready-made toolkit that incorporates a variety of cryptographic primitives for deploying secure-channel constructions, with support for different use cases and the associated trade-offs, such as latency and the need for key pre-sharing. KEX, by contrast, is a primitive abstraction with a more singular focus: capturing the KIND security of the underlying key exchange patterns and, in particular, preserving this security guarantee during the transition from classical to post-quantum cryptography. On the contrary, PQNoise cannot help any other protocol transition to post-quantum security, its only application is the Noise Protocol Framework. Our KEX primitive is distinct as well as narrower in its scope, which allows it to be applied to a broad range of cryptographic protocols, enabling generic transitions to post-quantum security

3 KEX Framework Construction

Here we formalize our KEX framework, explaining the expected functionality, corresponding algorithms and their outputs.

We consider a multi-user multi-instance key exchange construction KEX executed between two parties: an initiator I and a responder R. Such a KEX construction consists of a tuple of algorithms $\text{KEX} = (\text{LKGen}, \text{Start}, \text{Reply}, \text{Proc}, \text{Fin}, \text{Accept})$. We describe the algorithms below:

- $\text{LKGen}(1^\lambda) \xrightarrow{\$} (pk, sk, pid)$ is a potentially probabilistic key generation algorithm taking as input a security parameter 1^λ and outputting a public-key/secret-key pair (pk, sk) and (potentially) a party identifier pid . During execution LKGen is independently run by each party engaging in the KEX scheme. Depending on the instantiation, this algorithm models either long-term key-pair generation or the generation of *reusable* ephemeral key pairs by individual participants—such as the ephemeral public keys used in the Signal Double Ratchet protocol.
- $\text{Start}(sk_I, pk_R) \xrightarrow{\$} (\pi, \text{out}, k)$ is a potentially probabilistic algorithm, run by initiator, that takes as input potentially the output of the corresponding party’s LKGen execution, potentially the secret key from its own LKGen execution and outputs some state π , an externally visible output out (containing static and/or ephemeral public keys and/or encapsulated ciphertext) and a set of key(s) k . Depending on the instantiation Start may model,
 - a non-interactive predistributed key agreement by the initiator I or,
 - the initial round of a shared-key computation (potentially involving ephemeral key generation and/or ciphertext encapsulation under R’s static public key.
- $\text{Reply}(sk_R, cxt_R, pk_I) \xrightarrow{\$} (\pi, \text{out}, k)$ is a potentially probabilistic algorithm that takes as input potentially the output of the corresponding party’s LKGen execution, potentially the output of the corresponding party’s Start execution, and potentially the secret key from its own LKGen execution, and outputs some state π , an externally visible output

out (containing static and/or ephemeral public keys and/or encapsulated ciphertext) and key(s) k . Depending on the instantiation Reply may model,

- a non-interactive predistributed key agreement by the responder R or,
 - an intermediate round of a shared-key computation (potentially) involving ephemeral key generation and/or ciphertext encapsulation under I's static/ephemeral public key and/or ciphertext decapsulation under R's static secret key or,
 - final session key agreement and session update by responder R.
- $\text{Proc}(\pi, \text{ctxt}_I, pk_R) \xrightarrow{\$} (\pi, \text{out}, k)$ is a potentially probabilistic algorithm that takes as input some state π generated during Start (which potentially contains secret keys from I's own LKGen and/or Start executions), and the public-key output and/or encapsulated ciphertext of the corresponding party's Reply execution, and outputs some state π , an externally visible output out (containing I's static public key and/or the encapsulated ciphertext for R's static public key), and key(s) k . Depending on the instantiation Proc may model,
 - the final session key agreement and session update by the initiator I or,
 - an intermediate round of a shared-key computation (potentially) involving ciphertext encapsulation under R's static public key and/or ciphertext decapsulation under I's ephemeral secret key
 - $\text{Fin}(\pi, \text{ctxt}_I, pk_I) \xrightarrow{\$} (\pi, \text{out}, k)$ is a potentially probabilistic algorithm that takes as input some state π generated by Reply (which potentially contains the secret key from its own LKGen and/or Reply execution) and, the public-key output and/or encapsulated ciphertext from the initiator I's Proc execution, and outputs some state π , an externally visible output out (containing potentially the encapsulated ciphertext under I's static public key), and key(s) k . Fin models the final session key agreement and session update by the responder R.
 - $\text{Accept}(\pi, \text{ctxt}_I) \xrightarrow{\$} (\pi, \text{out}, k)$ is a deterministic algorithm that takes as input some state π , and the encapsulated ciphertext under initiator I's static public-key and outputs a set of key(s) k and some output out which for Accept contains either success ($\top$) or failure ($\perp$). Accept models an additional final stage in KEM-based KEX constructions, capturing session key agreement and the initiator I's session update.

Figures 3 and 4 illustrate the DH.TTEE and KEM.TTE patterns respectively, as instantiated within our framework. The remaining constructions are deferred to Appendix B.

REMARK 2 (KEX ALGORITHMS). *Our framework employs five algorithms to model key exchange between two parties for two reasons. First, we wish to capture every conceivable KEX construction, including settings where key predistribution is not possible. Second, the inherently interactive nature of KEM constructions requires additional processing steps compared to their non-interactive DH counterparts.*

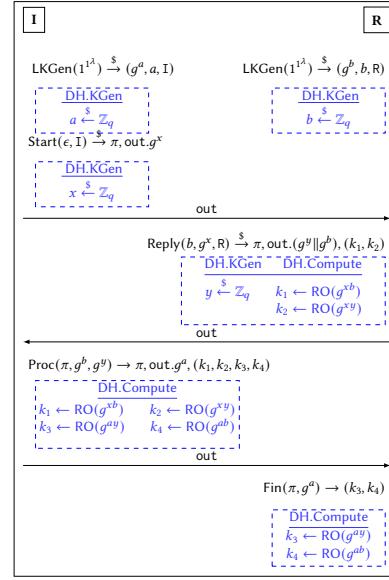


Figure 3: DH.TTEE Construction. Dashed boxes in blue provide cryptographic intuition behind each DH-KEX algorithm. Specifically, DH.KGen describes key generation (static or ephemeral), and DH.Compute describes the computation of shared keys. All raw DH keys are passed through random oracle RO. See Remark 5 for details.

A DH-KEX construction without static key predistribution, achieving the highest security level in our framework—key indistinguishability, forward secrecy, implicit authentication, and (informally) anonymity—requires at most four algorithms (Start, Reply, Proc, Fin) within 1.5 round trips. Achieving the same security level in the interactive KEM-KEX setting incurs one additional round trip and requires a fifth algorithm, Accept. This is because KEM encapsulation under a public key requires an extra round trip when static public key predistribution is not permitted. The five-algorithm design therefore accommodates KEX constructions across both the interactive and non-interactive paradigms, while handling restrictions on key predistribution.

4 KEX Security Model

A multi-user multi-instance key exchange protocol is a two-party protocol executed between an initiator I and responder R that securely establishes a shared key(s) among communicating parties over an insecure communication channel, ensuring that only the intended parties can derive the same secret key without revealing it to adversaries. The security of KEX is captured as a game played between an adversary and a challenger simulating parties executing a KEX protocol. In the game, the adversary interacts with the challenger's KEX executions via oracle queries that represent and abstract the capabilities of real-world attackers. For a KEX protocol, the adversary aims to distinguish between a “real” session key and a random key (as provided in the Test query) where breaking key indistinguishability implies some adversarial knowledge of the keys established in the KEX protocol. If the adversary can do this with

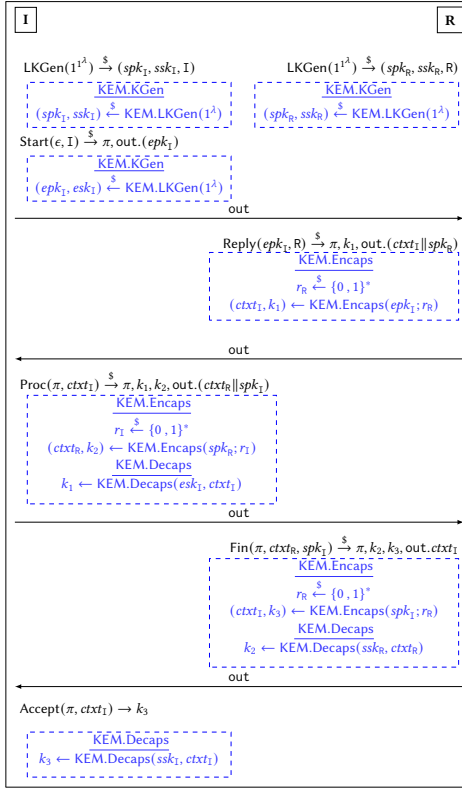


Figure 4: KEM.TTE Construction. Dashed boxes in blue provide cryptographic intuition behind each KEM-KEX algorithm. Specifically, KEM.KGen describes key generation (static or ephemeral), KEM.Encaps describes the encapsulation of a party’s public key together with the encapsulating party’s pseudorandom string r , and KEM.Decaps describes the decapsulation of a ciphertext ctxt under the corresponding secret key to recover a shared secret.

any non-negligible advantage (i.e., better than random guessing), then the KEX protocol is considered broken.

For all KEX patterns with distinct security guarantees, we prove key indistinguishability KIND as a baseline. Implicit authentication follows trivially for T/K-variants, and forward secrecy for EE-variants in DH-KEX and E-variants in KEM-KEX. Implicit authentication captures the guarantee that only the intended communicating parties can compute the shared secret produced by the KEX construction. For forward secrecy, we distinguish between *forward secrecy* and *weak forward secrecy*: the former denotes forward secrecy under mutual implicit authentication, while the latter denotes forward secrecy where either no authentication guarantee holds or implicit authentication holds for only one party.

A high-level description of a potential security game for an KEX protocol follows; a detailed description is shown in Figure 5. First, the challenger initializes the game, sets-up any necessary parameters, creates and records key pairs for the participating entities. The adversary can then interact with the challenger via a series of adversarial queries, allowing them to direct the execution

of the KEX between many simulated parties. Finally, the adversary outputs a bit guessing whether a given session key was honestly computed, or instead randomly sampled from the same distribution.

REMARK 3 (ADDITIONAL SECURITY GUARANTEES FOR AKE). Our framework is explicitly designed to focus on KEX at the primitive level; accordingly, indistinguishability (IND) serves as our primary security objective. We recognize that authenticated key exchange (AKE) protocols may capture additional security properties at the construction level, including transcript binding, deniability, and identity-hiding. We therefore discuss how our model can be extended to capture such additional properties.

We note that these security guarantees can be split into two categories: additive and subtractive. Additive properties are achieved by the higher-level protocol logic, and the KEX primitive cannot undermine them. Transcript binding is an example of an additive property where the construction guarantees the security property at the protocol-level (which holds independently of the constituent KEX primitive). More precisely, a KEX primitive neither guarantees nor undermines transcript binding; the protocol design achieves this (for example) by KDFs binding the session key to the transcript.

Conversely, subtractive properties are tightly coupled with all constituent cryptographic components of a construction, encompassing both the KEX primitive and the protocol itself. If any individual component violates the property, the corresponding guarantee does not hold for the protocol as a whole. Deniability and identity-hiding fall into this category. Consequently, for a protocol to guarantee these properties, they must hold across all constituent cryptographic components, including the underlying KEX primitive.

Broadly, we define an unlinkability (i.e., identity-hiding) property for an AKE-like protocol relative to an attacker seeking to link a user to particular session(s). Given the limited information provided by our KEX patterns, an adversary can break unlinkability by either linking a transmitted static public key (reused across sessions), or by inferring from the transmitted ciphertext(s) which public key was used to generate them, and thereby linking ephemeral ciphertexts. During the design of KEX, we anticipated the risks associated with plaintext transmission of public keys, and accordingly, several patterns are designed to prevent such attacks. For instance, in patterns such as TTEE and TTE, performing ephemeral shared key derivation(s) before transmitting static public keys enable the protocol to encrypt them if required.

With respect to ciphertext identity-hiding, the KEX framework inherits and preserves the property, but does not itself introduce it: if the underlying primitive is identity-hiding for the ciphertext, then so is KEX. We observe that Diffie-Hellman-based primitives are inherently identity-hiding for ciphertexts. Modeling a DH-based primitive as a KEM, the ‘ciphertext’—i.e., the responder’s public key-share—is sampled uniformly at random, and is therefore independent of the partner’s public key. Ciphertext identity-hiding in post-quantum KEM constructions has been studied by Xagawa [31] and Maram and Xagawa [15], who provide formal proofs of the property for numerous schemes, including the NIST-standardized ML-KEM.¹

¹These are QROM results resting on structural assumptions about the underlying PKE, and they address Kyber as submitted to the NIST process rather than ML-KEM as standardized.

$\text{Exp}_{\mathcal{K}, n_p, n_s}^{\text{KEX}, \text{clean}, \mathcal{A}}(1^\lambda):$ <pre> 1 $b \xleftarrow{\\$} \{0, 1\}, s, \text{ctr} \leftarrow 0, \text{KEY}, \text{IN}, \text{OUT} \leftarrow \emptyset, \text{out}, \text{in} \leftarrow \perp$ 2 for $i \in n_p$ do 3 $\text{KEX.LKGen}(1^\lambda) \xrightarrow{\\$} \text{spk}_i, \text{ssk}_i$ 4 $\text{PK} \xleftarrow{\cup} \text{spk}_i$ 5 end for 6 $\text{tested} \leftarrow (\perp, \perp, \perp)$ 7 $b' \leftarrow \mathcal{A}(\cdot)^{O(\text{Start}, \text{Reply}, \text{Proc}, \text{Fin}, \text{Accept}, \text{Corrupt}, \text{CorruptRand}, \text{Reveal}, \text{Test})}$ 8 $(i^*, s^*, \ell^*) \leftarrow \text{tested}$ 9 if $\neg \text{clean}(\pi_{s^*}^{i^*})$ then 10 return b 11 end if 12 return $(b' = b)$ </pre> $O\text{Reply}(i, \text{spk}, \text{in}) \rightarrow s, \text{out}:$ <pre> 1 $\text{PPK}_s^i \leftarrow \text{spk}$ 2 <i>// Track received key values</i> 3 $\text{IN}_s^i \leftarrow \text{in}$ 4 $\text{Reply}(\text{spk}, \text{in}, \text{R}) \xrightarrow{\\$} \pi_s^i, \text{out}, \vec{k}$ 5 <i>// Tracking sent values</i> 6 $\text{OUT}_s^i \xleftarrow{\cup} \text{out}$ 7 <i>// Tracking keys</i> 8 $\text{KEY}_s^i[\text{ctr}].\text{key} \xleftarrow{\cup} \vec{k}$ 9 $\text{ctr} \leftarrow \text{ctr} + 1$ 10 return s, out </pre> $O\text{Reveal}(i, s, \ell):$ <pre> 1 $\text{KEY}_s^i[\ell].\text{KFlag} \leftarrow \text{corrupt}$ 2 return $\text{KEY}_s^i[\ell].\text{key}$ </pre> $O\text{Test}(i, s, \ell):$ <pre> 1 if $(\text{KEY}_s^i[\ell].\text{KFlag} = \text{corrupt}) \vee ((\perp, \perp, \perp) \neq \text{tested})$ then 2 return $\perp$ 3 end if 4 $k_0 \xleftarrow{\\$} \mathcal{K}, k_1 \leftarrow \text{KEY}_s^i[\ell].\text{key}$ 5 $\text{tested} \leftarrow (i, s, \ell)$ 6 return k_b </pre>	$O\text{Start}(i, \text{spk}) \rightarrow s, \text{out}:$ <pre> 1 $\text{PPK}_s^i \leftarrow \text{spk}$ 2 $\text{Start}(\text{spk}, \text{I}) \xrightarrow{\\$} \pi_s^i, \text{out}, \vec{k}$ 3 <i>// Tracking sent key values</i> 4 $\text{OUT}_s^i \xleftarrow{\cup} \text{out}$ 5 <i>// Tracking keys</i> 6 $\text{KEY}_s^i[\text{ctr}].\text{key} \xleftarrow{\cup} \vec{k}$ 7 $\text{ctr} \leftarrow \text{ctr} + 1$ 8 return s, out </pre> $O\text{Proc}(i, s, \text{in}) \rightarrow s, \text{out}:$ <pre> 1 <i>// Track received key values</i> 2 $\text{IN}_s^i \leftarrow \text{in}$ 3 $\text{Proc}(\pi_s^i, \text{in}, \text{I}) \xrightarrow{\\$} \pi_s^i, \text{out}, \vec{k}$ 4 <i>// Track sent key values</i> 5 $\text{OUT}_s^i \xleftarrow{\cup} \text{out}$ 6 <i>// Track keys</i> 7 $\text{KEY}_s^i[\text{ctr}].\text{key} \xleftarrow{\cup} \vec{k}$ 8 $\text{ctr} \leftarrow \text{ctr} + 1$ 9 return s, out </pre> $O\text{Fin}(i, s, \text{in}) \rightarrow s, \text{out}:$ <pre> 1 <i>// Track received key values</i> 2 $\text{IN}_s^i \leftarrow \text{in}$ 3 $\text{Fin}(\pi_s^i, \text{in}, \text{R}) \xrightarrow{\\$} \pi_s^i, \text{out}, \vec{k}$ 4 <i>// Track sent key values</i> 5 $\text{OUT}_s^i \xleftarrow{\cup} \text{out}$ 6 <i>// Track keys</i> 7 $\text{KEY}_s^i[\text{ctr}].\text{key} \xleftarrow{\cup} \vec{k}$ 8 $\text{ctr} \leftarrow \text{ctr} + 1$ 9 return s, out </pre> $O\text{Accept}(i, s, \text{in}) \rightarrow \text{out}:$ <pre> 1 <i>// Track received key values</i> 2 $\text{IN}_s^i \leftarrow \text{in}$ 3 $\text{Accept}(\pi_s^i, \text{in}, \text{I}) \rightarrow \vec{k}, \text{out}$ 4 return out </pre> $O\text{Corrupt}(i):$ <pre> 1 $\text{LSK}_i \leftarrow \text{corrupt}$ 2 return ssk_i </pre> $O\text{CorruptRand}(i, s):$ <pre> 1 $\text{RAND}_s^i \leftarrow \text{corrupt}$ 2 return $\pi_s^i, \\$ </pre>
--	---

