Bundled Authenticated Key Exchange:
A Concrete Treatment of (Post-Quantum) Signal's Handshake Protocol
(full version)

Keitaro Hashimoto [1], Shuichi Katsumata [1,2], Thom Wiggers [2]

[1] National Institute of Advanced Industrial Science and Technology (AIST)

[2] PQShield

January 10, 2025

Abstract

The Signal protocol relies on a special handshake protocol, formerly X3DH and now PQXDH, to set up secure conversations. Prior analyses of these protocols (or proposals for post-quantum alternatives) have all used highly tailored models to the individual protocols and generally made ad-hoc adaptations to "standard" AKE definitions, making the concrete security attained unclear and hard to compare between similar protocols. Indeed, we observe that some natural Signal handshake protocols cannot be handled by these tailored models. In this work, we introduce *Bundled Authenticated Key Exchange* (BAKE), a concrete treatment of the Signal handshake protocol. We formally model prekey *bundles* and states, enabling us to define various levels of security in a unified model. We analyse Signal's classically secure X3DH and *harvest-now-decrypt-later*-secure PQXDH, and show that they do not achieve what we call *optimal* security (as is documented). Next, we introduce RingXKEM, a fully post-quantum Signal handshake protocol achieving optimal security; as RingXKEM shares states among many prekey bundles, it could not have been captured by prior models. Lastly, we provide a security and efficiency comparison of X3DH, PQXDH, and RingXKEM.

Contents

1. Introduction

The Signal protocol [MP16; PM16] is likely the most successful end-to-end encrypted messaging protocol. It is not just used by the Signal app, but also in many other applications that are used by billions, including WhatsApp [Wha23] and Facebook Messenger [Met23]. To send a Signal message to Blake, Alex needs to first set up a Signal conversation with Blake. This initial setup is done using a Signal handshake protocol, after which the messages are encrypted using the Double Ratchet protocol [PM16]. The Signal handshake protocol was initially X3DH [MP16], based on Triple Diffie–Hellman [KP05]. In late 2023, Signal rolled out a post-quantum iteration of X3DH, called PQXDH, offering security against a *harvest-now-decrypt-later* adversary: a step towards a fully post-quantum Signal protocol.

X3DH and later PQXDH have been analyzed computationally and symbolically using models tailored to the protocols [Beg+24; Bha+24; Coh+17; Coh+20; FG24; KBB17]. Proposals for fully post-quantum alternatives also devised protocol-specific models for analysis [Bre+20; Bre+22; Col+24; DG22; Has+21; Has+22], generally making ad-hoc adaptations to "standard" AKE models. This issue stems from the so-called *prekey bundles* used by the Signal handshake protocol, allowing *multiple* senders to establish a key with a possibly offline recipient. To reuse previous AKE models, this was usually modeled by treating each prekey bundle or even its components independently. Because of this, it is not possible to model some natural Signal handshake protocols that use the batched nature of generating prekeys and share state across prekeys. Moreover, because the prekey bundles are treated slightly differently in each model, sometimes deviating from how they are used in practice, it makes the concrete security attained unclear and hard to compare.

1.1. Contributions

In this paper, we provide a concrete treatment of the Signal handshake protocol. We formally model prekey bundles and their states, enabling us to capture new Signal handshake protocols while establishing various levels of security within a unified framework. We showcase this by directly comparing both the security properties and performance of X3DH, PQXDH, and our new proposal RingXKEM. In the following, we explain this in more detail.

1.1.1. A New Model for Signal Handshake Protocols

We introduce *Bundled Authenticated Key Exchange* (BAKE). It uses a specific function to upload a list of prekey bundles, modeling Signal's handshake protocols more true-to-practice. This enables us to capture protocols that share states across prekey bundles and facilitates a more formal analysis of security in the face of state compromises.

A security model for BAKE
Based on our syntax, we define a game-based security model that treats key indistinguishability and authentication properties separately. For key indistinguishability, the adversary can reveal both the long-term identity secret keys and states associated to the prekey bundles. However, allowing it to reveal secrets without restrictions leads to an *unavoidable attack* on key indistinguishability. We thus exclude the minimum set of all such unavoidable attacks that any BAKE protocol is vulnerable against, and define the *optimal* confidentiality properties of a BAKE protocol. If a specific protocol has further (accepted) weaknesses, we can include them as additional unavoidable attacks. By comparing the unavoidable attacks for different protocols, we get an immediate means of comparing their achieved security properties.

Explicit treatment of authentication
During the development of PQXDH, Bhargavan et al. discovered that the protocol is vulnerable against so-called "KEM re-encapsulation attacks" if the encapsulation key is not bound to the key exchange [Bha+24]. This attack forces two users to establish the same key, unknown to the adversary, while disagreeing on the encapsulation key being used. This was previously considered an implicit attack on key indistinguishability, though not immediately clear why key indistinguishability should fail. Another subtle attack is the potential replaying of messages, which the documentation mentions as a possibility and defers the analysis to be beyond the scope of the document [MP16, Sec. 4.2]. While Signal implements a countermeasure, replays seemingly were not covered by prior game-based security models as they do not break key indistinguishability and are very specific to the treatment of the so-called

last-resort[1] prekey bundles. (The one exception is [KBB17], which covers this using symbolic analysis.) In our work, we treat authentication as a primary goal, making it possible to capture both attacks as explicit breaks of authentication.

Classic, harvest-now-decrypt-later, and quantum adversaries

We can fine-tune the attacker to capture not just classical and quantum adversaries to key indistinguishability and authentication, but also the intermediate "harvest-now-decrypt-later" (HNDL) adversary. We can adjust the powers of the adversary depending on the attack attempted by the adversary: while certain attacks are unavoidable if the adversary is quantum from the outset of the security game, they may become avoidable assuming the adversary is classical up to some point. To the best of our knowledge, this is the first work to formally model what it means for a general Signal handshake protocol to be HNDL secure. Indeed, such a fine-grained security model is essential to formally proving security of PQXDH. We note that while there are some works [Bha+24; FG24] showing (a slight variant of) HNDL security of PQXDH, the security model is highly tailored to PQXDH and is non-reusable for general protocols.

1.1.2. Analyzing X3DH and PQXDH as BAKE Protocols

We instantiate and analyze both X3DH and PQXDH as BAKE protocols, and formally prove that they meet the security level described in the documents. Both of these protocols have well-documented weaknesses and thus cannot fully meet our optimal security targets. Both are known to be vulnerable to an attack in which a sender can be impersonated to a receiver if the receiver's state is compromised. Additionally, because a component of the prekey bundles is not signed, sender sessions only have weak forward secrecy. Finally, because PQXDH only gives HNDL security, we cannot allow the HNDL adversary to obtain any post-quantum KEM prekeys. We are able to explicitly quantify these known weaknesses as additional unavoidable attacks (and thus demonstrate the weaker security guarantees), which gives a very clear comparison to other protocols. As described above, we also show that by including replay protection in the protocol and adding so-called confirmation tags, these protocols are able to avoid replay and re-encapsulation attacks on authentication.

Real-world relevance

During the development of this work we have been in continuous dialog with the Signal developers. Our findings have been confirmed by Signal and, in response, Signal are considering ways to better separate the Signal handshake from the Double Ratchet protocol including the user's view into the key derivation function.

1.1.3. A Post-Quantum Signal Handshake Protocol

In Section 5, we present a fully post-quantum Signal handshake protocol called RingXKEM. This protocol relies on post-quantum ring signatures for (deniable) post-quantum authentication and post-quantum KEM key exchange for post-quantum secrecy, and was inspired by prior proposals [Bre+22; Has+21; Has+22]. We optimize prekey bundle storage by authenticating them using a Merkle tree, the root of which is signed by the identity key. This way, the server needs to store only a single large post-quantum signature instead of one per prekey bundle. This reduces the cost of uploading prekey bundles and the deployment of post-quantum authentication at the central server. It is worth highlighting that as RingXKEM shares states across many prekey bundles, it could not have been captured in previous models. Lastly, RingXKEM achieves optimal security[2] in our BAKE model against fully quantum adversaries.

Instantiations and efficiency comparison

We compare X3DH, PQXDH, and RingXKEM when instantiated with cryptographic primitives. For X3DH and PQXDH, we base the numbers on the deployed protocols. For RingXKEM, we base our numbers on the recent Gandalf ring signature scheme [GJK24]. By extrapolating from the runtime performance of the primitives, we

---

[1]Following the Signal source code and the specification for PQXDH.

[2]There is a slight ambiguity on what "optimal" means due to the leeway in the definition of the predicate Origin used to define BAKE protocols. However, regardless of this, RingXKEM satisfies stronger properties compared to X3DH and PQXDH. See Section 3.3.2 for more detail.

also estimate the runtime cost on mobile phones. These results show that RingXKEM can be deployed at a cost comparable to PQXDH, especially when considering the cost of storage of prekey bundles.

Related Work

Several prior works have looked at the security of Signal's original, classically-secure X3DH protocol. Cohn-Gordon et al. [Coh+17; Coh+20] provided a tailored game-based security model, capturing both the X3DH and Double Ratchet protocols. Kobeissi, Bhargavan, and Blanchet [KBB17] modeled the composition in the symbolic (namely ProVerif) and computational model (CryptoVerif). PQXDH has been recently developed alongside formal analysis by Bhargavan et al. [Bha+24]. Fiedler and Günther [FG24], building on the model of [Coh+17; Coh+20] and [Bre+22], provided a tailored game-based security model for PQXDH and proved its security. Both [Bha+24] and [FG24] analyze the HNDL security of PQXDH by (implicitly or explicitly) restricting a post-quantum adversary from being able to break the classical signature scheme. In contrast, our model makes no assumption on the cryptographic primitives being used and abstractly defines HNDL security against *any* Signal handshake protocol, making the security model general and reusable.

A fully post-quantum X3DH based on the isogeny-based SIDH key exchange was proposed by Dobson and Galbraith [DG22], but SIDH famously was broken [CD23; Mai+23]. Proposals based on lattices were put forward by Brendel et al. [Bre+22] and Hashimoto et al. [Has+21; Has+22], both basing their designs on ring signatures. Our proposal RingXKEM extends the Hashimoto et al. proposal by explicitly defining prekey bundles and using Merkle trees for more efficient server-side storage. We also prove the security of our proposal as a BAKE instead of as a (slight variant of a) standard AKE, allowing us to formally state the security properties as will be used in practice, and allows us to make direct comparisons on the obtained security properties to X3DH and PQXDH considered as BAKE. This is not true for the AKE-style security analyzes in the papers cited above papers, as each is tailored to the proposal. Dobson and Galbraith tailor their model to X3DH's achieved security properties, actively forgoing capturing stronger security properties not attained by X3DH. Brendel et al. adapt the Cohn-Gordon et al. model; both models require carefully constructed but hard to understand "clean" predicates to rule out attacks that X3DH does not protect against. Finally, Hashimoto et al. only sketch how their proposals can be used with prekey bundles, strictly limiting their analysis to the AKE setting.

Beyond Signal, Apple deployed an update to iMessage with post-quantum key exchange in early 2024, called PQ3 [App24]. Like PQXDH, PQ3 does not achieve post-quantum authentication. Stebila analyzed PQ3 using a reductionist approach and Basin, Linker, and Sasse used Tamarin, both with tailored models and considering hybrid security [BLS24; Ste24]. Collins et al. [Col+24] proposed K-Waay using Split-KEMs, which were initially proposed for use in X3DH in an early paper by Brendel et al. [Bre+20]. K-Waay deviates from prior protocols as it requires a receiver to verify the handshake messages in batches for security, and adds receiver prekey bundles.

