Security Analysis of Signal’s PQXDH Handshake

Rune Fiedler\textsuperscript{1} \hspace{1cm} Felix Günther\textsuperscript{2}

\textsuperscript{1} Cryptoplexity, Technische Universität Darmstadt, Germany
\textsuperscript{2} IBM Research Europe – Zurich, Switzerland

rune.fiedler@cryptoplexity.de, mail@felixguenther.info

Abstract. Signal recently deployed a new handshake protocol named PQXDH to protect against “harvest now, decrypt later” attacks of a future quantum computer. To this end, PQXDH adds a post-quantum KEM to the Diffie–Hellman combinations of the prior X3DH handshake.

In this work, we give a reductionist security analysis of Signal’s PQXDH handshake in a game-based security model that captures the targeted “maximum-exposure” security, allowing fine-grained compromise of user’s long-term, semi-static, and ephemeral key material. We augment prior such models to capture not only the added KEM component but also the signing of public keys, which prior analyses did not capture but which adds an additional flavor of post-quantum security in PQXDH. We then establish a fully parameterized, concrete security bound for the session key security of PQXDH, in particular shedding light on a KEM binding property we require for PQXDH’s security, and how to avoid it.

Our discussion of KEM binding complements the tool-based analysis of PQXDH by Bhargavan, Jacquemard, Kiefer, and Schmidt, which pointed out a potential re-encapsulation attack if the KEM shared secret does not bind the public key. We show that both Kyber (used in PQXDH) and its current NIST draft standard ML-KEM (foreseen to replace Kyber once standardized) satisfy a novel binding notion we introduce and rely on for our PQXDH analysis, which may be of independent interest.

1 Introduction

Billions of people today use messaging apps such as Facebook Messenger, Google Messages, Skype, Signal, or WhatsApp, in which the Signal end-to-end encryption protocol [Sig] secures the communication. The Signal protocol consists of two main components: The initial handshake protocol, which allows two parties to derive a shared session key while authenticating each other, and the Double Ratchet protocol, which allows renewing the session key in an ongoing session to achieve forward secrecy and post-compromise security. Until 2023, Signal used X3DH [MP16] as initial handshake protocol. In 2023, Signal released PQXDH [KS23, KS24], which modifies X3DH to add protection against quantum adversaries; in particular against “harvest now, decrypt later” attacks where an adversary records communication today to break its confidentiality when a cryptographically-relevant quantum computer becomes available in the future.

In X3DH, each user has Diffie–Hellman (DH) keys of different lifetimes (long-term, semi-static, and ephemeral). Combining several DH keys of two users Alice and Bob in the initial key agreement ensures mutual authentication and, as long as one of the combinations of keys remains uncompromised, that the derived session key is secure. To protect against “harvest now, decrypt later” quantum adversaries, PQXDH [KS24] adds a quantum-safe key encapsulation mechanism (KEM) to X3DH while keeping the existing, well-understood [CCD+17] handshake structure unmodified. Furthermore, X3DH signs the involved semi-static DH key and PQXDH additionally the KEM key under the the user’s long-term key.
In a nutshell, the protocol message flow is as follows (where the KEM appears only in PQXDH): Bob sends his long-term, semi-static, and ephemeral DH shares, as well as an ephemeral KEM public key for a full handshake; a reduced handshake omits the ephemeral keys and has a semi-static KEM public key instead. Alice generates an ephemeral DH key pair, encapsulates against Bob’s KEM key, and sends her ephemeral DH key and KEM ciphertext to Bob. Both parties derive the session key from DH shared secrets from three or four DH key combinations (long-term/semi-static, ephemeral/long-term, ephemeral/semi-static, and, if present, ephemeral/ephemeral) and the KEM shared secret.

Previous Security Analyses of X3DH and PQXDH

The development of PQXDH was accompanied by formal, tool-based verification conducted by Bhargavan, Jacomme, Kiefer, and Schmidt (BJKS) [BJKS23, BJK23]. Their analysis provided improvements and security assurance for PQXDH (leading to revisions of the protocol description [KS23, KS24]), using both the symbolic-analysis tool ProVerif [BC] and the computational-analysis tool CryptoVerif [Bla]. Security of Signal’s original X3DH handshake (as well as its ratcheting protocol) was established by Cohn-Gordon, Cremers, Dowling, Garratt, and Stebila (CCDGS) [CCD+17] through a reductionist security analysis in what we refer to as a “maximum-exposure” game-based security model. Vatandas, Gennaro, Ithurburn, and Krawczyk [VGIK20] analyzed the deniability of X3DH, and Fiedler and Janson [FJ24] the deniability of PQXDH.

The core difference between these prior analyses of PQXDH and X3DH is that while the tool-based analyses of BJKS provide machine-checked assurance, the ProVerif results remain on the symbolic, protocol-logic level and the CryptoVerif model does not account for all “maximum-exposure” attack vectors that Signal aims to protect against. Concretely, for proof complexity reasons, the CryptoVerif model of BJKS [BJKS23, BJK23] only captures the compromise of long-term user keys (but not of semi-static and ephemeral keys) and separately considers attacks against the classic DH and the post-quantum KEM security. Such compromise, however, is a main reason for the many key combination Signal’s X3DH and PQXDH handshake, and combined classic/post-quantum hybrid guarantees a main reason behind the PQXDH design specifically.

Our Contributions

This work completes the analysis picture for PQXDH by providing a reductionist security analysis of PQXDH (Revision 3 [KS24], throughout this paper) in a “maximum-exposure”, game-based security model following that for X3DH by CCDGS [CCD+17].

Augmented game-based security model. To this end, we first augment the coded game-based security model of Brendel, Fiedler, Günther, Janson, and Stebila (BFGJS) [BFG+22, BFG+21] to capture the added KEM component in PQXDH. In particular, we model that the KEM key and the semi-static DH keys in PQXDH are signed (instead of assuming them to be authentically distributed like in the prior game-based models [CCD+17, BFG+22, BFG+21]). This requires careful adaptation of related corruption modeling, which adds both model and proof complexity but leads to a security model which more closely captures the real-world deployment.

1The adversary may compromise nearly all secrets as long as trivial wins are excluded.
2Security modeling still always requires trade-offs in abstraction. For example, like all prior analyses [CCD+17, BFG+22, BFG+21, BJKS23, BJK23], we assume dedicated signing keys whereas Signal’s implementation re-uses the long-term DH user keys for signing.
Concrete security bound for PQXDH. We then analyze the PQXDH handshake, establishing its security based on the Gap Diffie–Hellman (GapDH) assumption holding in the DH group, the KEM providing one-way security under chosen-ciphertext attacks (OW-CCA), a certain form of binding property (which we expand on below), and low key-collision probability and correctness errors of the KEM, the signature scheme being existentially unforgeable, and the key derivation function behaving like a random oracle. In particular, we confirm that combining DH and KEM keys yields a combined, hybrid security bound, compared to the separate analysis in BJKS [BJKS23]. Our resulting security theorem for PQXDH is fully parameterized, giving concrete security bounds for each component.

Design discussion and KEM binding. We conclude with a discussion of the PQXDH design, with insights that complement those by BKJS [BJKS23]. In particular, we provide a point of view on the key derivation approach taken by Signal’s X3DH and PQXDH, why it leads to requiring a certain binding property from the KEM in our analysis, and how this requirement could be easily avoided.

For background, BKJS in their analysis [BJKS23, BJK23] discovered the following potential “KEM re-encapsulation attack” on PQXDH: An adversary can learn a KEM shared secret by compromising the involved KEM secret key, and then re-encapsulating this shared secret under another, uncompromised KEM public key. In consequence, decapsulations under two distinct public keys yield the same shared secret, and two sessions with different KEM public keys compute the same session key, violating the protocol’s security goals.

The BKJS analysis excludes this attack by modeling that the KEM public key is included in the associated data when AEAD-encrypting messages, a suggestion that the PQXDH description however only follows optionally: it argues that Kyber, used in the Signal implementation, already prevents the re-encapsulation attack [KS24, Sections 3.3 and 4.12]. Our analysis covers Signal’s approach, providing fine-grained insight into the properties required of the KEM to ensure security of the initial key agreement, without relying on the AEAD. Concretely, we require that the derived shared secret binds the involved public key and ciphertext, in a setting where the adversary knows the involved keys and the randomness to generate them. We prove that both Kyber [SAB+] (currently deployed in PQXDH) and the current NIST draft of ML-KEM [NIS23] (provisioned to replace Kyber in PQXDH once standardized) satisfy this binding property.

We view these KEM binding results as being of independent interest: They complement a range of binding notions for KEMs defined in concurrent work by Cremer, Dax, and Medinger (CDM) [CDM23], yet their notion that would fit the PQXDH setting is not satisfied by ML-KEM [Sch24]. We discuss the framework of CDM in more detail in Section 3 and relate the binding notion we introduce and require in the PQXDH analysis to theirs. Binding properties also arise in other key-combiner settings, e.g., in multi-recipient KEMs [AHK+23] and X-Wing [BCD+24], where the shared secret shall bind the ciphertext.

Related work

Several fully post-quantum replacements for X3DH have been considered in the literature: Hashimoto, Katsumata, Kwiatkowski, and Prest [HKKP22] propose SC-DAKE, a generic construction based on KEMs and a ring signature. BFGJS [BFG+22] propose a similar construction called SPQR based on KEMs and a designated verifier signature. Dobson and Galbraith [DG22] lift X3DH to supersingular isogenies with their construction called SI-X3DH, which is however broken by the SIDH attack [CD23, MMP+23, Rob23]. Collins, Huguenin-Dumittan, Nguyen, Rolin, and Vaudenay [CHDN+24] propose K-Waay, based on adapting split KEMs [BFG+20]. In this work, we focus on Signal’s deployed PQXDH protocol.
2 Preliminaries

2.1 Notation

We refer to the first element of a list \( l \) with \( l[1] \). For \( n \in \mathbb{N}, [n] \) denotes \{1, \ldots, n\}. We write a deterministic or randomized algorithm \( A \) with input \( x \) and output \( y \) as \( y \leftarrow A(x) \) resp. \( y \leftarrow s A(x) \) (or also \( A(x) \rightarrow y \) resp. \( A(x) \rightarrow y \), for the latter with explicit randomness \( r \) we write \( y \leftarrow B(x;r) \). We sample an element \( e \) uniformly at random from a set \( S \) with \( e \leftarrow s S \). We write \( \Pr[\mathcal{G}(A)] \) for the probability that a game \( \mathcal{G} \) involving an adversary \( \mathcal{A} \) outputs true.

2.2 Signatures

Definition 2.1 (Signature scheme). A signature scheme \( \text{SIG} = (\text{KGen}, \text{Sig}, \text{Vf}) \) with associated message space \( \mathcal{M} \) is a triple of PPT algorithms:

- \( \text{KGen}() \rightarrow (pk, sk) \): The probabilistic key generation outputs a public-key/secret-key pair \((pk, sk)\).
- \( \text{Sig}(sk, m) \rightarrow \sigma \): The probabilistic signing takes as input a secret key \( sk \) and a message \( m \) and outputs a signature \( \sigma \).
- \( \text{Vf}(pk, m, \sigma) \rightarrow d \): The deterministic verification takes as input a public key \( pk \), a message \( m \), and a signature \( \sigma \) and outputs a decision \( d \in \{0, 1\} \). If \( d = 1 \) we say the signature is valid.

A signature scheme is correct, if for all \((pk, sk) \leftarrow s \text{KGen}()\) and for all \( m \in \mathcal{M} \) it holds that \( \Pr[\text{Vf}(pk, m, \text{Sig}(sk, m))] = 1 \).

Definition 2.2 (EUF-CMA Security of Signatures). Let \( \text{SIG} = (\text{KGen}, \text{Sig}, \text{Vf}) \) be a signature scheme. We say that \( \text{SIG} \) is \((t, \epsilon, q_{\text{SIG}})\) -EUF-CMA-secure, if for any adversary \( \mathcal{A} \) against Game \( G_{\text{SIG}}^{\text{EUF-CMA}}(\mathcal{A}) \) defined in Figure 1 with running time at most \( t \) and making at most \( q_{\text{SIG}} \) queries to the \( \text{SIGN} \) oracle, we have that

\[
\text{Adv}_{\text{SIG}}^{\text{EUF-CMA}}(\mathcal{A}) := \Pr[\sigma_{\text{SIG}}^{\text{EUF-CMA}}(\mathcal{A})] \leq \epsilon.
\]

2.3 Hash functions

Definition 2.3 (Preimage resistance of hash functions). Let \( H : \{0, 1\}^* \rightarrow \{0, 1\}^n \) be a (hash) function. We say that \( H \) is \((t, \epsilon_d, \mathcal{M})\)-preimage resistant for randomly sampled preimages (PR-d-secure) and \((t, \epsilon_r)\)-preimage resistant for randomly sampled images (PR-r-secure), respectively, if for any adversary \( \mathcal{A} \) with running time at most \( t \) we have that

\[
\text{Adv}_{H, \mathcal{M}}^{\text{PR-d}}(\mathcal{A}) := \Pr[d = H(m') \mid m \leftarrow s \mathcal{M}, d \leftarrow H(m), m' \leftarrow s \mathcal{A}(d)] \leq \epsilon_d, \quad \text{resp.}
\]

\[
\text{Adv}_{H}^{\text{PR-r}}(\mathcal{A}) := \Pr[d = H(m') \mid d \leftarrow \{0, 1\}^n, m' \leftarrow s \mathcal{A}(d)] \leq \epsilon_r.
\]
Definition 2.4 (Collision resistance of hash functions). Let $H: \{0,1\}^* \to \{0,1\}^n$ be a (hash) function. We say that $H$ is $(t, \epsilon)$-collision-resistant (CR-secure), if for any adversary $A$ with running time at most $t$ we have that

$$\text{Adv}^\text{CR}_H(A) := \Pr[H(m) = H(m') \land m \neq m' \mid (m,m') \leftarrow A] \leq \epsilon_H.$$ 

Our analysis further uses that finding two input values that collide in the truncated (256-bit) output is hard. Prior work discusses the related (weak) near collision resistance [MvV97, PS14, JKN21], partial collision resistance [WY05, LT11, YCW14], and truncated collision resistance [JK18]. We model this as collision resistance of a hash function $H$ truncated to the first 256 bits, which we denote by $H_{256}$.

SHA3. Bertoni, Daemen, Peters, and Van Assche [BDPV08] show that SHA3 is indifferentiable from a random oracle when the underlying permutation is modeled as a random permutation. Thus, finding a collision or preimage in SHA3 or SHA3_{256} is as hard as finding it in a random oracle yielding digests of the same length.

2.4 Diffie–Hellman Key Exchange

Definition 2.5 (Diffie–Hellman Key Exchange). A Diffie–Hellman Key Exchange (DH) scheme is a tuple of algorithms $(\text{KGen}, \text{DH})$, defined as follows:

- $\text{KGen}() \to (pk, sk)$: This probabilistic algorithm returns a key pair $(pk, sk)$
- $\text{DH}(pk_A, sk_B) \to \text{DH}_{AB}$: On input a public key $pk_A$ and a secret key $sk_B$, this deterministic algorithm returns the shared DH secret $\text{DH}_{AB}$.

We say that a DH key exchange DH is correct if, for every $(pk_A, sk_A), (pk_B, sk_B) \leftarrow \text{KGen}()$, it holds that

$$\Pr[\text{DH}(pk_A, sk_B) = \text{DH}(pk_B, sk_A)] = 1.$$ 

For notational convenience we allow the arguments to be in arbitrary order, i.e., $\text{DH}(pk_A, sk_B) = \text{DH}(sk_A, pk_B)$. We write $\text{CDH}(pk_A, pk_B)$ to refer to the DH shared secret $\text{DH}_{AB}$ when we do not care which secret key is used.

We rely on the GapDH problem [OP01, CCD+17], i.e., that the computational DH problem is hard given a decisional DH oracle.

Definition 2.6 (GapDH problem). Let $(G, g,q)$ be a group of order $q$ with generator $g$. We say that the $(t, \epsilon_{\text{GDH}}, q_{\text{DDH}})$-GapDH problem holds in $(G, g,q)$ if for any adversary $A$ with running time at most $t$ making at most $q_{\text{DDH}}$ queries to its DDH oracle, we have that

$$\text{Adv}^\text{GDH}_{(G,g,q)}(A) := \Pr[G^\text{GDH}_{(G,g,q)}(A)] \leq \epsilon_{\text{GDH}},$$ 

where $G^\text{GDH}_{(G,g,q)}(A)$ is defined in Figure 2.
3 Key Encapsulation Mechanisms and Binding Properties

We first recap the basic syntax, correctness, and security of key encapsulation mechanisms (KEMs), before discussing the novel KEM binding properties.

Definition 3.1 (Key Encapsulation Mechanisms). A key encapsulation mechanism $KEM = (KGen, Enc, Dec)$ consists of the following three algorithms:

- $KGen() \rightarrow (pk, sk)$: The probabilistic key generation with randomness space $R_{KGen}$ outputs a public-key/secret-key pair $(pk, sk) \in PK \times SK$.
- $Enc(pk) \rightarrow (ct, ss)$: The probabilistic encapsulation algorithm with randomness space $R_{Enc}$ takes as input a public key $pk \in PK$ and outputs a ciphertext $ct \in C$ and the therein encapsulated shared secret $ss \in SS$.
- $Dec(sk, ct) \rightarrow ss'$: The deterministic decapsulation algorithm takes as input a ciphertext $ct \in C$ and secret key $sk$ and outputs $ss' \in SS \cup \{\bot\}$, where $\bot$ indicates an error.