Figure 5: Experiment for an adversary $\mathcal{A}$ against the key-indistinguishability security of protocol KEX. Our model allows adversary $\mathcal{A}$ to register dishonest long-term keys and inject adversarially chosen ephemeral keys into protocol executions. Such adversarial behaviors are tracked via the registers IN, OUT, PK and PPK.

For deniability, we observe that protocol designs achieving KEM-based authentication (such as ephemeral-static Diffie–Hellman) provide deniability guarantees—any ciphertext can be repudiated. Thus, authentication in all our patterns are implicit: capturing explicit authentication would be a protocol-layer decision. Similar to Noise, authentication resting on a symmetric token derived from the keys generated by the KEX primitive would be computable by either party and thus attributable to neither. Thus, since our corresponding KEX framework provides deniability, this property would be preserved if the composed protocol similarly facilitates deniability.

4.1 Execution Environment

During the experiment described in 5, the challenger maintains a series of variables and session states. Each session π that participates in the execution of the security model maintains the following session variables:

- $\rho \in \{\text{I}, \text{R}\}$: The role of the party in the current session.
- $i \in \{1, \dots, n_p\}$: Index of the session owner.
- $s \in \{1, \dots, n_s\}$: Current session index.
- ctr : Counter for keys.
- out/in : Stores public values from adversarial queries, initialized to $\perp$ and updated to $\perp/\top$ depending on the outcome. out : output returned to the adversary following an oracle query. in : input parsed from the adversary’s query prior to algorithm execution.

- OUT/IN : Registers tracking the outputs out returned to the adversary and the inputs in parsed from adversarial queries, respectively. Indexed by party and session identifiers.
- PK : Register tracking the honestly generated static public keys by the challenger.
- PPK : Register tracking static key input parsed from the adversary’s $O\text{Start}$ and $O\text{Reply}$ queries.
- KEY : Register tracking shared secrets. Indexed by party, session and key identifiers.

4.2 Adversarial Interaction

Here we describe the queries the adversary can make to the challenger. These oracles model different kinds of information that the adversary might gain access to in a real-world scenario. We thus capture an active adversary in control of the network, who attempts to compromise sessions via the following oracle queries:

- $O\text{Start}(i, \text{spk}) \rightarrow s, \text{out}$: allows $\mathcal{A}$ to initiate a KEX key exchange on behalf of party i with (potentially) static key spk . If spk is adversarially specified, the challenger validates and records it in the session state; in predistributed settings, it is additionally recorded in the PPK register. These checks allow the model to capture dishonestly generated static keys. Challenger internally runs Start and returns session identifier s and some output out to $\mathcal{A}$. Challenger also records any out and shared keys $\vec{k}$ in OUT , and KEY registers respectively.

- $O\text{Reply}(i, s, \text{spk}, \text{in}) \rightarrow s, \text{out}$: allows $\mathcal{A}$ to create a responder session π_s^i with (potentially) static key spk and with in set to the output of a preceding $O\text{Start}$ query. If spk is adversarially specified, the challenger validates and records it in the session state; in predistributed settings, it is additionally recorded in the PPK register. Challenger internally runs Reply and returns session identifier s and some output out to $\mathcal{A}$. Challenger also records all in , out and any shared keys $\tilde{k}$ in their respective registers.
- $O\text{Proc}(i, s, \text{in}) \rightarrow s, \text{out}$: allows $\mathcal{A}$ to invoke Proc for session π_s^i , with in set to the output of a preceding $O\text{Reply}$ query. The challenger runs Proc internally and returns the session identifier s and output out to $\mathcal{A}$, recording in , out , and any derived session keys $\tilde{k}$ in their respective registers.
- $O\text{Fin}(i, s, \text{in}) \rightarrow s, \text{out}$: allows $\mathcal{A}$ to invoke Fin for session π_s^i , with in set to the output of a preceding $O\text{Proc}$ query. The challenger runs Fin internally and returns the session identifier s and output out to $\mathcal{A}$, recording in , out , and any derived session keys $\tilde{k}$ in their respective registers.
- $O\text{Accept}(i, s, \text{in}) \rightarrow \text{out}$: allows $\mathcal{A}$ to invoke Accept for session π_s^i , with in set to the output of a preceding $O\text{Fin}$ query. The challenger runs Accept internally and returns output out set to $\perp/\top$ to $\mathcal{A}$, recording in , and any derived session keys $\tilde{k}$ in their respective registers.
- $O\text{Corrupt}(i, s)$: allows $\mathcal{A}$ to recover the long-term secrets of party i , which enables the framework to capture perfect forward secrecy. Challenger sets flag LSK_i to corrupt to track the corruption.
- $O\text{Reveal}(i, s, \ell)$: allows $\mathcal{A}$ to reveal *key* at index ℓ of the register KEY for session (i, s) . Challenger sets corresponding flag KFlag to corrupt to track key reveals. During the experiment, the adversary queries $O\text{Reveal}(i, s, \ell)$ with $\ell \in \{1, 2, 3, 4\}$ where each index targets a specific key as follows:
 - 1 - ephemeral-static *key*
 - 2 - ephemeral-ephemeral *key*
 - 3 - static-ephemeral *key*
 - 4 - static-static *key*

We note that our model enforces a global ordering on how keys are identified for Reveal queries within our clean Definitions. As a consequence, our constructions may compute shared secrets out of order, for instance, deriving k_3 without first computing k_0, k_1 or k_2 , which we consider a benign artifact of the ordering convention. This convention is adopted solely to maintain notational consistency and readability across our clean Definitions and formal analysis.

- $O\text{CorruptRand}(i, s)$: allows $\mathcal{A}$ to reveal the randomness for session (i, s) . Challenger sets corresponding flag RAND_s^i to corrupt to track the corruption.
- $O\text{Test}(i, s, \ell)$: returns to $\mathcal{A}$ the real-or-random *key* at index ℓ of the register KEY , computed by the *test session* π_s^i , allowing $\mathcal{A}$ to play the key indistinguishability game. This query can only be called once. Challenger sets session (i, s) to tested once $\mathcal{A}$ invokes $O\text{Test}$.

4.3 Cleanness Predicates

In this section, we describe, for selected KEX variants, the exact combinations of adversarial actions an adversary is allowed to perform without trivially breaking the construction. We defer the complete clean Definitions for all fundamental patterns to Appendix C, presenting in the main body only the most straightforward (i.e., XXEE and XE) and most complex (i.e., TTEE and TTE) KEX pattern variants as representative cases. We note that we provide explicit clean Definitions only for the K-variants KKXX and KX as baseline cases. Definitions for the remaining K-variants are omitted as they follow directly from their T-variant counterparts, differing only by an additional consistency check on the PPK register.

REMARK 4 (TRADEOFFS IN ADVERSARIAL ACTIONS). *We observe that there exists an inherent distinction between KEX-DH and KEX-KEM with respect to the permitted or forbidden adversarial actions necessary to guarantee KIND-security for each key, as summarized in Table 1. We label the corruption of static keys by flag LSK , indexed by party identifier, and the compromise of randomness by flag RAND , indexed by party and session identifiers. We draw attention to two observations.*

First, the corruption LSK_iLSK_j does not exist as a valid adversarial action in KEX-KEM constructions, as KEM key pairs are inherently asymmetric—encapsulation is performed by one party and decapsulation by the other. Second, simultaneous randomness compromise $\text{RAND}_s^i\text{RAND}_t^j$ for both parties cannot be permitted in any KEX-KEM variant, since randomness is consumed during both static and ephemeral key encapsulations—compromising both simultaneously would trivially reveal all shared secrets. These restrictions impose certain limitations relative to the DH setting, where an attacker may either fully compromise randomness or corrupt LSK_iLSK_j , provided at least one condition remains secure. This asymmetry must be accepted as an inherent cost of the DH-to-KEM transition, since KEM constructions are interactive in nature and cannot be mapped to their non-interactive DH counterparts in a one-to-one fashion.

In our construction described in Section 3 and the security model presented in Section 4, we abstract away the structural specifics of the variables out and in to avoid introducing unnecessary complexity into the discussion. However, the clean predicates must precisely capture the adversary’s capabilities, and therefore require a more fine-grained characterization of out and in . In particular, these variables may potentially contain ephemeral and/or static public keys of the communicating parties, i.e. the initiator I and the responder R . Since our model captures adversaries capable of injecting both ephemeral and static keys, the clean predicates must explicitly account for such behavior. Accordingly, within the clean predicates, we enforce the following specific structure on the variables out and in , as well as on the registers OUT and IN , which aggregate out and in , respectively:

- $\text{out} \leftarrow (\text{out}_0 \parallel \text{out}_1)$ (resp. $\text{in} \leftarrow (\text{in}_0 \parallel \text{in}_1)$), where out_0 (resp. in_0) contains the ephemeral public values epk/ect and out_1 (resp. in_1) contains the static public values spk/sct .
- For a given session π_s^i , $\text{OUT}_s^i[0]$ and $\text{OUT}_s^i[1]$ contain the sent ephemeral (epk/ect) and static (spk/sct) public values, respectively.

clean Pattern	LSK _i LSK _j	LSK _i RAND _t ⁱ	LSK _j RAND _s ⁱ	RAND _s ⁱ RAND _t ⁱ
KEX Pattern	DH.XXEE	-	-	-
	DH.TTX	corrupt	-	-
	DH.XTEX	-	-	corrupt
	DH.TTEX	corrupt	-	corrupt
	DH.TXEE	-	corrupt	-
	DH.TTEE	corrupt	corrupt	corrupt
	KEM.XXE	N/A	-	-
	KEM.TTX	N/A	corrupt	corrupt
	KEM.TXX	N/A	corrupt	-
	KEM.TXE	N/A	-	corrupt
	KEM.XTE	N/A	-	corrupt
	KEM.TTE	N/A	corrupt	corrupt

Table 1: Comparison of clean Corrupt patterns for KEX variants. For a given KEX pattern `corrupt` indicates an uncorrupted clean pattern where *at least one* pattern must hold. `corrupt` indicates a pattern the adversary is *forbidden* from corrupting/injecting to preserve KIND security. N/A indicates the pattern does not cryptographically exist for that KEX variant. – indicates the condition is not applicable.

- Similarly, $IN_s^i[0]$ and $IN_t^i[1]$ contain the received ephemeral and static public values for that session.

We start with the cleanness predicates for the the Diffie-Hellman KEX variants.

DEFINITION 1 ($\text{clean}_{\text{KEX-DH-XXEE}}$). For the XXEE pattern described in Figure 10, Test session (i, s) is clean for key ℓ in the security experiment defined in Figure 5, if the following conditions hold:

- (1) $RAND_s^i \neq \text{corrupt}$ (randomness not corrupted), **AND**
- (2) $\exists(j, t) \in n_P \times n_S$ s.t. $IN_s^i[0] = OUT_t^j[0]$, $RAND_t^j \neq \text{corrupt}$ (randomness not corrupted nor injected for partner), **AND**
- (3) $\text{Reveal}(i, s, \ell)$ has not been issued:
if $\pi_s^i, \rho = I$ (resp. R):
 $\forall(j, t) \in n_P \times n_S$ s.t. $(OUT_s^i[0] = IN_t^j[0] \wedge IN_s^i[0] = OUT_t^j[0])$, $\text{Reveal}(j, t, 2)$ has not been issued.

DEFINITION 2 ($\text{clean}_{\text{KEX-DH-TTEE}}$). For the TTEE pattern described in Figure 3, Test session (i, s) is clean for key ℓ in the security experiment defined in Figure 5, if the following conditions hold:

- (1) $LSK_i \neq \text{corrupt}$ (static secret not exposed), **OR**
- (2) $RAND_s^i \neq \text{corrupt}$ (randomness not corrupted), **AND**
- (3) $\exists(j, t) \in n_P \times n_S$ s.t. $IN_s^i[1] = PK[j]$, $LSK_j \neq \text{corrupt}$ (static secret not corrupted nor injected for partner), **OR**
- (4) $\exists(j, t) \in n_P \times n_S$ s.t. $IN_s^i[0] = OUT_t^j[0]$, $RAND_t^j \neq \text{corrupt}$ (randomness not corrupted nor injected for partner), **AND**
- (5) $\text{Reveal}(i, s, 1)$ has not been issued:
if $\pi_s^i, \rho = I$:
 $\forall(j, t) \in n_P \times n_S$ s.t. $(IN_s^i[1] = PK[j] \wedge OUT_s^i[0] = IN_t^j[0])$, $\text{Reveal}(j, t, 1)$ has not been issued.

elif $\pi_s^i, \rho = R$:

$\forall(j, t) \in n_P \times n_S$ s.t. $(OUT_s^i[1] = IN_t^j[1] \wedge IN_s^i[0] = OUT_t^j[0])$, $\text{Reveal}(j, t, 1)$ has not been issued, **OR**

(6) $\text{Reveal}(i, s, 2)$ has not been issued:

if $\pi_s^i, \rho = I$ (resp. R):

$\forall(j, t) \in n_P \times n_S$ s.t. $(OUT_s^i[0] = IN_t^j[0] \wedge IN_s^i[0] = OUT_t^j[0])$, $\text{Reveal}(j, t, 2)$ has not been issued, **OR**

(7) $\text{Reveal}(i, s, 3)$ has not been issued:

if $\pi_s^i, \rho = I$:

$\forall(j, t) \in n_P \times n_S$ s.t. $(OUT_s^i[1] = IN_t^j[1] \wedge IN_s^i[0] = OUT_t^j[0])$, $\text{Reveal}(j, t, 3)$ has not been issued.

elif $\pi_s^i, \rho = R$:

$\forall(j, t) \in n_P \times n_S$ s.t. $(IN_s^i[1] = PK[j] \wedge OUT_s^i[0] = IN_t^j[0])$, $\text{Reveal}(j, t, 3)$ has not been issued, **OR**

(8) $\text{Reveal}(i, s, 4)$ has not been issued:

if $\pi_s^i, \rho = I$ (resp. R):

$\forall(j, t) \in n_P \times n_S$ s.t. $(OUT_s^i[1] = IN_t^j[1] \wedge IN_s^i[1] = PK[j])$, $\text{Reveal}(j, t, 4)$ has not been issued.