Organization

In the following, we start by introducing our notion of bundled AKE in Section 2 and our security model in Section 3. This is followed by a discussion of first X3DH and then PQXDH in Section 4. In Section 5, we propose RingXKEM, our proposal for a fully post-quantum Signal handshake protocol. Finally, in Section 6 we summarize the security properties of the protocols that we have discussed, and discuss the bandwidth, storage, and computational requirements.

2. Bundled Authenticated Key Exchange

In this section, we define the syntax for a (two-round) *bundled authenticated key exchange* (BAKE) protocol. This definition is tailored to the semantics and flow of Signal handshake protocols like X3DH. While we build on prior approaches (e.g., [Bre+22; Coh+17; Coh+20; Col+24; FG24; Has+21; Has+22]), our concrete modeling of the uploading of prekey bundles and the users' state, allow a more formal modeling of forward secrecy and state reuse.

2.1. Syntax of Bundled AKE

We give our syntax for BAKE protocols in Definition 1. Signal protocols pre-generate and publish a number of so-called *prekey bundles* to the central server, which can be viewed as the first message in standard AKE. We model this through the BAKE.PreKeyBundleGen function, which is the most significant difference to prior models; prior work typically treated prekey bundles individually. This function explicitly returns a single state that contains all (private) information for the prekey bundles. We use this to model attacks on the ephemeral keys stored by the users. In the second round of the key agreement, the person that wants to start a conversation, whom we refer to as *sender*, downloads a prekey bundle and uses it to complete the cryptographic handshake and obtain a shared secret to encrypt their message with. This is modeled by the BAKE.Send function. Finally, the *receiver* (whose previously uploaded prekey bundle was used by the sender) takes this generated message and its current state to complete the handshake in BAKE.Receive.

Definition 1. A *two-round bundled authenticated key exchange* protocol BAKE consists of the following four PPT algorithms, where $L \in \mathsf{poly}(\lambda)$.

BAKE.IdKeyGen(1^λ) $\overset{\$}{\to}$ (ik, isk)**:** The identity key generation algorithm takes as input the security parameter 1^λ and outputs an identity public key ik and an associated secret key isk.

BAKE.PreKeyBundleGen(isk_u) $\overset{\$}{\to}$ ($\vec{\mathsf{prek}}_u$, st_u)**:** The prekey bundle generation algorithm takes a user u's identity secret key as input and outputs a number of prekey bundles $\vec{\mathsf{prek}}_u = (\mathsf{prek}_{u,t})_{t \in [L] \cup \{\perp\}}$, and a user state st_u. Prekey bundles with $t \neq \perp$ are called *one-time* prekey bundles and the special prekey bundle $\mathsf{prek}_{u,\perp}$ is called the *last-resort* prekey bundle (cf. Section 2.1.2). The state may for example include the associated (ephemeral) secret keys to public keys included in $\vec{\mathsf{prek}}_u$.

BAKE.Send(isk_s, ik_r, $\mathsf{prek}_{r,t}$) $\overset{\$}{\to}$ (K, ρ)**:** The sender algorithm takes as input a sender s's identity secret key isk_s and the intended receiver r's identity key ik_r and a particular prekey bundle $\mathsf{prek}_{r,t}$, and outputs a session key K and a handshake message ρ.

BAKE.Receive(isk_r, st_r, ik_s, t, ρ) $\to$ (K', st_r)**:** The (deterministic) receiver algorithm takes as input a receiver r's identity secret key isk_r and state st_r, a sender's identity key ik_s, along with the identifier of the used prekey bundle $t \in [L] \cup \{\perp\}$, and the initial message ρ. It then outputs a key K' and a possibly updated state st_r. Key agreement may fail, in which case $K' = \perp$ is returned, and the state is rolled back to before running the algorithm.

2.1.1. A Single State for Prekey Bundles

A BAKE protocol uses a single state for all prekey bundles uploaded by a single BAKE.PreKeyBundleGen call. We use this state in Section 3.3 to model forward secrecy properties related to state compromises that leak the private keys for prekey bundles that have not been used and deleted. The singular shared state is one of the functionalities missing in prior formalization. Looking ahead, our fully post-quantum Signal handshake protocol RingXKEM could not been captured by prior work as prekey bundles were treated independently.

Running the BAKE.PreKeyBundleGen algorithm will refresh all prekey bundles and the state. Signal clients call this function frequently, both to ensure enough prekey bundles are available at the server, and to rotate last-resort prekey bundles, which we will describe in the next paragraph. In our security model described in Section 3.3 we use *epochs* to track the expiration of secret key material obtained from the state.

2.1.2. Availability Versus Ephemeral Keys

If each prekey bundle would be single use, the number of prekey bundles uploaded would pose a limit on the number of Signal handshakes that can be completed. Thus, to ensure availability of the recipient even if they are offline for extended amounts of time, so-called *last-resort prekey bundles* are used if the list of *one-time* prekey bundles is depleted. The last-resort prekey bundle is a specially designated prekey bundle and, when used, is not deleted from the list of available prekeys at the server, and its associated secrets are not deleted from the receiver's

state. Because of this, any exchanges that use the last-resort prekey bundle are vulnerable to state compromises even after the handshake completes, until the next call of BAKE.PreKeyBundleGen, which replaces the last-resort prekey bundle and the receiver's state.

For bookkeeping in our models, we will designate a specific label $\perp$ to refer to a last-resort prekey bundle. In protocol execution, the server will distribute first all one-time prekey bundles until they are exhausted, after which the last-resort prekey bundle $\mathsf{prek}_{u,\perp}$ will be used.

3. Correctness and Security of Bundled AKE

We define the correctness and security of a BAKE protocol borrowing the formalism from recent (standard) AKE protocol designs [Bre+22; Col+24; Has+22; Jag+21]. The unique feature of our formalism comes from handling the state of the prekey bundles, especially the last-resort prekey bundle that can be reused multiple times.

3.1. Execution Environment

The correctness and security of a BAKE protocol is defined by an interactive game between an adversary and a challenger, formally illustrated in Algorithms 1 and 2. The challenger plays the role of the users and the adversary can arbitrarily interact with the users and execute algorithms BAKE.PreKeyBundleGen, BAKE.Send, and BAKE.Receive through oracle queries. As in standard AKE definitions, we rely on a so-called *instance identifier* (iID) to track all the information maintained by the game. In *bundled* AKE, we must extend prior definitions of instance identifiers to capture (last-resort) prekey bundles. Due to its complexity, we first provide an overview of the information maintained by the game below.

We consider a system of N users, where each user is represented by an identity $u \in \mathcal{U}$. Each user has an identity key pair $(\mathsf{ik}_u, \mathsf{isk}_u)$ and will periodically publish its prekey bundles.[3] As explained in Section 2.1.1, each prekey bundle is assigned a value called epoch. The initial prekey bundle generated by user u has epoch = 1, and every time u generates a new set of prekey bundles, epoch is incremented by one.

The adversary can instruct the users to perform the following three tasks: (i) ask a receiver to create new prekey bundles $\vec{\mathsf{prek}}_r = (\mathsf{prek}_{r,t})_{t \in [L] \cup \{\perp\}}$ (via $O_{\mathsf{PubNewPrekeyBundle}}$); (ii) ask a sender to send a handshake message ρ (via O_{Send}); and (iii) ask a receiver to process a handshake message (via O_{Receive}). Task (i) generates $L + 1$ new instances for the receiver and task (ii) generates a single new instance for the sender. The game records the creation of new instances by using an instance identifier $\mathsf{iID} = (\overline{\mathsf{iID}}, \mathsf{ctr}) \in \mathbb{N} \times (\{0, \perp\} \cup \mathbb{N})$. The *base* instance identifier $\overline{\mathsf{iID}}$ is a unique integer assigned to each instance, created when tasks (i) and (ii) are performed. We use $\mathsf{base}(\mathsf{iID})$ to extract $\overline{\mathsf{iID}}$ from iID. We also may simply refer to iID as an instance. The counter ctr is used to distinguish between a receiver instance using the last-resort prekey bundle from other instances. Concretely, when task (i) is performed, the game creates $L + 1$ instances: L instances of the type $\mathsf{iID}_t := (\overline{\mathsf{iID}}_t, 0)$ for $t \in [L]$ (associated to the one-time prekey bundles) and one instance of type $\mathsf{iID}_\perp := (\overline{\mathsf{iID}}_\perp, \perp)$ (associated to the last-resort prekey bundle). When task (ii) is performed, the game creates one sender instance with $\mathsf{iID} := (\overline{\mathsf{iID}}, 0)$. The reader can think of instances with ctr = 0 as a normal AKE instance.

What is unique to a BAKE protocol is that receivers can reuse the last-resort prekey bundle, i.e., instances with ctr = $\perp$. More precisely, many senders can use the same prekey bundle associated to the instance $\mathsf{iID}_\perp$ to send a handshake message to the receiver. To this end, we use ctr $\in \mathbb{N}$ to model the fact that multiple instances can be associated to $\mathsf{iID}_\perp$ when task (iii) is performed on $\mathsf{iID}_\perp$. When task (iii) is performed on $\mathsf{iID}_\perp$ for the ith ($i \in \mathbb{N}$) time, the game creates a new instance $\mathsf{iID}_{\perp,i} := (\overline{\mathsf{iID}}_\perp, i)$, where $\mathsf{base}(\mathsf{iID}_\perp) = \mathsf{base}(\mathsf{iID}_{\perp,i})$. Importantly, unlike receiver instances of the type $\mathsf{iID} = (\overline{\mathsf{iID}}, 0)$ that can be completed, $\mathsf{iID}_\perp$ will always remain an incomplete instance. Namely, the game will never assign a session key to the instance $\mathsf{iID}_\perp$ as the session key will be assigned to a newly created instance $\mathsf{iID}_{\perp,i}$ with the same base instance identifier (see Algorithm 2 for more details).

Capturing last-resort prekey bundles separately from one-time prekey bundles allows for a fine-grained notion of security where we can model session key compromise of, say $\mathsf{iID}_{\perp,i}$, while still arguing session key secrecy of

---

[3]Technically speaking, the adversary will instruct the user to generate a new set of prekey bundles via the oracle $O_{\mathsf{PubNewPrekeyBundle}}$, but we ignore this detail for better readability. See Algorithm 1 for more detail.

$\text{iID}_{\perp,j}$ for $j \neq i$. Moreover, letting the instance identifiers $\text{iID}_{\perp,i}$ and $\text{iID}_{\perp,j}$ share the same base instance identifier $\overline{\text{iID}}_\perp$ allows to succinctly define security as we show in Section 3.3.3.

The game uses these instances iID to record all the information handled by the instance associated with iID. Looking ahead, the game keeps track of multiple lists, initialized to a special empty symbol ϵ, and updated when the game oracles are called by the adversary. They are defined as follows:

role[iID] $\in \{$ sender, receiver $\}$ records the instance's role, i.e., whether the instance acts as the sender or the receiver.