We say that a KEM $KEM = (KGen, Enc, Dec)$ is $\delta$-correct if, for every key pair $(pk, sk) \leftarrow KGen()$, and every encapsulation $(ct, ss) \leftarrow Enc(pk)$, we have

$$\Pr[ss' \neq ss | ss' \leftarrow Dec(sk, ct)] \leq \delta.$$ 

In our analysis, we sometimes need to rule out that honestly generated KEM public keys collide. We capture that probability in the following. For Kyber [SAB+17] and ML-KEM [NIS23], such public-key collisions boil down to random 256-bit seeds colliding under SHA3-512.

Definition 3.2 (KEM public-key collision probability). We define the public-key collision probability of a KEM via a function $\gamma_{coll} : \mathbb{N} \rightarrow [0, 1]$, letting $\gamma_{coll}(n)$ denote the probability that two among $n$ honestly generated public keys collide:

$$\gamma_{coll}(n) := \Pr[pk_i = pk_j \land i \neq j \mid (pk_i, sk_i) \leftarrow KGen() \text{ for } i \in [1, n]].$$

Our analysis requires KEM one-way security under chosen-ciphertext attacks (OW-CCA), which is implied by standard IND-CCA security.

Definition 3.3 (OW-CCA Security of KEMs). Let $KEM = (KGen, Enc, Dec)$ be a KEM. We say that KEM is $(t, \epsilon, q_{Dec})$-OW-CCA-secure, if for any adversary $A$ against Game $G_{KEM}^{IND-CCA}(A)$ defined in Figure 2 with running time at most $t$ and making at most $q_{Dec}$ queries to the Decaps oracle, we have that

$$\text{Adv}_{KEM}^{OW-CCA}(A) := \Pr[G_{KEM}^{OW-CCA}(A)] \leq \epsilon.$$ 

3.1 Binding Properties

Due to the way PQXDH includes (only) the KEM shared secret in its key derivation (but not the KEM public key or ciphertext), our security analysis relies on a novel binding property of the KEM scheme. In a nutshell, we ask that it is hard to find two distinct ciphertexts or (honestly generated) public keys so that the corresponding decapsulations output the same shared secrets, even when given full control over the ciphertexts and knowing corresponding secret keys and the randomness used to generate these keys. The latter is a result of capturing the “maximum-exposure” security of PQXDH, allowing an adversary to reveal a session’s random coins (cf. Section 4). The PQXDH setting, in which an adversary interacts with many KEM keys, also motivates our notion being multi-user.
Aligning with language of Cremers, Dax, and Medinger (CDM) [CDM23] from their concurrent work systematizing binding notions for KEMs, our notion asks that the KEM shared secret binds both public key and ciphertext used to produce it, under leakage of key generation randomness (and hence secret keys), which is a novel variant we denote as \( \text{LEAK}^+\text{-BIND-SS-\{CT,PK\}} \). In general, their notion \( X\text{-BIND-P-Q} \) captures that the components in set \( P \in \{\{ss\},\{ct\},\{ss,ct\}\} \) bind the components in set \( Q \in \{\{pk\},\{ss\},\{ct\}\} \), where the KEM keys are chosen by the adversary \( X = \text{MAL} \), honestly generated and leaked to the adversary \( X = \text{LEAK} \), or honestly generated without leakage of secrets \( X = \text{HON} \). We reproduce their \( \text{LEAK} \) and \( \text{MAL} \) notions for \( ss \) simultaneously binding \( pk \) and \( ct \) in our syntax\(^3\) on the right-hand side of Figure 3.

In the following, we formalize our new \( \text{LEAK}^+\text{-BIND-SS-\{CT,PK\}} \) binding property and relate it to the notions in the CDM framework [CDM23]. In particular, we show in Section 3.3 below that both Kyber and the current NIST draft standard of ML-KEM satisfy the \( \text{LEAK}^+\text{-BIND-SS-\{CT,PK\}} \) property, while the notion from [CDM23] that implies ours, \( \text{MAL-BIND-SS-\{CT,PK\}} \), is not satisfied by ML-KEM [Sch24].\(^4\) While Signal’s current implementation of PQXDH uses Kyber [SAB\(^+\)], it is already prepared\(^5\) to transition to the NIST standard ML-KEM [NIS23].

**Definition 3.4 (LEAK\(^+\)-BIND-SS-\{CT,PK\}).** Let KEM = (KGen, Enc, Dec) be a KEM and \( n \geq 2 \). We say that KEM is \( (t,\epsilon,n)\text{-LEAK}^+\text{-BIND-SS-\{CT,PK\}} \)-secure, if for any adversary \( A \) against Game \( \mathcal{G}^{\text{LEAK}^+\text{-BIND-SS-\{CT,PK\}}}_{\text{KEM},n}(A) \) defined in Figure 3 with running time at most \( t \), we have that

\[
\text{Adv}^{\text{LEAK}^+\text{-BIND-SS-\{CT,PK\}}}_{\text{KEM},n}(A) := \Pr \left[ \mathcal{G}^{\text{LEAK}^+\text{-BIND-SS-\{CT,PK\}}}_{\text{KEM},n}(A) \right] \leq \epsilon.
\]

Figure 4 visualizes the relations between \( \text{LEAK}^+\text{-BIND-SS-\{CT,PK\}} \) and the corresponding notions from [CDM23]: The \( \text{MAL} \) variant is strictly stronger than the \( \text{LEAK}^+ \) variant, which in turn is strictly stronger than the \( \text{LEAK} \) variant. Notably, ML-KEM separates the \( \text{MAL} \) and our \( \text{LEAK}^+ \) variants; we show the other implications and separations below, incl. relating our multi-user notion to the 2-user case.

\(^3\)Note that in the syntax of [CDM23] the KEMs shared secret \( ss \) is simply called a “key”, denoted \( k \). Accordingly, in their notation, \( P \in \{\{k\},\{ct\},\{k,ct\}\} \) and \( Q \in \{\{pk\},\{ss\},\{ct\}\} \). We will stick to our syntax and hence, e.g., write \( \text{MAL-BIND-SS-CT} \) corresponding to their notion \( \text{MAL-BIND-K-CT} \).

\(^4\)CDM [CDM23, Appendices B and E] conjecture that Kyber is \( \text{MAL-BIND-SS-\{CT,PK\}} \)-secure as it hashes both \( pk \) and \( ct \) into the key derivation. Their generic argument [CDM23, Thm. E.1] requires including both via an injective function, a hash function as used in Kyber however is clearly not injective. We leave it as an open question whether Kyber achieves \( \text{MAL-BIND-SS-\{CT,PK\}} \) security; for us, the \( \text{LEAK}^+ \) variant suffices.

\(^5\)https://github.com/signallab/libsignal/commit/0670f0d
Figure 4: Relations between our $\text{LEAK}^+$-$\text{BIND}^-$-$\text{SS}$-$\{CT, PK\}$ binding notion and the corresponding $\text{MAL}$, $\text{LEAK}$ notions from [CDM23] for KEMs. Solid arrows indicate implications, dashed ones loose implications, and crossed-out ones separations. Annotations indicate the respective theorems.

### 3.2 Relations of Binding Properties

First, we show the implication from $\text{MAL}$ to $\text{LEAK}^+$ (Theorem 3.5) and from $\text{LEAK}^+$ to $\text{LEAK}$ (Theorem 3.6). Second, we show that our notion is equivalent for $n$ and $2$ users, up to a factor of $n^2$ (Theorem 3.7). Third, we give a separation between $\text{LEAK}$ and $\text{LEAK}^+$ (Theorem 3.8). Lastly, we show that $\text{ML-KEM}$ and $\text{Kyber}$ are $\text{LEAK}^+$ (Theorem 3.9 and Theorem 3.10), while $\text{ML-KEM}$ is known to not be $\text{MAL}$ [Sch24], giving us a separation for $\text{LEAK}^+$ and $\text{MAL}$.

Note that for the following theorems we assume KEMs with a public key space $|PK| \geq 2$.

**Theorem 3.5** ($\text{MAL-BIND-SS-}\{CT, PK\} \implies \text{LEAK}^+\text{-BIND-SS-}\{CT, PK\}$). A $\text{MAL-BIND-SS-}\{CT, PK\}$-secure KEM KEM is also $\text{LEAK}^+\text{-BIND-SS-}\{CT, PK\}$-secure. Concretely, for any $\text{LEAK}^+\text{-BIND-SS-}\{CT, PK\}$ adversary $A$ against KEM there exist an adversary $B$ such that

$$\text{Adv}^{\text{LEAK}^+\text{-BIND-SS-}\{CT, PK\}}_{\text{KEM}}(A) \leq \text{Adv}^{\text{MAL-BIND-SS-}\{CT, PK\}}_{\text{KEM}}(B).$$

**Proof.** To win in the $\text{LEAK}^+\text{-BIND-SS-}\{CT, PK\}$ game, $A$ must produce ciphertexts $ct_i, ct_j$ such that $ss_i = ss_j$ and one (or both) of $pk_i \neq pk_j$ and $ct_i \neq ct_j$ hold.

We let $B$ truthfully simulate $G^{\text{LEAK}^+\text{-BIND-SS-}\{CT, PK\}}_{\text{KEM}}(A)$ for $A$, i.e., honestly computing keys $pk_k$ for $k \in [n]$, obtaining $i, ct_i, j, ct_j$ from $A$. Then, $B$ outputs $(pk_i, sk_i, pk_j, sk_j, ct_i, ct_j)$. If $A$ wins, we have that $ss_i = ss_j$ while $pk_i \neq pk_j$ or $ct_i \neq ct_j$, so $B$ also wins in the $\text{MAL-BIND-SS-}\{CT, PK\}$ game. \hfill $\square$

**Theorem 3.6** ($\text{LEAK}^+\text{-BIND-SS-}\{CT, PK\} \implies \text{LEAK-BIND-SS-}\{CT, PK\}$). A $\text{LEAK}^+\text{-BIND-SS-}\{CT, PK\}$-secure KEM KEM is also $\text{LEAK-BIND-SS-}\{CT, PK\}$-secure. Concretely, for any $\text{LEAK-BIND-SS-}\{CT, PK\}$ adversary $A$ against KEM there exist an adversary $B$ such that

$$\text{Adv}^{\text{LEAK-BIND-SS-}\{CT, PK\}}_{\text{KEM}}(A) \leq \text{Adv}^{\text{LEAK}^+\text{-BIND-SS-}\{CT, PK\}}_{\text{KEM}, 2}(B).$$

**Proof.** We let $B$ start $A$ on its own inputs except the key generation randomness, thereby truthfully simulating the $\text{LEAK-BIND-SS-}\{CT, PK\}$ game for $A$. When $A$ terminates, $B$ returns the output of $A$. If $A$ wins, then so does $B$. \hfill $\square$

**Theorem 3.7** ($\text{LEAK}^+\text{-BIND-SS-}\{CT, PK\}$: $n$ users $\iff$ $2$ users). Let $n \geq 2$. $A$ ($t, \epsilon, n$)-$\text{LEAK}^+\text{-BIND-SS-}\{CT, PK\}$-secure KEM KEM is also ($t, \epsilon, 2$)-$\text{LEAK}^+\text{-BIND-SS-}\{CT, PK\}$-secure and a ($t, \epsilon, 2$)-$\text{LEAK}^+\text{-BIND-SS-}\{CT, PK\}$-secure KEM KEM is also ($t', n^2 \cdot \epsilon, n$)-$\text{LEAK}^+\text{-BIND-SS-}\{CT, PK\}$-secure
(for \( t' \approx t \)). Concretely, for any \( \text{LEAK}^\ast\text{-BIND-SS-}\{CT, PK\} \) adversary \( \mathcal{A} \) against \( \text{KEM} \) there exist an adversary \( \mathcal{B} \) such that

\[
\text{Adv}^{\text{LEAK}^\ast\text{-BIND-SS-}\{CT, PK\}}_{\text{KEM}, 2}(\mathcal{A}) \leq \text{Adv}^{\text{LEAK}^\ast\text{-BIND-SS-}\{CT, PK\}}_{\text{KEM}, n}(\mathcal{A}) \quad \text{and} \\
\text{Adv}^{\text{LEAK}^\ast\text{-BIND-SS-}\{CT, PK\}}_{\text{KEM}, n}(\mathcal{A}) \leq n^2 \cdot \text{Adv}^{\text{LEAK}^\ast\text{-BIND-SS-}\{CT, PK\}}_{\text{KEM}, 2}(\mathcal{B}).
\]

**Proof.** The “\( \Rightarrow \)” direction straightforwardly holds by letting \( \mathcal{A} \) ignore the remaining \( n - 2 \) keys.

The “\( \Leftarrow \)” direction holds via a guessing argument: Let \( \mathcal{B} \) guess two distinct key indices \( i, j \in [n] \), embed the two keys obtained in its game in these positions, and sample the remaining \( n - 2 \) keys itself. If \( \mathcal{A} \) uses \( i \) (and possibly \( j \)) for its attack, \( \mathcal{B} \) is successful if \( \mathcal{A} \) is. The chance of \( \mathcal{B} \) guessing correctly is \( \frac{1}{n^2} \), establishing the claim. \( \square \)

**Theorem 3.8 (LEAK-BIND-SS-\{CT, PK\} \( \Rightarrow \) LEAK\(^\ast\)-BIND-SS-\{CT, PK\}).** Assuming a \( \text{LEAK-BIND-SS-}\{CT, PK\} \)–secure \( \text{KEM} \) \( \text{KEM}' \) and a preimage-resistant (PR-d) hash function \( H \) (when sampling from \( \{0, 1\}^{256} \)), there exists a \( \text{KEM} \) \( \text{KEM} \) which is \( \text{LEAK-BIND-SS-}\{CT, PK\} \)–secure but not \( \text{LEAK}^\ast\text{-BIND-SS-}\{CT, PK\} \)–secure. Concretely, for any \( \text{LEAK-BIND-SS-}\{CT, PK\} \) adversary \( \mathcal{A} \) against \( \text{KEM}' \) there exist adversaries \( \mathcal{B}_1, \mathcal{B}_2, \mathcal{B}_3 \) such that

\[
\text{Adv}^{\text{LEAK}^\ast\text{-BIND-SS-}\{CT, PK\}}_{\text{KEM}, 2}(\mathcal{B}_1) = 1 \quad \text{and} \\
\text{Adv}^{\text{LEAK}^\ast\text{-BIND-SS-}\{CT, PK\}}_{\text{KEM}}(\mathcal{A}) \leq \text{Adv}^\text{PR-d}_{H}(\mathcal{B}_2) + \text{Adv}^{\text{LEAK}^\ast\text{-BIND-SS-}\{CT, PK\}}_{\text{KEM}', \text{KEM}}(\mathcal{B}_3).
\]

**Proof.** Let \( \text{KEM}' = (\text{KGen}', \text{Enc}', \text{Dec}') \) with shared secret space \( \mathcal{SS}' \) and randomness space \( \mathcal{R}_{\text{KGen}'} \) for key generation be \( \text{LEAK-BIND-SS-}\{CT, PK\} \)–secure, and consider the following modification \( \text{KEM} \) with shared secret space \( \mathcal{SS}' \cup \{\mathcal{ss}^*\} \) where \( \mathcal{ss}^* \) denotes a distinguished shared secret with \( \mathcal{ss}^* \notin \mathcal{SS}' \) and randomness space \( \{0, 1\}^{256} \times \mathcal{R}_{\text{KGen}'} \) for key generation.

<table>
<thead>
<tr>
<th>( \text{KGen}(;r); )</th>
<th>( \text{Enc}(pk); )</th>
<th>( \text{Dec}(sk, pk, ct); )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 ( (x, r_{KG}) \leftarrow r )</td>
<td>5 ( \text{return } (\text{Enc}'(pk), \perp) )</td>
<td>6 ( (sk', y) \leftarrow sk )</td>
</tr>
<tr>
<td>2 ( y \leftarrow H(x) )</td>
<td></td>
<td>7 ( (ct', z) \leftarrow ct )</td>
</tr>
<tr>
<td>3 ( (pk', sk') \leftarrow \text{KGen}'(;r_{KG}) )</td>
<td></td>
<td>8 ( \text{if } y = H(z) )</td>
</tr>
<tr>
<td>4 ( \text{return } (pk', (sk', y)) )</td>
<td></td>
<td>9 ( \text{return } \mathcal{ss}^* )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10 ( \text{return } \text{Dec}'(sk', pk, ct') )</td>
</tr>
</tbody>
</table>

\( \text{KEM} \) is not \( \text{LEAK}^\ast\text{-BIND-SS-}\{CT, PK\} \)-secure: An adversary \( \mathcal{B}_1 \) gets \( (pk_1, sk_1, r_1, pk_2, sk_2, r_2) \) as input, parses the randomness as \( (x_1, r_{KG,1}) \leftarrow r_1 \) and \( (x_2, r_{KG,2}) \leftarrow r_2 \), and outputs the two ciphertexts \( (0, x_1), (1, x_2) \) (omitting the indices for the two users). Both decapsulations trigger the special case. Hence, the shared secrets collide, while the ciphertexts differ (at least) in their first component and \( \mathcal{B}_1 \) always wins.