Next, we define the cleanness predicates for the KEM KEX variants.

DEFINITION 3 ($\text{clean}_{\text{KEX-KEM-XXE}}$). For the XXE pattern described in Figure 17, Test session (i, s) is clean for key ℓ in the security experiment defined in Figure 5, if the following conditions hold:

- (1) $RAND_s^i \neq \text{corrupt}$ (randomness not corrupted) **AND**
 $\exists(j, t) \in n_P \times n_S$ s.t. $(OUT_s^i[0] = IN_t^j[0] \wedge IN_s^i[0] = OUT_t^j[0])$,
 $RAND_t^j \neq \text{corrupt}$ (randomness not corrupted for partner session), **AND**
 - (2) $\text{Reveal}(i, s, 2)$ has not been issued, **AND**
if $\pi_s^i, \rho = I$:
 $\forall(j, t) \in n_P \times n_S$ s.t. $(OUT_s^i[0] = IN_t^j[0] \wedge IN_s^i[0] = OUT_t^j[0])$, $\text{Reveal}(j, t, 2)$ has not been issued.
- elif** $\pi_s^i, \rho = R$:
- $\forall(j, t) \in n_P \times n_S$ s.t. $(IN_s^i[0] = OUT_t^j[0] \wedge OUT_s^i[0] = IN_t^j[0])$, $\text{Reveal}(j, t, 2)$ has not been issued.

DEFINITION 4 ($\text{clean}_{\text{KEX-KEM-TTE}}$). For the TTE pattern described in Figure 4, Test session (i, s) is clean for key ℓ in the security experiment defined in Figure 5, if the following conditions hold:

- (1) $LSK_i \neq \text{corrupt}$ (static secret has not been exposed) **AND**
 $\exists(j, t) \in n_P \times n_S$ s.t. $(PK[i] = IN_t^j[1] \wedge IN_s^i[1] = OUT_t^j[1])$,
 $RAND_t^j \neq \text{corrupt}$ (randomness not corrupted for partner), **OR**
- (2) $RAND_s^i \neq \text{corrupt}$ (randomness not corrupted) **AND**
 $\exists(j, t) \in n_P \times n_S$ s.t. $(IN_s^i[1] = PK[j] \wedge OUT_s^i[1] = IN_t^j[1])$,
 $LSK_j \neq \text{corrupt}$ (static secret not exposed for partner), **OR**
- (3) $RAND_s^i \neq \text{corrupt}$ (randomness not corrupted) **AND**
 $\exists(j, t) \in n_P \times n_S$ s.t. $(OUT_s^i[0] = IN_t^j[0] \wedge IN_s^i[0] = OUT_t^j[0])$,
 $RAND_t^j \neq \text{corrupt}$ (randomness not corrupted for partner session), **AND**
- (4) $\text{Reveal}(i, s, 1)$ has not been issued, **AND**
if $\pi_s^i, \rho = I$:
 $\forall(j, t) \in n_P \times n_S$ s.t. $(IN_s^i[1] = PK[j] \wedge OUT_s^i[1] = IN_t^j[1])$, $\text{Reveal}(j, t, 1)$ has not been issued.

elif $\pi_s^i, \rho = R$:
 $\forall (j, t) \in n_P \times n_S$ s.t. $(\text{OUT}_s^i[1] = \text{IN}_t^j[1] \wedge \text{IN}_s^i[1]) = \text{OUT}_t^j[1]$, **Reveal**($j, t, 1$) *has not been issued*, **OR**
 (5) **Reveal**($i, s, 2$) *has not been issued*, **AND**
if $\pi_s^i, \rho = I$:
 $\forall (j, t) \in n_P \times n_S$ s.t. $(\text{OUT}_s^i[0] = \text{IN}_t^j[0] \wedge \text{IN}_s^i[0] = \text{OUT}_t^j[0])$, **Reveal**($j, t, 2$) *has not been issued*.
elif $\pi_s^i, \rho = R$:
 $\forall (j, t) \in n_P \times n_S$ s.t. $(\text{IN}_s^i[0] = \text{OUT}_t^j[0] \wedge \text{OUT}_s^i[0] = \text{IN}_t^j[0])$, **Reveal**($j, t, 2$) *has not been issued*, **OR**
 (6) **Reveal**($i, s, 3$) *has not been issued*, **AND**
if $\pi_s^i, \rho = I$:
 $\forall (j, t) \in n_P \times n_S$ s.t. $(\text{OUT}_s^i[1] = \text{IN}_t^j[1] \wedge \text{IN}_s^i[1] = \text{OUT}_t^j[1])$, **Reveal**($j, t, 3$) *has not been issued*.
elif $\pi_s^i, \rho = R$:
 $\forall (j, t) \in n_P \times n_S$ s.t. $(\text{IN}_s^i[1] = \text{PK}[j] \wedge \text{OUT}_s^i[1] = \text{IN}_t^j[1])$, **Reveal**($j, t, 3$) *has not been issued*.

Finally, we formalize the advantage of an adversary $\mathcal{A}$ in winning the KEX key indistinguishability experiment in the following way:

DEFINITION 5 (KEX KEY INDISTINGUISHABILITY). Let KEX be a multi-user multi-instance key exchange construction, and $n_P, n_S \in \mathbb{N}$. For a particular given predicate *clean*, an algorithm $\mathcal{A}$, and some key-space distribution $\mathcal{K}$, we define the advantage of $\mathcal{A}$ in the KEX key-indistinguishability game to be,

$$\text{Adv}_{\mathcal{K}, n_P, n_S}^{\text{KEX}, \text{clean}, \mathcal{A}}(1^\lambda) = 2 \cdot \left| \Pr \left[\text{Exp}_{\mathcal{K}, n_P, n_S}^{\text{KEX}, \text{clean}, \mathcal{A}}(1^\lambda) = 1 \right] - \frac{1}{2} \right|. \quad (1)$$

We say that KEX is post-quantum KEX-secure if, for all QPT algorithms $\mathcal{A}$, $\text{Adv}_{\mathcal{K}, n_P, n_S}^{\text{KEX}, \text{clean}, \mathcal{A}}(1^\lambda)$ is negligible in the security parameter 1^λ . We say that KEX is classically KEX-secure if, for all PPT algorithms $\mathcal{A}$, $\text{Adv}_{\mathcal{K}, n_P, n_S}^{\text{KEX}, \text{clean}, \mathcal{A}}(1^\lambda)$ is negligible in the security parameter 1^λ .

5 Security Analysis

In this section we analyze the cryptographic security of KEX fundamental patterns TTEE and TTE by utilizing the KEX model described in Section 4. We defer the proofs for the remaining KEX patterns to the Appendix D. We divide our analysis by the KEX pattern variant—i.e., DH-KEX and KEM-KEX—and provide formal security proofs for our selected exemplar patterns. We omit dedicated security proofs for prime KEX patterns and patterns with preshared static keys (K-variants), as their analysis is identical to that of the corresponding transmitted (T-variant) counterpart, to which we direct the reader. We likewise omit explicit analysis for variants related by a swap of key ownership that leaves security guarantees unchanged, e.g. TTEX and TTXE.

Our formal analysis proceeds via a sequence of games. We bound the difference in the adversary's advantage in each game with the underlying cryptographic assumptions until the adversary reaches a game where the advantage of that game equals 0, which shows that adversary $\mathcal{A}$ cannot win with non-negligible advantage. Where a KEX variant produces multiple keys, we index the analysis by case and define $\mathcal{A}$'s advantage in Case X as $\text{Adv}_{\mathcal{K}, n_P, n_S, C_X}^{\text{KEX}, \text{clean}, \mathcal{A}}(1^\lambda)$. We present proof sketches in lieu of full proofs due to space constraints.

We begin by presenting the analysis of DH-KEX variants. For our DH-KEX analysis we index shared secrets as outputs of a random oracle as follows, where the first exponent corresponds to a key of I and the second to a key of R :

- $k_1 \leftarrow \text{RO}(g^{xb})$ (ephemeral-static)
- $k_2 \leftarrow \text{RO}(g^{xy})$ (ephemeral-ephemeral)
- $k_3 \leftarrow \text{RO}(g^{ay})$ (static-ephemeral)
- $k_4 \leftarrow \text{RO}(g^{ab})$ (static-static)

REMARK 5 (USE OF RANDOM ORACLES). Readers may question our use of random oracles in the security analysis of DH-KEX patterns, given that our constructions consist solely of pure KEX components and do not incorporate additional post-processing of raw DH outputs such as hashing or key derivation. Our reasoning is twofold. First, we intentionally model keys as the raw output of the key exchange operation, generalizing the construction so that the precise method of key derivation is delegated to individual implementations rather than rigidly enforced at the framework level. Second, to the best of our knowledge, no DDH-like assumption exists that simultaneously permits oracle access to secret keys and reasons about the indistinguishability of raw DH outputs without passing through a random oracle—as discussed earlier, this is precisely what motivates our use of GDH with a random oracle rather than a purely decisional assumption.

THEOREM 1 (TTEE.KEX SECURITY). The TTEE construction is KEX-secure with cleanness predicate Definition 2 (capturing implicit authentication and forward secrecy). That is, for any PPT algorithm $\mathcal{A}$ against the key indistinguishability game (defined in Figure 5), $\text{Adv}_{\mathcal{K}, n_P, n_S}^{\text{KEX}, \text{clean}, \mathcal{A}}(1^\lambda)$ is negligible under the GDH assumption.

PROOF. We split the analysis into four cases:

- **Case 1:** Corrupt has not been issued for spk_R nor CorruptRand has been issued for initiator session (i.e., ephemeral secret $x \xleftarrow{\$} \mathbb{Z}_p$ of I and static secret $b \xleftarrow{\$} \mathbb{Z}_p$ of R remain secure).
- **Case 2:** CorruptRand has not been issued for initiator nor responder sessions (i.e., ephemeral secrets $x \xleftarrow{\$} \mathbb{Z}_p$ of I and $y \xleftarrow{\$} \mathbb{Z}_p$ of R remain secure).
- **Case 3:** Corrupt has not been issued for spk_I nor CorruptRand has been issued for responder session (i.e., static secret $a \xleftarrow{\$} \mathbb{Z}_p$ of I and ephemeral secret $y \xleftarrow{\$} \mathbb{Z}_p$ of R remain secure).
- **Case 4:** Corrupt has not been issued for spk_I nor spk_R (i.e., static secrets $a \xleftarrow{\$} \mathbb{Z}_p$ of I and $b \xleftarrow{\$} \mathbb{Z}_p$ of R remain secure).

Here we provide a proof sketch for **Case 1**. **Game 0** is the initial KIND security game. In **Game 1** we rule out the low-probability event that two randomly-generated DH keys collide. We do this by programming the challenger to maintain a list L of all honestly generated DH private keys. If a private key appears twice, challenger aborts the experiment and $\mathcal{A}$ automatically loses. We say there are n_P static keys in the game and a maximum of 2 ephemeral private keys per session for a total maximum of $2n_S$ ephemeral secrets. This yields a total of $n_P + 2n_S$ DH keys in the list L , every pair of which must not collide—otherwise the challenger aborts the experiment. There are $\binom{|L|}{2}$ such pairs of DH keys to consider

and each DH key in L is in the same group of order q so it collides with another key in L with probability $1/q$ and gives us the bound $\frac{\binom{n_P+2n_S}{2}}{q}$. In **Game 2**, we guess the index (i, s) of the session π_s^i , and the index (j, t) of the partner π_t^j respectively, incurring a tightness loss of $n_P^2 \cdot n_S^2$. In **Game 3**, we replace the value g^{x^b} used by the test session (and potentially, its communicating partner) as the RO input for computing k_1 with an independent random $g^z \leftarrow \mathcal{G}$. We argue that if an adversary can distinguish this change, then we can build a reduction to break the GDH assumption. We construct a reduction $\mathcal{B}$ that initializes a GDH challenger receiving (g^α, g^β) , which it uses to replace the static key g^b and the ephemeral key g^x respectively. For every RO query on input in , $\mathcal{B}$ submits (g^α, g^β, in) to the GDH's DDH oracle. If the oracle returns true, then $in = g^{\alpha\beta}$ and $\mathcal{B}$ submits in as its GDH solution. If $\mathcal{A}$ never queries $g^{\alpha\beta}$, replacing it with a uniformly random g^z is undetectable since RO outputs on unqueried inputs are uniformly random. The only way for the adversary distinguish this change then is to query in to the RO. However, doing so allows our reduction to solve the GDH challenge.

Since $\mathcal{B}$ does not know α or β , it must simulate computations involving these values. For honest long-term keys g^t , $\mathcal{B}$ computes $(g^\alpha)^t, (g^\beta)^t$ using its knowledge of t . We note that in order to handle adversarially generated ephemeral keys g^t in a consistent manner, $\mathcal{B}$ samples random group elements g^z to represent each such computed DH value $(g^{\alpha t}, g^{\beta t}, \text{etc.})$ and records the associated tuples (g^α, g^t, g^z) . Consistency is maintained by checking all adversarial RO queries against stored tuples via the DDH oracle, reprogramming the RO accordingly to ensure an indistinguishable simulation. Proofs for **Case 2**, **3** and **4** follow identically to **Case 1** up to a change in notation, with $\mathcal{A}$ targeting keys k_2, k_3 and k_4 respectively. Thus we have:

$$\begin{aligned} \text{Adv}_{\mathcal{K}, n_P, n_S}^{\text{KEX, clean}, \mathcal{A}}(1^\lambda) &\leq \frac{4\binom{n_P+2n_S}{2}}{q} + n_P^2 n_S^2 \cdot \left(\text{Adv}_{\mathcal{K}, C_1}^{\text{GDH}, \mathcal{A}}(1^\lambda) + \right. \\ &\quad \left. \text{Adv}_{\mathcal{K}, C_2}^{\text{GDH}, \mathcal{A}}(1^\lambda) + \text{Adv}_{\mathcal{K}, C_3}^{\text{GDH}, \mathcal{A}}(1^\lambda) + \text{Adv}_{\mathcal{K}, C_4}^{\text{GDH}, \mathcal{A}}(1^\lambda) \right) \end{aligned} \quad (2)$$

□

Next, we proceed with the analysis of the KEM-KEX variants. For our KEM-KEX analysis we index shared secrets as follows:

- $k_1 \leftarrow \text{KEM.Encaps}(spk_R, r_I)$ (ephemeral-static)
- $k_2 \leftarrow \text{KEM.Encaps}(epk_I, r_R)$ (ephemeral-ephemeral)
- $k_3 \leftarrow \text{KEM.Encaps}(spk_I, r_R)$ (static-ephemeral)

THEOREM 2 (TTE.KEX SECURITY). *The TTE construction is KEX-secure with cleanness predicate Definition 4 (capturing implicit authentication and forward secrecy). That is, for any QPT algorithm $\mathcal{A}$ against the key indistinguishability game (defined in Figure 5), $\text{Adv}_{\mathcal{K}, n_P, n_S}^{\text{KEX, clean}, \mathcal{A}}(1^\lambda)$ is negligible under the ind-cca and ind-1cca security of KEM primitives.*

PROOF. We split the analysis into three cases:

- **Case 1:** CorruptRand has not been issued for initiator nor responder sessions.