(Sender[iID], Receiver[iID]) $\in (\mathcal{U} \cup \{\perp\}) \times \mathcal{U}$ records the identities of the sender and the receiver relative to the instance iID. Sender[iID] $= \perp$ captures the fact that the sender is undefined when the receiver creates the prekey bundles.

prek[iID] records the prekey used by the instance iID.

prekidx[iID] records the index of the prekey used by the *receiver* instance iID (i.e., role[iID] = receiver) in the corresponding prekey bundle. When role[iID] = sender, the sender is not assumed to know the index of the receiver's prekey bundle, i.e., prekidx[iID] $= \epsilon$.

epoch[iID] records the epoch in which the prekey bundle used by the *receiver* instance iID was published. Similarly to prekidx, we do not assume the sender to know this, i.e., epoch[iID] $= \epsilon$ when role[iID] = sender.

prekreuse[iID] records the number of time a last-resort prekey of a receiver instance has being reused. Specifically, we have prekreuse[iID] $\neq \epsilon$ only for iID $\in \mathbb{N} \times \{\perp\}$.

ρ[iID] records the handshake message used by the instance iID.

key[iID] records the session key computed by the instance iID. This is set to $\perp$ if iID does not accept the protocol execution. As explained above, we have key[iID] $= \epsilon$ for iID $\in \mathbb{N} \times \{\perp\}$.

For more detail, we refer the readers to Algorithms 1 and 2. While the game records more information associated to iID, we postpone their explanations to Section 3.4 as they only relate to security. In the next subsection, we define correctness.

3.2. Correctness of BAKE

Correctness requires that when all the users in the system honestly execute the BAKE protocol without the adversary tampering the protocol messages, then they derive an identical session key except with all but a negligible probability. Formally, we model this through a game between a *passive* adversary $\mathcal{P}$ and a challenger. Here, a passive adversary $\mathcal{P}$ can arbitrary interact with the users under the restriction that it must honestly deliver the protocol messages. For instance, if a sender s outputs a handshake message ρ to sender r, then $\mathcal{P}$ can not invoke receiver r on anything other than ρ.

Definition 2 (Correctness). We define the correctness game in Algorithm 1 and define the advantage of a *passive* adversary $\mathcal{P}$ as

$$\text{Adv}_{\text{BAKE},\mathcal{P}}^{\text{CORR}}(1^\lambda) := \Pr\left[\text{Game}_{\text{BAKE},\mathcal{P}}^{\text{CORR}}(1^\lambda) = 1\right].$$

We say a BAKE protocol is *correct* if $\text{Adv}_{\text{BAKE},\mathcal{P}}^{\text{CORR}}(1^\lambda) = \text{negl}(\lambda)$ for any efficient passive adversary $\mathcal{P}$.

3.3. Security of BAKE: Key Indistinguishability

We model the security of a BAKE protocol via a *key indistinguishability* game. Informally, we want to argue that a particular session key key[iID] established by an instance iID looks random to the adversary. However, as with any standard AKE protocol, to formally argue this, we must establish a set of *unavoidable attacks*[4] through a predicate

---

[4]This is often termed *trivial* attacks in the literature. We chose the term unavoidable as the triviality of an attack is in many cases subjective. Indeed, as we see later, some attacks are quite contrived yet unavoidably necessary to rule out for some protocols.

called safe and declare the adversary to be successful only if the predicate safe holds true at the end of the game. The set of unavoidable attacks is to some degree protocol dependent, and as such, an appropriate predicate safe must be defined for each protocol.

Below, we provide the definition of key indistinguishability assuming the existence of such a predicate safe and defer the definition of the predicate safe to Sections 3.4 and 3.5.

3.3.1. Key Indistinguishability

We define key indistinguishability of a BAKE protocol as follows, assuming a predicate safe defined in Sections 3.4 and 3.5.

Definition 3 (Key Indistinguishability). We define the key indistinguishability security in Algorithm 1 (with respect to a predicate safe) and define the advantage of an adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ as

$$\mathsf{Adv}^{\mathsf{KIND}}_{\mathsf{BAKE}, \mathcal{A}}(\lambda) \coloneqq \left| \Pr\left[\mathsf{Game}^{\mathsf{KIND}}_{\mathsf{BAKE}, \mathcal{A}}(1^\lambda) = 1 \right] - \frac{1}{2} \right|.$$

A BAKE protocol is *key indistinguishable* if $\mathsf{Adv}^{\mathsf{KIND}}_{\mathsf{BAKE}, \mathcal{A}}(1^\lambda) = \mathsf{negl}(\lambda)$ for any efficient $\mathcal{A}$.

As a special case, if $\mathcal{A}_1$ is classical, but $\mathcal{A}_2$ is quantum, then we say it is key indistinguishable against *harvest-now-decrypt-later* adversaries.

In a *harvest now, decrypt later (HNDL) attack* a classical adversary records the communication in the present time and then retroactively tries to attack the protocol when quantum computers are available. Namely, the quantum adversary tries to retroactively break the security after the communication has terminated. This is considered to be one of the largest threats of quantum computers to currently-deployed systems and a major driver for government and industry bodies to transition towards post-quantum cryptography. Indeed, the motivation for Signal to update X3DH to PQXDH was to exactly secure against HNDL attacks; note the authentication, which must happen in present time, is still only classically secure.

Our formalization of a two-stage adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ allows to explicitly distinguish the security property of X3DH and PQXDH. We allow $\mathcal{A}_1$ to interact with the users through oracle queries but $\mathcal{A}_2$ is only given $\mathcal{A}_1$'s state. We can model classical, HNDL, and quantum adversaries by setting $(\mathcal{A}_1, \mathcal{A}_2)$ to be (classical, classical), (classical, quantum), and (quantum, quantum), respectively. Note that if $\mathcal{A}_2$ is quantum, this might result in (additional) unavoidable attacks.

3.3.2. Origin Instances and Partners

Before defining the predicate safe, we define the predicates Origin (cf. Algorithm 2, Line 4) and Partner (cf. Algorithm 2, Line 6) that is used internally by the predicate safe (see Sections 3.4 and 3.5). These are fundamental predicates used by any standard AKE protocol to define the set of "unavoidable" attacks. Recall that for key indistinguishability, we must argue that a session key derived by some instance is indistinguishable from random. Then clearly, we must *at least* restrict the adversary from obtaining the session key derived by an "associating" peer instance.

Below, we rely on the concept of *origin instances* and *partners*, defined through the predicates Origin and Partner, to formalize the adversarial capabilities. These concepts are also used in standard AKE literature, but looking ahead, we must appropriately extend prior definitions to capture last-resort prekey bundles that are unique to BAKE protocols.

Origin instances. Consider the following example: The adversary invokes a receiver r to create prekey bundles $\vec{\mathsf{prek}}_r$ and invokes a sender s to create a handshake message with respect to one of the prekey bundles in $\vec{\mathsf{prek}}_r$. Accordingly, the game creates two instances iID' and iID, one for the receiver r and the other for the sender s. We also have $\mathsf{key}[\mathsf{iID}'] = \epsilon$ but $\mathsf{key}[\mathsf{iID}] \neq \perp$ as the receiver has not processed the handshake message while the sender has derived a session key. Now, assume the adversary declares iID as the test instance. While $\mathsf{key}[\mathsf{iID}'] \neq \mathsf{key}[\mathsf{iID}]$, it is clear that we cannot allow the adversary to obtain both the receiver's identity public key ik_r and the receiver

Algorithm 1 Games for correctness, key indistinguishability, and match soundness. Below, $\mathcal{U}$ denotes the set of users in the system, $\mathcal{P}$ denotes a passive adversary, $O^\star$ denotes the set $\{\,O_{\mathsf{PubNewPrekeyBundle}}, O_{\mathsf{Send}}, O_{\mathsf{Receive}}\,\}$, and O denotes the set of all oracles defined in Algorithm 2. Additionally, $\mathsf{mode} \in \{\,\mathsf{KIND}, \mathsf{MATCH}\,\}$.

```
 1: function Game_BAKE,P^CORR(1^λ)            14: function Game_BAKE,A^mode(1^λ)
 2:     S_ilD := ∅ ▷ Admin variable for O^*:   15:     b ←$ {0,1}
        Set of existing instances.            16:     S_ilD := ∅ ▷ Admin variable: Set of existing instances.
 3:     NumilD := 0 ▷ Admin variable: Number  17:     NumilD := 0 ▷ Admin variable: Number of instances.
        of instances.                         18:     ilD* := ⊥ ▷ Tested instance.
 4:     for user u ∈ U do                      19:     for user u ∈ U do
 5:         ▷ Initialize epoch and counter. ◁ 20:         ▷ Initialize epoch and counter. ◁
 6:         (epoch_u, ctr_u) := (0,0)          21:         (epoch_u, ctr_u) := (0,0)
 7:         (ik_u, isk_u) ←$ BAKE.IdKeyGen(1^λ) 22:         (ik_u, isk_u) ←$ BAKE.IdKeyGen(1^λ)
 8:     1 ← P^{O^*}((ik_u)_{u∈U}) ▷ P always  23:     st ←$ A_1^O((ik_u)_{u∈U})
        terminates with 1                     24:     b' ←$ A_2(st)
 9:     for (ilD, ilD') ∈ S_ilD × S_ilD do    25:     if ⟦mode = KIND⟧ then ▷ Key ind. game.
        cond := ⟦role[ilD] ≠ role[ilD']⟧       26:         if ⟦ilD* = ⊥⟧ ∨ ⟦safe(ilD*) = false⟧ then
            ∧ ⟦sender[ilD] = sender[ilD']⟧     27:             b' ←$ {0,1}
10:         ∧ ⟦receiver[ilD] = receiver[ilD']⟧ 28:         return ⟦b = b'⟧
            ∧ ⟦prekidx[ilD] = prekidx[ilD']⟧   29:     else ▷ mode = MATCH, Match soundness game
            ∧ ⟦ρ[ilD] = ρ[ilD']⟧ ∧ ⟦key[ilD] ≠ key[ilD']⟧ 30:         return ⟦Match(S_ilD) = false⟧
11:     if cond then
12:         return 1
13:     return 0
```

state st_r. If we allow such an attack, the adversary can simply run BAKE.Receive by himself and derive the same session key as the sender.

To disallow such an unavoidable attack, we will say that the receiver instance ilD' is an *origin instance* to the sender instance ilD, and disqualify the adversary from performing certain types of attacks on the origin instance. A common way is to define the origin instance of a sender instance ilD to be the receiver instance ilD' such that $\mathsf{prek}[\mathsf{ilD}] = \mathsf{prek}[\mathsf{ilD}']$; in the AKE literature, this corresponds to setting the origin instance identifier as the first message of a two-round AKE protocol [CF12; CF15; Jag+21; PRZ24; PWZ23]. More generally, we use an *origin function* Φ_{origin} and say that ilD' is the origin instance of ilD if $\Phi_{\mathsf{origin}}(\mathsf{ilD}) = \Phi_{\mathsf{origin}}(\mathsf{ilD}')$. In some cases where $\mathsf{prek}[\mathsf{ilD}]$ contains malleable components unnecessary for the secrecy of the session key (e.g., $\mathsf{prek}[\mathsf{ilD}]$ contains a non-strongly unforgeable signature), this general definition captures security more appropriately (see Li and Schäge [LS17] for more detail).

Formally, we define origin instances as follows.