\( \text{KEM} \) is \( \text{LEAK-BIND-SS-}\{CT, PK\} \)-secure: If \( \mathcal{A} \) triggers the special condition in the decapsulation, then it has found a preimage under \( H \) (for randomly sampled 256-bit preimages), breaking preimage resistance. Otherwise, the adversary cannot take advantage of the extra components in the secret key and ciphertext. Hence, breaking \( \text{LEAK-BIND-SS-}\{CT, PK\} \) of \( \text{KEM} \) implies breaking \( \text{LEAK-BIND-SS-}\{CT, PK\} \) of \( \text{KEM}' \), which we have excluded by assumption. \( \square \)
3.3 Kyber and ML-KEM

Figure 5 gives an algorithmic description of Kyber [SAB+] and its corresponding current draft NIST standard ML-KEM [NIS23]. On a high level, the scheme uses a public key encryption scheme PKE in an FO transform [FO99], as well as three functions H, G, and J (ML-KEM) resp. KDF (Kyber), instantiated as SHA3-256, SHA3-512, and SHAKE256 with 256 bits output, respectively. Our interest being in the binding properties, we focus here on the internals of decapsulation. The decapsulation algorithm first decrypts the PKE ciphertext $ct$ to $m'$. Then, it hashes $m'$ and $h$ (the hashed public key, stored in the secret key) under $G$ onto $ss', r'$. It uses $r'$ as randomness for re-encrypting $m'$. If the re-encrypted ciphertext matches $ct$, Kyber outputs $KDF(ss' || H(ct))$ as shared secret while ML-KEM outputs $ss'$ directly. Otherwise, perform an implicit rejection, i.e., compute the shared secret as $KDF(z || H(ct))$ for Kyber resp. $J(z || ct)$ for ML-KEM, where $z$ is a secret random 256-bit string, which is part of the secret key.

In the following, we show that ML-KEM and Kyber satisfy the \textit{LEAK$^+$-BIND-SS-\{CT, PK\}} binding property, assuming collision resistance of all involved hash functions, (only for ML-KEM) random-oracle properties of H and J, and (only for Kyber) preimage resistance of $G_{256}$ for randomly sampled images.

**Theorem 3.9 (ML-KEM is LEAK$^+$-BIND-SS-\{CT, PK\}–secure).** Assuming H is $(t, \epsilon_H)$–collision-resistant, $G_{256}$ is $(t, \epsilon_G)$–collision-resistant, J is $(t, \epsilon_J)$–collision-resistant, and modeling H, J as independent random oracles, ML-KEM is $(t', \epsilon, n)$–LEAK$^+$-BIND-SS-\{CT, PK\}–secure for $t' \approx t$ and

$$
\epsilon = \frac{n^2}{2^{256}} + 2\epsilon_J + \frac{q_{RO}^2}{2^{256}} + \epsilon_H + 2\epsilon_G,
$$

where $q_{RO}$ is the number of random oracle queries made by the adversary.

**Proof.** We analyze the probability of $A$ winning over a series of game hops.

**Game 0.** We start with the original binding game $G_{ML-KEM,n}^{LEAK^+-BIND-SS-\{CT,PK\}}(A)$:

$$
Adv_{ML-KEM,n}^{LEAK^+-BIND-SS-\{CT,PK\}}(A) = Adv_{ML-KEM}^{G_{ML-KEM,n}}(A).
$$

We distinguish several cases for the adversary to win, depending on how the adversary fulfills the winning condition and if the decapsulation accepts (line 15 in Figure 5 evaluates to true) or implicitly rejects (i.e.,
decapsulation branches into line 17). We denote line 15 being taken with $A_x$ and line 17 with $R_x$, where $x \in \{i, j\}$ are the key indices in the binding game. We index any intermediate values from decapsulation of $c_t_i$ and $c_t_j$ with $i$ and $j$, respectively. We assume a successful adversary, i.e., $ss_i = ss_j$. Note that $R_x$ and $A_x$ are mutually exclusive. The cases are:

A. Both decapsulations reject and the ciphertexts are for two distinct public keys, i.e., $R_i \land R_j \land pk_i \neq pk_j$.

B. Both decapsulations reject and the ciphertexts are distinct, i.e., $R_i \land R_j \land c_t_i \neq c_t_j$.

C. One decapsulation rejects, i.e., wlog. $R_i \land A_j$.

D. Both decapsulations accept and the ciphertexts are for two distinct public keys, i.e., $A_i \land A_j \land pk_i \neq pk_j$.

E. Both decapsulations accept and the ciphertexts are distinct, i.e., $A_i \land A_j \land pk_i = pk_j \land c_t_i \neq c_t_j$.

We treat these cases as events in $G_0$ and indicate the occurrence of event $X$ by $G_0[X]$. By the union bound we get:

$$\text{Adv}_{\text{ML-KEM}}^G(A) \leq \sum_{X \in \{A,B,C,D,E\}} \text{Adv}_{\text{ML-KEM}}^G[X](A)$$

Case A (Both decapsulations reject, distinct public keys: $R_i \land R_j \land pk_i \neq pk_j$).

Here, we bound the probability of the adversary producing ciphertexts that both get implicitly rejected during decapsulation and decapsulate to the same shared secret under distinct public keys.

Game A.0. This is the game conditioned on $R_i \land R_j \land pk_i \neq pk_j$ being satisfied.

$$\text{Adv}_{\text{ML-KEM}}^G(A) = \text{Adv}_{\text{ML-KEM}}^{G_0[R_i \land R_j \land pk_i \neq pk_j]}(A).$$

Game A.1 (Colliding $z$ values). We let $G_{A,0}$ return false if there is a collision among the $z$ values in the $n$ secret keys. Since the $z$ values are 256-bit strings sampled uniformly at random, by the birthday bound we get:

$$\text{Adv}_{\text{ML-KEM}}^{G_{A,1}}(A) \leq \text{Adv}_{\text{ML-KEM}}^{G_{A,1}}(A) + \frac{n^2}{2^{256}}.$$

Game A.2 (Collision in $J$). Now, we have established $z_i || c_t_i \neq z_j || c_t_j$ (since $z_i \neq z_j$), yet $J(z_i || c_t_i) = J(z_j || c_t_j)$. We build a reduction $B_1$ that outputs $z_i || c_t_i, z_j || c_t_j$, breaking $(t, \epsilon_j)$-collision resistance of $J$:

$$\text{Adv}_{\text{ML-KEM}}^{G_{A,1}}(A) \leq \epsilon_j.$$

Case B (Both decapsulations reject, distinct ciphertexts: $R_i \land R_j \land c_t_i \neq c_t_j$).

Here, we bound the probability of the adversary producing two distinct ciphertexts that both get implicitly rejected during decapsulation and decapsulate to the same shared secret.

Game B.0. This is the game conditioned on $R_i \land R_j \land c_t_i \neq c_t_j$ being satisfied.

$$\text{Adv}_{\text{ML-KEM}}^G(A) = \text{Adv}_{\text{ML-KEM}}^{G_0[R_i \land R_j \land c_t_i \neq c_t_j]}(A).$$

Since $c_t_i \neq c_t_j$, we also have $z_i || c_t_i \neq z_j || c_t_j$, yet $J(z_i || c_t_i) = J(z_j || c_t_j)$. We build a reduction $B_2$ that outputs $z_i || c_t_i, z_j || c_t_j$, breaking $(t, \epsilon_j)$-collision resistance of $J$:

$$\text{Adv}_{\text{ML-KEM}}^{G_{B,0}}(A) \leq \epsilon_j.$$
Case C (One decapsulation rejects: $R_i \land A_j$).

Here, we bound the probability of the adversary producing exactly one ciphertext that gets implicitly rejected during decapsulation while both ciphertexts decapsulate to the same shared secret.

**Game C.0.** This is the game conditioned on $R_i \land A_j$ being satisfied.

$$\text{Adv}_{\text{ML-KEM}}^{G_{C.0}}(A) = \text{Adv}_{\text{ML-KEM}}^{G_{0}[R_i \land A_j]}(A).$$

For $A$ to win, it needs to create a collision $J(z_i || ct_i) = ss_i = ss_j = G_{256}(m_j' || h_j)$ between $J$ and $G$. Assuming both $J$ and $G$ behave like (independent) random oracles, the probability of finding such a collision with $q_{RO}$ many random oracle queries is upper bounded by:

$$\text{Adv}_{\text{ML-KEM}}^{G_{C.0}}(A) \leq \frac{q_{RO}}{2^{256}}.$$

Case D (Both decapsulations accept, distinct public keys: $A_i \land A_j \land pk_i \neq pk_j$).

Here, we bound the probability of the adversary producing ciphertexts that both get accepted during decapsulation and decapsulate to the same shared secret under distinct public keys.

**Game D.0.** This is the game conditioned on $A_i \land A_j \land pk_i \neq pk_j$ being satisfied.

$$\text{Adv}_{\text{ML-KEM}}^{G_{D.0}}(A) = \text{Adv}_{\text{ML-KEM}}^{G_{0}[A_i \land A_j \land pk_i \neq pk_j]}(A).$$

Since we know that $pk_i \neq pk_j$ and $G_{256}(m_i || H(pk_i)) = ss_i = ss_j = G_{256}(m_j || H(pk_j))$, it must be that $A$ provides us with a collision either in $H$ or in $G_{256}$. Once more, we can distinguish two cases:

1. The first case is $m_i || H(pk_i) = m_j || H(pk_j)$. Then we can build a reduction $B_3$ that outputs $pk_i, pk_j$ as collision in $H$, breaking $(t, \epsilon_H)$-collision resistance of $H$.

2. Otherwise, $m_i || H(pk_i) \neq m_j || H(pk_j)$, and we can build reduction $B_4$ that outputs $m_i || H(pk_i), m_j || H(pk_j)$ as collision in $G_{256}$. Hence, $B_2$ breaks the $(t, \epsilon_G)$-collision resistance of $G_{256}$.

   
   Jointly,

   $$\text{Adv}_{\text{ML-KEM}}^{G_{D.0}}(A) \leq \epsilon_H + \epsilon_G.$$

Case E (Both decapsulations accept, distinct ciphertexts: $A_i \land A_j \land pk_i = pk_j \land ct_i \neq ct_j$).

Here, we bound the probability of the adversary producing ciphertexts that both get accepted during decapsulation and decapsulate to the same shared secret under the same public key.

**Game E.0.** This is the game conditioned on $A_i \land A_j \land pk_i = pk_j \land ct_i \neq ct_j$ being satisfied.

$$\text{Adv}_{\text{ML-KEM}}^{G_{E.0}}(A) = \text{Adv}_{\text{ML-KEM}}^{G_{0}[A_i \land A_j \land pk_i = pk_j \land ct_i \neq ct_j]}(A).$$

We know that $pk_i = pk_j$ (and hence $h_i = h_j$) for $G_{256}(m_i || h_i) = ss_i = ss_j = G_{256}(m_j || h_j)$. Note that we must have $m_i \neq m_j$; Otherwise, the re-encryption step in both decapsulations would result in identical ciphertexts $ct_i = ct_j$, which contradicts the condition of event $E$.

Now, we can build a reduction $B_5$ that outputs $m_i || H(pk_i), m_j || H(pk_j)$ as collision in $G_{256}$. Since $G$ collides in its first output, which is 256 bits, $B_3$ breaks the $(t, \epsilon_G)$-collision resistance of $G_{256}$.

$$\text{Adv}_{\text{ML-KEM}}^{G_{E.0}}(A) \leq \epsilon_G.$$

Collecting the bounds yields the claim. \qed
For the Kyber proof, we do not require any collisions across hash functions, allowing us to avoid relying on the random oracle. Instead, we additionally require $G_{256}$ to be $(t, \epsilon_G)$–preimage-resistant for randomly sampled images.

**Theorem 3.10** (Kyber is $LEAK^+\cdot$BIND-SS-{$CT,PK$}–secure). Assuming $H$ is $(t, \epsilon_H)$–collision-resistant, $G_{256}$ is $(t, \epsilon_G)$–collision-resistant and $(t, \epsilon_G')$–preimage-resistant ($PR_r$, i.e., when sampling from the range), and KDF is $(t, \epsilon_{KDF})$–collision-resistant, Kyber is $(t', \epsilon, n)$–$LEAK^+\cdot$BIND-SS-{$CT,PK$}–secure for $t' \approx t$ and

$$
\epsilon = \epsilon_H + \epsilon_{KDF} + \frac{n^2}{2^{256}} + \epsilon_{KDF} + n \epsilon_G' + \epsilon_{KDF} + \epsilon_G + \epsilon_H
$$

$$
= 2 \epsilon_H + \epsilon_G + n \epsilon_G' + 4 \epsilon_{KDF} + \frac{n^2}{2^{256}}.
$$

**Proof.** We analyze the probability of $A$ winning over a series of game hops.

**Game 0.** We start with the original binding game $G_{Kyber,n}^{LEAK^+\cdot$BIND-SS-{$CT,PK$}$(A)$:

$$
\text{Adv}_{Kyber,n}^{LEAK^+\cdot$BIND-SS-{$CT,PK$}$(A) = \text{Adv}_{Kyber}^{G_0}(A).
$$

We distinguish several cases for the adversary to win, depending on how the adversary fulfills the winning condition and if the decapsulation accepts (line 15 in Figure 5 evaluates to true) or implicitly rejects (i.e., decapsulation branches into line 17). We denote line 15 being taken with $A_x$ and line 17 with $R_x$, where $x \in \{i, j\}$ are the key indices in the binding game. We index any intermediate values from decapsulation of $ct_i$ and $ct_j$ with $i$ and $j$, respectively. We assume a successful adversary, i.e., $ss_i = ss_j$. Note that $R_x$ and $A_x$ are mutually exclusive. The cases are:

A. Both ciphertexts are distinct, i.e., $ct_i \neq ct_j$.

B. Both decapsulations reject and the ciphertexts are for two distinct public keys, i.e., $R_i \land R_j \land pk_i \neq pk_j$.

C. One decapsulation rejects and the ciphertexts are for two distinct public keys, i.e., wlog. $R_i \land A_j \land pk_i \neq pk_j$.

D. Both decapsulations accept and the ciphertexts are for two distinct public keys, i.e., $A_i \land A_j \land pk_i \neq pk_j$.

We treat these cases as events in $G_0$ and indicate the occurrence of event $X$ by $G_0[X]$. By the union bound we get:

$$
\text{Adv}_{Kyber}^{G_0}(A) \leq \sum_{X \in \{A,B,C,D\}} \text{Adv}_{Kyber}^{G_0[X]}(A)
$$

**Case A (distinct ciphertexts: $ct_i \neq ct_j$).**

Here, we bound the probability of the adversary producing two distinct ciphertexts that decapsulate to the same shared secret.

**Game A.0.** This is the game conditioned on $ct_i \neq ct_j$ being satisfied.

$$
\text{Adv}_{Kyber}^{G_{A.0}}(A) = \text{Adv}_{Kyber}^{G_0[ct_i \neq ct_j]}(A).
$$

We know that $KDF(x_i||H(ct_i)) = ss_i = ss_j = KDF(x_j||H(ct_j))$ for $x_i$ being either $ss_i'$ (in case of an accepting decapsulation) or $z_i$ (in case of an implicitly rejecting decapsulation), and $x_j$ likewise. Since we
know that $ct_i \neq ct_j$, we can now either build a reduction $B_1$ that outputs $ct_i, ct_j$, breaking $(t, \epsilon_H)$-collision-resistance of $H$, or a reduction $B_2$ that outputs $x_i||H(ct_i), x_j||H(ct_j)$, breaking $(t, \epsilon_{KDF})$-collision-resistance of KDF:

$$\text{Adv}_{KDF}^G(\mathcal{A}) \leq \epsilon_H + \epsilon_{KDF}.$$

**Case B (Both decapsulations reject, distinct public keys: $R_i \land R_j \land pk_i \neq pk_j$).**

Here, we bound the probability of the adversary producing ciphertexts that both get implicitly rejected during decapsulation and decapsulate to the same shared secret under distinct public keys.

**Game B.0.** This is the game conditioned on $R_i \land R_j \land pk_i \neq pk_j$ being satisfied.

$$\text{Adv}_{Kyber}^G(\mathcal{A}) = \text{Adv}_{Kyber}^G[R_i \land R_j \land pk_i \neq pk_j](\mathcal{A}).$$

**Game B.1 (Colliding $z$ values).** We let $G_{B.0}$ return false if there is a collision among the $z$ values in the $n$ secret keys. Since the $z$ values are 256-bit strings sampled uniformly at random, by the birthday bound we get:

$$\text{Adv}_{G_{B.0}}^G(\mathcal{A}) \leq \text{Adv}_{Kyber}^G(\mathcal{A}) + \frac{n^2}{2^{256}}.$$

**Game B.2 (Collision in KDF).** Now, we have established $z_i||H(\text{ct}_i) \neq z_j||H(\text{ct}_j)$ (since $z_i \neq z_j$), yet $KDF(z_i||H(\text{ct}_i)) = KDF(z_j||H(\text{ct}_j))$. We build a reduction $B_3$ that outputs $z_i||H(\text{ct}_i), z_j||H(\text{ct}_j)$, breaking $(t, \epsilon_{KDF})$-collision resistance of KDF:

$$\text{Adv}_{KDF}^G(\mathcal{A}) \leq \epsilon_{KDF}.$$

**Case C (One decapsulation rejects, distinct public keys: $R_i \land A_j \land pk_i \neq pk_j$).**

Here, we bound the probability of the adversary producing exactly one ciphertext that gets implicitly rejected during decapsulation, while both ciphertexts decapsulate to the same shared secret under distinct public keys.