- **Case 2:** Corrupt has not been issued for spk_R nor CorruptRand has been issued for initiator session.
- **Case 3:** Corrupt has not been issued for spk_I nor CorruptRand has been issued for responder session.

Here we provide a proof sketch for **Case 1**. **Game 0** is the initial KIND security game. In **Game 1** we rule out the low-probability event that two randomly-generated KEM keys collide. We do this by programming the challenger to maintain a list L of all honestly generated KEM private keys. If a private key appears twice, challenger aborts the experiment and $\mathcal{A}$ automatically loses. We say there are n_P static keys in the game and a maximum of 1 ephemeral private keys per session for a total maximum of n_S ephemeral secrets. This yields a total of $n_P + n_S$ KEM keys in the list L , none of which must collide—otherwise the challenger aborts the experiment. Each KEM private key is sampled from the keyspace SK and therefore the probability of collision is $1/|SK|$ thus giving us the bound $\frac{n_P+n_S}{|SK|}$. In **Game 2**, we guess the index (i, s) of the session π_s^i , and the index (j, t) of the partner π_t^j respectively, incurring a tightness loss of $n_P^2 \cdot n_S^2$. In **Game 3** challenger replaces the key k_2 derived in the test session π_s^i (resp. π_t^j) with the uniformly random and independent value $\tilde{k}_2$ by interacting with an ind-1cca KEM challenger, which restricts $\mathcal{A}$ to a single decapsulation oracle call in the following manner. We define a reduction $\mathcal{B}$ who interacts with a ind-1cca challenger and replaces the public key epk_I input of the initiator with the public-key $\tilde{epk}$ received from the challenger. When the responder should compute the ciphertext $ctxt_I$, $\mathcal{B}$ instead replaces $ctxt_I$ and k_2 with $\tilde{ctxt}$ and $\tilde{k}_2$ received from the ind-1cca challenger, respectively. If $\mathcal{A}$ injects a malicious ciphertext to the test session $\mathcal{B}$ will use their single call to ind-1cca decapsulation to obtain some key k' . An adversary capable of distinguishing this change is an adversary against the ind-1cca security of KEM.

Next, we proceed to **Case 2**. **Game 0** is the initial KIND security game. **Game 1** follows identical to **Case 1 Game 1**. In **Game 2**, we guess the index (i, s) of the session π_s^i , and the index (j, t) of the partner π_t^j respectively, incurring a tightness loss of $n_P^2 n_S^2$. In **Game 3** challenger replaces the key k_1 derived in the test session π_s^i (resp. π_t^j) with the uniformly random and independent value $\tilde{k}_1$ by interacting with an ind-cca KEM challenger. **Game 3** follows identical to **Case 1 Game 3**, up to a change in notation, except whenever party j requires the use of the secret key to decapsulate a ciphertext $ctxt' \neq ctxt$, $\mathcal{B}$ simply queries $ctxt'$ to the ind-cca challenger, and replaces the computation of k' with the output of the challenger. An adversary capable of distinguishing this change is an adversary against the ind-cca security of KEM. The analysis of **Case 3** is identical to **Case 2**, up to a change in notation, with $\mathcal{A}$ trying to distinguish $\tilde{k}_3$ from k_3 . Thus we have:

$$\begin{aligned} \text{Adv}_{\mathcal{K}, n_P, n_S}^{\text{KEX, clean}, \mathcal{A}}(1^\lambda) &\leq \frac{3(n_P + n_S)}{|SK|} + n_P^2 n_S^2 \cdot \left(\text{Adv}_{\mathcal{K}, C_1}^{\text{ind-1cca}, \mathcal{A}}(1^\lambda) + \right. \\ &\quad \left. \text{Adv}_{\mathcal{K}, C_2}^{\text{ind-cca}, \mathcal{A}}(1^\lambda) + \text{Adv}_{\mathcal{K}, C_3}^{\text{ind-cca}, \mathcal{A}}(1^\lambda) \right) \end{aligned} \quad (3)$$

□

6 Discussion and Applications

In this Section we discuss the practical applications and observed implications arising from our KEX framework. Table 2 consolidates the insights gained from our framework and its security analysis, providing a broad characterization of the implications of migrating from DH-KEX to KEM-KEX constructions. The comparison is intended to convey a realistic account of the unavoidable costs and tradeoffs inherent to the post-quantum KEX migration.

6.1 Application to Transport Layer Security

In this section we discuss how to apply our techniques to existing real-world cryptographic protocols, and how to update the analysis to capture our new approach. For our first motivating example, we use an existing game-based analysis of TLS 1.3 [10]. In TLS 1.3, the handshake protocol is effectively divided into two phases: the first phase (key exchange) establishes secret session keys between the two parties using ephemeral Diffie-Hellman, and the second phase (authentication) uses those keys to encrypt signatures that authenticate one or both parties.

Thus, we identify the Diffie-Hellman pattern used in TLS 1.3 as XXEE. Going to Table 2, we can see that the Diffie-Hellman pattern XXEE maps to KEM pattern XKE, achieving the same security with the same communication pattern. Thus, we are able to replace the exchange of Diffie-Hellman public keys with a KEM public key (on the first flow from the client) and a KEM encapsulating ciphertext (on the second flow from the server) without difficulty.

Now we need to demonstrate how to transition an existing proof of TLS 1.3 (that specifically uses Diffie-Hellman) to instead use our generic KEX assumption that enables a proof of both Diffie-Hellman and KEMs with the same security notions. We give an example high-level replacement of the Diffie-Hellman operations with our KEX syntax in Figure 6. Next, we identify the relevant proof step of the analysis of TLS 1.3 [10].

In **Theorem 5.2, Game B.2** the authors use a PRF-ODH assumption to replace the output of $HS = \text{HKDF.Extract}(\text{dES}, g^{xy})$ with a uniformly random string $\widetilde{HS}$. Our approach will now split this into two game hops: the first replaces the output of g^{xy} with a key sampled indistinguishable from the same distribution (or more accurately, an output from a random oracle); the second replaces the output of HKDF.Extract with a uniformly random string via a PRF assumption. Since the latter is straightforward, we focus on the first game hop.

Game B.2': In this game, we replace the derivation of the key g^{xy} established by the test session and its partner session with a uniformly random and independent $\widetilde{g^{xy}}$. We employ the KEX assumption with cleanness predicate $\text{clean}_{\text{KEX-DH-XXEE}}$ (note that the cleanness predicate $\text{clean}_{\text{KEX-KEM-XXE}}$ matches identically). Specifically, we turn any adversary that can distinguish this change against the KEX security of DH – XXEE. In particular, at the beginning of the experiment we initialize a KEX challenger. When the guessed test session owned by party i is required to compute Start , we instead issue a $\text{OStart}(i, \perp)$ query to the KEX challenger, which returns $s, \widetilde{\text{out}}$. The reduction saves s and replaces the out computed by the protocol with $\widetilde{\text{out}}$ output by the challenger. When the adversary issues a Send query to the guessed partner session, we replace the computation of Reply with a query

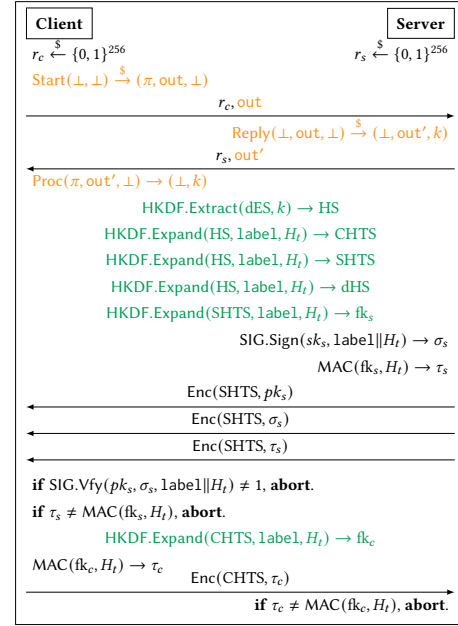


Figure 6: The TLS 1.3 full (EC)DHE handshake protocol with our KEX syntax replacing the Diffie-Hellman operations, highlighted in orange. Operations performed by both parties are centered in green.

to the KEX challenge $\text{OReply}(j, \perp, \widetilde{\text{out}})$, which outputs $t, \widetilde{\text{out}}'$, which the reduction uses to replace the computation of out' . The reduction immediately replaces the computation of k in both the test session and its communicating partner with the output of the $\text{OTest}(i, s, 2) \rightarrow \widetilde{k}$ query made to the KEX challenger. Thus, if the bit b sampled by the KEX challenger is 0, we are in Game B.1, otherwise we are in Game B.2, and any adversary distinguishing this change could be used to distinguish the bit b sampled by the KEX challenger and we have:

$$\text{Adv}_{\text{TLS1.3-fu11-1RTT}, \mathcal{A}}^{\text{GB.1}}(1^\lambda) \leq \text{Adv}_{\text{TLS1.3-fu11-1RTT}, \mathcal{A}}^{\text{GB.2}}(1^\lambda) + \text{Adv}_{\mathcal{K}, n_P, n_S}^{\text{XXEE, clean}, \mathcal{A}}(1^\lambda)$$

REMARK 6 (DISCUSSION ON TIGHTNESS). We highlight a useful feature of our KEX assumption. It is parameterized by n_P and n_S , denoting respectively the number of parties executing the protocol and the number of sessions per party. This parametrization allows KEX to be applied in two distinct ways: either as a security game in its own right, as illustrated by the security proofs for the various KEX patterns; or as an intermediate key establishment game-hop, inheriting the ‘guessing’ of sessions and partners from earlier hops in an existing security game without incurring additional security loss. This second usecase allows an assumption to be tailored to the specific game-hop in the security proof, thereby controlling the corresponding loss in reduction tightness.

For TLS 1.3, setting $n_P = 2$ and $n_S = 1$ eliminates the tightness loss introduced by the underlying security assumption, reducing it to a single PRF-ODH assumption.

6.2 Application to Signal Double Ratchet

For our second motivating example, we abstract an epoch of the Signal Double Ratchet protocol using our KEX primitive, and modify the existing game-based analysis of Signal Double Ratchet protocol [8]. Within the Signal protocol, the key establishment comprises two distinct phases: an initial key exchange, in which the X3DH protocol establishes a master secret, and the subsequent Double Ratchet protocol. The latter derives a root key and a chain key from the master secret, advancing the chain key with each message sent and the root key with each Diffie–Hellman ratchet step, thereby providing forward secrecy and post-compromise security.

Given the interleaved evolution of the sending and receiving ratchets, and the resulting complexity of capturing the protocol’s security within the KEX framework, we model the Double Ratchet protocol as an exemplar KEX use case, that highlights the flexibility of our framework. Unlike the symmetric ratchet, which solely applies a KDF to existing state, the asymmetric ratchet introduces new DH shared secrets into the protocol state. We capture this as an instance of the DH pattern XKEE between two parties Alice and Bob in Figure 7. Looking at Table 2, we see that DH pattern XKEE (i.e., the preshared variant of XTEE) maps to the KEM pattern XKE (i.e., the preshared variant of XTE’) that matches the roundtrip cost of XKEE while preserving its security guarantees.

We next demonstrate how a proof of the Signal Double Ratchet that relies on Diffie–Hellman can be transitioned to our generic KEX assumption, which supports both Diffie–Hellman and KEM-based constructions under the same security notions. We give an example high-level replacement of the Diffie–Hellman operations with our KEX syntax in Figure 7. We then identify the relevant proof step of the analysis of Signal Double Ratchet [8]. The core challenge in capturing the security of the Signal Double Ratchet within our framework is the reuse of ratchet keys both within and across epochs. The robustness of KEX primitive is what allows us to overcome this: because the primitive is flexible, we can choose a pattern that extends naturally across consecutive ratchet epochs. In each new epoch of asymmetric ratcheting, the last received ephemeral ratchet key of Bob becomes the new ‘long-term’ *known* ratchet key for Alice, thus illustrating an instance of *reusable* ephemeral keys within KEX.

In **Theorem 1 Case 3.1** the authors [8] substitute the ratchet keys $g^{rchk_A^u}$ and $g^{rchk_B^o}$ with the values from the GDH challenger. The original proof treats the two keys interchangeably. In contrast, our approach separates the analysis into two subcases to explicitly distinguish between *ephemeral–static* and *ephemeral–ephemeral* shared keys and the corresponding adversarial capabilities in each case.

Case 3.1.1: In this sub-case, we know that Corrupt has not been issued for receiver’s long-term key nor CorruptRand has been issued for initiator session (i.e., Alice’s ephemeral secret $rchk_A^x \xleftarrow{\$} \mathbb{Z}_p$ and Bob’s static secret $rchk_B^{x-1} \xleftarrow{\$} \mathbb{Z}_p$ remain secure).

Case 3.1.2: In this sub-case, we know that CorruptRand has been not been issued for either parties (i.e., Alice and Bob’s ephemeral secrets, $rchk_A^x$ and $rchk_B^x$, respectively, remain secure).

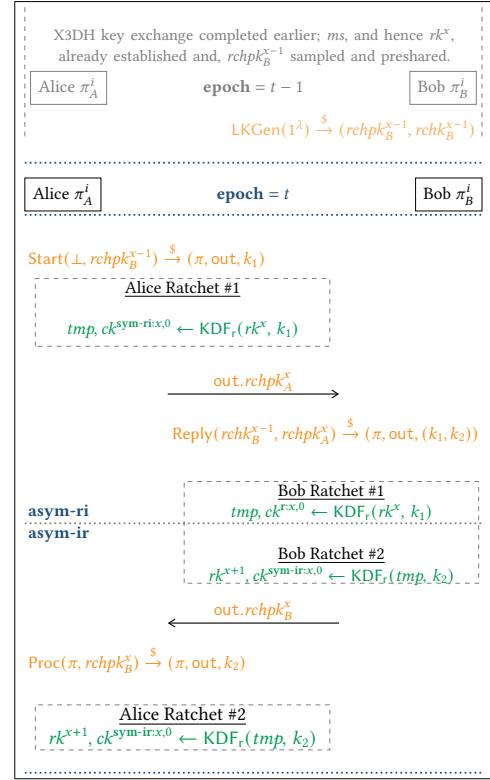


Figure 7: The Signal Double Ratchet protocol with our KEX syntax replacing the Diffie–Hellman operations, highlighted in orange. Operations necessary for the protocol but outside KEX are shown in green. This highlights an instance of modeling reusable ephemeral keys within KEX.

Next we provide a proof sketch for both subcases. We use the KEX assumption with cleanliness predicate $\text{clean}_{\text{KEX-DH-XKEE}}$, the preshared variant of XTEE; the cleanliness predicate for XKE, the preshared variant of XTE’, matches identically. At the beginning of the experiment for both subcases, we initialize a KEX challenger who runs $\text{LKGen}(1^\lambda) \xrightarrow{\$} \overline{rchk_B^{x-1}}, \overline{rchpk_B^{x-1}}$ and embeds the long-term key $\overline{rchpk_B^{x-1}}$ into party i for use in the simulation.