Definition 4 (Origin Instance)**.** Let Φ_{origin} be an efficiently computable function called an *origin function*. An instance $\mathsf{ilD}' \in \mathbb{N} \times (\{\,0, \bot\,\} \cup \mathbb{N})$ is an *origin instance* of $\mathsf{ilD} \in \mathbb{N} \times (\{\,0, \bot\,\} \cup \mathbb{N})$ if the predicate $\mathsf{Origin}(\mathsf{ilD}, \mathsf{ilD}')$ defined below holds true:

$$\llbracket \mathsf{Receiver}[\mathsf{ilD}] = \mathsf{Receiver}[\mathsf{ilD}']\rrbracket$$
$$\wedge \; \llbracket (\mathsf{role}[\mathsf{ilD}], \mathsf{role}[\mathsf{ilD}']) = (\mathsf{sender}, \mathsf{receiver})\rrbracket$$
$$\wedge \; \llbracket \Phi_{\mathsf{origin}}(\mathsf{ilD}) = \Phi_{\mathsf{origin}}(\mathsf{ilD}')\rrbracket.$$

We define the predicate Origin in an asymmetric manner. A receiver does not need an origin instance as it either has no information of the sender to begin with or can use the predicate $\mathsf{Partner}$, defined next, to specify the peer instance.

Partners. The notion of partners concerns two instances that have agreed on the communicating user and a same session key. Clearly, if the adversary challenges one of the instances, then we must disallow the adversary from revealing the session key from the other partnered instance. Formally, we define partners as follows.

Definition 5 (Partner)**.** Two instances $\mathsf{ilD}, \mathsf{ilD}' \in \mathbb{N} \times (\{\,0, \bot\,\} \cup \mathbb{N})$ are *partners* if the predicate $\mathsf{Partner}(\mathsf{ilD}, \mathsf{ilD}')$

Algorithm 2 Oracles used by the correctness, key indistinguishability, and match soundness games. We assume all oracles to only take users in the system as input, i.e., $u, s, r \in \mathcal{U}$.

1: **function** $O_{\mathsf{PubNewPrekeyBundle}}(u)$
2:    $\mathsf{epoch}_u \leftarrow \mathsf{epoch}_u + 1 \; \triangleright$ *Move to next epoch*
3:    $\mathsf{ctr}_u \leftarrow 0 \; \triangleright$ *Reset counter*
4:    $(\vec{\mathsf{prek}}_u, \mathsf{st}_u) \xleftarrow{\$} \mathsf{BAKE.PreKeyBundleGen}(\mathsf{isk}_u)$
5:    $\triangleright$ *Assign instances to prekeys* $\triangleleft$
6:    **for** $t \in [L] \cup \{\bot\}$ **do**
7:      $\triangleright$ *Create new base instance* $\triangleleft$
8:      $\mathsf{NumiID} \leftarrow \mathsf{NumiID} + 1$
9:      **if** $[\![t \neq \bot]\!]$ **then** $\triangleright$ *One-time prekey bundle*
10:       $\mathsf{iID} := (\mathsf{NumiID}, 0)$
11:      **else** $\triangleright$ *Last-resort prekey bundle*
12:       $\mathsf{iID} := (\mathsf{NumiID}, \bot)$
13:       $\mathsf{prekreuse}[\mathsf{iID}] \leftarrow 0 \; \triangleright$ *Record number of reuses*
14:      $\mathcal{S}_{\mathsf{iID}} \leftarrow \mathcal{S}_{\mathsf{iID}} \cup \{\mathsf{iID}\}$
15:      $(\mathsf{role}[\mathsf{iID}], \mathsf{Sender}[\mathsf{iID}], \mathsf{Receiver}[\mathsf{iID}]) \leftarrow$
       $(\mathsf{receiver}, \bot, u)$
16:      $(\mathsf{prek}[\mathsf{iID}], \mathsf{prekidx}[\mathsf{iID}]) \leftarrow (\vec{\mathsf{prek}}_u[t], t)$
17:      $\mathsf{epoch}[\mathsf{iID}] \leftarrow \mathsf{epoch}_u$
18:    **return** $\vec{\mathsf{prek}}_u$

19: **function** $O_{\mathsf{Send}}(s, r, \mathsf{prek})$
20:    $\mathsf{NumiID} \leftarrow \mathsf{NumiID} + 1 \; \triangleright$ *Create new base instance*
21:    $\mathsf{iID} := (\mathsf{NumiID}, 0)$
22:    $\mathcal{S}_{\mathsf{iID}} \leftarrow \mathcal{S}_{\mathsf{iID}} \cup \{\mathsf{iID}\}$
23:    $(K, \rho) \xleftarrow{\$} \mathsf{BAKE.Send}(\mathsf{isk}_s, \mathsf{ik}_r, \mathsf{prek})$
24:    $(\mathsf{role}[\mathsf{iID}], \mathsf{Sender}[\mathsf{iID}], \mathsf{Receiver}[\mathsf{iID}]) \leftarrow$
     (sender, s, r)
25:    $(\mathsf{prek}[\mathsf{iID}], \rho[\mathsf{iID}]) \leftarrow (\mathsf{prek}, \rho)$
26:    $\mathsf{key}[\mathsf{iID}] \leftarrow K$
27:    $\mathsf{PeerCorr}[\mathsf{iID}] \leftarrow \mathsf{RevIK}[r] \; \triangleright$ *Check if peer's* isk *is corrupted*
28:    **return** ρ

29: **function** $O_{\mathsf{Receive}}(r, s, \rho)$
30:    $\mathsf{ctr}_r \leftarrow \mathsf{ctr}_r + 1$
31:    **if** $[\![\mathsf{ctr}_r \leq L]\!]$ **then** $\triangleright$ *One-time prekey exists*
32:      $t := \mathsf{ctr}_r$
33:      Fetch iID s.t. $(\mathsf{epoch}[\mathsf{iID}], \mathsf{prekidx}[\mathsf{iID}]) =$
     $(\mathsf{epoch}_u, t) \triangleright$ *Unique*
34:    **else** $\triangleright$ *One-time prekey depleted*
35:      $t := \bot$
36:      Fetch $\mathsf{iID}_\bot$ s.t. $(\mathsf{epoch}[\mathsf{iID}_\bot], \mathsf{prekidx}[\mathsf{iID}_\bot]) =$
     $(\mathsf{epoch}_u, t) \triangleright$ *Unique*

37:    $(K', \mathsf{st}_r) \leftarrow \mathsf{BAKE.Receive}(\mathsf{isk}_r, \mathsf{st}_r, \mathsf{ik}_s, t, \rho)$
38:    **if** $[\![t = \bot]\!]$ **then**
39:      $\triangleright$ *Create new completed last-resort instance* $\triangleleft$
40:      $\mathsf{prekreuse}[\mathsf{iID}] \leftarrow \mathsf{prekreuse}[\mathsf{iID}] + 1$
     $\triangleright \mathsf{prekreuse}[\mathsf{iID}] = \mathsf{ctr}_r - L$
41:      $\mathsf{iID} := (\mathsf{base}(\mathsf{iID}_\bot), \mathsf{prekreuse}[\mathsf{iID}])$
42:      $\mathcal{S}_{\mathsf{iID}} \leftarrow \mathcal{S}_{\mathsf{iID}} \cup \{\mathsf{iID}\}$
43:      $\triangleright$ *Copy information into new* iID $\triangleleft$
44:      $\mathsf{role}[\mathsf{iID}] \leftarrow \mathsf{role}[\mathsf{iID}_\bot]$
45:      $\mathsf{Receiver}[\mathsf{iID}] \leftarrow \mathsf{Receiver}[\mathsf{iID}_\bot]$
46:      $\mathsf{prek}[\mathsf{iID}] \leftarrow \mathsf{prek}[\mathsf{iID}_\bot]$
47:      $\mathsf{prekidx}[\mathsf{iID}] \leftarrow \mathsf{prekidx}[\mathsf{iID}_\bot]$
48:      $\mathsf{epoch}[\mathsf{iID}] \leftarrow \mathsf{epoch}[\mathsf{iID}_\bot]$
49:    $\triangleright$ *Record completed instance* $\triangleleft$
50:    $(\mathsf{Sender}[\mathsf{iID}], \rho[\mathsf{iID}], \mathsf{key}[\mathsf{iID}]) \leftarrow (s, \rho, K')$
51:    $\triangleright$ *Check if peer's* isk *is corrupted* $\triangleleft$
52:    $\mathsf{PeerCorr}[\mathsf{iID}] \leftarrow \mathsf{RevIK}[s]$
53:    $\triangleright$ *Check if own* st *is corrupted* $\triangleleft$
54:    $\mathsf{StateRev}[\mathsf{iID}] \leftarrow \mathsf{RevUserSt}[r]$
55:    $\triangleright$ *Inform success of* $\mathsf{BAKE.Receive}$ $\triangleleft$
56:    **return** $[\![K' \neq \bot]\!]$

57: **function** $O_{\mathsf{RevSessKey}}(\mathsf{iID})$
58:    **require** $[\![\mathsf{base}(\mathsf{iID}) \leq \mathsf{NumiID}]\!] \triangleright$ *Existing instance*
59:    $\mathsf{RevSessKey}[\mathsf{iID}] \leftarrow \mathsf{true}$
60:    **return** $\mathsf{key}[\mathsf{iID}]$

61: **function** $O_{\mathsf{RevIK}}(u)$
62:    $\mathsf{RevIK}[u] \leftarrow \mathsf{true}$
63:    **return** isk_u

64: **function** $O_{\mathsf{RevState}}(u)$
65:    $\triangleright$ st_u *for* epoch_u *can't have been corrupted before* $\triangleleft$
66:    **require** $[\![\mathsf{UserStCtr}[u, \mathsf{epoch}_u] = \epsilon]\!]$
67:    $\mathsf{RevUserSt}[u, \mathsf{epoch}_u] \leftarrow \mathsf{true}$
68:    $\mathsf{UserStCtr}[u, \mathsf{epoch}_u] \leftarrow \mathsf{ctr}_u$
69:    **return** st_u

70: **function** $O_{\mathsf{Test}}(\mathsf{iID})$
71:    **require** $[\![\mathsf{base}(\mathsf{iID}) \leq \mathsf{NumiID}]\!] \triangleright$ *Existing instance*
72:    **require** $[\![\mathsf{iID}^* = \bot]\!] \wedge [\![\mathsf{key}[\mathsf{iID}] \neq \bot]\!]$
73:    $\mathsf{iID}^* \leftarrow \mathsf{iID}$
74:    $K_0 := \mathsf{key}[\mathsf{iID}]; K_1 \xleftarrow{\$} \mathcal{K}$
75:    **return** K_b

11

defined below holds true:

$$[\![\mathsf{Sender}[\mathsf{iID}] = \mathsf{Sender}[\mathsf{iID}']]\!]$$
$$\wedge\ [\![\mathsf{Receiver}[\mathsf{iID}] = \mathsf{Receiver}[\mathsf{iID}']]\!]$$
$$\wedge\ [\![\mathsf{role}[\mathsf{iID}] \neq \mathsf{role}[\mathsf{iID}']]\!] \wedge [\![\mathsf{key}[\mathsf{iID}] = \mathsf{key}[\mathsf{iID}']]\!].$$

We note that the partnering definition captures unknown key share attacks [BM99]. The two instances will not be partnered as they disagree on the view of the peer, even if they derive the same key. Also, it is worth noting that an instance of the form $\mathsf{iID}_\perp = (\overline{\mathsf{iID}}, \perp)$ (i.e., a receiver instance associated with a last-resort prekey bundle) cannot be partnered with any other instance as $\mathsf{key}[\mathsf{iID}_\perp] = \epsilon$ by definition. Here, we implicitly use the fact that for a two-round protocol, a sender instance iID will always satisfy $\mathsf{key}[\mathsf{iID}] \neq \epsilon$.