**Game C.0.** This is the game conditioned on $R_i \land A_j \land pk_i \neq pk_j$ being satisfied.

$$\text{Adv}_{Kyber}^G(\mathcal{A}) = \text{Adv}_{Kyber}^G[R_i \land A_j \land pk_i \neq pk_j](\mathcal{A}).$$

If $ss'_i \neq z_j$ (for $ss' = G_{256}(m'_i||h_i)$), we can build a reduction $B_4$ that returns $ss'_i||H(\text{ct}_i), z_j||H(\text{ct}_j)$ to break $(t, \epsilon_{KDF})$-collision resistance of KDF. Otherwise, we can build a reduction $B_5$ that guesses $j \in [n]$ and embeds an image of $G_{256}$ in $z_j$, breaking the $(t, \epsilon'_G)$-preimage resistance of $G_{256}$ for randomly sampled images (PR-r) if it guesses $j$ correctly. Hence,

$$\text{Adv}_{Kyber}^G(\mathcal{A}) \leq \epsilon_{KDF} + n \cdot \epsilon'_G.$$

**Case D (Both decapsulations accept, distinct public keys: $A_i \land A_j \land pk_i \neq pk_j$).**

Here, we bound the probability of the adversary producing ciphertexts that both get accepted during decapsulation and decapsulate to the same shared secret under distinct public keys.

**Game D.0.** This is the game conditioned on $A_i \land A_j \land pk_i \neq pk_j$ being satisfied.

$$\text{Adv}_{Kyber}^G(\mathcal{A}) = \text{Adv}_{Kyber}^G[A_i \land A_j \land pk_i \neq pk_j](\mathcal{A}).$$
Now, $KDF(G_{256}(m_i||h_i)||H(ct_i)) = ss_i = ss_j = KDF(G_{256}(m_j||h_j)||H(ct_j))$ for $pk_i \neq pk_j$. It must be that $\mathcal{A}$ provides us with a collision either in $KDF$, $G_{256}$, or $H$. Once more, we can distinguish the cases:

1. In case $G_{256}(m_i||h_i)||H(ct_i) \neq G_{256}(m_j||h_j)||H(ct_j)$, then we can build a reduction $\mathcal{B}_6$ that outputs $G_{256}(m_i||h_i)||H(ct_i), G_{256}(m_j||h_j)||H(ct_j)$, breaking $(t, \epsilon_{KDF})$–collision-resistance of $KDF$.

2. Otherwise, if $m_i||h_i \neq m_j||h_j$, then we can build a reduction $\mathcal{B}_7$ that outputs $m_i||h_i, m_j||h_j$, breaking $(t, \epsilon_G)$–collision-resistance of $G_{256}$.

3. Otherwise, we must have $m_i||h_i = m_j||h_j$ and $H(pk_i) = h_i = h_j = H(pk_j)$. By $G_{C.0}$, $pk_i \neq pk_j$ and we can build a reduction $\mathcal{B}_8$ that outputs $pk_i, pk_j$, breaking $(t, \epsilon_H)$–collision-resistance of $H$.

Jointly,

$$\text{Adv}_{\text{Kyber}}^{G_{C.0}}(\mathcal{A}) \leq \epsilon_{KDF} + \epsilon_G + \epsilon_H.$$

Collecting the bounds yields the claim.

4 Security Model

We analyze security of the PQXDH protocol in a computational, game-based security model for authenticated key exchange protocols, following the tradition of Bellare and Rogaway (BR) [BR94], but refined towards the maximum-exposure security that Signal’s handshakes (both X3DH and PQXDH) aim at. Our model is based on prior Signal and Signal-like security models by CCDGS [CCD+17] and particularly BFGJS [BFG+22, BFG+21], the latter introducing syntax and modeling for the particular class of asynchronous key exchange protocols that the Signal handshakes belong to.

In BR-style models, a computational adversary interacts with multiple users across multiple sessions of the key exchange protocol, fully controlling the network and being able to reroute, modify, inject, and drop messages at will (through a $\text{Send}$ oracle). For maximum-exposure, the model allows the adversary to compromise a user’s long-term key (via a $\text{CorruptLTKey}$ oracle) and semi-static keys ($\text{CorruptSSKey}$), as well as reveal a session’s key ($\text{RevealSSKey}$) and ephemeral randomness ($\text{RevealRand}$). The targeted security property finally is key indistinguishability, asking that keys established in so-called “fresh” sessions are indistinguishable from random keys. “Freshness” here encodes that a session is not trivially compromised and that the protocol under analysis aims to protect against the involved key compromises (defined through a set of “clean” predicates). We will detail those technical bits below.

For the analysis of PQXDH, we extend the prior models in several ways:

- Prior models [CCD+17, BFG+22, BFG+21] assumed semi-static keys being authentically distributed. However, only long-term keys can be verified out-of-band in Signal, and semi-static keys are signed with those keys instead. We are the first to capture these $\text{signed semi-static keys}$ in a game-based model for a computational security analysis. In both the model and the security analysis (cf. Section 5), capturing these signatures leads to notable added complexity.

- In contrast to Signal’s classical X3DH handshake, PQXDH also involves $\text{signed ephemeral (KEM)}$ keys. We capture this, too, expanding on the set of attack vectors our model covers.

- PQXDH adding KEM keys means that $\text{multiple semi-static keys}$ (DH and KEM) can now be involved in a handshake. We capture this by accordingly modified identification of semi-static keys.

Note that CCDGS [CCD+17] also analyzed the ratcheting protocol, working in a multi-stage key exchange model [FG14]; PQXDH does not affect the ratcheting part, hence we focus on the initial handshake. BFGJS [BFG+22, BFG+21] in turn also studied deniability, which we do not consider in this work.
In the following, we will first review the syntax required to express both the PQXDH protocol and the security model formally; we mostly follow BFGJS [BFG+22, BFG+21] here, in parts verbatim, and focus on highlighting (with light-gray background) the main differences introduced in this work to account for signed semi-static and ephemeral keys as well as multiple identifiers for semi-static keys.

4.1 Syntax and Notation

Key exchange syntax. We define a two-party key exchange protocol via the following probabilistic algorithms:

- **KGenLT** \( \rightarrow (ltpk, ltsk) \): The *long-term key generation* algorithm that outputs a party’s public-key/secret-key pair.

- **KGenSS** \( ltsk \rightarrow ((sspk_1, sssk_1, \sigma_1), \ldots) \): The *semi-static key generation* algorithm that takes as input a long-term secret key \( ltsk \) and outputs a vector of public-key/secret-key pairs and corresponding signatures.\(^6\)

- **Run** \( (ltsk, sssk, ltpk, \pi, m) \rightarrow (\pi', m') \): The *session execution algorithm* that takes as input a party’s long-term secret key \( ltsk \), that party’s semi-static secret keys \( sssk \), all parties’ long-term public keys \( ltpk \), a session state \( \pi \), and an incoming message \( m \), and outputs an updated session state \( \pi' \) and a (possibly empty) outgoing message \( m' \). The session sending the first message is set up by calling Run with a distinguished message \( m = (\text{create}, (\text{ssid}, \text{type})) \), where \( \text{ssid} \) indicates the semi-static keys\(^7\) to be used and \( \text{type} \) whether a full (\( \text{type} = \text{full} \)) or reduced (\( \text{type} = \text{reduced} \)) handshake should be performed.

Note that an explicit ephemeral key generation algorithm is not mandatory and can happen inside of Run.

Parties and sessions. In our model, parties \( P \in [n_p] \), each holding a long-term public-key/secret-key pair generated by **KGenLT** may run multiple instances of the protocol (simultaneously or sequentially); we denote the \( i \)th such session of party \( P \) by \( \pi_i^P \). Each session maintains the following information:

- \( \text{oid} \in [n_p] \): The identity of the session owner.
- \( \text{pid} \in [n_p] \cup \{\star\} \): The identity of the intended peer, which may initially be unknown (indicated by \( \star \)).
- \( \text{role} \in \{\text{initiator}, \text{responder}\} \): The role of the party.
- \( \text{st}_{\text{exec}} \in \{\bot, \text{running}, \text{accepted}, \text{rejected}\} \): The status of this session’s execution.
- \( \text{sid} \in \{0,1\}^* \cup \{\bot\} \): A session identifier defining partnering.
- \( \text{cid} \in \{0,1\}^* \cup \{\bot\} \): A contributive identifier, defining a preliminary form of partnering (often as a substring or prefix of the session identifier) for the case the session is not yet bound to an authenticated peer [DFGS15].
- \( K \in K_{KE} \cup \{\bot\} \): The session key established in this session, initialized to \( \bot \).

\( ^6 \)The length of this vector is protocol-dependent; e.g., in X3DH, semi-static keys consist of one DH key pair, and PQXDH adds a second semi-static key (for a KEM scheme). For simplicity, we have a set of semi-static keys be generated in one operation, yet in the protocol each key can be used independently.

\( ^7 \)In the model, semi-static keys are identified by some value \( \text{ssid} = "s,n" \) where \( s \) indicates the **KGenSS** call through which they were generated and \( n \) the key’s position in that call’s output. In practice, the latter signifies the type of semi-static key if there are several; for PQXDH, \( n = 1 \) for DH keys and \( n = 2 \) for KEM keys.
• \( \text{type} \in \{\text{full}, \text{reduced}\} \): Indicator whether an ephemeral pre-key was used (\( \text{type} = \text{full} \)) for key establishment, or not (\( \text{type} = \text{reduced} \)).

• \( \text{coins} \in \mathcal{R}_{KE} \): The random coins from the randomness space \( \mathcal{R}_{KE} \) used in the execution of \( \text{Run} \); set by the game and read-only thereafter.

• \( \text{sspks} \in (\{0, 1\}^*) \cup \{\perp\} \): The semi-static public keys used in this session.

For bookkeeping in the security game we additionally introduce the following flags, which are not accessible by the protocol sessions:

• \( \text{revrand} \in \{\text{true}, \text{false}\} \) indicates whether the random coins \( \pi \cdot \text{coins} \) have been revealed via a \( \text{RevealRand} \) query. The default value is \( \text{false} \).

• \( \text{pcorr} \in \{\text{true}, \text{false}\} \) indicates whether the peer’s long-term key was corrupted at the point in time when this session accepted. The default value is \( \text{false} \).

Session partnering. We say two sessions \( \pi^i_U \) and \( \pi^j_V \) are partnered if they agree on the session identifier: \( \pi^i_U \cdot \text{sid} = \pi^j_V \cdot \text{sid} \neq \perp \). Contributive identifiers (\( \text{cid} \)) indicate when sessions may eventually derive the same key but are not fully partnered (yet); we use these to model security of initiator’s keys in incomplete handshakes.

4.2 Security Game

The security property of an authenticated key exchange protocol \( \text{KE} \) we consider in this work is indistinguishability of session keys (KI), formalized through the game \( G^\text{KI}_{\text{KE}}(A) \) in Figure 6 played by an adversary \( A \).

At the start of the game, a random challenge bit \( b_{\text{test}} \leftarrow \{0, 1\} \) is fixed and long-term public-key/secret-key pairs are generated for all \( n_p \) honest parties and their public keys \( \vec{ltpk} \) provided to the adversary. The adversary is then able to interact with honest parties via the following queries:

• \( \text{Send}(U, i, m) \): Sends message \( m \) to session \( \pi^i_U \), which corresponds to executing \( \text{Run}(ltsk_U, \vec{sssk}_U, \vec{ltpk}, \pi^i_U, m) \), saving the updated session state \( \pi' \) as \( \pi^i_U \), and returning the outgoing message \( m' \) to the adversary.

• \( \text{CorruptLTKey}(U) \): Returns party \( U \)'s long-term secret key \( ltsk_U \) to the adversary; recorded through the flag \( \text{corrltk}_U \).

• \( \text{CorruptSSKey}(U, ssid) \): Returns party \( U \)'s semi-static secret key \( \vec{sssk}^{ssid}_U \) to the adversary; recorded through the flag \( \text{corrssk}_{sspk} \), where \( \vec{sspk} \) is the public key corresponding to \( \vec{sssk}^{ssid}_U \).

• \( \text{RevealRand}(U, i) \): Returns the random coins of session \( \pi^i_U \) to the adversary, recorded through the flag \( \text{revrand} \).

• \( \text{RevealSessKey}(U, i) \): If session \( \pi^i_U \) has accepted, return its session key \( \pi^i_U \cdot K \) to the adversary.

• \( \text{Test}(U, i) \): If the \( \text{Test} \) query has been called before or session \( \pi^i_U \) has not accepted, then return \( \perp \). Otherwise; if \( b_{\text{test}} = 0 \), return \( \pi^i_U \cdot K \), otherwise return a randomly sampled session key from the protocol’s key space \( \mathcal{K}_{KE} \). Record the test session as \( \pi^* \leftarrow \pi^i_U \).

---

8Since we model that semi-static keys are not authentically distributed (but signed), sessions only know the public key received (\( \vec{sspk} \)), but not the game label \( ssid \), which was used in prior models [CCD⁺¹⁷, BFG⁺²², BFG⁺²¹] to track compromise more easily.
At the end of the game, the adversary outputs a bit $b'$. The adversary is said to win if $b' = b_{\text{test}}$ and the test session $\pi^*$ is fresh. Formally, if the test session is fresh, the experiment outputs 1 if $b' = b_{\text{test}}$ and 0 otherwise; if the test session is not fresh, then the experiment outputs a random bit. The adversary’s advantage in the key indistinguishability game measured as the experiment outputting 1 minus the adversary’s guessing chance, $\frac{1}{2}$.

**Definition 4.1 (Key indistinguishability).** Let $KE$ be a key exchange protocol and $A$ an adversary against the key indistinguishability ($KI$) game $G_{KE}^K(A)$ in Figure 6. We say that $KE$ achieves $(t, \epsilon, (q_{\text{Snd}}, q_{\text{CorrLT}}, q_{\text{CorrSS}}, q_{\text{RevR}}, q_{\text{RevSK}}))$–key indistinguishability, if for any adversary $A$ against $G_{KE}^K(A)$ with running time at most $t$ and making at most $q_{\text{Snd}}, q_{\text{CorrLT}}, q_{\text{CorrSS}}, q_{\text{RevR}}, \text{resp. } q_{\text{RevSK}}$ queries to its $\text{SEND}$, $\text{CORRUPTLTKEY}$, $\text{CORRUPTSSKEY}$, $\text{REALVERAND}$, resp. $\text{REALSESSKEY}$ oracles, we have that

$$\text{Adv}_{KE}^K(A) = \Pr[KE_{\text{test}}(A)] - \frac{1}{2} \leq \epsilon.$$ 

Note that the model restricts the adversary to a single query to the $\text{TEST}$ oracle.

**Soundness.** The model also captures soundness (via the predicate $\text{sound}$), i.e., that session identifiers appropriately reflect correct protocol executions. Concretely, soundness demands that an adversary cannot create one of the following situations (or else it will win the game immediately):

(i) Two sessions accept with the same session identifier, but derive different session keys, indicate different handshake types (full vs. reduced), or do not agree on their contributive identifiers (Fig. 6, line 18).

(ii) Two initiator sessions accept with the same session identifier (Fig. 6, line 19).

(iii) Three sessions accept with the same session identifier in full handshake type (Fig. 6, line 20).

**Freshness.** Unrestricted access to the game oracles allows for trivial wins, e.g., by testing a session key and also revealing it (or its partner). The freshness predicate $\text{fresh}$ rules out such trivial wins, and further encodes that the adversary has not obtained sufficiently many secrets, via $\text{CORRUPTLTKEY}$ and/or $\text{CORRUPTSSKEY}$ and/or $\text{REALVERAND}$ queries, to derive the session key of the test session itself and/or substituted signed keys in its execution; this is encoded through a set of so-called $\text{clean}$ predicates (Figure 6, line 16 and lines 46ff.).

**Clean predicates.** Following the terminology of CCDGS [CCD++17], the set of $\text{clean}$ predicates capture the maximum-exposure properties of the protocol being analyzed. Figure 6, lines 46ff. formally capture these for PQXDH. The core properties closely follow those for the classic X3DH handshake [CCD++17, BFG++22]: in handshakes where the long-term/semi-static, ephemeral/long-term, or ephemeral/semi-static (or, for type = full handshakes, ephemeral/ephemeral) secrets combination between initiator/responder is not revealed, we expect security of the derived session key. Jumping ahead, for those combinations where PQXDH employs both DH and KEM keys, we will obtain hybrid/combiner security terms in our analysis (see Section 5), capturing that an adversary has to break both primitives to successfully attack the protocol via these compromise angles.