In **Case 3.1.1** we replace the derivation of the key k_1 for initiator (resp. responder) with uniformly random and independent $\tilde{k}$. Specifically, we turn any adversary that can distinguish this change against the KEX security of DH-XKEE. When the guessed test session—from the MSKE game of [8], which our KEX game inherits—owned by the initiator is required to compute Send for stage *asym-ri*, we instead issue a $\text{OStart}(i, \overline{rchpk_B^{x-1}})$ to the KEX challenger, which returns $s, \widetilde{out}$. The reduction saves s and replaces the out computed by the protocol with $\widetilde{out}$ output by the challenger. When the adversary issues a Send query to the guessed partner session, we replace the computation of Reply with a $\text{OReply}(j, \perp, \widetilde{out})$ query to the KEX challenger which outputs $t, \widetilde{out}'$, which the reduction uses to replace the computation of out' . The reduction may also issue Reveal queries

DH	Comm KEM	Communication Cost	Sec KEM	Security Cost
KXXX		× no KEM equivalent – non-interactive pattern		
TTXX	TTX	+0.5RTT, >[SIZE]	TTX	=[AUTH], ≠[KIND]
XTEX	XTX	+0.5RTT	XTX XTE	=[SEC] +[FS,RANON]
TXXE	TXX	✓	TXX TXE'	=[SEC] +[FS]
XXEE	XXEX	✓	XXEX	=[SEC]
TXEE	TXE'	−0.5RTT, >[SIZE]	TXE	=[SEC]
TXEE'	TXE'	✓, >[SIZE]	TXE'	=[SEC]
XTEE	XTX	+0.5RTT, ~~FS~~	XTE	=[SEC]
TTEX	TTE'	✓, >[SIZE]	TTE	+ [FS, ANON]
TTEX'	TTE'	+0.5RTT	TTE'	+ [FS, RANON]
TTXE	TTE'	✓, >[SIZE]	TTE'	+ [FS, RANON]
TTXE'	TTE'	+0.5RTT, >[SIZE]	TTE'	+ [FS]
TTEE	TTE'	✓, >[SIZE]	TTE'	= [SEC]
TTEE'	TTE'	+0.5RTT, >[SIZE]	TTE'	= [SEC]

×	no KEM match exists	✓	RTT-exact match	= [SEC]	identical security guarantees
+0.5RTT	half RTT overhead	−0.5RTT	half RTT saving	= [AUTH]	authentication preserved
>[SIZE]	increased message size	~~FS~~	forward secrecy lost	+ [FS]	additional forward secrecy
≠[KIND]	differs in key indistinguishability	+ [ANON]	additional anonymity	+ [RANON]	additional responder anonymity

Table 2: Mapping from DH-KEX to KEM-KEX patterns, preserving key-ownership structure. Comm KEM is the closest match by round-trip count; Sec KEM is the closest match by security guarantees—these may differ. Where two Sec KEM rows appear for one DH pattern, both options are viable with distinct security trade-offs. The communication cost of the KEM match is relative to its DH-KEX counterpart. For DH.TTXX security mapping, preserves = [AUTH] authentication security but key indistinguishability (KIND) differs due to randomness sensitivity in KEM constructions. = [SEC]: all security guarantees preserved. + []: additional guarantee over the DH variant. ANON/RANON denote full and responder anonymity; FS denotes forward secrecy; KIND denotes key indistinguishability. We note that the mappings identified represent best-case matches with respect to round-trip efficiency and security guarantees, and are not claimed to be exhaustive. We further note that additional bandwidth and round-trip optimizations may be achieved by mapping to KEM-KEX K-variants in settings where pre-distribution of static keys is feasible.

for any KEX key other than the Test-ed key k_1 , including k_2 . The reduction immediately replaces the computation of k_1 , in both the test session and its communicating partner, with the output of the query $O\text{Test}(i^*, s, 1) \rightarrow \tilde{k}$ made to the KEX challenger, where $i^* = i$ if the test session is owned by the initiator and $i^* = j$ if owned by the responder. Additionally, we need to replace the computation of k_2 : the reduction does so by issuing a $\text{Reveal}(i^*, s, 2)$ query and using that to compute the other key generated within the KEX challenger. We note that when the bit b sampled by the KEX challenger is 0, k_1 is computed normally, otherwise k_1 is replaced with a uniformly and random value $\tilde{k}$. Thus, any adversary distinguishing this change could be used to distinguish the bit b sampled by the KEX challenger and we have:

$$\text{Adv}_3^{C_{3.1.1}, \text{cleanEE}}(u, i, [0], [0]) (1^\lambda) \leq \text{Adv}_{\mathcal{K}, n_P, n_S}^{\text{XKEE}, \text{clean}, \mathcal{A}} (1^\lambda)$$

Now we proceed to **Case 3.1.2**. Here we replace the derivation of the key k_2 with uniformly random and independent $\tilde{k}$. This case follows identically to **Case 3.1.1** up to a change in notation, and

the protocol stage is now **asym-ir**. The reduction may also issue Reveal queries for any KEX key other than the Test-ed key k_2 , including k_1 . The reduction immediately replaces the computation of k_2 , in both the test session and its communicating partner, with the output of the query $O\text{Test}(i^*, s, 2) \rightarrow \tilde{k}$ made to the KEX challenger, where $i^* = i$ if the test session is owned by the initiator and $i^* = j$ if owned by the responder. Additionally, we need to replace the computation of k_1 : the reduction does so by issuing a $\text{Reveal}(i^*, s, 1)$ query and using that to compute the other key generated within the KEX challenger. We note that when the bit b sampled by the KEX challenger is 0, k_2 is computed normally, otherwise k_2 is replaced with a uniformly and random value $\tilde{k}$. Thus, any adversary distinguishing this change could be used to distinguish the bit b sampled by the KEX challenger and we have:

$$\text{Adv}_3^{C_{3.1.2}, \text{cleanEE}}(u, i, [0], [0]) (1^\lambda) \leq \text{Adv}_{\mathcal{K}, n_P, n_S}^{\text{XKEE}, \text{clean}, \mathcal{A}} (1^\lambda)$$

6.3 Discussion on Applications

As our applications to TLS 1.3 and the Signal Double Ratchet demonstrate, KEX can be used to accommodate both the TLS 1.3 handshake,

where each session establishes its keys afresh, and the Signal Double Ratchet, where the asymmetric ratchet reuses the same ephemeral key-shares both within and across epochs. The two cases exercise different aspects of the framework, and together they evidence its robustness. We note that the application to the Signal Double Ratchet proved the more nuanced of the two, further demonstrating the adaptability of the KEX framework. Our final mapping adopts the XKEE pattern, though the protocol admits several other valid mappings (e.g., TXEE, KKEE); the choice among them rests on the modeling requirements of the practitioner.

In each case, the security guarantees of the KEX assumption can be chosen to match the existing security analysis without requiring a complete re-proof of the protocol, and without introducing further loss in reduction tightness. Our KEX assumption is scoped specifically to the key exchange primitive guarantees of an analysis, and therefore the changes to the existing proof are correspondingly narrow: the only hops requiring replacement are those establishing the indistinguishability of the established secret. Whether that secret is established via DH or a post-quantum KEM makes no difference to the KEX abstraction.

Additionally, the absence of a tightness penalty follows from parameterizing the assumption by n_P and n_S . Because the existing proof has already guessed the party and the session to be tested, and the assumption is instantiated to match that guess, our game hop need not guess further to capture all parties and sessions.

We acknowledge that adapting the KEX assumption to existing analyses requires familiarity with the underlying security model and the accompanying proof structure. An ad-hoc alternative, however, would demand a full re-proof, whereas our framework confines the task to replacing the hops that establish key exchange secret indistinguishability, leaving the existing analysis intact while translating its guarantees to the post-quantum setting.

We further note that the framework does not currently capture implicit failures; extensions capturing both implicit and explicit failures is left for future work.

7 Acknowledgments

This research was supported by the UK Research and Innovation grant UKRI156.

References

- [1] Heather Adkins and Sophie Schmieg. 2026. Quantum Frontiers May Be Closer Than They Appear. <https://blog.google/innovation-and-ai/technology/safety-security/cryptography-migration-timeline/>. Accessed: 2026-04-20.
- [2] Yawning Angel, Benjamin Dowling, Andreas Hülsing, Peter Schwabe, and Florian Weber. 2022. Post quantum noise. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. 97–109.
- [3] Apple Security Engineering and Architecture (SEAR). 2024. iMessage with PQ3: The New State of the Art in Quantum-Secure Messaging at Scale. Apple Security Research Blog, <https://security.apple.com/blog/imessage-pq3/>. Accessed: 2026-04-20.
- [4] Ryan Babbush, Adam Zalcman, Craig Gidney, Michael Broughton, Tanuj Khattar, Hartmut Neven, Thiago Bergamaschi, Justin Drake, and Dan Boneh. 2026. Securing Elliptic Curve Cryptocurrencies against Quantum Vulnerabilities: Resource Estimates and Mitigations. *arXiv:2603.28846 [quant-ph]* <https://arxiv.org/abs/2603.28846>
- [5] Benjamin Benčina, Benjamin Dowling, Varun Maram, and Keita Xagawa. 2025. Post-quantum Cryptographic Analysis of SSH. *Cryptology ePrint Archive*, Paper 2025/684. <https://eprint.iacr.org/2025/684>
- [6] Jacqueline Brendel, Marc Fischlin, Felix Günther, Christian Janson, and Douglas Stebila. 2020. Towards post-quantum security for Signal’s X3DH handshake. In *International Conference on Selected Areas in Cryptography*. Springer, 404–430.
- [7] Madelyn Cain, Qian Xu, Robbie King, Lewis RB Picard, Harry Levine, Manuel Endres, John Preskill, Hsin-Yuan Huang, and Dolev Bluvstein. 2026. Shor’s algorithm is possible with as few as 10,000 reconfigurable atomic qubits. *arXiv preprint arXiv:2603.28627* (2026).
- [8] Katriel Cohn-Gordon, Cas Cremers, Benjamin Dowling, Luke Garratt, and Douglas Stebila. 2020. A Formal Security Analysis of the Signal Messaging Protocol. *Journal of Cryptology* 33, 4 (2020).
- [9] Yevgeniy Dodis, Daniel Jost, Shuichi Katsumata, Thomas Prest, and Rolfe Schmidt. 2025. Triple ratchet: A bandwidth efficient hybrid-secure signal protocol. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 302–331.
- [10] Benjamin Dowling, Marc Fischlin, Felix Günther, and Douglas Stebila. 2021. A cryptographic analysis of the TLS 1.3 handshake protocol. *Journal of Cryptology* 34, 4 (2021), 37.
- [11] Benjamin Dowling, Paul Rösler, and Jörg Schwenk. 2020. Flexible authenticated and confidential channel establishment (FACCE): Analyzing the noise protocol framework. In *IACR International Conference on Public-Key Cryptography*. Springer, 341–373.
- [12] Alex Escala, Gottfried Herold, Eike Kiltz, Carla Ràfols, and Jorge Villar. 2017. An Algebraic Framework for Diffie–Hellman Assumptions. *Journal of Cryptology* 30, 1 (01 Jan 2017), 242–288. doi:10.1007/s00145-015-9220-6
- [13] Ehren Kret and Rolfe Schmidt. 2023. The PQXDH key agreement protocol. *Signal* (2023). <https://signal.org/docs/specifications/pqxdh/pqxdh.pdf>
- [14] Caroline Kudla and Kenneth G Paterson. 2005. Modular security proofs for key agreement protocols. In *International conference on the theory and application of cryptography and information security*. Springer, 549–565.
- [15] Varun Maram and Keita Xagawa. 2023. Post-quantum anonymity of Kyber. In *IACR International Conference on Public-Key Cryptography*. Springer, 3–35.
- [16] National Institute of Standards and Technology. 2022. PQC Standardization Process: Announcing Four Candidates to be Standardized, Plus Fourth Round Candidates. <https://www.nist.gov/news-events/news/2022/07/pqc-standardization-process-announcing-four-candidates-be-standardized-plus>. Selects CRYSTALS-Kyber, CRYSTALS-Dilithium, FALCON, and SPHINCS+ for standardization following the third round of the NIST PQC Standardization Process. Rationale described in NIST IR 8413. Accessed: 2026-04-20.
- [17] National Institute of Standards and Technology. 2023. *SP 800-186: Recommendations for Discrete Logarithm-Based Cryptography: Elliptic Curve Domain Parameters*. NIST Special Publication 800-186. National Institute of Standards and Technology (NIST). Specifies elliptic curve domain parameters for U.S. Government use, including P-256 (Section 3). Accessed: 2026-04-20. doi:10.6028/NIST.SP-800-186
- [18] National Institute of Standards and Technology. 2024. *FIPS 203: Module-Lattice Based Key-Encapsulation Mechanism Standard*. Federal Information Processing Standard 203. National Institute of Standards and Technology (NIST). Accessed: 2026-04-20. doi:10.6028/NIST.FIPS.203
- [19] National Institute of Standards and Technology. 2024. *FIPS 204: Module-Lattice Based Digital Signature Standard*. Federal Information Processing Standard 204. National Institute of Standards and Technology (NIST). Accessed: 2026-04-20. doi:10.6028/NIST.FIPS.204
- [20] National Institute of Standards and Technology. 2024. *FIPS 205: Stateless Hash-Based Digital Signature Standard*. Federal Information Processing Standard 205. National Institute of Standards and Technology (NIST). Accessed: 2026-04-20. doi:10.6028/NIST.FIPS.205
- [21] National Security Agency. 2022. *Announcing Commercial National Security Algorithm Suite 2.0*. Cybersecurity Advisory U/OO/194427-22. National Security Agency (NSA). Accessed: 2026-04-20. <https://www.nsa.gov/Press-Room/News-Highlights/Article/Article/3148990/nsa-releases-future-quantum-resistant-qr-algorithm-requirements-for-national-se/>
- [22] Tatsuaki Okamoto and David Pointcheval. 2001. The gap-problems: A new class of problems for the security of cryptographic schemes. In *International workshop on public key cryptography*. Springer, 104–118.
- [23] Trevor Perrin. 2018. The noise protocol framework. <https://noiseprotocol.org/noise.html>. *noiseprotocol, Protocol Revision* 34 (2018).
- [24] PQShield. 2026. PQC Roadmaps and Transition Guidance. Accessed: 2026-04-19. <https://pqshield.com/pqc-transition-roadmaps-and-guidance/>
- [25] Peter Schwabe, Douglas Stebila, and Thom Wiggers. 2020. Post-quantum TLS without handshake signatures. In *Proceedings of the 2020 ACM SIGSAC conference on computer and communications security*. 1461–1480.
- [26] Peter Schwabe, Douglas Stebila, and Thom Wiggers. 2021. More efficient post-quantum KEMTLS with pre-distributed public keys. In *European Symposium on Research in Computer Security*. Springer, 3–22.
- [27] P.W. Shor. 1994. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*. 124–134. doi:10.1109/SFCS.1994.365700
- [28] Standards for Efficient Cryptography Group (SECG). 2010. *SEC 2: Recommended Elliptic Curve Domain Parameters*. Standard. Certicom Research. secp256k1 parameters specified in Section 2.4.1. Accessed: 2026-04-20. <https://www.secg.org/sec2-v2.pdf>

- [29] Douglas Stebila. 2024. Security analysis of the iMessage PQ3 protocol. Cryptology ePrint Archive, Paper 2024/357. (2024). <https://eprint.iacr.org/2024/357>
- [30] Bas Westerbaan. 2026. Cloudflare Targets 2029 for Full Post-Quantum Security. Cloudflare Blog, <https://blog.cloudflare.com/post-quantum-roadmap/>. Accessed: 2026-04-20.
- [31] Keita Xagawa. 2022. Anonymity of NIST PQC round 3 KEMs. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 551–581.