Similarly to origin instances, we can define partners via a general function $\Phi_{\mathsf{part}}(\mathsf{iID})$ as opposed to using $\mathsf{key}[\mathsf{iID}]$. However, for the notion of partners, it has been shown that they are essentially identical for natural schemes [Brz+24], and as such, we opt to use the simpler definition. This comes with the benefit of the partner definition being much more intuitive and easier to compare between different protocols. At this point, we would like to highlight that our usage of the protocol-specific origin function $\Phi_{\mathsf{origin}}(\mathsf{iID})$ will have implications when defining what an "optimally" secure BAKE protocol is. See Section 3.4 for more discussion.

3.3.3. Match Soundness

Lastly, we provide soundness guarantees for the predicates Origin and Partner. Observe that an origin instance highly depends on the definition of Φ_{origin}. For instance, we can define $\Phi_{\mathsf{origin}}(\mathsf{iID}) = \perp$ for any instance iID, making every receiver instance to be an origin instance to every sender instance. However, such a definition does not seem "good" (i.e., sound). We thus use a predicate Match to define the classes of sound predicates Origin and Partner.

Definition 6 (Predicate Match). Let $\mathcal{S}_{\mathsf{iID}} \subset \mathbb{N} \times (\{0, \perp\} \cup \mathbb{N})$ be the set of instances generated in the game (cf. Algorithm 1). The predicate $\mathsf{Match}(\mathcal{S}_{\mathsf{iID}})$ holds true if and only if for any $\mathsf{iID}, \mathsf{iID}', \mathsf{iID}'' \in \mathcal{S}_{\mathsf{iID}}$, we have the following.

1. If $\mathsf{Partner}(\mathsf{iID}, \mathsf{iID}') = \mathsf{true}$, then either $\mathsf{Origin}(\mathsf{iID}, \mathsf{iID}') = \mathsf{true}$ or $\mathsf{Origin}(\mathsf{iID}', \mathsf{iID}) = \mathsf{true}$.

2. If $\mathsf{Partner}(\mathsf{iID}, \mathsf{iID}') = \mathsf{Partner}(\mathsf{iID}, \mathsf{iID}'') = \mathsf{true}$, then $\mathsf{iID}' = \mathsf{iID}''$.

3. If $\mathsf{Origin}(\mathsf{iID}, \mathsf{iID}') = \mathsf{Origin}(\mathsf{iID}, \mathsf{iID}'') = \mathsf{true}$, then $\mathsf{base}(\mathsf{iID}') = \mathsf{base}(\mathsf{iID}'')$. Moreover, we have the following two cases:

    a) If $\mathsf{iID}' \in \mathbb{N} \times \{0\}$ (i.e., a receiver instance associated with a one-time prekey bundle), then $\mathsf{iID}' = \mathsf{iID}''$.

    b) Otherwise, if $\mathsf{iID}' \in \mathbb{N} \times (\{\perp\} \cup \mathbb{N}^*)$ (i.e., a receiver instance associated with a last-resort prekey bundle), then there exists a unique instance $\mathsf{iID}_\perp = (\mathsf{base}(\mathsf{iID}'), \perp) \in \mathcal{S}_{\mathsf{iID}}$, and we have

    $$\left\{ \mathsf{iID}'' \ \middle| \ \begin{array}{c} \exists \mathsf{iID},\ \mathsf{Origin}(\mathsf{iID}, \mathsf{iID}'') = \mathsf{true} \\ \wedge\ \mathsf{iID}'' \neq \mathsf{iID}_\perp \end{array} \right\} = \{(\mathsf{base}(\mathsf{iID}_\perp), i)\}_{i \in [\mathsf{prekreuse}[\mathsf{iID}_\perp]]}.$$

Item 1 demands that if two instances iID and iID' are partners, then one of them must be an origin instance of the other. Due to the asymmetry of the definition of Origin, $\mathsf{Origin}(\mathsf{iID}, \mathsf{iID}') = \mathsf{true}$ when $\mathsf{role}(\mathsf{iID}) = \mathsf{sender}$. Item 2 demands that if a partner exists, then it is unique. The first part of Item 3 demands that if receiver instances iID' and iID'' are origin instances of a sender instance iID, then iID' and iID'' must share the same base instance identifier $\overline{\mathsf{iID}} = \mathsf{base}(\mathsf{iID})$. That is, $\mathsf{iID}' = (\overline{\mathsf{iID}}, \mathsf{ctr}')$ and $\mathsf{iID}'' = (\overline{\mathsf{iID}}, \mathsf{ctr}'')$ for $\mathsf{ctr}', \mathsf{ctr}'' \in \{0, \perp\} \cup \mathbb{N}$.

Items 3a and 3b add additional checks to Item 3. The first, Item 3a, demands that if iID' used a one-time prekey bundle, then $\mathsf{ctr}' = \mathsf{ctr}'' = 0$. Put differently, a sender instance has a unique origin instance. This reflects the fact that a one-time prekey bundle can only be used once, and is a common check performed for standard two-round AKE protocols. The second, Item 3b, demands that if iID' used a last-resort key bundle (i.e., $\mathsf{ctr}' \in \{\perp\} \cup \mathbb{N}$), then a sender instance may have multiple origin instances, all of which having the same base instance identifier. Moreover, we demand that there exists one unique instance $\mathsf{iID}_\perp$ that must have been generated

during BAKE.PreKeyBundleGen and all other instances are of the form $\{\,(\mathsf{base}(\mathsf{iID}'), i)\,\}_{i\in[\mathsf{prekreuse}[\mathsf{iID}_\perp]]}$, where recall prekreuse[$\mathsf{iID}_\perp$] is the number of time BAKE.Receive was called on the last-resort prekey bundle. This reflects the fact that a last-resort prekey bundle can be reused multiple times and many instances sharing the same prekey bundle exists.

Finally, we check whether predicate Match holds via the following security game.

Definition 7 (Match Soundness). We define the match soundness game in Algorithm 1 (with respect to a predicate Match and origin function Φ_{origin}) and define the advantage of an adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ as

$$\mathsf{Adv}^{\mathsf{MATCH}}_{\mathsf{BAKE}, \mathcal{A}}(\lambda) \coloneqq \Pr\left[\mathsf{Game}^{\mathsf{MATCH}}_{\mathsf{BAKE}, \mathcal{A}}(1^\lambda) = 1\right].$$

We say a BAKE protocol is *match sound* if $\mathsf{Adv}^{\mathsf{MATCH}}_{\mathsf{BAKE}, \mathcal{A}}(1^\lambda) = \mathsf{negl}(\lambda)$ for any efficient $\mathcal{A}$.

Note that for match soundness, we only need to define the game using $\mathcal{A}_1$ as $\mathcal{A}_2$ has no effect on the outcome of the game. We define it as above for readability and consistency with the key indistinguishability game. Moreover, notice that match soundness allows the adversary to arbitrary corrupt the users without consequence, as unlike key indistinguishability, it is not limited by any predicate safe.

Remark 1 (KEM re-encapsulation attack on PQXDH). *The KEM re-encapsulation attack on Signal's PQXDH has been documented in numerous places [Bha+23; Bha+24; FG24; KS23].*[5] *This attack forces two users to establish the same key, unknown to the adversary, while disagreeing on the encapsulation key being used. This was previously considered an implicit attack on key indistinguishability, though it is not immediately clear why key indistinguishability should fail. In contrast, we consider this as an explicit goal as such an attack will violate the first requirement in predicate* Match. *This helps better understand the scope of the attack and prevent similar vulnerabilities in future works. Indeed, we are able to capture replay attacks (see Section 4), seemingly never covered by any game-based security model. The one exception being [KBB17], covering this using symbolic analysis.*

3.4. Predicate safe$_{\mathsf{BAKE}}$: Optimal Security

As discussed in Section 3.3, the predicate safe defines a set of unavoidable attacks that break the key indistinguishability of a BAKE protocol. While the set of such attacks are protocol dependent, we first identify the *minimal* set of unavoidable attacks that no BAKE protocol can be secured against and define the associated predicate safe$_{\mathsf{BAKE}}$. This allows us to define the "optimal" key indistinguishability security as it provides the *maximum* attack freedom to the adversary.

Looking ahead, our RingXKEM in Section 5 will satisfy this best possible key indistinguishability. In contrast, X3DH and PQXDH are known to be insecure against some attacks not included in the above minimal set of unavoidable attacks (i.e., the adversary breaks key indistinguishability even if safe$_{\mathsf{BAKE}}$ evaluates to true). To this end, we will define more restrictive predicates safe$_{\mathsf{X3DH}}$ and safe$_{\mathsf{PQXDH}}$ in Section 3.5.

Keeping track of adversary's knowledge.   To define predicate safe$_{\mathsf{BAKE}}$, we must know *what* and *when* secret information is revealed. To do so, the security game keeps track of the adversary's knowledge by managing the following lists.

RevSessKey[iID] $\in$ { true, false } records whether the session key of the instance iID is revealed.

RevIK[u] $\in$ { true, false } records whether the identity secret key of the user u is revealed.

RevUserSt[u, epoch] $\in \mathbb{N} \cup$ { false } records whether the user state of user u in epoch $\in \mathbb{N}$, denoted as $\mathsf{st}_{u,\mathsf{epoch}}$, is revealed. If not, it records false. Otherwise, it records an integer value indicating how many times $\mathsf{st}_{u,\mathsf{epoch}}$ was used by the Receive algorithm on time of reveal. For instance, if $t = \mathsf{RevUserSt}[u, \mathsf{epoch}]$ satisfies $t \leq L$, then it indicates that u used t of the one-time prekeys, otherwise if $t > L$, then u used the last-resort prekey.

---

[5]Although it is called an "attack", PQXDH is not vulnerable against this attack thanks to the design of Kyber. Moreover, there are easy ways to thwart the attack without relying on these special properties. See [KS23, Sec. 4.2] and Section 4 for more details.

PeerCorr[iID] $\in$ { true, false } records whether the identity secret key of the peer of instance iID has been revealed when iID computed the session key, to model forward secrecy.

StateRev[iID] $\in$ { true, false } records whether the user state of the owner of the instance iID is revealed when iID computes the session key, conditioning on iID being a *receiver* instance. That is, the game does not need to keep track if iID is a sender instance; StateRev[iID] = false if role[iID] = sender (see Remark 2 for more details).

Algorithm 3 The predicates $\text{safe}_{\text{protocol}}$ where protocol $\in$ { BAKE, X3DH, PQXDH }.