In addition to prior work [CCD++17, BFG++22, BFG++21], our clean predicates further model the signatures on the responder’s (semi-static and ephemeral public) keys and allow the adversary to replace semi-static and ephemeral keys sent over the network. Instead of assuming authentic distribution of semi-static public keys, we merely preclude the adversary from compromising the (long-term) signing key prior
Figure 6: Key indistinguishability (KI) game for key exchange protocol $KE$ (top), in which adversary $A$ has access to oracles $SEND$, $TEST$, $CORRUPTLTKEY$, $CORRUPTSSKEY$, $REVEALRAND$, and $REVEALSESSKEY$ (middle), and wrt. to clean predicates for PQXDH (bottom). Highlighted code reflects the main changes compared to the model of BFGJS [BFG$^{+} 22$, BFG$^{+} 21$].
to the targeted session having accepted (cf. Figure 6, lines 54 and 60). Treating signatures explicitly, we also capture the additional guarantees that signing KEM keys give: for the ephemeral/ephemeral combination to contribute security, it is now (also) sufficient to have the peer’s contribution be cleanly signed (cf. Figure 6, line 48). To that end we set a flag \texttt{pcorr} in a session if its peer was corrupted upon acceptance (Figure 6, line 29). For \texttt{PQXDH}, this part of the model ensures that a quantum-later adversary cannot, in addition to a “harvest now, decrypt later” attack, replace the ephemeral KEM public key with its own without being noticed.

In more formal detail, let \(\pi^*\) denote the test session. Depending on whether an ephemeral pre-key was used in the key derivation of \(\pi^*\) or not, we apply either the \texttt{cleanfull} or the \texttt{cleanreduced} predicate to \(\pi^*\).

Since \texttt{cleanreduced} is part of the description of \texttt{cleanfull}, we first discuss the case \(\pi^*.\texttt{type} = \texttt{reduced}\). Intuitively, a session key derived in such a session remains unknown to the adversary, if one of the four keys (i.e., excluding the unused \(DH_4\)) that constitute the master secret is “clean”, i.e., cannot be computed by the adversary. This is the case if one of the following three \texttt{clean} predicates holds for the test session \(\pi^*\):

- \texttt{cleanLTSS}: This predicate indicates whether the combination of the long-term key of the initiator and the semi-static keys of the responder are unknown to the adversary. If the test session is an initiator, the responder’s signing key must further be uncompromised upon acceptance.

- \texttt{cleanELT}: This predicate indicates whether the combination of the ephemeral contribution of the initiator and the long-term key of the responder is unknown to the adversary.

- \texttt{cleanESS}: This predicate indicates whether the combination of the ephemeral contribution of the initiator and the semi-static keys of the responder are unknown to the adversary. If the test session is an initiator, the responder’s signing key must further be uncompromised upon acceptance.

If the test session \(\pi^*\) is a responder session, the evaluation of \texttt{cleanELT} and \texttt{cleanESS} necessitates a further predicate called \texttt{cleanpeerE} (in all other cases, it is sufficient to consider the compromise of keys and randomness via the flags \texttt{corrltk}, \texttt{corrssk}, \texttt{revrand}, and \texttt{pcorr}).

- \texttt{cleanpeerE}: This (sub)predicate indicates that the randomness used within any (\texttt{sid-} or \texttt{cid-})partnered session is unknown to the adversary.

For test sessions in full handshake mode, i.e., where \(\pi^*.\texttt{type} = \texttt{full}\), it must either hold that \texttt{cleanreduced} is true or that the additional input to the master secret computation is clean. The latter is captured by the following predicate:

- \texttt{cleanEE}: This predicate indicates that the ephemeral contribution of the test session is unknown to the adversary and that the ephemeral contribution of the peer is unknown to the adversary (captured by \texttt{cleanpeerE}) or was “cleanly” signed (captured by the new \texttt{cleansigE} predicate we introduce). It is the second part of this “or” statement by which our model captures that KEM ephemeral keys are protected by signatures against replacement.

Again, the predicate \texttt{cleanpeerE} helps to determine within \texttt{cleanEE} whether the randomness of the test session’s (contributive) partners is unrevealed. The following predicate precludes that the adversary substitutes the responder’s ephemeral key and signs it with the signing key obtained from corrupting the long-term key.

- \texttt{cleansigE}: This (sub)predicate indicates that for initiator test sessions the peer was not corrupted at the time of acceptance and, if the test session has a contributively partnered session, then the ephemeral contribution of that session is not known to the adversary.
5 PQXDH Analysis

We formalize the PQXDH handshake in Figure 7, highlighting the changes compared to X3DH to achieve post-quantum security in dark gray. Ours is the first reductionist analysis to model the signatures on public keys, which we accent with light gray. Long-term key pairs consist of a DH key pair and a signing key pair. Semi-static key pairs consist of a DH key pair and a KEM key pair, each signed under the long-term signing key. The protocol then works as follows.

First, Bob produces a pre-key bundle with semi-static and ephemeral keys as indicated by the ssid vector and type: The pre-key bundle always includes the semi-static DH key and for reduced handshakes a semi-static KEM key. Only for full handshakes, the pre-key bundle includes ephemeral DH and KEM keys.

Signatures on the semi-static DH key and the KEM key (regardless if it is semi-static or ephemeral) are always included.

Second, Alice verifies both signatures and samples an ephemeral DH key pair of her own. She computes several DH secret combinations, namely long-term/semi-static, ephemeral/long-term, ephemeral/semi-static, and ephemeral/ephemeral (the last one only for full handshakes), and encapsulates against Bob’s KEM public key. She derives the session key via a key derivation function KDF on input all three/four DH shared secrets and the KEM shared secret and sends her sampled DH ephemeral public key and the KEM ciphertext to Bob.

Third, Bob computes the same DH shared secrets and decapsulates the KEM ciphertext. He derives the session key in the same manner and accepts.

Security

We show that PQXDH achieves key indistinguishability wrt. to the maximum-exposure attack vectors formalized through the clean predicates in our model (cf. Section 4). As for the prior analysis of X3DH [CCD+17] whose proof structure we follow in parts, the bound is highly non-tight; we still give it in concrete terms for clarity.

**Theorem 5.1 (Key indistinguishability of PQXDH).** The PQXDH protocol given in Figure 7 with randomness space $R_{KE} = R_{DH,KGen} \times R_{KEM,KGen} \times R_{KEM,Enc} \times R_{SIG,Sig}$ achieves $(t, \epsilon, (q_{Snd}, q_{CorrLT}, q_{CorrSS}, q_{RevR}, q_{RevSK}))$-key indistinguishability, assuming (for $t' \approx t$) the GapDH problem in the group $(G, g, q)$ is $(t', \epsilon_{GDH}, 1.5(q_{RO} + q_{Snd}))$-hard, KEM is a $(t', \epsilon_{CCA}, n_s)$-OW-CCA-secure and $(t', \epsilon_{LEAK+}, n_p, n_{ss} + n_s)$-LEAK+-BIND-SS-{$CT, PK$}-secure KEM with public-key collision probability $\gamma_{coll} + \epsilon_{corr}$ and correctness error $\delta_{corr}$, SIG is a $(t', \epsilon_{SIG}, 2n_{ss} + n_s)$-unforgeable signature scheme, and KDF behaves like a (programmable) random oracle. Concretely, for any efficient adversary $A$ we have

$$\text{Adv}_{\text{PQXDH}}(A) \leq \frac{(n_p + n_p \cdot n_{ss} + n_s)^2}{q} + \gamma_{coll}(n_p \cdot n_{ss} + n_s) + n_s \cdot \delta_{corr} + \epsilon_{LEAK+}$$

$$+ \begin{pmatrix}
(n_p \cdot (\epsilon_{SIG} + n_p \cdot n_{ss} \cdot \epsilon_{GDH})) \\
+ (n_s \cdot n_p \cdot \epsilon_{GDH}) \\
+ (n_p \cdot (\epsilon_{SIG} + n_{ss} \cdot n_s \cdot \epsilon_{GDH})) \\
+ (n_p \cdot (\epsilon_{SIG} + n_{ss} \cdot n_s \cdot \epsilon_{GDH})) \\
+ (n_p \cdot (\epsilon_{SIG} + n_{ss} \cdot \epsilon_{GDH}, \epsilon_{CCA})) \\
+ (n_{ss} \cdot \epsilon_{GDH}, \epsilon_{CCA}) \\
+ (n_p \cdot (\epsilon_{SIG} + n_{ss} \cdot q_{RO} \cdot \epsilon_{CCA})) \\
\end{pmatrix}$$

For easier accessibility, we first give a summary of the proof; the full proof follows below.

---

9The PQXDH description [KS24] suggests to use a single key pair for both DH and the signature scheme XEdDSA [Per16], modeling which we leave to future work.

10Ephemeral DH and KEM pre-keys are used as long as there are some left on the Signal server. In practice, ephemeral DH and KEM pre-keys may run out at different points in time. We do not model this to improve accessibility of the proof.
Figure 7: The PQXDH protocol. Darker gray code lines are additions compared to the original X3DH protocol to add postquantum security. Lighter gray code lines are signature elements of the X3DH and PQXDH not covered in the prior analyses of [CCD+17].
Proof summary. The proof proceeds via a series of game hops. First, we exclude collisions in the DH and KEM public keys, yielding the terms \( (n_p + n_p n_s + n_s)^2 + \gamma_{\text{coll}}(n_p \cdot n_s + n_s) \). Second, we exclude KEM correctness errors when decapsulating in sessions \((n_s \cdot \delta_{\text{corr}})\). Now, soundness holds, as the session identifiers involving only DH secrets (and hence do not provide post-quantum security), or, for the sixth case (compared to no guarantee under this attack vector for \(PQXDH\)), this completes the proof case. Third, we ensure that KEM shared secrets bind the public keys and ciphertexts, introducing the \( \epsilon_{\text{LEAK}+} \) term; this in particular rules out re-encapsulation attacks where sessions that are not partnered possibly derive the same session key.\(^{11}\)

At this stage we separate the proof into several cases based on the clean predicate (cf. the annotated sum in the bound of Theorem 5.1). Let us illustrate the fourth case \((\text{cleanESS} \land \text{type} = \text{reduced})\) here since it covers the most interesting proof techniques, including handling the signatures and a hybrid DH/KEM argument; the other cases follow in a similar fashion. In this case, the cleanESS predicate is satisfied and the adversary is testing a session running a reduced \(\text{type} = \text{reduced}\) handshake. In the proof, we first guess the responder identity involved in the test session, introducing a \(n_p\) loss. Second, we ensure that the guessed responder’s semi-static KEM and DH keys are not tampered with in the test session, reducing to the unforgeability of the signature scheme \((\epsilon_{\text{SIG}})\). Third and fourth, we guess the two semi-static key identifiers of the responder as well as the initiator session, with a loss of \(n_s^2 \cdot n_s\). Fifth, we abort if the adversary queries the key derivation function KDF, modeled as random oracle, on the master secret of the test session. To bound this step, we embed both a GapDH challenge and a OW-CCA challenge into the test session (concretely, into the ephemeral DH and KEM encapsulation contribution of the initiator and the semi-static DH and KEM keys of the responder, which we both guessed before). If the adversary detects this change, the reduction can win both games, yielding the minimum of the advantages as the hybrid bound, i.e., \(\min(\epsilon_{\text{GDH}}, \epsilon_{\text{CCA}})\). Finally, we can replace the session key of the test session with a random key, making it independent of the challenge bit \(b_{\text{test}}\). Concluding that the session key of the test session is now independent of \(b_{\text{test}}\) since non-partnered sessions derive distinct keys (in particular due to the KEM binding hop in the beginning), this completes the proof case. \(\square\)

We get hybrid guarantees for the fourth and fifth proof cases \((\text{cleanESS} \text{ in reduced mode and cleanEE in full mode with a clean peer, i.e., cleanpeerE})\), showing that for these attack vectors, an adversary would need to break both GapDH in the DH group and OW-CCA security of the KEM. The other attack vectors either involve only DH secrets (and hence do not provide post-quantum security), or, for the sixth case \((\text{cleanEE})\) in full mode with clean signatures, the adversary can manipulate the (unsigned) ephemeral DH key, hence only the KEM secret ensures security here. All in all, \(PQXDH\) maintains the classical guarantees of \(X3DH\) for the first three cases (as intended), extends the guarantees for the fourth and fifth case to provide hybrid guarantees, and adds a post-quantum guarantee for the last case due to the signed ephemeral KEM key (compared to no guarantee under this attack vector for \(X3DH\)).

We now give the full proof for Theorem 5.1.

Proof. Game 0. We start with Game \(G_0\) being the original key indistinguishability game \(G_{\text{PQXDH}}^{\text{KI}}(A)\), i.e.,

\[
\text{Adv}_{\text{PQXDH}}^{\text{KI}}(A) = \text{Adv}_{G_0}^{\text{PQXDH}}(A).
\]

Game 1 (DH and KEM key collisions). We let \(G_0\) abort (overwriting the adversary’s output with 0) if any two honestly generated DH keys or KEM keys coincide. There are \(n_p\) many long-term DH keys, \(n_p \cdot n_{\text{ss}}\) many semi-static DH keys, and at most \(n_s\) many ephemeral DH keys, so by the birthday bound

\(^{11}\)As we will detail in the proof (Game 3) and discuss in Section 6, if \(PQXDH\) included the session context (e.g., the values in the session identifier) in the key derivation, as is good practice, this problem would not occur: different KEM public keys or ciphertexts would lead to different session keys (by being part of the KDF input as session context).
the probability for any two DH keys colliding can be upper-bounded by \( \frac{(n_p + n_p \cdot n_{ss} + n_s)^2}{q} \). As for KEM keys, there are \( n_p \cdot n_{ss} \) semi-static ones and at most \( n_s \) ephemeral ones, which collide with probability at most \( \gamma_{coll}(n_p \cdot n_{ss} + n_s) \); cf. Definition 3.1. Put together, we have

\[
\text{Adv}_{G_0}^{\text{PQXDH}}(A) \leq \frac{(n_p + n_p \cdot n_{ss} + n_s)^2}{q} + \gamma_{coll}(n_p \cdot n_{ss} + n_s) + \text{Adv}_{G_1}^{\text{PQXDH}}(A).
\]

**Game 2 (KEM correctness).** We next let \( G_1 \) abort if any honestly created KEM encapsulation fails to decapsulate correctly. This introduces a KEM correctness error term for each of the \( n_s \) many sessions:

\[
\text{Adv}_{G_1}^{\text{PQXDH}}(A) \leq n_s \cdot \delta_{\text{corr}} + \text{Adv}_{G_2}^{G_1}(A).
\]

**Soundness.** At this point soundness holds unconditionally; we consider the three sub-conditions of the sound predicate:

- **Agreement on shared key, type, contributive identifiers (Figure 6, line 18):** The DH keys and the KEM key and the KEM ciphertext in the session identifier determine all inputs to the key derivation function \( \text{KDF} \). By \( G_2 \), KEM ciphertexts decapsulate correctly and both parties derive the same session key. The session identifier further determines the type by \( \text{epk}_B \) being empty (\( \text{type} = \text{reduced} \)) or not (\( \text{type} = \text{full} \)). Finally, the contributive identifier is a subset of the session identifier, hence matching if the latter match.

- **No initiator session identifiers collide (Figure 6, line 19):** Every initiator samples a fresh ephemeral DH key \( \text{epk}^{\text{DH}}_A \) included in the session identifier. These DH keys do not collide by Game \( G_1 \). Hence, initiator session identifiers do not collide.

- **No three session identifiers collide in full mode (Figure 6, line 20):** Three colliding session identifiers implies colliding identifiers for two initiator sessions or for two responder sessions. Collisions of initiator session are ruled out above already. Collisions of responder sessions imply colliding ephemeral DH keys \( \text{epk}^{\text{DH}}_B \), which are ruled out by Game \( G_1 \).

**Game 3 (KEM shared secret collisions).** We will need later that two sessions that use distinct (semi-static or ephemeral) KEM public keys or distinct ciphertexts do not end up using the same shared secret \( ss \) (and hence possibly the same session key, despite not being partnered). If PQXDH included the session context (e.g., the values in the session identifier) in the key derivation, we would not need this game hop, since different KEM public keys or ciphertexts would lead to different session keys. See also the discussion in Section 6.

\[12\]
output of decapsulation for responder sessions) involving two distinct public keys $pk_i$, $pk_j$ or two distinct ciphertexts $ct_i$, $ct_j$, $B_1$ outputs $(i, ct_i, j, ct_j)$ and wins. Hence,

$$\text{Adv}_{\text{PQXDH}}^{G_2}(\mathcal{A}) \leq \epsilon_{\text{LEAK}} + \text{Adv}_{\text{PQXDH}}^{G_1}(\mathcal{A}).$$

**Separating the clean cases.** At this point, we will divide the proof into six sub-cases following the structure of the clean_full resp. clean_reduced predicates evaluated on the test session $\pi^*$. We can bound the advantage of the adversary in Game $G_3$ by the sum of its advantages in Games $G_3[c]$ which are $G_3$ conditioned on a sub-predicate $c$:

$$\text{Adv}_{\text{PQXDH}}^{G_3}(\mathcal{A}) \leq \sum_{c \in \{\text{clean}_{\text{LTSS}}(\pi^*), \text{clean}_{\text{ELT}}(\pi^*), \text{clean}_{\text{ESS}}(\pi^*) \land \pi^*.\text{type} = \text{full}, \text{clean}_{\text{ESS}}(\pi^*) \land \pi^*.\text{type} = \text{full}, \text{clean}_{\text{EE}}(\pi^*) \land \text{clean}_{\text{peerE}}(\pi^*) \land \pi^*.\text{type} = \text{full} \}} \text{Adv}_{\text{PQXDH}}^{G_3[c]}(\mathcal{A}).$$

In the remainder of the proof, we will bound each of these cases, numbered A–F, separately.