A Open Science

All artifacts necessary for evaluating the contributions of this submission are self-contained within this manuscript. These comprise a novel KEX framework capable of modeling both Diffie-Hellman and Key Encapsulation Mechanism based key exchange constructions, an accompanying security model capturing the desired security properties of such constructions, and a security analysis of the fundamental KEX patterns identified within our framework. We further demonstrate the practical applicability of our framework through a concrete instantiation using TLS 1.3 and Signal Double Ratchet a case studies. This work has not produced any external artifacts.

B Fundamental KEX Pattern Constructions

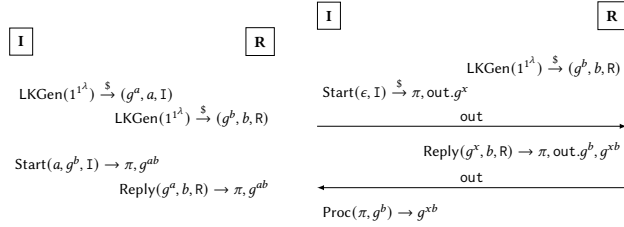


Figure 8: KKXX

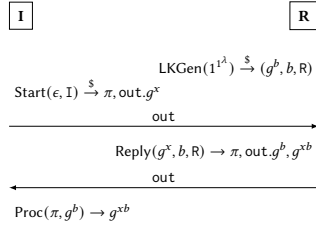


Figure 9: XTEX

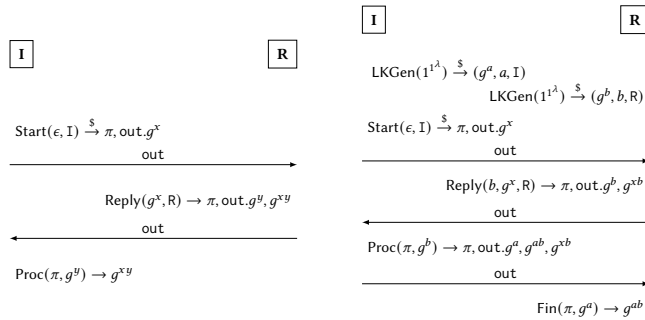


Figure 10: XXEE

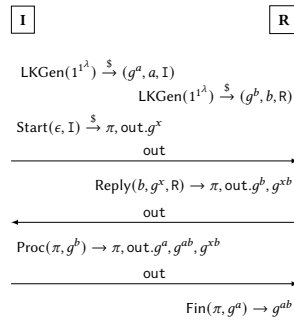


Figure 11: TTEX

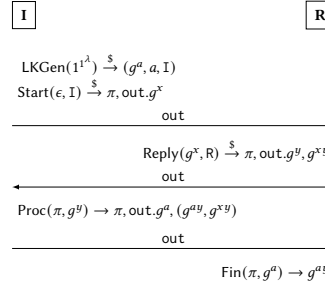


Figure 12: TXEE

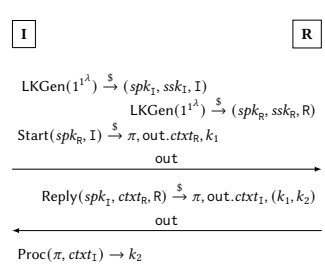


Figure 14: KKK

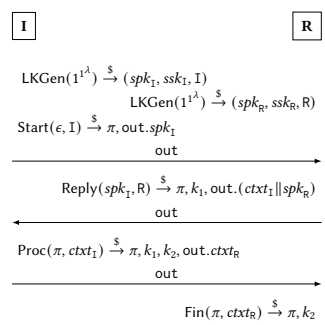


Figure 16: TTX

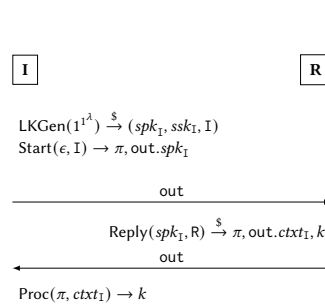


Figure 18: TXX

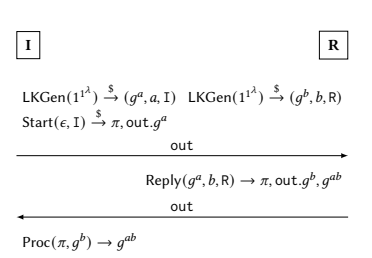


Figure 13: TTXX

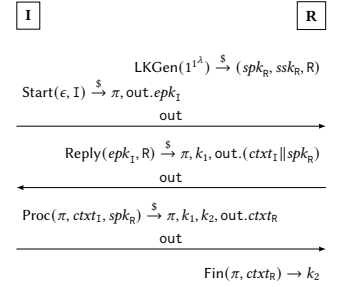


Figure 15: XTE

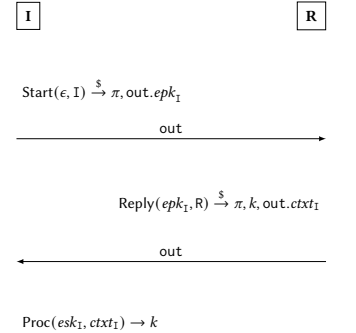


Figure 17: XXE

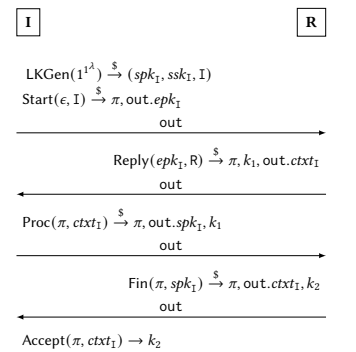


Figure 19: TXE

C Complete Cleanness Predicates

DEFINITION 6 ($\text{clean}_{\text{KEX-DH-KKXX}}$). For the KKXX pattern described in Figure 8, Test session (i, s) is *clean* for key ℓ in the security experiment defined in Figure 5, if the following conditions hold:

- (1) $\text{LSK}_i \neq \text{corrupt}$ (static secret not corrupted), **AND**
- (2) $\exists(j, t) \in n_P \times n_S$ s.t. $\text{PPK}_t^j.\text{spk} = \text{PK}[j]$, $\text{LSK}_j \neq \text{corrupt}$ (static key not corrupted for partner), **AND**
- (3) $\text{Reveal}(i, s, \ell)$ has not been issued:
if $\pi_s^i.\rho = \text{I}$ (resp. R):
 $\forall(j, t) \in n_P \times n_S$ s.t. $(\text{PPK}_s^i.\text{spk} = \text{PK}[i] \wedge \text{PPK}_t^j.\text{spk} = \text{PK}[j])$, $\text{Reveal}(j, t, 4)$ has not been issued.

DEFINITION 7 ($\text{clean}_{\text{KEX-DH-TTXX}}$). For the TTXX pattern described in Figure 13, Test session (i, s) is *clean* for key ℓ in the security experiment defined in Figure 5, if the following conditions hold:

- (1) $\text{LSK}_i \neq \text{corrupt}$ (static secret not corrupted), **AND**
- (2) $\exists(j, t) \in n_P \times n_S$ s.t. $\text{IN}_s^i[1] = \text{PK}[j]$, $\text{LSK}_j \neq \text{corrupt}$ (static secret not corrupted nor injected for partner), **AND**
- (3) $\text{Reveal}(i, s, \ell)$ has not been issued:
if $\pi_s^i.\rho = \text{I}$ (resp. R):
 $\forall(j, t) \in n_P \times n_S$ s.t. $(\text{OUT}_s^i[1] = \text{IN}_t^j[1] \wedge \text{IN}_s^i[1] = \text{PK}[j])$, $\text{Reveal}(j, t, 4)$ has not been issued.

DEFINITION 8 ($\text{clean}_{\text{KEX-DH-XTEX}}$). For the XTEX pattern described in Figure 9, Test session (i, s) is *clean* for key ℓ in the security experiment defined in Figure 5, if the following conditions hold:

- (1) $\text{RAND}_s^i \neq \text{corrupt}$ (randomness not corrupted), **AND**
- (2) $\exists(j, t) \in n_P \times n_S$ s.t. $\text{IN}_s^i[1] = \text{PK}[j]$, $\text{LSK}_j \neq \text{corrupt}$ (static secret not corrupted nor injected for partner), **AND**
- (3) $\text{Reveal}(i, s, 1)$ has not been issued:
if $\pi_s^i.\rho = \text{I}$:
 $\forall(j, t) \in n_P \times n_S$ s.t. $(\text{IN}_s^i[1] = \text{PK}[j] \wedge \text{OUT}_s^i[0] = \text{IN}_t^j[0])$, $\text{Reveal}(j, t, 1)$ has not been issued.
elif $\pi_s^i.\rho = \text{R}$:
 $\forall(j, t) \in n_P \times n_S$ s.t. $(\text{OUT}_s^i[1] = \text{IN}_t^j[1] \wedge \text{IN}_s^i[0] = \text{OUT}_t^j[0])$, $\text{Reveal}(j, t, 1)$ has not been issued.

DEFINITION 9 ($\text{clean}_{\text{KEX-DH-TTEX}}$). For the TTEX pattern described in Figure 11, Test session (i, s) is *clean* for key ℓ in the security experiment defined in Figure 5, if the following conditions hold:

- (1) $\text{LSK}_i \neq \text{corrupt}$ (static secret not exposed), **OR**
- (2) $\text{RAND}_s^i \neq \text{corrupt}$ (randomness not corrupted), **AND**
- (3) $\exists(j, t) \in n_P \times n_S$ s.t. $\text{IN}_s^i[1] = \text{PK}[j]$, $\text{LSK}_j \neq \text{corrupt}$ (static secret not corrupted nor injected for partner), **AND**
- (4) $\text{Reveal}(i, s, 1)$ has not been issued:
if $\pi_s^i.\rho = \text{I}$:
 $\forall(j, t) \in n_P \times n_S$ s.t. $(\text{IN}_s^i[1] = \text{PK}[j] \wedge \text{OUT}_s^i[0] = \text{IN}_t^j[0])$, $\text{Reveal}(j, t, 1)$ has not been issued.
elif $\pi_s^i.\rho = \text{R}$:
 $\forall(j, t) \in n_P \times n_S$ s.t. $(\text{OUT}_s^i[1] = \text{IN}_t^j[1] \wedge \text{IN}_s^i[0] = \text{OUT}_t^j[0])$, $\text{Reveal}(j, t, 1)$ has not been issued, **OR**
- (5) $\text{Reveal}(i, s, 4)$ has not been issued:
if $\pi_s^i.\rho = \text{I}$ (resp. R):
 $\forall(j, t) \in n_P \times n_S$ s.t. $(\text{OUT}_s^i[1] = \text{IN}_t^j[1] \wedge \text{IN}_s^i[1] = \text{PK}[j])$, $\text{Reveal}(j, t, 4)$ has not been issued.

DEFINITION 10 ($\text{clean}_{\text{KEX-DH-TXEE}}$). For the TXEE pattern described in Figure 12, Test session (i, s) is *clean* for key ℓ in the security experiment defined in Figure 5, if the following conditions hold:

- (1) $\text{LSK}_i \neq \text{corrupt}$ (static secret not exposed), **OR**
- (2) $\text{RAND}_s^i \neq \text{corrupt}$ (randomness not corrupted), **AND**
- (3) $\exists(j, t) \in n_P \times n_S$ s.t. $\text{IN}_s^i[0] = \text{OUT}_t^j[0]$, $\text{RAND}_t^j \neq \text{corrupt}$ (randomness not corrupted nor injected for partner), **AND**
- (4) $\text{Reveal}(i, s, 2)$ has not been issued:
if $\pi_s^i.\rho = \text{I}$ (resp. R):
 $\forall(j, t) \in n_P \times n_S$ s.t. $(\text{OUT}_s^i[0] = \text{IN}_t^j[0] \wedge \text{IN}_s^i[0] = \text{OUT}_t^j[0])$, $\text{Reveal}(j, t, 2)$ has not been issued, **OR**
- (5) $\text{Reveal}(i, s, 3)$ has not been issued:
if $\pi_s^i.\rho = \text{I}$:
 $\forall(j, t) \in n_P \times n_S$ s.t. $(\text{OUT}_s^i[1] = \text{IN}_t^j[1] \wedge \text{IN}_s^i[0] = \text{OUT}_t^j[0])$, $\text{Reveal}(j, t, 3)$ has not been issued.
elif $\pi_s^i.\rho = \text{R}$:
 $\forall(j, t) \in n_P \times n_S$ s.t. $(\text{IN}_s^i[1] = \text{PK}[j] \wedge \text{OUT}_s^i[0] = \text{IN}_t^j[0])$, $\text{Reveal}(j, t, 3)$ has not been issued.

DEFINITION 11 ($\text{clean}_{\text{KEX-DH-XTEE}}$). For the XTEE pattern, Test session (i, s) is *clean* for key ℓ in the security experiment defined in Figure 5, if the following conditions hold:

- (1) $\text{RAND}_s^i \neq \text{corrupt}$ (randomness not corrupted), **AND**
- (2) $\text{LSK}_j \neq \text{corrupt}$ (partner's static secret not exposed), **OR**
- (3) $\exists(j, t) \in n_P \times n_S$ s.t. $\text{IN}_s^i[0] = \text{OUT}_t^j[0]$, $\text{RAND}_t^j \neq \text{corrupt}$ (randomness not corrupted nor injected for partner), **AND**
- (4) $\text{Reveal}(i, s, 1)$ has not been issued:
if $\pi_s^i.\rho = \text{I}$:
 $\forall(j, t) \in n_P \times n_S$ s.t. $(\text{IN}_s^i[1] = \text{PK}[j] \wedge \text{OUT}_s^i[0] = \text{IN}_t^j[0])$, $\text{Reveal}(j, t, 1)$ has not been issued.
elif $\pi_s^i.\rho = \text{R}$:
 $\forall(j, t) \in n_P \times n_S$ s.t. $(\text{OUT}_s^i[1] = \text{IN}_t^j[1] \wedge \text{IN}_s^i[0] = \text{OUT}_t^j[0])$, $\text{Reveal}(j, t, 1)$ has not been issued, **OR**
- (5) $\text{Reveal}(i, s, 2)$ has not been issued:
if $\pi_s^i.\rho = \text{I}$:
 $\forall(j, t) \in n_P \times n_S$ s.t. $(\text{IN}_s^i[1] = \text{PK}[j] \wedge \text{OUT}_s^i[0] = \text{IN}_t^j[0])$, $\text{Reveal}(j, t, 1)$ has not been issued.
elif $\pi_s^i.\rho = \text{R}$:
 $\forall(j, t) \in n_P \times n_S$ s.t. $(\text{OUT}_s^i[1] = \text{IN}_t^j[1] \wedge \text{IN}_s^i[0] = \text{OUT}_t^j[0])$, $\text{Reveal}(j, t, 1)$ has not been issued, **OR**
- (5) $\text{Reveal}(i, s, 2)$ has not been issued:
if $\pi_s^i.\rho = \text{I}$ (resp. R):
 $\forall(j, t) \in n_P \times n_S$ s.t. $(\text{OUT}_s^i[0] = \text{IN}_t^j[0] \wedge \text{IN}_s^i[0] = \text{OUT}_t^j[0])$, $\text{Reveal}(j, t, 2)$ has not been issued.