1: **function** $\text{safe}_{\text{protocol}}(\text{iID}^*)$
2:     $(s^*, r^*) \leftarrow (\text{Sender}[\text{iID}^*], \text{Receiver}[\text{iID}^*])$
3:     ▷ *Origin instances* ◁
4:     $\mathfrak{O}(\text{iID}^*) \leftarrow \{ \text{iID} \in \mathcal{S}_{\text{iID}} \mid \text{Origin}(\text{iID}^*, \text{iID}) = \text{true} \}$
5:     ▷ *Partner instances* ◁
6:     $\mathfrak{P}(\text{iID}^*) \leftarrow \{ \text{iID} \in \mathcal{S}_{\text{iID}} \mid \text{Partner}(\text{iID}^*, \text{iID}) = \text{true} \}$
7:     **if** $[\![\forall \text{Attack} \in \text{Table 1} : \text{Attack}(\text{iID}^*) = \text{false}]\!]$ **then**
8:        ▷ $\mathcal{A}$ *did not execute any unavoidable attacks* ◁
9:        **return** true
10:    **else if** $[\![\text{protocol} \in \{ \text{X3DH, PQXDH} \}]\!] \wedge [\![\forall \text{Attack} \in \text{Table 3} \setminus \{ \text{Attack-6\&7} \} : \text{Attack}(\text{iID}^*) = \text{false}]\!]$ **then**
11:       ▷ X3DH/PQXDH $\mathcal{A}$ *does not execute classical attacks* ◁
12:       **return** true
13:    **if** $[\![\text{protocol} = \text{PQXDH}]\!] \wedge [\![\text{Attack-6}(\text{iID}^*) = \text{false}]\!] \wedge [\![\text{Attack-7}(\text{iID}^*) = \text{false}]\!]$ **then**
14:       ▷ $\mathcal{A}$ *does not execute a specific HNDL attack in* **Table 3** ◁
15:       **return** true
16:    **return** false

Predicate $\text{safe}_{\text{BAKE}}$ $\Leftarrow$ Unavoidable attacks against any BAKE protocol. We first specify the set of unavoidable attacks that no BAKE protocol can prevent in Table 1.

Attack 1 The adversary reveals the session key of the tested instance iID^*.

Attack 2 Assume the tested instance iID^* has a partner instance iID and consider an adversary that reveals the session key of iID. This is an unavoidable attack since partner instances derive the same session keys (cf. Definition 5).

Attack 3 Assume the tested instance iID^* is owned by a sender (resp. receiver), it has an origin (resp. partner) instance iID, and it used a *one-time* prekey bundle. Consider an adversary that reveals the receiver's identity secret key and the receiver's user state containing the secret of the used one-time prekey bundle. This is an unavoidable attack since BAKE.Receive is deterministic; the adversary can simply run it as the receiver to derive the session key of the tested instance. We divide into Attacks 3-1 (role = sender) and 3-2 (role = receiver) by the tested instance's role.

Attack 4 Assume the tested instance iID^* is owned by a sender (resp. receiver), it has an origin (resp. a partner) instance iID, and it used a *last-resort* prekey bundle. Consider an adversary that reveals the receiver's identity secret key and the receiver's user state containing the last-resort prekey secret. Similarly to Attack 3, the adversary can compute the session key of the tested instance. We divide Attack 4 into Attacks 4-1 (role = sender) and 4-2 (role = receiver).

Attack 5 Assume that the tested instance iID^* has no origin or partner instance. Consider an adversary that corrupts the identity secret key of the peer of iID^* before iID^* computed the session key. This results in an unavoidable attack since if the adversary knew the peer's identity secret key, it can trivially impersonate the peer of the tested instance, thus computing the same session key. We divide Attack 5 into Attacks 5-1 (role = sender) and 5-2 (role = receiver).

Table 1: Minimal set of unavoidable attacks against any BAKE protocol. Each row denotes the predicate Attack-xx(iID*) returning the logical AND of the conditions specified in each column. Variables s^* = Sender[iID*] and r^* = Receiver[iID*] denote the sender and receiver relative to tested instance iID*; one of them is the identity of the user in iID* and the other of its (supposed) peer. ep^* denotes the epoch in which the used prekey was issued. "—" means that the variable can take any value.

	Status of the tested iID*					Adversary's activities							
Attack	$\mathrm{role}[iID^*]$	$\lvert\mathfrak{O}(iID^*)\rvert$	$\lvert\mathfrak{P}(iID^*)\rvert$	$\exists iID \in \mathfrak{O}(iID^*):\ \mathrm{prekidx}[iID]$	$\mathrm{prekidx}[iID^*]$	$\mathrm{PeerCorr}[iID^*]$	$\mathrm{StateRev}[iID^*]$	$\mathrm{RevSessKey}[iID^*]$	$\exists iID \in \mathfrak{P}(iID^*):\ \mathrm{RevSessKey}[iID]$	$\mathrm{RevIK}[s^*]$	$\mathrm{RevIK}[r^*]$	$\mathrm{RevUserSt}[r^*,ep^*]$	Explanation
1	—	—	—	—	—	—	—	true	—	—	—	—	key[iID*] is revealed.
2	—	—	1	—	—	—	—	—	true	—	—	—	The session key key[iID] of the partner instance of iID* is revealed.
3-1	sender	1	—	$\leq L$	—	—	—	—	—	—	true	$< \mathrm{prekidx}[iID]$	When the same one-time prekey is used by the sender and the receiver, both isk_{r^*} and st_{r^*,ep^*} are revealed before the one-time prekey is used by r^*.
3-2	receiver	—	1	—	$\leq L$	—	—	—	—	—	true	$< \mathrm{prekidx}[iID^*]$	Same as 3-1
4-1	sender	1	—	$\perp$	—	—	—	—	—	—	true	$\neq$ false	When the same last-resort prekey is used by the sender and the receiver, both isk_{r^*} and st_{r^*,ep^*} are revealed.
4-2	receiver	—	1	—	$\perp$	—	—	—	—	—	true	$\neq$ false	Same as 4-1
5-1	sender	0	—	—	—	true	—	—	—	—	—	—	The peer of the tested instance may be impersonated by the adversary.
5-2	receiver	—	0	—	—	true	—	—	—	—	—	—	Same as 5-1

Note: $\mathfrak{O}(iID)$ and $\mathfrak{P}(iID)$ give the set of origin and partner sessions, respectively, for iID (see Algorithm 3).

Remark 2 (Asymmetry between sender and receiver). *Notice that Attacks 3 and 4 only consider an adversary revealing the* receiver's *identity secret key and user state. In particular, an adversary revealing the* sender's *identity secret key and user state is not considered an unavoidable attack. This is because a* BAKE *protocol is two-round and the* BAKE.Send *algorithm is probabilistic and does not use the user's state. As such, there is no immediate way for the adversary to compute the tested session key given the sender's secrets.*

Definition 8 (Predicate safe$_\mathsf{BAKE}$). We define the *optimal* predicate safe$_\mathsf{BAKE}$ for any BAKE protocol in Algorithm 3 based on the set of unavoidable attacks in Table 1.

The rows of Table 1 work as predicates that return the logical AND of the conditions specified in each column. Predicate safe$_\mathsf{BAKE}$ checks if there is a row in the table that returns true. If any rows returns true, then the adversary has executed an unavoidable attack. In this case, the tested instance is deemed unsafe. In other words, if all rows return false, the session key derived by the tested instance must be secure (if the protocol and primitives used are secure).

Notice predicate safe$_\mathsf{BAKE}$ is parameterized by the predicates Origin and Partner. As mentioned in Section 3.3.2, while predicate Partner is defined unambiguously by Definition 5 between different protocols, predicate Origin has some ambiguity due to our usage of the origin function Φ_origin (see Definition 4). As such, it is worth highlighting that "optimal" security is defined implicitly with respect to a specific choice of Φ_origin.

3.4.1. Avoidable Attacks on BAKE Protocols

To prove security of a BAKE protocol, we must show that it is secure against any adversary that does *not* execute any of the unavoidable attacks in Table 1. Taking the counter-positive, we consider every attack strategy for which predicate safe$_\mathsf{BAKE}$ evaluates to true, and then prove key indistinguishability for each. Such attack strategies can be derived by enumerating the combinations of variables such that the value of each row of Table 1 is false. As this is a useful tool for any security proof, we formally depict this in Table 2. Specifically, if the predicate safe$_\mathsf{BAKE}$ evaluates to true, then the adversary must take one of the attack strategies shown in Table 2. Note that all standard

AKE security proofs either implicitly or explicitly follow this proof strategy [Fuj+12; Han+21; Has+21; Has+22; Höv+20; Jag+21; PQR21].

Table 2: Every allowed adversary attack strategy (i.e., attacks for which $\mathsf{safe}_\mathsf{BAKE}$ evaluates to true). See Table 1 for notation. Each type is split depending on the role of the tested instance.

Attack	Status of the tested iID*					Adversary's activities							Explanation
	role[iID*]	$\lvert\mathfrak{C}(\mathsf{iID}^*)\rvert$	$\lvert\mathfrak{B}(\mathsf{iID}^*)\rvert$	$\exists\mathsf{iID}\in\mathfrak{O}(\mathsf{iID}^*): \text{prekidx}[\mathsf{iID}]$	prekidx[iID*]	PeerCorr[iID*]	StateRev[iID*]	RevSessKey[iID*]	$\exists\mathsf{iID}\in\mathfrak{P}(\mathsf{iID}^*): \text{RevSessKey}[\mathsf{iID}]$	RevIK[s*]	RevIK[r*]	RevUserSt[r*, ep*]	
1-1	sender	1	—	—	—	—	—	false	false	—	false	—	Reveal user-state $\mathsf{st}_{r^*,\mathsf{ep}^*}$ but not identity key isk_{r^*}.
1-2	receiver	—	1	—	—	—	—	false	false	—	false	—	Same as 1-1.
2-1	sender	1	—	≤ L	—	—	—	false	false	—	—	≥ prekidx[iID]	Reveal $\mathsf{st}_{r^*,\mathsf{ep}^*}$ *after* the one-time prekey is used by r^* and reveal isk_{r^*}.
2-2	receiver	—	1	—	≤ L	—	—	false	false	—	—	≥ prekidx[iID*]	Same as 2-1.
3-1	sender	1	—	⊥	—	—	—	false	false	—	—	false	Only reveal isk_{r^*} if the last-resort key is used.
3-2	receiver	—	1	—	⊥	—	—	false	false	—	—	false	Same as 3-1.
4-1	sender	0	—	—	—	false	—	false	n/a	—	—	—	Attack against key-compromise impersonation security and full forward secrecy is allowed.
4-2	receiver	—	0	—	—	false	—	false	n/a	—	—	—	Same as 4-1.

To get some intuition behind the attacks, we will map the attack strategies to known attacks documented in standard AKE protocols.

Maximal exposure attack [CK01; Fuj+12; Kra05; LLM07]: This is captured by Types 1, 2, and 3. In this attack, the adversary can obtain any combinations of the identity secret and user-state of partnering and origin instances, except for those that lead to the unavoidable Attacks 3 and 4. Note that since sender's user-state is not used to generate the handshake message (cf. Remark 2), we always allow the adversary to reveal the sender's identity secret and its user-state.

Key-compromise impersonation (KCI) attack [BJM97; Kra05]: This is captured by Type 4. In this attack, the adversary can obtain the identity secret key of the tested instance and uses it to impersonate another user against the tested instance.

Attack against full forward security [CK01; DOW92]: This is also captured by Type 4. In this attack, an active adversary (i.e., the tested instance has no origin/partner instance) can obtain the identity secret key of the peer of the tested instance *after* the session key has been computed.