**Case A (clean_{LTSS}(\pi^*))**.

In this proof case, the clean_{LTSS} predicate ensures for the test session $\pi^*$ that

1. the initiator’s long-term key is uncompromised,
2. the responder’s semi-static key is uncompromised, and
3. if $\pi^*$ is an initiator, the responder’s long-term (signing) key was uncompromised upon acceptance.

Via the last point, we can guarantee that initiator and responder indeed agree on the semi-static key (identifier), given signatures are unforgeable. Then, similar to the classical Signal proof [CCD+17], the uncompromised long-term/semi-static Diffie–Hellman combination then ensures key indistinguishability for the test session.

**Game A.0.** This case begins with Game $G_3$ conditioned on clean_{LTSS}(\pi^*) being satisfied. 

$$\text{Adv}_{\text{PQXDH}}^{G_{A.0}}(\mathcal{A}) = \text{Adv}_{\text{PQXDH}}^{G_{3[\text{clean}_{\text{LTSS}}(\pi^*)]}}(\mathcal{A}).$$

**Game A.1 (Guess responder identity $V^*$).** We first guess the identity $V^*$ of the responder involved in the test session, overwriting the adversary’s bit guess with 0 if this guess was incorrect. This step loses at most a factor of the number of users $n_p$:

$$\text{Adv}_{\text{PQXDH}}^{G_{A.0}}(\mathcal{A}) \leq n_p \cdot \text{Adv}_{\text{PQXDH}}^{G_{A.1}}(\mathcal{A}).$$

**Game A.2 (Signature unforgeability).** We now abort the game (again, returning 0 as the adversary’s bit guess) in the event that the test session $\pi^*$ is an initiator session and accepts using a semi-static DH public key $\text{sspk}_{V^*}$ that was not generated through a $KGenSS$ run for $V^*$. This ensures that the test session accepts with $\pi^*.\text{sspk} = (\text{sspk}_{V^*}, \cdot)$ corresponding to a DH key pair of which the adversary does
not know the secret key, since cleanLTSS($\pi^*$) guarantees that the semi-static secret belong to $sspk_{V^*}^{DH}$ is not compromised. The probability of such an abort can be bounded by the advantage of the following reduction $B_2$ against the $(t, \epsilon_{SIG}, 2n_{ss} + n_s)$-unforgeability of $SIG$.

The reduction $B_2$ samples all key components itself except for the signature key of $V^*$: In place of the long-term public signature key $ltk_{V^*}^{SIG}$ it uses the public key $pk$ obtained in its unforgeability game. In its simulation of Game $G_{A,1}$, $B_2$ uses its signing oracle to obtain signatures under $ltk_{V^*}^{SIG}$: two per semi-static key (for DH and KEM public keys) and up to $n_s$ signatures on ephemeral KEM keys. Since cleanLTSS($\pi^*$) = true, we know that $\pi^*.pcorr = false$, i.e., the long-term key of $V^*$ was not corrupted when the test session accepted and so $B_2$ does not have to answer a CORRUPTLTKEY($V^*$) query prior to the abort event. Hence, $B_2$ provides a perfect simulation of Game $G_{A,1}$ up to when the abort would happen, and if $\pi^*$ receives a signature $\sigma_{V^*}$ on a semi-static DH public key that $V^*$ did not generate, then $B_2$ can output this as its forgery and wins. Thus,

$$Adv_{PQXDH}^{G_{A,1}}(A) \leq \epsilon_{SIG} + Adv_{PQXDH}^{G_{A,2}}(A).$$

**Game A.3 (Guess initiator identity $U^*$).** We guess the identity $U^*$ of the initiator to the test session, overwriting the adversary’s bit guess with 0 if this guess was incorrect. This step again loses at most a factor of the number of users $n_p$:

$$Adv_{PQXDH}^{G_{A,2}}(A) \leq n_p \cdot Adv_{PQXDH}^{G_{A,3}}(A).$$

**Game A.4 (Guess semi-static key identifier ssid* of $V^*$).** We now guess the identifier ssid of the responder $V^*$’s (uncorrupted) semi-static DH key $sspk_{V^*}^{DH}$. Note that depending on the role of $\pi^*$ this is either the test session’s own key (if $\pi^*.role = responder$), or of the intended peer (if $\pi^*.role = initiator$). We denote the guessed identifier by ssid*, and abort, setting the adversary’s output bit to 0, if this guess is incorrect, losing at most a factor of the number of semi-static keys per user $n_{ss}$:

$$Adv_{PQXDH}^{G_{A,3}}(A) \leq n_{ss} \cdot Adv_{PQXDH}^{G_{A,4}}(A).$$

**Game A.5 (GapDH).** We now abort the game (again, returning 0 as the adversary’s bit guess) in the event that the adversary queries the random oracle on a value formatted like a master secret and beginning with CDH($ltk_{V^*}^{DH}, sspk_{V^*}^{DH}$), i.e., the shared DH key between $ltk_{V^*}^{DH}$ and $sspk_{V^*}^{DH}$, where $sspk_{V^*}^{DH}$ is the semi-static DH key with identifier ssid*. The probability of such an abort can be bounded by the advantage of the following reduction $B_3$ against the $(t, \epsilon_{GDH}, g_{DDH})$-hardness of the GapDH problem in $(G, g, q)$.

Reduction $B_3$ samples all key components itself except for replacing the long-term DH key $ltk_{U^*}^{DH}$ of $U^*$ and the semi-static DH key $sspk_{V^*}^{DH}$ of $V^*$ with its own challenge values $g^a$ and $g^b$. Since cleanLTSS($\pi^*$) = true, $B_3$ never has to answer the queries CORRUPTLTKEY($U^*$) or CORRUPTSSKEY($V^*, ssid^*$) in its simulation of Game $G_{A,4}$. Though, $B_3$ may have to answer for REVEALSESSKEY and SEND queries that involve the substituted keys. Hence, the reduction patches the random oracle and the SEND oracle to generate the session keys in a consistent manner. In consequence, the reduction answers queries to the REVEALSESSKEY oracle consistently as well.

Figure 8 gives an algorithmic description of the changes. In particular, the reduction gives special treatment to SEND queries that involve any of the challenge secrets, i.e., for SEND queries where the initiator is the initiator of the test session $U^*$ partnered with the responder of the test session $V^*$ using the semi-static key ssid* of the test session, the responder is $U^*$, or the responder is $V^*$ with semi-static key ssid* and the initiator is not $U^*$. In these cases the reduction cannot compute all DH shared secrets.
$RO'(x)$:
1. if $DH_1 || DH_2 || DH_3 || DH_4 || ss \equiv x$ // if we can parse the query string $x$ as three or four DH shared secrets ($DH_i$ may be empty) and a KEM shared secret...
2. if $DDH(ltpk_U^{DH}, sspk_V^{DH}, DH_1)$ // referring to $V$'s key from $sid^*$
3. $B_3$ halts and returns $DH_1$ as its GapDH solution // change for Game $G_{A.5}$
4. return patch($DH_1, DH_2, DH_3, DH_4, ss$) // extra routine to ensure consistency
5. return $RO(x)$

patch($PoS_1, PoS_2, PoS_3, DH_4, ss$):
6. foreach $(e_1, e_2, e_3, e_4, e_5, y)$ in $L$ // iterate over list entries
7. if $DDH-eq(PoS_1, e_1) \land DDH-eq(PoS_2, e_2) \land DDH-eq(PoS_3, e_3) \land DH_4 = e_4 \land ss = e_5$
8. return $y$ // query matches a prior one, respond with recorded value $y$
9. $y \leftarrow \{0, 1\}^{256}$ // new query, sample value at random
10. $L \leftarrow L \cup \{(PoS_1, PoS_2, PoS_3, DH_4, ss, y)\}$ // record response $y$ on this query
11. return $y$

$DDH-eq(PoS_1, PoS_2)$:
12. if $(pk_1, pk_2) = PoS_1 \land (pk_3, pk_4) = PoS_2$ // if both PoS$_1$ and PoS$_2$ parse as two public keys...
13. return $(pk_1, pk_2) = (pk_3, pk_4)$
14. if $(pk_1, pk_2) = PoS_1$ // if PoS$_1$ parses as two public keys...
15. return DDH($pk_1, pk_2, PoS_2$)
16. if $(pk_1, pk_2) = PoS_2$ // if PoS$_2$ parses as two public keys...
17. return DDH($pk_1, pk_2, PoS_1$)
18. return $PoS_1 = PoS_2$ // else, both are DH shared secrets

Send($W, i, m$) substitutes KDF call in Run where GapDH challenge keys are involved:
19. ...
20. $U \leftarrow \pi_{W, \text{initiator}}$ // $V \leftarrow \pi_{W, \text{responder}}$ $ssid \leftarrow \pi_{W, \text{ssid}}$
21. $(\ldots, epk_U^{DH}, \ldots) \leftarrow \pi_{W, \text{sid}}$
22. if $U = U^* \land V = V^* \land ssid = ssid^*$
23. compute $DH_2, DH_3, ss$ with the corresponding secret keys as specified in Run
24. $\pi_{W, K} \leftarrow \text{patch}(ltpk_U^{DH}, sspk_{V^*}^{DH}, DH_2, (epk_U^{DH}, sspk_{V^*}^{DH}, DH_4, ss))$
25. else if $V = U^*$
26. compute $DH_1, DH_3, ss$ with the corresponding secret keys as specified in Run
27. $\pi_{W, K} \leftarrow \text{patch}(DH_1, (epk_U^{DH}, ltpk_U^{DH}), DH_3, DH_4, ss))$
28. else if $U \neq U^* \land V = V^* \land ssid = ssid^*$
29. compute $DH_1, DH_2, DH_4, ss$ with the corresponding secret keys as specified in Run
30. $\pi_{W, K} \leftarrow \text{patch}(DH_1, DH_2, (epk_U^{DH}, sspk_{V^*}^{DH}), DH_4, ss))$
31. else // $B_3$ can compute all secrets
32. compute $DH_1, DH_2, DH_3, DH_4, ss$ with the corresponding secret keys as specified in Run
33. $\pi_{W, K} \leftarrow \text{patch}(DH_1, DH_2, DH_3, DH_4, ss))$
34. ...

Figure 8: Algorithmic description of how $B_3$ simulates the random oracle and the Send oracle in game $G_{A.5}$. To ensure consistency of queries we use an additional (global) list $L$, which is initialized as empty. This list $L$ contains six-tuples consisting of three pairs of DH public keys or DH shared secrets (PoS$_i$), a DH shared secret, a KEM shared secret, and the associated RO output. The patch routine checks if incoming queries match list entries with the DDH-eq routine: DDH-eq returns true if both PoS argument are the same two public keys, if the two PoS arguments are a valid DH tuple according to the DDH oracle, or if the two PoS arguments are identical DH shared secrets. We patch the Send oracle to use the patch routine in lieu of computing the KDF, whenever there are challenge secret keys involved that $B_3$ does not know.
entering the key derivation function and chooses a session key via the patch routine. Specifically, the patch routine takes as arguments three pairs of DH public keys or DH shared secrets PoS (since \(DH_1, DH_2, DH_3\) are the DH shared secrets that the reduction can potentially not compute), a DH shared secret, and a KEM shared secret, and returns a RO output that is consistent with previous queries. To this end, patch saves all queries together with their RO response in a list \(L\) and checks new queries against \(L\). Note here that two PoS are equivalent if the pair of public keys in one of them forms a valid Diffie–Hellman triple with the shared secret in the other one according to the DDH oracle or if the PoS components are equivalent, as described by the DDH-eq routine. If a query does not match any previous query, the patch routine randomly chooses a new session key and records the query with the response in \(L\).

Since the Send oracle computes consistent session keys for each session, \(B_3\) also consistently answers to the RevealSessKey oracle queries. To ensure consistency with queries to the random oracle, the reduction also uses the patch routine for RO queries. Finally, in case the adversary queries the random oracle on a value beginning with \(DH\) shared secrets that the reduction can potentially not compute, a DH shared secret, and a KEM shared secret, and returns a RO output that is consistent with previous queries. To this end, patch saves all queries together with their RO response in a list \(L\) and checks new queries against \(L\). Note here that two PoS are equivalent if the pair of public keys in one of them forms a valid Diffie–Hellman triple with the shared secret in the other one according to the DDH oracle or if the PoS components are equivalent, as described by the DDH-eq routine. If a query does not match any previous query, the patch routine randomly chooses a new session key and records the query with the response in \(L\).

Since the Send oracle computes consistent session keys for each session, \(B_3\) also consistently answers to the RevealSessKey oracle queries. To ensure consistency with queries to the random oracle, the reduction also uses the patch routine for RO queries. Finally, in case the adversary queries the random oracle on a value beginning with \(DH\) shared secrets that the reduction can potentially not compute, a DH shared secret, and a KEM shared secret, and returns a RO output that is consistent with previous queries. To this end, patch saves all queries together with their RO response in a list \(L\) and checks new queries against \(L\). Note here that two PoS are equivalent if the pair of public keys in one of them forms a valid Diffie–Hellman triple with the shared secret in the other one according to the DDH oracle or if the PoS components are equivalent, as described by the DDH-eq routine. If a query does not match any previous query, the patch routine randomly chooses a new session key and records the query with the response in \(L\).

Unless \(A\) queries the random oracle on \(DH(ltpk_{V^*}^{DH}, sspk_{V^*}^{DH})\) in the \(DH_1\) position, \(B_3\) provides a perfect simulation of game \(G_{A,4}\). If \(A\) makes such a random oracle query, then \(B_3\) will detect this and win its GapDH game. Thus,

\[
\text{Adv}_{G_{A,4}}^{PQXDH}(A) \leq \epsilon_{GDH} + \text{Adv}_{G_{A,5}}^{PQXDH}(A).
\]

**Game A.6 (Replacing the session key).** We now replace the session key of the test session \(\pi^*\) with a uniformly sampled key. Since Game \(G_{A,5}\) has ruled out that the adversary queries the RO on the master secret \(DH_1||DH_2||DH_3||DH_4\|ss\) of the test session \(\pi^*\), the adversary has no chance of detecting this change. Thus,

\[
\text{Adv}_{G_{A,5}}^{PQXDH}(A) = \text{Adv}_{G_{A,6}}^{PQXDH}(A).
\]

To conclude this proof case, observe that in Game \(G_{A,6}\) the session key of the test session \(\pi^*\) is now a uniformly random key, independent of \(b_{\text{test}}\). Furthermore, \(A\) cannot reveal the session key of the test session \(\pi^*\) via a RevealSessKey query on \(\pi^*\) or any partnered session which might hold the same key. Finally, any non-partnered session will derive a different session key: any difference in the identities \(A, B\) yields different DH public keys by Game \(G_1\), any difference in the DH shares implies a difference in the DH[1]–DH[4] inputs, while differing KEM public keys or ciphertexts yield a different ss input by Game \(G_3\). Thus, \(A\) cannot gain any information about the test bit \(b_{\text{test}}\) and can do no better than to guess:

\[
\text{Adv}_{G_{A,6}}^{PQXDH}(A) \leq 0.
\]

\(^{13}\)If PQXDH would include the session context (e.g., the values in the session identifier) in the key derivation, as is good practice, the reduction would need less DDH oracle queries: Using the context, the reduction can detect when a challenge DH key is combined with a maliciously generated (ephemeral) key. In the modified Send oracle in Figure 8, only the first PoS in line 24 remains a pair of public keys, while the other three occurrences are solved via context. In consequence, at maximum PoS per entry in \(L\) contains two public keys and we get a maximum of \(q_{RO} + q_{Snd}\) DDH queries.

28
Case B (clean_{ELT}(\pi^*))

In this proof case, the clean_{ELT} predicate ensures for the test session \pi^* that

1. the initiator's randomness is not revealed, and
2. the responder's long-term key is uncompromised.

Similar to the classical Signal proof [CCD+17], the uncompromised ephemeral/long-term Diffie–Hellman combination ensures key indistinguishability for the test session.