DEFINITION 12 ($\text{clean}_{\text{KEX-KEM-TTX}}$). For the TTX pattern described in Figure 16, Test session (i, s) is *clean* for key ℓ in the security experiment defined in Figure 5, if the following conditions hold:

- (1) $\text{LSK}_i \neq \text{corrupt}$ (static secret has not been exposed) **AND**
 $\exists(j, t) \in n_P \times n_S$ s.t. $(\text{PK}[i] = \text{IN}_t^j[1] \wedge \text{IN}_s^i[1] = \text{OUT}_t^j[1])$,
 $\text{RAND}_t^j \neq \text{corrupt}$ (randomness not corrupted for partner), **OR**
- (2) $\text{RAND}_s^i \neq \text{corrupt}$ (randomness not corrupted) **AND**
 $\exists(j, t) \in n_P \times n_S$ s.t. $(\text{IN}_s^i[1] = \text{PK}[j] \wedge \text{OUT}_s^i[1] = \text{IN}_t^j[1])$,
 $\text{LSK}_j \neq \text{corrupt}$ (static secret not exposed for partner), **AND**
- (3) $\text{Reveal}(i, s, 1)$ has not been issued, **AND**

if $\pi_s^i, \rho = I$:
 $\forall (j, t) \in n_P \times n_S$ s.t. $(\text{IN}_s^i[1] = \text{PK}[j] \wedge \text{OUT}_s^i[1] = \text{IN}_t^j[1]), \text{Reveal}(j, t, 1))$ has not been issued.
elif $\pi_s^i, \rho = R$:
 $\forall (j, t) \in n_P \times n_S$ s.t. $(\text{OUT}_s^i[1] = \text{IN}_t^j[1] \wedge \text{IN}_s^i[1] = \text{OUT}_t^j[1]), \text{Reveal}(j, t, 1)$ has not been issued, **OR**
 (4) $\text{Reveal}(i, s, 3)$ has not been issued, **AND**
if $\pi_s^i, \rho = I$:
 $\forall (j, t) \in n_P \times n_S$ s.t. $(\text{OUT}_s^i[1] = \text{IN}_t^j[1] \wedge \text{IN}_s^i[1] = \text{OUT}_t^j[1]), \text{Reveal}(j, t, 3))$ has not been issued.
elif $\pi_s^i, \rho = R$:
 $\forall (j, t) \in n_P \times n_S$ s.t. $(\text{IN}_s^i[1] = \text{PK}[j] \wedge \text{OUT}_s^i[1] = \text{IN}_t^j[1]), \text{Reveal}(j, t, 3)$ has not been issued.

DEFINITION 13 ($\text{clean}_{\text{KEX-KEM-KKX}}$). For the KKX pattern described in Figure 14, Test session (i, s) is clean for key ℓ in the security experiment defined in Figure 5, if the following conditions hold:

- (1) $\text{LSK}_i \neq \text{corrupt}$ (static secret not corrupted) **AND**
 $\exists (j, t) \in n_P \times n_S$ s.t. $(\text{PPK}_s^i.\text{spk} = \text{PK}[i] \wedge \text{IN}_s^i[1] = \text{OUT}_t^j[1]), \text{RAND}_t^j \neq \text{corrupt}$ (randomness not corrupted for partner), **OR**
- (2) $\text{RAND}_s^i \neq \text{corrupt}$ (randomness not corrupted) **AND**
 $\exists (j, t) \in n_P \times n_S$ s.t. $(\text{PPK}_t^j.\text{spk} = \text{PK}[j] \wedge \text{OUT}_s^i[1] = \text{IN}_t^j[1]), \text{LSK}_j \neq \text{corrupt}$ (static secret not exposed for partner session), **AND**
- (3) $\text{Reveal}(i, s, 1)$ has not been issued:
if $\pi_s^i, \rho = I$:
 $\forall (j, t) \in n_P \times n_S$ s.t. $(\text{PPK}_s^j.\text{spk} = \text{PK}[j] \wedge \text{OUT}_s^i[1] = \text{IN}_t^j[1]), \text{Reveal}(j, t, 1))$ has not been issued.
elif $\pi_s^i, \rho = R$:
 $\forall (j, t) \in n_P \times n_S$ s.t. $(\text{PPK}_s^i.\text{spk} = \text{PK}[i] \wedge \text{IN}_s^i[1] = \text{OUT}_t^j[1]), \text{Reveal}(j, t, 1)$ has not been issued, **OR**
- (4) $\text{Reveal}(i, s, 3)$ has not been issued:
if $\pi_s^i, \rho = I$:
 $\forall (j, t) \in n_P \times n_S$ s.t. $(\text{PPK}_s^i.\text{spk} = \text{PK}[i] \wedge \text{IN}_s^i[1] = \text{OUT}_t^j[1]), \text{Reveal}(j, t, 3)$ has not been issued.
elif $\pi_s^i, \rho = R$:
 $\forall (j, t) \in n_P \times n_S$ s.t. $(\text{PPK}_t^j.\text{spk} = \text{PK}[j] \wedge \text{OUT}_s^i[1] = \text{IN}_t^j[1]), \text{Reveal}(j, t, 3)$ has not been issued.

DEFINITION 14 ($\text{clean}_{\text{KEX-KEM-XTE}}$). For the XTE pattern described in Figure 15, Test session (i, s) is clean for key ℓ in the security experiment defined in Figure 5, if the following conditions hold:

- (1) $\text{RAND}_s^i \neq \text{corrupt}$ (randomness not corrupted) **AND**
 $\exists (j, t) \in n_P \times n_S$ s.t. $(\text{IN}_s^i[1] = \text{PK}[j] \wedge \text{OUT}_s^i[1] = \text{IN}_t^j[1]), \text{LSK}_j \neq \text{corrupt}$ (static secret not exposed for partner), **OR**
- (2) $\text{RAND}_s^i \neq \text{corrupt}$ (randomness not corrupted) **AND**
 $\exists (j, t) \in n_P \times n_S$ s.t. $(\text{OUT}_s^i[0] = \text{IN}_t^j[0] \wedge \text{IN}_s^i[0] = \text{OUT}_t^j[0]), \text{RAND}_t^j \neq \text{corrupt}$ (randomness not corrupted for partner session), **AND**
- (3) $\text{Reveal}(i, s, 1)$ has not been issued, **AND**
if $\pi_s^i, \rho = I$:
 $\forall (j, t) \in n_P \times n_S$ s.t. $(\text{IN}_s^i[1] = \text{PK}[j] \wedge \text{OUT}_s^i[1] = \text{IN}_t^j[1]), \text{Reveal}(j, t, 1))$ has not been issued.

elif $\pi_s^i, \rho = R$:
 $\forall (j, t) \in n_P \times n_S$ s.t. $(\text{OUT}_s^i[1] = \text{IN}_t^j[1] \wedge \text{IN}_s^i[1] = \text{OUT}_t^j[1]), \text{Reveal}(j, t, 1)$ has not been issued, **OR**
 (4) $\text{Reveal}(i, s, 2)$ has not been issued, **AND**
if $\pi_s^i, \rho = I$:
 $\forall (j, t) \in n_P \times n_S$ s.t. $(\text{OUT}_s^i[0] = \text{IN}_t^j[0] \wedge \text{IN}_s^i[0] = \text{OUT}_t^j[0]), \text{Reveal}(j, t, 2))$ has not been issued.
elif $\pi_s^i, \rho = R$:
 $\forall (j, t) \in n_P \times n_S$ s.t. $(\text{IN}_s^i[0] = \text{OUT}_t^j[0] \wedge \text{OUT}_s^i[0] = \text{IN}_t^j[0]), \text{Reveal}(j, t, 2)$ has not been issued.

DEFINITION 15 ($\text{clean}_{\text{KEX-KEM-TXX}}$). For the TXX pattern described in Figure 18, Test session (i, s) is clean for key ℓ in the security experiment defined in Figure 5, if the following conditions hold:

- (1) $\text{LSK}_i \neq \text{corrupt}$ (static secret has not been exposed) **AND**
 $\exists (j, t) \in n_P \times n_S$ s.t. $(\text{PK}[i] = \text{IN}_t^j[1] \wedge \text{IN}_s^i[1] = \text{OUT}_t^j[1]), \text{RAND}_t^j \neq \text{corrupt}$ (randomness not corrupted for partner), **AND**
- (2) $\text{Reveal}(i, s, 3)$ has not been issued, **AND**
if $\pi_s^i, \rho = I$:
 $\forall (j, t) \in n_P \times n_S$ s.t. $(\text{OUT}_s^i[1] = \text{IN}_t^j[1] \wedge \text{IN}_s^i[1] = \text{OUT}_t^j[1]), \text{Reveal}(j, t, 3)$ has not been issued.
elif $\pi_s^i, \rho = R$:
 $\forall (j, t) \in n_P \times n_S$ s.t. $(\text{IN}_s^i[1] = \text{PK}[j] \wedge \text{OUT}_s^i[1] = \text{IN}_t^j[1]), \text{Reveal}(j, t, 3)$ has not been issued.

DEFINITION 16 ($\text{clean}_{\text{KEX-KEM-TXE}}$). For the TXE pattern described in Figure 19, Test session (i, s) is clean for key ℓ in the security experiment defined in Figure 5, if the following conditions hold:

- (1) $\text{LSK}_i \neq \text{corrupt}$ (static secret has not been exposed) **AND**
 $\exists (j, t) \in n_P \times n_S$ s.t. $(\text{PK}[i] = \text{IN}_t^j[1] \wedge \text{IN}_s^i[1] = \text{OUT}_t^j[1]), \text{RAND}_t^j \neq \text{corrupt}$ (randomness not corrupted for partner), **OR**
- (2) $\text{RAND}_s^i \neq \text{corrupt}$ (randomness not corrupted) **AND**
 $\exists (j, t) \in n_P \times n_S$ s.t. $(\text{OUT}_s^i[0] = \text{IN}_t^j[0] \wedge \text{IN}_s^i[0] = \text{OUT}_t^j[0]), \text{RAND}_t^j \neq \text{corrupt}$ (randomness not corrupted for partner session), **AND**
- (3) $\text{Reveal}(i, s, 2)$ has not been issued, **AND**
if $\pi_s^i, \rho = I$:
 $\forall (j, t) \in n_P \times n_S$ s.t. $(\text{OUT}_s^i[0] = \text{IN}_t^j[0] \wedge \text{IN}_s^i[0] = \text{OUT}_t^j[0]), \text{Reveal}(j, t, 2))$ has not been issued.
elif $\pi_s^i, \rho = R$:
 $\forall (j, t) \in n_P \times n_S$ s.t. $(\text{IN}_s^i[0] = \text{OUT}_t^j[0] \wedge \text{OUT}_s^i[0] = \text{IN}_t^j[0]), \text{Reveal}(j, t, 2)$ has not been issued, **OR**
- (4) $\text{Reveal}(i, s, 3)$ has not been issued, **AND**
if $\pi_s^i, \rho = I$:
 $\forall (j, t) \in n_P \times n_S$ s.t. $(\text{OUT}_s^i[1] = \text{IN}_t^j[1] \wedge \text{IN}_s^i[1] = \text{OUT}_t^j[1]), \text{Reveal}(j, t, 3)$ has not been issued.
elif $\pi_s^i, \rho = R$:
 $\forall (j, t) \in n_P \times n_S$ s.t. $(\text{IN}_s^i[1] = \text{PK}[j] \wedge \text{OUT}_s^i[1] = \text{IN}_t^j[1]), \text{Reveal}(j, t, 3)$ has not been issued.

D Full Proofs

THEOREM 3 (XXEE.KEX SECURITY). The XXEE construction is KEX-secure with cleanness predicate Definition 1 (capturing weak forward

secrecy). That is, for any PPT algorithm $\mathcal{A}$ against the key indistinguishability game (defined in Figure 5), $\text{Adv}_{\mathcal{K},n_P,n_S}^{\text{KEX, clean}, \mathcal{A}}(1^\lambda)$ is negligible under the GDH assumption.

PROOF. Here we provide a proof sketch. By definition of clean we know CorruptRand has not been issued for either parties. Hence, the proofs for the theorem follows identical to **Case 2** of Theorem 1 where $\mathcal{A}$ tries to distinguish k_2 . Thus we have:

$$\text{Adv}_{\mathcal{K},n_P,n_S}^{\text{KEX, clean}, \mathcal{A}}(1^\lambda) \leq \frac{\binom{n_P+2n_S}{2}}{q} + n_P^2 n_S^2 \cdot \text{Adv}_{\mathcal{K},C_1}^{\text{GDH}, \mathcal{A}}(1^\lambda) \quad (4)$$

□

THEOREM 4 (TTXX.KEX SECURITY). *The TTXX construction is KEX-secure with cleanness predicate Definition 7 (capturing implicit authentication). That is, for any PPT algorithm $\mathcal{A}$ against the key indistinguishability game (defined in Figure 5), $\text{Adv}_{\mathcal{K},n_P,n_S}^{\text{KEX, clean}, \mathcal{A}}(1^\lambda)$ is negligible under the GDH assumption.*

PROOF. Here we provide a proof sketch. By definition of clean we know CorruptRand has not been issued for either parties. Hence, the proofs for the theorem follows identical to **Case 4** of Theorem 1 where $\mathcal{A}$ tries to distinguish k_4 . Thus we have:

$$\text{Adv}_{\mathcal{K},n_P,n_S}^{\text{KEX, clean}, \mathcal{A}}(1^\lambda) \leq \frac{\binom{n_P+2n_S}{2}}{q} + n_P^2 n_S^2 \cdot \text{Adv}_{\mathcal{K},C_1}^{\text{GDH}, \mathcal{A}}(1^\lambda) \quad (5)$$

□

THEOREM 5 (XTEX.KEX SECURITY). *The XTEX construction is KEX-secure with cleanness predicate Definition 8 (capturing implicit responder authentication). That is, for any PPT algorithm $\mathcal{A}$ against the key indistinguishability game (defined in Figure 5), $\text{Adv}_{\mathcal{K},n_P,n_S}^{\text{KEX, clean}, \mathcal{A}}(1^\lambda)$ is negligible under the GDH assumption.*

PROOF. Here we provide a proof sketch. By definition of clean we know CorruptRand has not been issued for initiator nor Corrupt for responder. Hence, the proofs for the theorem follows identical to **Case 1** of Theorem 1 where $\mathcal{A}$ tries to distinguish k_1 . Thus we have:

$$\text{Adv}_{\mathcal{K},n_P,n_S}^{\text{KEX, clean}, \mathcal{A}}(1^\lambda) \leq \frac{\binom{n_P+2n_S}{2}}{q} + n_P^2 n_S^2 \cdot \text{Adv}_{\mathcal{K},C_1}^{\text{GDH}, \mathcal{A}}(1^\lambda) \quad (6)$$

□

THEOREM 6 (TTEX.KEX SECURITY). *The TTEX construction is KEX-secure with cleanness predicate Definition 9 (capturing implicit authentication). That is, for any PPT algorithm $\mathcal{A}$ against the key indistinguishability game (defined in Figure 5), $\text{Adv}_{\mathcal{K},n_P,n_S}^{\text{KEX, clean}, \mathcal{A}}(1^\lambda)$ is negligible under the GDH assumption.*

PROOF. We split the analysis into two cases:

- **Case 1:** Corrupt has not been issued for spk_I nor spk_R .
- **Case 2:** CorruptRand has not been issued for initiator session and Corrupt has not been issued for spk_R .