3.5. Predicates ($\mathsf{safe}_\mathsf{X3DH}$, $\mathsf{safe}_\mathsf{PQXDH}$): Achievable Security

In addition to the unavoidable attacks specified in the previous section for any BAKE protocol, Signal's X3DH and PQXDH have some documented and accepted weaknesses in specific powerful compromise scenarios. Below, we specify these additional unavoidable attacks to exclude them from our security analysis.

Predicates ($\mathsf{safe}_\mathsf{X3DH}$, $\mathsf{safe}_\mathsf{PQXDH}$) $\Leftarrow$ Unavoidable attacks specific to (X3DH, PQXDH). The unavoidable attacks specific to X3DH and/or PQXDH are given in Table 3.

The first attack assumes a harvest-now-decrypt-later (HNDL) adversary (cf. Section 3.3.1) and only concerns PQXDH.

Attack 6 Assume the tested instance iID* is owned by a sender (resp. receiver), it has an origin (resp. a partner) instance iID, and it used a *one-time PQKEM prekey*. Consider an adversary that reveals the receiver's state before the origin (resp. tested) instance computes the session key.

Attack 7 Assume the tested instance iID* is owned by a sender (resp. receiver), it has an origin (resp. a partner) instance iID, and it used a *last-resort PQKEM prekey*. Consider an adversary that reveals the receiver's state.

Although these attacks may not be formally documented, it is implied since PQXDH is not fully quantum secure, only aiming to be secure against HNDL adversaries. Namely, the above attack exploits the fact that if a (harvest-now) classical adversary $\mathcal{A}_1$ obtains the secret associated to the PQKEM prekey, then all security is lost against a (decrypt-later) quantum adversary $\mathcal{A}_2$ since $\mathcal{A}_2$ can break all the Diffie–Hellman secrets to compute the session key.

The next attack is on the *full forward secrecy* of the *sender*.

Attack 8-1 Assume the tested instance iID* is owned by a sender without an origin instance, and consider an adversary that has revealed the receiver's identity secret key after the tested instance computed the session key.

In X3DH and PQXDH, an adversary can mount Attack 8-1 by providing a sender with a prekey in which the *unsigned* ephemeral Diffie–Hellman public key opk is replaced by an adversarial opk*. Since the prekey is modified, the sender will no longer have an origin instance, and as such, the adversary is able to reveal the receiver's user state containing the secret to the prekey. Combined with the receiver's identity secret key, the adversary can now compute the session key.

The final attack is a *user-state compromise impersonation* attack of the *receiver*.

Attack 8-2 Assume the tested instance iID* is owned by a receiver and has no partner instance. Consider an adversary that has revealed the receiver's user state before the tested instance computed the session key.

This attack against X3DH and PQXDH is well-known and is documented in the Signal documentation [MP16, Sec. 4.6] and [KS23, Sec. 4.6].[6] Notably, once the receiver's state is revealed, an adversary can impersonate any user to the receiver.

We now define the predicates safe$_{\text{X3DH}}$ and safe$_{\text{PQXDH}}$.

Definition 9 (Predicates safe$_{\text{X3DH}}$ and safe$_{\text{PQXDH}}$)**.** We define the predicates safe$_{\text{X3DH}}$ and safe$_{\text{PQXDH}}$ for a BAKE protocols X3DH and PQXDH, respectively, in Algorithm 3 based on the set of unavoidable attacks in Table 3.

Table 3: Additional unavoidable attacks specific to X3DH and PQXDH, where Attacks 6-x and 7-x are unique to PQXDH as we consider a HNDL adversary (i.e., $\mathcal{A}_2$ is quantum; $\mathcal{A}_1$ is always classical). Refer to Table 1 for the notation used in this table.

Attack	role[iID*]	$|\Sigma(\text{iID}^*)|$	$|\mathfrak{B}(\text{iID}^*)|$	$\exists \text{iID} \in \Sigma(\text{iID}^*):$ prekidx[iID]	prekidx[iID*]	$\mathcal{A}_2$ is quantum	PeerCorr[iID*]	StateRev[iID*]	RevSessKey[iID*]	$\exists \text{iID} \in \mathfrak{B}(\text{iID}^*):$ RevSessKey[iID]	RevIK[r*]	RevIK[r**]	RevUserStr[r*, ep*]	Explanation
6-1	sender	1	—	≤ L	—	true	—	—	—	—	—	—	< prekidx[iID]	**PQXDH**: The KEM prekey is known to *quantum* adversaries.
6-2	receiver	—	1	—	≤ L	true	—	—	—	—	—	—	< prekidx[iID]	Same as 6-1
7-1	sender	1	—	⊥	—	true	—	—	—	—	—	—	≠ false	**PQXDH**: The KEM prekey is known to *quantum* adversaries.
7-2	receiver	—	1	—	⊥	true	—	—	—	—	—	—	≠ false	Same as 7-1
8-1	sender	0	—	—	—	—	false	—	—	—	—	true	—	Unavoidable attack against full forward secrecy for *sender*.
8-2	receiver	—	0	—	—	—	—	true	—	—	—	—	—	Unavoidable attack against user-state compromised impersonation security for *receiver*.

---

[6]While the documentation uses the term "key" compromise impersonation attack, we use "user-state" as that is what the adversary reveals.

Table 4: Every allowed adversary attack strategy for X3DH and PQXDH. The differences with Table 2 are indicated in red. As in Table 3, adversary $\mathcal{A}_1$ is always classical; $\mathcal{A}_2$ possibly being quantum is only considered for PQXDH: in X3DH it is classical. See Table 2 for notation.

| Attack | Status of the tested iID* | | | | | Adversary's activities | | | | | | | | Explanation |
	role[iID*]	$|\Sigma(\text{iID}^*)|$	$|\Re(\text{iID}^*)|$	$\exists\text{iID} \in \Sigma(\text{iID}^*):$ prekidx[iID]	prekidx[iID*]	$\mathcal{A}_2$ is quantum	PeerCorr[iID*]	StateRev[iID*]	RevSesskey[iID*]	$\exists\text{iID} \in \Re(\text{iID}^*):$ RevSesskey[iID]	RevIK[s*]	RevIK[r*]	RevUserSt[r*, ep*]	
1-1	sender	1	—	—	—	false	—	—	false	false	—	false	—	Only for classical adversary, reveal user-state $\text{st}_{r^*,\text{ep}^*}$ but not secret key isk_{r^*}.
1-2	receiver	—	1	—	—	false	—	—	false	false	—	false	—	Same as 1-1
2-1	sender	1	—	≤ L	—	—	—	—	false	false	—	—	≥ prekidx[iID]	Reveal $\text{st}_{r^*,\text{ep}^*}$ *after* the one-time prekey is used by r^* and reveal isk_{r^*}.
2-2	receiver	—	1	—	≤ L	—	—	—	false	false	—	—	≥ prekidx[iID*]	Same as 2-1
3-1	sender	1	—	⊥	—	—	—	—	false	false	—	—	false	Only reveal isk_{r^*} if the last-resort key is used.
3-2	receiver	—	1	—	⊥	—	—	—	false	false	—	—	false	Same as 4-1
4	sender	0	—	—	—	false	false	—	false	n/a	—	false	—	Only attack against *weak* forward secrecy is allowed for the *sender* but can reveal user-state.
5	receiver	—	0	—	—	—	false	false	false	n/a	—	—	—	Attack against full forward secrecy is allowed for the *receiver* but cannot reveal user-state.

3.5.1. Avoidable attacks.

Similarly to $\text{safe}_{\text{BAKE}}$, for completeness, we take the counter positive and list all the allowed adversary attack strategies. To show key indistinguishability, we prove that the protocol remains secure with respect to each attack strategies. This is given in Table 4.

Notice that Types 2 and 3 are identical to the allowed attack strategies for the optimal BAKE protocol (cf. Table 2). Type 1 is relaxed by only allowing classical adversaries when the user-state is revealed. Type 4 captures *weak* forward secrecy for the sender as apposed to full forward secrecy. Lastly, while Type 5 captures full forward secrecy for the receiver, it restricts the adversary from compromising the receiver's user-state.

4. Signal's X3DH and PQXDH

The X3DH protocol [MP16] was proposed in 2016 by Marlinspike and Perrin based on the Triple Diffie–Hellman AKE protocol [KP05]. In 2023, Signal introduced PQXDH to protect the Signal handshake protocol against harvest now, decrypt later attacks [KS23]. In this section, we will first describe X3DH and PQXDH, then we discuss their security.

4.1. Descriptions of X3DH and PQXDH

The descriptions of X3DH and PQXDH are given in Algorithms 4 to 6. As PQXDH mainly consists of the addition of a post-quantum KEM to X3DH, it is described in the same figures, marked with a gray dotted box. Below, we first focus on the shared features before discussing PQXDH's additions.

The key agreement in these protocols proceeds roughly as follows. The identity keys of both users are Diffie–Hellman (DH) values. The prekey bundle contains a signed DH key, and, if it is a one-time prekey bundle, an ephemeral DH key. Finally, the sender generates an ephemeral key. These keys are used pairwise in DH computations before combining them into a shared secret ss (c.f. Algorithm 5, Lines 6 to 13).

While our description of X3DH and PQXDH closely follows Signal's documentation [KS23; MP16], we incorporated several minor modifications based on discussions with Signal developers that may be included in future updates [Sch24].

Algorithm 4 PQX3DH identity key and prekey bundle generation algorithms. PQXDH-exclusive code is marked like this.

1: **function** PQX3DH.IdKeyGen(1^λ)
2: $\quad \overline{\text{isk}} \overset{\$}{\leftarrow} \mathbb{Z}_p; \overline{\text{ik}} := [\overline{\text{isk}}]G$
3: $\quad (\text{vk}, \text{sk}) \leftarrow \text{Sig.KeyGen}(1^\lambda)$
4: $\quad$ **return** $(\text{ik} := (\overline{\text{ik}}, \text{vk}), \text{isk} := (\overline{\text{isk}}, \text{sk}))$

1: **function** PQX3DH.PreKeyBundleGen(isk_u)
2: $\quad (\overline{\text{isk}}_u, \text{sk}_u) \leftarrow \text{isk}_u$
3: $\quad D_{\text{prek}}, D_{\rho_\perp} := \emptyset \quad \triangleright$ *Initialize empty lists*
4: $\quad \triangleright$ *Generate what Signal calls the* signed prekey $\quad\triangleleft$
5: $\quad \text{spksec}_u \overset{\$}{\leftarrow} \mathbb{Z}_p; \text{spk}_u := [\text{spksec}]G$
6: $\quad \sigma_{\text{spk}_u} \leftarrow \text{Sig.Sign}(\text{sk}_u, \text{spk}_u)$
7: $\quad \triangleright$ *Create the L one-time* prekey bundles $\quad\triangleleft$
8: $\quad$ **for** $t \in [L]$ **do**
9: $\quad\quad \text{osk}_{u,t} \overset{\$}{\leftarrow} \mathbb{Z}_p; \text{opk}_{u,t} := [\text{osk}_{u,t}]G$
10: $\quad\quad (\text{ek}_{u,t}, \text{dk}_{u,t}) \overset{\$}{\leftarrow} \text{KEM.KeyGen}(1^\lambda)$
11: $\quad\quad \sigma_{\text{ek}_{u,t}} \overset{\$}{\leftarrow} \text{Sig.Sign}(\text{sk}_u, \text{ek}_{u,t})$
12: $\quad\quad \text{prek}_{u,t} := (\text{spk}_u, \sigma_{\text{spk}_u}, \text{opk}_{u,t}, \text{ek}_{u,t}, \sigma_{\text{ek}_{u,t}})$
13: $\quad\quad D_{\text{prek}}[t] \leftarrow (\text{prek}_{u,t}, (\text{spksec}_u, \text{osk}_{u,t}, \text{dk}_{u,t}))$
14: $\quad \triangleright$ *Set up the* last-resort *prekey bundle* $\quad\triangleleft$
15: $\quad (\text{ek}_{u,\perp}, \text{dk}_{u,\perp}) \overset{\$}{\leftarrow} \text{KEM.KeyGen}(1^\lambda)$
16: $\quad \sigma_{\text{ek}_{u,\perp}} \overset{\$}{\leftarrow} \text{Sig.Sign}(\text{sk}_u, \text{ek}_{u,\perp})$
17: $\quad \text{prek}_{u,\perp} := (\text{spk}_u, \sigma_{\text{spk}_u}, \perp, \text{ek}_{u,\perp}, \sigma_{\text{ek}_{u,\perp}})$
18: $\quad D_{\text{prek}}[\perp] \leftarrow (\text{prek}_{u,\perp}, (\text{spksec}_u, \perp, \text{dk}_{u,\perp}))$
19: $\quad$ **return** $(\vec{\text{prek}}_u, \text{st}_u := (D_{\text{prek}}, D_{\rho_\perp}))$