Game B.0. This case begins with Game G_3 conditioned on clean_{ELT}(\pi^*) being satisfied.

$$
\text{Adv}_{PQXDH}^{G_{0,0}}(A) = \text{Adv}_{PQXDH}^{G_{0,[\text{clean}_{ELT}(\pi^*)]}(A)}.
$$

Game B.1 (Guess initiator session). We guess the initiator session \pi_i^* contributing to the test session \pi^* (i.e., either the test session itself if it is an initiator session, or the initiator session partnered to the test session), overwriting the adversary's bit guess with 0 if this guess was incorrect. This step loses at most a factor of the number of sessions \(n_s\):

$$
\text{Adv}_{PQXDH}^{G_{0,1}}(A) \leq n_s \cdot \text{Adv}_{PQXDH}^{G_{0,2}}(A).
$$

Game B.2 (Guess responder identity V^*). We guess the identity V^* of the responder involved in the test session, overwriting the adversary’s bit guess with 0 if this guess was incorrect. This step loses at most a factor of the number of users \(n_p\):

$$
\text{Adv}_{PQXDH}^{G_{0,2}}(A) \leq n_p \cdot \text{Adv}_{PQXDH}^{G_{0,3}}(A).
$$

Game B.3 (GapDH). We now abort the game (again, returning 0 as the adversary’s bit guess) in the event that the adversary queries the random oracle on a value formatted like a master secret and beginning with CDH(epk^{DH}, ltpk^{DH}), i.e., the shared DH key between the ephemeral key of the test session’s initiator epk^{DH} and ltpk^{DH}. The probability of such an abort can be bounded by the advantage of the following reduction \(B_4\) against the \((t, \epsilon_{GDH}, q_{DDH})\)-hardness of the GapDH problem in \((G, g, q)\).

The reduction follows the idea of Game G_{A.5}, only that now we embed the GapDH challenge keys as the ephemeral key in the initiator session \pi_i^* and as the long-term key of \(V^*\). The reduction does not need to answer corresponding REVEALRAND queries on \pi_i^* or CORRUPTLTKEY queries on \(V^*\) due to clean_{ELT}. It can use a similar strategy to the one in Game G_{A.5} to patch the random oracle, yet it has an easier time ensuring consistency of the Send oracle: It can always compute \DH_1 and \DH_2 by using the responder’s semi-static secret key, and \DH_3 (if it is a full handshake) with the honest party’s ephemeral secret key. Outside of the test session \(B_4\) can use the initiator’s ephemeral secret key to compute \DH_2 if the ephemeral key was honestly generated. Hence, \(B_4\) can compute all inputs to the KDF for all Send queries itself, except for the Send queries to \(V^*\) with a malicious ephemeral key on the initiator side. In consequence, \(B_4\) queries the DDH oracle at most once per Send query and once when checking RO queries for the \DH_2 position. All in all, \(B_4\) needs a maximum of \(q_{Send} + q_{RO}\) queries to the DDH oracle, which is well below the number in Game G_{A.5}.

Unless \(A\) queries the random oracle on CDH(epk^{DH}, ltpk^{DH}) in the \DH_2 position, \(B_4\) provides a perfect simulation of G_{B.2}. If \(A\) does make such a random oracle query, then \(B_4\) will detect this and win its GapDH game. Thus,

$$
\text{Adv}_{PQXDH}^{G_{B.2}}(A) \leq \epsilon_{GDH} + \text{Adv}_{PQXDH}^{G_{B.3}}(A).
$$

29
Game B.4 (Replacing the session key). We now replace the session key of the test session \( \pi^* \) with a uniformly sampled key. Since Game \( \mathcal{G}_{B,3} \) has ruled out that the adversary queries the RO on the master secret \( DH_1 \| DH_2 \| DH_3 \| DH_4 \| ss \) of the test session \( \pi^* \), the adversary has no chance of detecting this change. Thus,
\[
\text{Adv}_{\mathcal{G}_{B,4}}(A) = \text{Adv}_{\mathcal{G}_{B,3}}(A).
\]

To conclude this proof case, observe that in Game \( \mathcal{G}_{B,4} \) the session key of the test session \( \pi^* \) is now a uniformly random key, independent of \( b_{\text{test}} \). As before, \( A \) can neither reveal the session key of the test session \( \pi^* \) nor of any partnered session, nor does any other session derive the same session key by Game \( \mathcal{G}_3 \). Thus, \( A \) cannot gain any information about the test bit \( b_{\text{test}} \) and can do no better than to guess:
\[
\text{Adv}_{\mathcal{G}_{B,4}}(A) \leq 0.
\]

Case C (\( \text{cleanESS}(\pi^*) \land \pi^*.\text{type} = \text{full} \)).

In this proof case, the test session performs a full handshake and the \( \text{cleanESS} \) predicate ensures for the test session \( \pi^* \) that
1. the initiator’s randomness is not revealed, and
2. the responder’s semi-static key is uncompromised, and
3. if \( \pi^* \) is an initiator, the responder’s long-term (signing) key was uncompromised upon acceptance.

Via the last point, similarly to Case A, we can guarantee that initiator and responder indeed agree on the semi-static key (identifier), given signatures are unforgeable. Then, similar to the classical Signal proof [CCD+17], the uncompromised ephemeral/semi-static Diffie–Hellman combination ensures key indistinguishability for the test session.

Game C.0. This case begins with Game \( \mathcal{G}_3 \) conditioned on \( \text{cleanESS}(\pi^*) \land \pi^*.\text{type} = \text{full} \) being satisfied.
\[
\text{Adv}_{\mathcal{G}_{C,0}}(A) = \text{Adv}_{\mathcal{G}_{[\text{cleanESS}(\pi^*) \land \pi^*.\text{type} = \text{full}]}}(A).
\]

Game C.1 (Guess responder identity \( V^* \)). We first guess the identity \( V^* \) of the responder to the test session, overwriting the adversary’s bit guess with 0 if this guess was incorrect. This step loses at most a factor of the number of users \( n_p \):
\[
\text{Adv}_{\mathcal{G}_{C,0}}(A) \leq n_p \cdot \text{Adv}_{\mathcal{G}_{C,1}}(A).
\]

Game C.2 (Signature unforgeability). We now abort the game (again, returning 0 as the adversary’s bit guess) in the event that the test session \( \pi^* \) is an initiator session and accepts using a semi-static DH public key \( \text{sspk}_{V^*}^{\text{DH}} \) that was not generated through a \( \text{KGenSS} \) run for \( V^* \). This ensures that the test session accepts with \( \pi^*.\text{sspsk} = (\text{sspk}_{V^*}^{\text{DH}}, \cdot) \) corresponding to a DH key pair of which the adversary does not know the secret key, since \( \text{cleanESS}(\pi^*) \) guarantees that the semi-static secret belong to \( \text{sspk}_{V^*}^{\text{DH}} \) is not compromised. Similar to Game \( \mathcal{G}_{A,2} \), the probability of such an abort can be bounded by the advantage of a reduction against the \( (t, \epsilon_{\text{SIG}}, 2n_{ss} + n_s) \)-unforgeability of \( \text{SIG} \). The argument is identical to the Game \( \mathcal{G}_{A,2} \). Thus,
\[
\text{Adv}_{\mathcal{G}_{C,1}}(A) \leq \epsilon_{\text{SIG}} + \text{Adv}_{\mathcal{G}_{C,2}}(A).
\]
Game C.3 (Guess semi-static key identifier $\text{ssid}^*$ of $V^*$). We now guess the identifier $\text{ssid}$ of the responder $V^*$'s (uncorrupted) semi-static key $\text{sspk}_{V^*}$. Note that depending on the role of $\pi^*$ this is either the test session’s own key (if $\pi^*.\text{role} = \text{responder}$), or of the intended peer (if $\pi^*.\text{role} = \text{initiator}$). We denote the guessed identifier by $\text{sspk}_{V^*}^*$, and abort, setting the adversary’s output bit to 0, if this guess is incorrect, losing at most a factor $n_{ss}$ of the number of semi-static keys per user:

$$\text{Adv}_{\text{PQXDH}}^{G_{C.3}}(A) \leq n_{ss} \cdot \text{Adv}_{\text{PQXDH}}^{G_{C.3}}(A).$$

Game C.4 (Guess initiator session). We guess the initiator session $\pi_i^*$ involved in the test session $\pi^*$ (i.e., either the test session itself if it is an initiator session, or the initiator session partnered to the test session), overwriting the adversary’s bit guess with 0 if this guess was incorrect. This step loses at most a factor of the number of sessions $n_s$:

$$\text{Adv}_{\text{PQXDH}}^{G_{C.4}}(A) \leq n_s \cdot \text{Adv}_{\text{PQXDH}}^{G_{C.4}}(A).$$

Game C.5 (GapDH). We now abort the game (again, returning 0 as the adversary’s bit guess) in the event that the adversary queries the random oracle on a value formatted like a master secret, beginning with $\text{CDH}(epk_{V^*}, \text{sspk}_{V^*})$, i.e., the shared DH key between the ephemeral key of the test session’s initiator $epk_{V^*}$ and $\text{sspk}_{V^*}$, where $\text{sspk}_{V^*}$ is the semi-static DH key with identifier $\text{ss}$.

By contrast, Game C.3 has an advantage $\text{Adv}_{\text{PQXDH}}^{G_{C.3}}(A)$ of $n_{ss}$ over the GapDH problem in $(G, g, q)$, which is well below the number in Game C.5.

The proof follows the idea of Games $G_{A.5}$ and $G_{B.3}$. Though, now we embed the GapDH challenge keys as the ephemeral key in the initiator session $\pi_i^*$ and as semi-static key of $V^*$ with semi-static id $\text{ssid}^*$. The reduction does not need to answer corresponding $\text{REVEALRAND}$ queries on $\pi_i^*$ or $\text{CORRUPTSSKETCH}(V^*, \text{ss}^*)$ queries due to $\text{cleanESS}$. It can use a similar strategy to patch the random oracle: It can always compute $\text{DH}_1$ and $\text{DH}_2$ by using the initiator’s and responder’s long-term secret key, respectively, and $\text{DH}_4$ with the honest party’s ephemeral secret key. Outside of the test session $B_5$ can use the initiator’s ephemeral secret key to compute $\text{DH}_3$ if the ephemeral key was honestly generated. Hence, $B_5$ can compute all inputs to the KDF for all $\text{SEND}$ queries itself, except for the $\text{SEND}$ queries to the test session or to $V^*$ with $\text{ss}$ and a malicious ephemeral key on the initiator side. In consequence, $B_5$ queries the DDH oracle at most once per $\text{SEND}$ query and once when checking RO queries for the $\text{DH}_3$ position. All in all, $B_5$ needs a maximum of $q_{\text{ss}} + q_{\text{RO}}$ queries to the DDH oracle, which is well below the number in Game $G_{A.5}$.

Unless $A$ queries the random oracle on $\text{CDH}(epk_{V^*}, \text{sspk}_{V^*})$ in the $\text{DH}_2$ position, $B_5$ provides a perfect simulation of $G_{C.4}$. If $A$ does make such a random oracle query, then $B_5$ will detect this and win its GapDH game. Thus,

$$\text{Adv}_{\text{PQXDH}}^{G_{C.5}}(A) \leq \epsilon_{\text{GDH}} + \text{Adv}_{\text{PQXDH}}^{G_{C.4}}(A).$$

Game C.6 (Replacing the session key). We now replace the session key of the test session $\pi^*$ with a uniformly sampled key. Since Game $G_{C.5}$ has ruled out that the adversary queries the RO on the master secret $\text{DH}_1 || \text{DH}_2 || \text{DH}_3 || \text{SS} \cdot s$ of the test session $\pi^*$, the adversary has no chance of detecting this change. Thus,

$$\text{Adv}_{\text{PQXDH}}^{G_{C.6}}(A) = \text{Adv}_{\text{PQXDH}}^{G_{C.3}}(A).$$

To conclude this proof case, observe that in Game $G_{C.6}$ the session key of the test session $\pi^*$ is now a uniformly random key, independent of $b_{\text{test}}$. As before, $A$ can neither reveal the session key of the test session $\pi^*$ nor of any partnered session, nor does any other session derive the same session key by Game $G_3$.

Thus, $A$ cannot gain any information about the test bit $b_{\text{test}}$ and can do no better than to guess:

$$\text{Adv}_{\text{PQXDH}}^{G_{C.6}}(A) \leq 0.$$
Case D ($\text{cleanESS}(\pi^*) \land \pi^*.\text{type} = \text{reduced}$).

In this proof case, the test session performs a reduced handshake and the $\text{cleanESS}$ predicate ensures for the test session $\pi^*$ that

1. the initiator’s randomness is not revealed, and
2. the responder’s semi-static key is uncompromised, and
3. if $\pi^*$ is an initiator, the responder’s long-term (signing) key was uncompromised upon acceptance.

Via the last point, we can guarantee that initiator and responder indeed agree on the semi-static key (identifier), given signatures are unforgeable. The difference to Case C is that, in a reduced handshake, the semi-static KEM public key of the responder is used (and there is no ephemeral/ephemeral Diffie–Hellman combination). This means we get the following hybrid guarantee: both the ephemeral/semi-static Diffie–Hellman combination being uncompromised and the semi-static KEM key being uncompromised are, on their own, sufficient to ensure key indistinguishability for the test session.

Game D.0. This case begins with Game $G_3$ conditioned on $\text{cleanESS}(\pi^*) \land \pi^*.\text{type} = \text{reduced}$ being satisfied.

$$\text{Adv}^{G_0}_{\text{PQXDH}}(A) = \text{Adv}^{G_3|\text{cleanESS}(\pi^*) \land \pi^*.\text{type}=\text{reduced}}_{\text{PQXDH}}(A).$$

Game D.1 (Guess responder identity $V^*$). We first guess the identity $V^*$ of the responder to the test session, overwriting the adversary’s bit guess with 0 if this guess was incorrect. This step loses at most a factor of the number of users $n_p$:

$$\text{Adv}^{G_0}_{\text{PQXDH}}(A) \leq n_p \cdot \text{Adv}^{G_1}_{\text{PQXDH}}(A).$$

Game D.2 (Signature unforgeability). In this game case, we are interested in the authenticity of both semi-static DH and KEM keys, and hence now abort the game (again, returning 0 as the adversary’s bit guess) in the event that the test session $\pi^*$ is an initiator session and accepts using a semi-static DH public key $\text{sspk}_{V^*}^{\text{DH}}$ or a semi-static KEM public key $\text{sspk}_{V^*}^{\text{KEM}}$ that was not generated through a $\text{KGenSS}$ run for $V^*$. Since for reduced handshakes both public keys are recorded in $\pi^*.\text{sspks} = (\text{sspk}_{V^*}^{\text{DH}}, \text{sspk}_{V^*}^{\text{KEM}})$, this ensures that the test session accepts with semi-static DH and KEM public keys of which the adversary does not know the secret, since $\text{cleanESS}(\pi^*)$ guarantees that the semi-static secrets corresponding to $\text{sspk}_{V^*}^{\text{DH}}$ or to $\text{sspk}_{V^*}^{\text{KEM}}$ are not compromised.

Again similar to Game $G_{A_2}$, we can reduce this hop to the $(t, \epsilon_{\text{SIG}}, 2n_{ss} + n_s)$-unforgeability of $\text{SIG}$:

$$\text{Adv}^{G_1}_{\text{PQXDH}}(A) \leq \epsilon_{\text{SIG}} + \text{Adv}^{G_{A_2}}_{\text{PQXDH}}(A).$$

Game D.3 (Guess semi-static key identifiers $\text{ssid}_1^*$, $\text{ssid}_2^*$ of $V^*$). We now guess the ssid identifiers of the responder $V^*$’s (uncorrupted) semi-static DH and KEM keys, $\text{sspk}_{V^*}^{\text{DH}}$ resp. $\text{sspk}_{V^*}^{\text{KEM}}$. Note that depending on the role of $\pi^*$ this is either the test session’s own key (if $\pi^*.\text{role} = \text{initiator}$), or of the intended peer (if $\pi^*.\text{role} = \text{responder}$). We denote the guessed identifiers by $\text{ssid}_1^*$ (for the DH key) and $\text{ssid}_2^*$ (for the KEM key), and abort, setting the adversary’s output bit to 0, if this guess is incorrect, losing at most a factor $n_{ss}$ of the number of semi-static keys of each type per user, for each of the two guesses:

$$\text{Adv}^{G_2}_{\text{PQXDH}}(A) \leq n_{ss}^2 \cdot \text{Adv}^{G_{D_3}}_{\text{PQXDH}}(A).$$

32
**Game D.4 (Guess initiator session).** We guess the initiator session $\pi^*_i$ involved in the test session $\pi^*$ (i.e., either the test session itself if it is an initiator session, or the initiator session partnered to the test session), overwriting the adversary’s bit guess with 0 if this guess was incorrect. This step loses at most a factor of the number of sessions $n_s$:

$$\text{Adv}^G_{\pi\text{XDH}}(A) \leq n_s \cdot \text{Adv}^G_{\pi\text{XDH}}(A).$$

**Game D.5 (GapDH + OW-CCA).** We now abort the game (again, returning 0 as the adversary’s bit guess) in the event that the adversary queries the random oracle on a value formatted like a master secret, beginning with \(\text{CDH}(epk^{DH}, sspk^{DH}_V^*)\), i.e., the shared DH key between the ephemeral key of the test session’s initiator $e_{\text{KEM}}^{DH}$ and $sspk^{DH}_V^*$, and ending with the KEM shared secret $ss^*$ resulting from $\pi^*_i$ encapsulating against $sspk^{KEM}_V^*$, where $sspk^{DH}_V^*$ and $sspk^{KEM}_V^*$ are the semi-static DH and KEM keys in $sspk^{ssid^*_i}$. The probability of such an abort can be bounded by the minimum of the advantages of the following reduction $B_6$ playing simultaneously against the $(t, \epsilon_{GDH}, q_{\text{DDH}})$-hardness of the GapDH problem in $(G, g, q)$ and $(t, \epsilon_{CCA}, n_s)$-OW-CCA security of KEM.