Proofs for **Case 1** and **Case 2** follow identical to **Case 4** and **Case 1** of Theorem 1, respectively, where $\mathcal{A}$ targets k_4 and k_1 , respectively. Thus we have:

$$\text{Adv}_{\mathcal{K},n_P,n_S}^{\text{KEX, clean}, \mathcal{A}}(1^\lambda) \leq \frac{2\binom{n_P+2n_S}{2}}{q} + n_P^2 n_S^2 \cdot \left(\text{Adv}_{\mathcal{K},C_1}^{\text{GDH}, \mathcal{A}}(1^\lambda) + \text{Adv}_{\mathcal{K},C_2}^{\text{GDH}, \mathcal{A}}(1^\lambda) \right) \quad (7)$$

□

THEOREM 7 (TXEE.KEX SECURITY). *The TXEE construction is KEX-secure with cleanness predicate Definition 10 (capturing initiator implicit authentication and weak forward secrecy). That is, for any PPT algorithm $\mathcal{A}$ against the key indistinguishability game (defined in Figure 5), $\text{Adv}_{\mathcal{K},n_P,n_S}^{\text{KEX, clean}, \mathcal{A}}(1^\lambda)$ is negligible under the GDH assumption.*

PROOF. We split the analysis into two cases:

- **Case 1:** Corrupt has not been issued for spk_I nor CorruptRand has been issued for responder session.
- **Case 2:** CorruptRand has not been issued for initiator nor responder session.

Proofs for **Case 1** and **Case 2** follow identical to **Case 3** and **Case 2** of Theorem 1, respectively, where $\mathcal{A}$ targets k_3 and k_2 , respectively. Thus we have:

$$\text{Adv}_{\mathcal{K},n_P,n_S}^{\text{KEX, clean}, \mathcal{A}}(1^\lambda) \leq \frac{2\binom{n_P+2n_S}{2}}{q} + n_P^2 n_S^2 \cdot \left(\text{Adv}_{\mathcal{K},C_1}^{\text{GDH}, \mathcal{A}}(1^\lambda) + \text{Adv}_{\mathcal{K},C_2}^{\text{GDH}, \mathcal{A}}(1^\lambda) \right) \quad (8)$$

□

THEOREM 8 (TTX.KEX SECURITY). *The TTX construction is KEX-secure with cleanness predicate Definition 12 (capturing implicit authentication). That is, for any QPT algorithm $\mathcal{A}$ against the key indistinguishability game (defined in Figure 5), $\text{Adv}_{\mathcal{K},n_P,n_S}^{\text{KEX, clean}, \mathcal{A}}(1^\lambda)$ is negligible under the ind-cca security of KEM primitives.*

PROOF. We split the analysis into two cases:

- **Case 1:** Corrupt has not been issued for spk_I nor CorruptRand has been issued for responder session.
- **Case 2:** Corrupt has not been issued for spk_R nor CorruptRand has been issued for initiator session.

The proofs of **Case 1** and **Case 2** follow identical to the proofs of **Case 3** and **Case 2** of Theorem 2. Thus we have:

$$\text{Adv}_{\mathcal{K},n_P,n_S}^{\text{KEX, clean}, \mathcal{A}}(1^\lambda) \leq \frac{2(n_P + n_S)}{|\text{SK}|} + n_P^2 n_S^2 \cdot \left(\text{Adv}_{\mathcal{K},C_1}^{\text{ind-cca}, \mathcal{A}}(1^\lambda) + \text{Adv}_{\mathcal{K},C_2}^{\text{ind-cca}, \mathcal{A}}(1^\lambda) \right) \quad (9)$$

□

THEOREM 9 (XTE.KEX SECURITY). *The XTE construction is KEX-secure with cleanness predicate Definition 14 (capturing implicit receiver authentication and forward secrecy). That is, for any QPT algorithm $\mathcal{A}$ against the key indistinguishability game (defined in Figure*

5), $\text{Adv}_{\mathcal{K},n_P,n_S}^{\text{KEX, clean}, \mathcal{A}}(1^\lambda)$ is negligible under the ind-cca and ind-1cca security of KEM primitives.

PROOF. We split the analysis into two cases:

- **Case 1:** Corrupt has not been issued for spk_R nor CorruptRand has been issued for initiator session.
- **Case 2:** CorruptRand has not been issued for initiator nor responder sessions.

The proofs of **Case 1** and **Case 2** follow identical to the proofs of **Case 2** and **Case 1** of Theorem 2. Thus we have:

$$\text{Adv}_{\mathcal{K},n_P,n_S}^{\text{KEX, clean}, \mathcal{A}}(1^\lambda) \leq \frac{2(n_P + n_S)}{|\text{SK}|} + n_P^2 n_S^2 \cdot \left(\text{Adv}_{\mathcal{K},C_1}^{\text{ind-cca}, \mathcal{A}}(1^\lambda) + \text{Adv}_{\mathcal{K},C_2}^{\text{ind-1cca}, \mathcal{A}}(1^\lambda) \right) \quad (10)$$

□

THEOREM 10 (TXE.KEX SECURITY). *The TXE construction is KEX-secure with cleanness predicate Definition 16 (capturing implicit initiator authentication and forward secrecy). That is, for any QPT algorithm $\mathcal{A}$ against the key indistinguishability game (defined in Figure 5), $\text{Adv}_{\mathcal{K},n_P,n_S}^{\text{KEX, clean}, \mathcal{A}}(1^\lambda)$ is negligible under the ind-cca and ind-1cca security of KEM primitives.*

PROOF. We split the analysis into two cases:

- **Case 1:** Corrupt has not been issued for spk_I nor CorruptRand has been issued for responder session.
- **Case 2:** CorruptRand has not been issued for initiator nor responder sessions.

The proofs of **Case 1** and **Case 2** follow identical to the proofs of **Case 3** and **Case 1** of Theorem 2. Thus we have:

$$\text{Adv}_{\mathcal{K},n_P,n_S}^{\text{KEX, clean}, \mathcal{A}}(1^\lambda) \leq \frac{2(n_P + n_S)}{|\text{SK}|} + n_P^2 n_S^2 \cdot \left(\text{Adv}_{\mathcal{K},C_1}^{\text{ind-cca}, \mathcal{A}}(1^\lambda) + \text{Adv}_{\mathcal{K},C_2}^{\text{ind-1cca}, \mathcal{A}}(1^\lambda) \right) \quad (11)$$

□

THEOREM 11 (XXE.KEX SECURITY). *The XXE construction is KEX-secure with cleanness predicate Definition 3 (capturing weak forward secrecy). That is, for any QPT algorithm $\mathcal{A}$ against the key indistinguishability game (defined in Figure 5), $\text{Adv}_{\mathcal{K},n_P,n_S}^{\text{KEX, clean}, \mathcal{A}}(1^\lambda)$ is negligible under the ind-1cca security of KEM primitives.*

PROOF. Here we provide a proof sketch. By the definition of clean we know CorruptRand has not been issued for initiator nor responder sessions. Hence, the proofs for the theorem follows identical to **Case 1** of Theorem 2 Thus we have:

$$\text{Adv}_{\mathcal{K},n_P,n_S}^{\text{KEX, clean}, \mathcal{A}}(1^\lambda) \leq \frac{(n_P + n_S)}{|\text{SK}|} + n_P^2 n_S^2 \cdot \text{Adv}_{\mathcal{K},C_1}^{\text{ind-1cca}, \mathcal{A}}(1^\lambda) \quad (12)$$

□

THEOREM 12 (TXX.KEX SECURITY). *The TXX construction is KEX-secure with cleanness predicate Definition 15 (capturing implicit initiator authentication). That is, for any QPT algorithm $\mathcal{A}$ against the key indistinguishability game (defined in Figure 5), $\text{Adv}_{\mathcal{K},n_P,n_S}^{\text{KEX, clean}, \mathcal{A}}(1^\lambda)$ is negligible under the ind-cca security of KEM primitives.*

PROOF. Here we provide a proof sketch. By the definition of clean we know Corrupt has not been issued for spk_I nor CorruptRand has been issued for responder session. Hence, the proofs for the theorem follows identical to **Case 3** of Theorem 2 Thus we have:

$$\text{Adv}_{\mathcal{K},n_P,n_S}^{\text{KEX, clean}, \mathcal{A}}(1^\lambda) \leq \frac{(n_P + n_S)}{|\text{SK}|} + n_P^2 n_S^2 \cdot \text{Adv}_{\mathcal{K},C_1}^{\text{ind-cca}, \mathcal{A}}(1^\lambda) \quad (13)$$

□

E Mapped Protocols

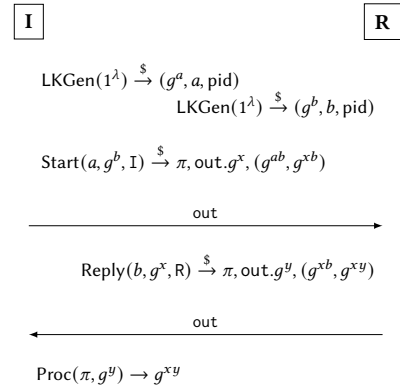


Figure 20: SignalDR(XKEE Pattern)

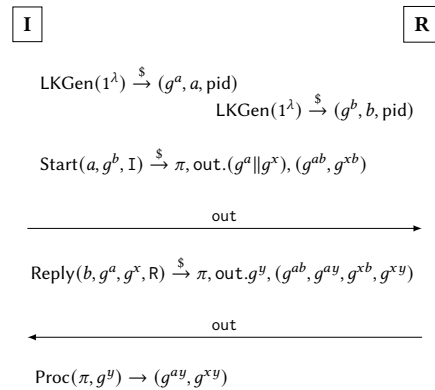


Figure 21: WireGuard (TKEE Pattern)

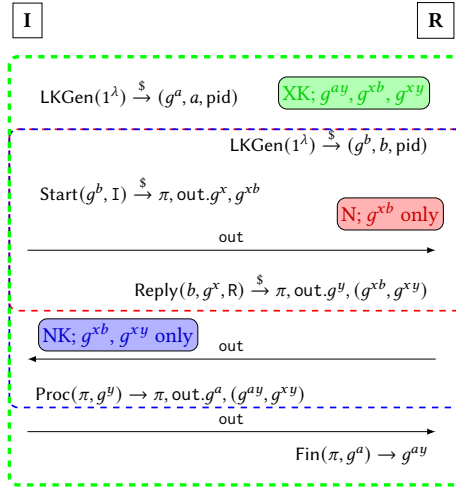


Figure 22: Noise (TKEE/XKEX/XKEE Pattern)

F Security Assumptions

In this section we define the cryptographic formalisms and assumptions that we utilize for the formal analysis of KEX in Section 5.

Escala et al. [12] defines Decisional Diffie-Hellman as follows:

DEFINITION 17 (DECISIONAL DIFFIE-HELLMAN DDH PROBLEM). Let $\mathcal{G}$ be a group of prime order q and g be a generator of $\mathcal{G}$, and let $a, b, c \xleftarrow{\$} \mathbb{Z}_q$. The advantage of an algorithm $\mathcal{A}$ in solving the DDH problem for (g, q) is $\text{Adv}_{g,q}^{\text{ddh}}(\mathcal{A})$ is defined as $\text{Adv}_{g,q}^{\text{ddh}}(\mathcal{A}) = |\Pr(\mathcal{A}(g, g^a, g^b, g^{ab}) = 1) - \Pr(\mathcal{A}(g, g^a, g^b, g^c) = 1)|$. We say that the DDH problem is hard if for all PPT $\mathcal{A}$ $\text{Adv}_{g,q}^{\text{ddh}}(\mathcal{A})$ is negligible.

Okamoto and Pointcheval [22] and Kudla and Paterson [14] define Gap Diffie-Hellman as follows:

DEFINITION 18 (GAP DIFFIE-HELLMAN GDH PROBLEM). Let $f : X \times Y \rightarrow \{0, 1\}$ be a relation on sets X and Y . The computational problem of f is, given $x \in X$, to compute any $y \in Y$ such that $f(x, y) = 1$ if such a y exists, or to return $\perp$ otherwise. The decisional problem of f is, given $(x, y) \in X \times Y$, to decide whether $f(x, y) = 1$ or not. The gap problem of f is to solve the computational problem of f using an oracle which solves the decisional problem of f . Let p and q be primes where $q \mid p - 1$. Let $\mathcal{G}$ be a multiplicative subgroup of $\mathbb{Z}_p^*$ of order q , and let $g \in \mathcal{G}$ generate $\mathcal{G}$. Given $a, b, c \in \mathbb{Z}_q$, define the Diffie-Hellman relation f_{DH} as:

$$f_{\text{DH}} : (\mathcal{G} \times \mathcal{G}) \times \mathcal{G} \rightarrow \{0, 1\}, \text{ where } f_{\text{DH}}(g^a, g^b, g^c) = \begin{cases} 1 & \text{if } g^{ab} = g^c, \\ 0 & \text{otherwise.} \end{cases}$$

This gives rise to three standard problems:

- **Computational Diffie-Hellman (CDH):** Given $g^a, g^b \in \mathcal{G}$ where $a, b \in_R \mathbb{Z}_q$, compute $g^{ab} \bmod p$.
- **Decisional Diffie-Hellman (DDH):** Given $g^a, g^b, g^c \in \mathcal{G}$ where $a, b \in_R \mathbb{Z}_q$, determine whether $c = ab \bmod q$ or not.
- **Gap Diffie-Hellman (GDH):** Given $g^a, g^b \in \mathcal{G}$ where $a, b \in_R \mathbb{Z}_q$, as well as access to an oracle that solves DDH on $\mathcal{G}$, compute $g^{ab} \bmod p$.

The corresponding assumptions are that the above problems are hard, i.e. infeasible to solve in polynomial time in the security parameter.

Next we describe the assumptions for the post-quantum KEM-KEX security of our framework by defining ind-cca security for KEMs. Compared to an ind-cpa attacker, an ind-cca adversary can encrypt messages of their choosing as well as access decryptions/decapsulations of ciphertexts of their choosing even after the challenge has been received. A special variant of ind-cca is ind-1cca in which the adversary is allowed to make only one decryption/decapsulation query.

DEFINITION 19 (ind-cca SECURITY OF KEMs). A key encapsulation mechanism (KEM) is a triple of algorithms $\text{KEM} = \{\text{KGen}, \text{Encaps}, \text{Decaps}\}$ with an associated keyspace $\mathcal{K}$.

We define the ind-cca security of a key encapsulation mechanism in the following game played between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$.

- (1) $\mathcal{C}$ generates a public-key pair $\text{KGen}(1^\lambda) \xrightarrow{\$} (pk, sk)$
- (2) $\mathcal{C}$ generates a ciphertext and key $\text{Encaps}(pk) \xrightarrow{\$} (c, k_0)$
- (3) $\mathcal{C}$ samples a key $k_1 \xleftarrow{\$} \mathcal{K}$ and a bit b uniformly at random.
- (4) $\mathcal{A}$ is given (pk, c, k_b)
- (5) The adversary may adaptively query the challenger; for each query value ctxt_i the challenger responds with $k_i = \text{Decaps}(sk, \text{ctxt}_i)$
- (6) The adversary outputs a guess bit b'

We say that $\mathcal{A}$ wins the ind-cca security game if $b' = b$ and define the advantage of an algorithm $\mathcal{A}$ in breaking the ind-cca security of a key encapsulation mechanism KEM as $\text{Adv}_{\text{KEM}, \mathcal{A}}^{\text{ind-cca}}(1^\lambda) = |2 \cdot \Pr(b' = b) - 1|$. We say that a post-quantum KEM is ind-cca-secure if for all QPT algorithms $\mathcal{A}$, $\text{Adv}_{\text{KEM}, \mathcal{A}}^{\text{ind-cca}}(1^\lambda)$ is negligible in the security parameter 1^λ .