The reduction embeds the GapDH challenge as the ephemeral DH share in the initiator session $\pi^*_i$ and as semi-static DH key of $V^*$ with semi-static id $ssid^*_i$ (as in Game $G_{C.5}$), and the KEM public key $p_{\text{KEM}}^{DH}$ from the OW-CCA game as the semi-static key $sspk^{DH}_V^*$ with semi-static id $ssid^*_i$. Furthermore, it uses the KEM challenge ciphertext $ct^*$ in $\pi^*_i$. Similar to Game $G_{C.5}$, the reduction does not need to answer corresponding $\text{REVEALRAND}$ on $\pi^*_i$ or $\text{CORRUPTSSKEY}(V^*, ssid^*_i/ssid^*_i)$ queries due to $\text{cleanESS}$. Furthermore, as for Game $G_{C.5}$, patching the $\text{Snd}$ oracle is relatively easy: Except for queries to the test session or to $V^*$ using $sspk^{DH}_V^*$ and a malicious ephemeral DH key on the initiator side, the reduction can compute all DH shared secrets with the DH secret keys and it can decapsulate the KEM shared secret with the KEM secret key or, for sessions using $sspk^{DH}_V^*$, the $\text{DECAPS}$ oracle of the OW-CCA game. Otherwise (and in particular in the test session), the reduction uses a freshly sampled key as session key. If the DDH oracle accepts a query, then the reduction returns the $DH_3$ and $ss^*$ from the corresponding RO query to its GapDH and OW-CCA games after finishing the simulation for $A$.

The reduction ensures consistency with the random oracle by checking RO queries against the above mentioned patches. In consequence, the remaining queries to the DDH oracle are once per $\text{Snd}$ query and when checking RO queries for the $DH_3$ position. All in all, $B_6$ needs a maximum of $q_{\text{Snd}} + q_{\text{RO}}$ queries to the DDH oracle, which is well below the number in $G_{A.5}$; it further makes at most $n_s$ many $\text{DECAPS}$ query in the OW-CCA game.

Unless $A$ queries the random oracle on $\text{CDH}(epk^{DH}_V, sspk^{DH}_V^*)$ in the $DH_3$ position and $ss^*$ in the $ss$ position, $B_6$ provides a perfect simulation of $G_{D.4}$. If $A$ does make such a random oracle query, then $B_6$ will detect this and win both its GapDH game and its OW-CCA game. We can thus bound the game hop by the minimum advantage against the two games:

$$\text{Adv}^G_{\pi\text{XDH}}(A) \leq \min(\epsilon_{GDH}, \epsilon_{CCA}) + \text{Adv}^G_{\pi\text{XDH}}(A).$$

**Game D.6 (Replacing the session key).** We now replace the session key of the test session $\pi^*$ with a uniformly sampled key. Since Game $G_{D.5}$ has ruled out that the adversary queries the RO on the master secret $DH_1 \| DH_2 \| DH_3 \| DH_4 \| ss$ of the test session $\pi^*$, the adversary has no chance of detecting this change. Thus,

$$\text{Adv}^G_{\pi\text{XDH}}(A) = \text{Adv}^G_{\pi\text{XDH}}(A).$$

To conclude this proof case, observe that in Game $G_{D.6}$ the session key of the test session $\pi^*$ is now a uniformly random key, independent of $b_{\text{test}}$. As before, $A$ can neither reveal the session key of the test
session $\pi^*$ nor of any partnered session, nor does any other session derive the same session key by Game $G_3$. Thus, $A$ cannot gain any information about the test bit $b_{\text{test}}$ and can do no better than to guess:

$$\text{Adv}^{G_{D,6}}_{PQXDH}(A) \leq 0.$$  

**Case E** ($\text{clean}_{EE}(\pi^*) \land \text{clean}_{peerE}(\pi^*) \land \text{type}(\pi^*) = \text{full}$).

In this proof case, the test session performs a full handshake and the $\text{clean}_{EE}$ predicate and its $\text{clean}_{peerE}$ sub-predicate ensure for the test session $\pi^*$ that

1. the test session’s owner’s randomness is not revealed, and
2. the randomness of the session (contributively) partnered to the test session is not revealed.

Similar to Case D, we get a *hybrid* guarantee here: both the ephemeral/ephemeral Diffie–Hellman combination being uncompromised and the ephemeral KEM key being uncompromised are, on their own, sufficient to ensure key indistinguishability for the test session.

**Game E.0.** This case begins with Game $G_3$ conditioned on $\text{clean}_{EE}(\pi^*) \land \text{clean}_{peerE}(\pi^*) \land \pi^*.\text{type} = \text{full}$ being satisfied.

$$\text{Adv}^{G_{E,0}}_{PQXDH}(A) = \text{Adv}^{G_3[\text{clean}_{EE}(\pi^*) \land \text{clean}_{peerE}(\pi^*) \land \pi^*.\text{type} = \text{full}]}_{PQXDH}(A).$$

**Game E.1 (Guess initiator and responder sessions).** We guess the initiator and responder sessions $\pi^*_i$ resp. $\pi^*_r$ involved in the test session $\pi^*$ (i.e., both the test session itself and its sid- resp. cid-partner), overwriting the adversary’s bit guess with 0 if this guess was incorrect. This step loses at most a factor of the number of sessions $n_s$ squared:

$$\text{Adv}^{G_{E,0}}_{PQXDH}(A) \leq n_s^2 \cdot \text{Adv}^{G_{E,1}}_{PQXDH}(A).$$

**Game E.2 (GapDH + OW-CCA).** We now abort the game (again, returning 0 as the adversary’s bit guess) in the event that the adversary queries the random oracle on a value formatted like a master session $\pi^*$, beginning with $\text{CDH}(\text{epk}^\text{DH}_{U^*}, \text{epk}^\text{DH}_{V^*})$, i.e., the shared DH key between the ephemeral keys $\text{epk}^\text{DH}_{U^*}$ and $\text{epk}^\text{DH}_{V^*}$ of the guessed initiator session $\pi^*_i$, resp. responder session $\pi^*_r$, and ending with the KEM shared secret $ss^*$ resulting from $\pi^*_i$ encapsulating against $\text{epk}^\text{KEM}_{V^*}$. The probability of such an abort can be bounded by the *minimum* of the advantages of the following reduction $B_7$ playing *simultaneously* against the $(t, \epsilon_{GDH}, q_{DDH})$-hardness of the GapDH problem in $(G, g, q)$ and $(t, \epsilon_{CCA}, n_s)$–OW-CCA security of KEM.

The reduction embeds the GapDH challenge as the ephemeral keys in the sessions $\pi^*_i$ and $\pi^*_r$, and the KEM public key $\text{pk}^\text{KEM}$ from the OW-CCA game as ephemeral key $\text{epk}^\text{KEM}$ in $\pi^*_r$. It uses the KEM challenge ciphertext $ct^*$ in $\pi^*_r$. Similar to Game $G_{D,5}$, the reduction does not need to answer corresponding $\text{REVEALRAND}$ queries on $\pi^*_i$ or $\pi^*_r$ due to $\text{clean}_{EE} \land \text{clean}_{peerE}$. Furthermore, as for Game $G_{D,5}$, patching the $\text{SEND}$ oracle is relatively easy: Except for queries to the test session or to $\pi^*_i$ with a modified initiator ephemeral key, the reduction can compute all DH shared secrets itself. For the latter query, $B_7$ uses a single call to the DDH oracle of GapDH resp. $\text{DECAPS}$ oracle of OW-CCA to simulate. In the test session, the reduction uses a freshly sampled key as session key. If the DDH oracle accepts a query, then the

---

14This can happen if the test session is a responder session and the adversary mauls the initiator public key on the wire.
reduction returns the $DH_3$ and $ss$ from the corresponding RO query to its GapDH and OW-CCA games after finishing the simulation for $A$.

The reduction ensures consistency with the random oracle by checking RO queries against the above mentioned patches. It queries the DDH oracle possibly once for $\pi^*_U$ and when checking RO queries for the $DH_4$ position, so overall a maximum of $q_{RO} + 1$ queries, which is well below the number in Game $G_{A,5}$. It further makes at most one Decaps query in the OW-CCA game.

Unless $A$ queries the random oracle on $CDH(epk_{B_4}^{DH}, epk_{V_4}^{DH})$ in the $DH_4$ position and $ss^*$ in the $ss$ position, $B_7$ provides a perfect simulation of $G_{E,1}$. If $A$ does make such a random oracle query, then $B_7$ will detect this and win both its GapDH game and its OW-CCA game. We can thus bound the game hop by the minimum advantage against the two games:

$$Adv_{PQXDH}^{G_{E,1}}(A) \leq \min(\epsilon_{GDH}, \epsilon_{CCA}) + Adv_{PQXDH}^{G_{E,2}}(A).$$

**Game E.3 (Replacing the session key).** We now replace the session key of the test session $\pi^*$ with a uniformly sampled key. Since Game $G_{E,2}$ has ruled out that the adversary queries the RO on the master secret $DH_4 || DH_2 || DH_3 || DH_4 || ss$ of the test session $\pi^*$, the adversary has no chance of detecting this change. Thus,

$$Adv_{PQXDH}^{G_{E,2}}(A) = Adv_{PQXDH}^{G_{E,3}}(A).$$

To conclude this proof case, observe that in Game $G_{E,3}$ the session key of the test session $\pi^*$ is now a uniformly random key, independent of $b_{test}$. As before, $A$ can neither reveal the session key of the test session $\pi^*$ nor of any partnered session, nor does any other session derive the same session key by Game $G_3$. Thus, $A$ cannot gain any information about the test bit $b_{test}$ and can do no better than to guess:

$$Adv_{PQXDH}^{G_{E,3}}(A) \leq 0.$$

**Case F (cleanEE($\pi^*$) \land clean_{sigE}($\pi^*$) \land type($\pi^*$) = full).**

In this proof case, the test session performs a full handshake and the cleanEE predicate and its clean_{sigE} sub-predicate ensure for the test session $\pi^*$ that

1. the test session’s owner’s randomness is not revealed, and
2. if $\pi^*$ is an initiator, the responder’s long-term (signing) key was uncompromised upon acceptance and a potential responder partner session’s randomness was not revealed.

Via the second point, we can guarantee that initiator and responder agree on the ephemeral KEM key, given signatures are unforgeable. The KEM encapsulation then ensures key indistinguishability for the test session.

**Game F.0.** This case begins with Game $G_3$ conditioned on cleanEE($\pi^*$) \land clean_{sigE}($\pi^*$) \land type($\pi^*$) = full being satisfied.

$$Adv_{PQXDH}^{G_{F,0}}(A) = Adv_{PQXDH}^{G_3[\text{cleanEE}(\pi^*) \land \text{clean}_{sigE}(\pi^*) \land \text{type}(\pi^*)=\text{full}]}(A).$$

**Game F.1 (Guess responder identity $V^*$).** We first guess the identity $V^*$ of the responder to the test session, overwriting the adversary’s bit guess with 0 if this guess was incorrect. This step loses at most a factor of the number of users $n_p$:

$$Adv_{PQXDH}^{G_{F,0}}(A) \leq n_p \cdot Adv_{PQXDH}^{G_{F,1}}(A).$$
Game F.2 (Signature unforgeability). We now abort the game (again, returning 0 as the adversary’s bit guess) in the event that the test session $\pi^*$ is an initiator session and accepts using an ephemeral KEM public key $\text{epk}_V^{\text{KEM}}$ that was not generated by $V^*$. This ensures that the test session accepts with an ephemeral KEM public key of which the adversary does not know the secret key, since the adversary cannot reveal $V^*$’s session randomness. Similar to Game $G_{A.2}$, the probability of such an abort can be bounded by the advantage of a reduction against the $(t, \epsilon_{\text{CCA}}, n_\text{ss} + n_s)$-unforgeability of $\text{SIG}$. The proof is identical to the one for Game $G_{A.2}$. Thus,

$$\text{Adv}_{\text{PQXDH}}^G(A) \leq \epsilon_{\text{SIG}} + \text{Adv}_{\text{PQXDH}}^G(A).$$

Game F.3 (Guess initiator and responder sessions). We guess the initiator and responder sessions $\pi^*$ resp. $\pi^*$ involved in the test session $\pi^*$ (i.e., both the test session itself and its partner), overwriting the adversary’s bit guess with 0 if this guess was incorrect. This step loses at most a factor of the number of sessions $n_s$ squared:

$$\text{Adv}_{\text{PQXDH}}^G(A) \leq n_s^2 \cdot \text{Adv}_{\text{PQXDH}}^G(A).$$

Game F.4 (OW-CCA). We now abort the game (again, returning 0 as the adversary’s bit guess) in the event that the adversary queries the random oracle on a value formatted like a master secret and ending with the KEM shared secret $ss^*$ resulting from $\pi^*$ encapsulating against $\text{epk}_V^{\text{KEM}}$. The probability of such an abort can be bounded by the advantage of the following reduction $B_8$ playing against $(t, \epsilon_{\text{CCA}}, n_s)$-OW-CCA security of KEM.

The reduction embeds the challenge $pk_{\text{KEM}}, ct^*$ from the OW-CCA game as ephemeral key $\text{epk}_V^{\text{KEM}}$ of $V^*$, and as KEM ciphertext in $\pi^*$ due to clean$_{\text{EE}}$. When $A$ halts, the reduction guesses $i \in [q_{\text{RO}}]$ and returns the $ss$ position of the $i$th RO query (assuming this query is formatted like a master secret) as the target shared secret.

Unless $A$ queries the random oracle on $ss^*$ in the $ss$ position, $B_8$ provides a perfect simulation of Game $G_{A.3}$. If $A$ does make such a random oracle query, then $B_8$ wins its OW-CCA game if it guesses the $i$th RO query correctly. We can thus bound the game hop by the advantage against OW-CCA times a loss of $q_{\text{RO}}$:

$$\text{Adv}_{\text{PQXDH}}^G(A) \leq q_{\text{RO}} \cdot \epsilon_{\text{CCA}} + \text{Adv}_{\text{PQXDH}}^G(A).$$

Game F.5 (Replacing the session key). We now replace the session key of the test session $\pi^*$ with a uniformly sampled key. Since Game $G_{F.4}$ has ruled out that the adversary queries the RO on the master secret $DH_1 || DH_2 || DH_3 || DH_4 || ss$ of the test session $\pi^*$, the adversary has no chance of detecting this change. Thus,

$$\text{Adv}_{\text{PQXDH}}^G(A) = \text{Adv}_{\text{PQXDH}}^G(A).$$

To conclude this proof case, observe that in Game $G_{F.5}$ the session key of the test session $\pi^*$ is now a uniformly random key, independent of $b_{\text{test}}$. As before, $A$ can neither reveal the session key of the test session $\pi^*$ nor of any partnered session, nor does any other session derive the same session key by Game $G_3$. Thus, $A$ cannot gain any information about the test bit $b_{\text{test}}$ and can do no better than to guess:

$$\text{Adv}_{\text{PQXDH}}^G(A) \leq 0.$$
6 Discussion and Conclusion

In this work, we provided a reductionist security analysis of Signal’s PQXDH in a “maximum-exposure”, game-based security model, augmented compared to prior versions [CCD⁺17, BFG⁺22, BFG⁺21] to capture the KEM component but also key signing, and gave a fully-parameterized, concrete security bound for PQXDH.

Our bound relies (among other things) on the LEAK⁺-BIND-SS-{CT,PK} binding property of the KEM. While we show that this property is satisfied by both Kyber and ML-KEM, the current and future KEMs in PQXDH may not satisfy this property; so achieving PQXDH-like security without relying on a binding property is of general interest. We can indeed forgo this assumption (and the corresponding advantage term), if in the key derivation of PQXDH, we include the KEM public key and ciphertext (or, ideally, the whole session context as is good practice). This supports and complements a proposal discussed in the tool-based formal verification of PQXDH by BJKS [BJK23, BJKS23], to make sure the key agreement in the initial handshake does not rely on the follow-up AEAD encryption (which would indeed violate key indistinguishability as we prove it). As a side note, including context into the key derivation also supports tighter security proofs [CCG⁺19].

Our reduction for PQXDH has an additional case (compared to the X3DH analysis [CCD⁺17]), where key indistinguishability hinges solely on the signed ephemeral KEM key. This guarantees security against an active adversary who later gets quantum powers. Hence, PQXDH protects against an even stronger class of attacks than the “harvest now, decrypt later” attack which motivated the design.

Like prior Signal analyses [CCD⁺17, BJKS23, BJK23], we model the long-term DH and signing keys as separate ones; we leave it to future work to more accurately reflect the implementation which re-uses the same keys for both purposes.

Acknowledgments

We thank Ehren Kret and Rolfe Schmidt for discussions on the PQXDH protocol, and Franziskus Kiefer for insights on the KEM re-encapsulation attack. R.F. was supported by the German Federal Ministry of Education and Research and the Hessian Ministry of Higher Education, Research, Science and the Arts within their joint support of the National Research Center for Applied Cybersecurity ATHENE.

References