It is worth noting that the Signal implementation also deviates from the documentation in various ways.[7] Though the documentation is titled "The PQXDH Key Agreement Protocol" [KS23], the described protocol additionally transmits an initial protocol message, encrypted using some unspecified authenticated encryption with associated data (AEAD). The same key used to encrypt this message is also the key that is output from the key exchange protocol. This lack of key separation and the inclusion of a user-specified message make it not just harder to consider X3DH and PQXDH as a modular "handshake" component to the Signal messaging protocol, but also harder to model.

Arguably, the sending of a message and lack of key separation are (over)simplifications made in a somewhat informal description. The Signal implementation actually interleaves the initial messages of the Double Ratchet algorithm with the PQXDH handshake, using Double Ratchet to derive new keys to encrypt and authenticate the message (using AES-CBC and HMAC). For ease of presentation, modeling, and to prove the security of a modular PQXDH handshake without having to consider Double Ratchet, we remove the AEAD and include protocol specific contents into the key derivation function (KDF) to generate a confirmation tag τ_{conf} in our protocol descriptions. At a high level, the confirmation tag acts as an implicit *one-time* MAC, replacing the need of an AEAD, where the message being signed is the sender's view of the protocol. We discussed this with Signal, who indicated that, in response to these findings, they may follow our suggestion to make a better separation between the handshake protocol and Double Ratchet. Looking ahead, such a modification allows us to prevent the KEM re-encapsulation attack on PQXDH without making non-standard assumptions on the underlying KEM (cf. Remark 1).

We further modify the users to keep track of the received handshake messages with respect to the last-resort prekey bundle using a list $D_{\rho_\perp}$. The receiver will reject any handshake message ρ such that $\rho \in D_{\rho_\perp}$. See Algorithm 6, Line 8. Note that we could further compress $D_{\rho_\perp}$ by hashing ρ, adding an assumption on collision resistance. Since a last-resort prekey can be reused, this protects against an adversary mounting a

---

[7]This was also noted by [Coh+17], who also heavily refer to source code.

Algorithm 5 PQX3DH sender algorithms. prek is not indexed by $t \in [L] \cup \{\bot\}$ as they are oblivious to the sender. PQXDH-exclusive code is marked like this.

1: **function** PQX3DH.Send(isk_s, ik_r, prek_r)
2:    $(\overline{\mathsf{isk}}_s, \mathsf{sk}_s) \leftarrow \mathsf{isk}_s$; $(\overline{\mathsf{ik}}_r, \mathsf{vk}_r) \leftarrow \mathsf{ik}_r$
3:    $\left(\mathsf{spk}_r, \sigma_{\mathsf{spk}_r}, \mathsf{opk}_r, \boxed{\mathsf{ek}_r, \sigma_{\mathsf{ek}_r}}\right) \leftarrow \mathsf{prek}_r$ ▷ $\mathsf{opk}_r = \bot$ *if* prek_r *is a last-resort key bundle*
4:    **require** $[\![\, \mathsf{Sig}.\mathsf{Verify}(\mathsf{vk}_r, \mathsf{spk}_r, \sigma_{\mathsf{spk}_r}) = 1 \,]\!]$
5:    $\boxed{\textbf{require } [\![\, \mathsf{Sig}.\mathsf{Verify}(\mathsf{vk}_r, \mathsf{ek}_r, \sigma_{\mathsf{ek}_r}) = 1 \,]\!]}$
6:    $\mathsf{esk} \xleftarrow{\$} \mathbb{Z}_p$, $\mathsf{epk} := [\mathsf{esk}]G$
7:    $\mathsf{ss}_1 := [\overline{\mathsf{isk}}_s]\mathsf{spk}_r$
8:    $\mathsf{ss}_2 := [\mathsf{esk}]\overline{\mathsf{ik}}_r$
9:    $\mathsf{ss}_3 := [\mathsf{esk}]\mathsf{spk}_r$
10:   $\mathsf{ss} := \mathsf{ss}_1 \| \mathsf{ss}_2 \| \mathsf{ss}_3$
11:   **if** $[\![\, \mathsf{opk}_r \neq \bot \,]\!]$ **then** ▷ *One-time prekey bundle*
12:     $\mathsf{ss}_4 := [\mathsf{esk}]\mathsf{opk}_r$
13:     $\mathsf{ss} := \mathsf{ss}_1 \| \mathsf{ss}_2 \| \mathsf{ss}_3 \| \mathsf{ss}_4$
14:   $\boxed{(\mathsf{ss}_{\mathsf{KEM}}, \mathsf{ct}) \xleftarrow{\$} \mathsf{KEM}.\mathsf{Encaps}(\mathsf{ek}_r)}$
15:   $\mathsf{content} := \mathsf{ik}_s \| \mathsf{ik}_r \| \mathsf{prek}_r \| \mathsf{epk} \boxed{\| \mathsf{ct}}$
16:   $K \| \tau_{\mathsf{conf}} := \mathsf{KDF}(\mathsf{ss} \boxed{\| \mathsf{ss}_{\mathsf{KEM}}}, \mathsf{content})$
17:   $\rho := (\mathsf{epk}, \boxed{\mathsf{ct},} \tau_{\mathsf{conf}})$
18:   **return** (K, ρ)

---

replay attack that makes a receiver derive the same session key multiple times. Observe a protocol vulnerable against this replay attack explicitly violates our match soundness as it allows creating multiple partner instances (cf. Definition 6, Item 2). Replay attacks appear to have been overlooked in prior analyses (although mentioned in the documentation [KS23, Sec. 4.2], and covered by a symbolic analysis [KBB17]), which illustrates the usefulness of our security model. We highlight that Signal implements the countermeasure suggested by the documentation.

We further clarify the differences between Algorithms 5 and 6, the Signal X3DH protocol description, and the `libsignal` implementation [Sig] in Appendix B. Lastly, common to prior work, we separate the identity key into separate keys for ECDH key agreement and EdDSA signatures. In practice, Signal uses this key in both roles, using the X25519 secret as an XEd25519 signing key [Per16].

4.2. HNDL-Security for PQXDH

PQXDH only attempts to give post-quantum security against HNDL attacks, and thus still relies on elliptic curve cryptography for authentication. While the identity keys are the same as X3DH, signed post-quantum KEM keys are added to the prekey bundles. In the functions PQXDH.Send and PQXDH.Receive one can see how these additional KEM keys are used to inject a KEM-encapsulated quantum-safe shared secret into the key returned by the handshake.

Note that although the Signal specification and implementation of PQXDH supports prekey bundles without KEM prekeys (as this gives backwards compatibility with X3DH), we do not to model this.[8] Classic security of PQXDH without KEM prekeys follows directly from X3DH.

Downgrade resilience of PQXDH. As long as PQXDH clients do not enforce the usage of KEM prekeys, i.e., run in "compatibility mode", a network attacker or malicious server may omit them from prekey bundles and force a classically-secure session. This is because the prekey bundle's composition is not authenticated. Though it appears receivers might notice that prekey bundle prek_t contained a KEM prekey when it was generated, in the Signal implementation, prekey bundles are actually assembled piece-wise on the server and the DH and KEM (one-time) prekeys are individually identified (i.e., in practice identifier t can be considered a tuple (t_{DH}, t_{KEM})). The protocols do not try to authenticate protocol version or algorithms supported by the sender or receiver, as, e.g.,

---

[8] This DH-only mode will eventually be disabled [Sch24].

Algorithm 6 PQX3DH receiver algorithms. PQXDH-exclusive code is marked like this.

1: **function** PQX3DH.Receive($\text{isk}_r, \text{st}_r, \text{ik}_s, t, \rho$)
2:     $(\overline{\text{isk}}_r, \text{sk}_r) \leftarrow \text{isk}_r$; $(\overline{\text{ik}}_s, \text{vk}_s) \leftarrow \text{ik}_s$
3:     $(D_{\text{prek}}, D_{\rho_\perp}) \leftarrow \text{st}_r$
4:     **if** $[\![t \neq \perp]\!]$ **then** ▷ *One-time prekey bundle*
5:         **require** $[\![D_{\text{prek}}[t] \neq \perp]\!]$ ▷ *Check if unused.*
6:         $(\text{prek}_{r,t}, (\text{spksec}_r, \text{osk}_{r,t}, \boxed{\text{dk}_{r,t}})) \leftarrow D_{\text{prek}}[t]$
7:     **else** ▷ *Last-resort prekey bundle (i.e., $t = \perp$)*
8:         **require** $[\![\rho \notin D_{\rho_\perp}]\!]$ ▷ *Check ρ is not replayed.*
9:         $D_{\rho_\perp} \leftarrow D_{\rho_\perp} \cup \{ \rho \}$
10:        $(\text{prek}_{r,t}, (\text{spksec}_r, \perp, \boxed{\text{dk}_{r,t}})) \leftarrow D_{\text{prek}}[t]$
11:     $(\text{epk}, \boxed{\text{ct},} \tau_{\text{conf}}) \leftarrow \rho$
12:     $\text{ss}_1 \coloneqq [\text{spksec}_r]\overline{\text{ik}}_s$; $\text{ss}_2 \coloneqq [\overline{\text{isk}}_r]\text{epk}$
13:     $\text{ss}_3 \coloneqq [\text{spksec}_r]\text{epk}$; $\text{ss} \coloneqq \text{ss}_1\|\text{ss}_2\|\text{ss}_3$
14:     **if** $[\![t \neq \perp]\!]$ **then** ▷ *One-time prekey bundle*
15:        $\text{ss}_4 \coloneqq [\text{osk}_{r,t}]\text{epk}$; $\text{ss} \coloneqq \text{ss}_1\|\text{ss}_2\|\text{ss}_3\|\text{ss}_4$
16:     $\boxed{\text{ss}_{\text{KEM}} \leftarrow \text{KEM.Decaps}(\text{dk}_{r,t}, \text{ct})}$
17:     $\text{content} \coloneqq \text{ik}_s\|\text{ik}_r\|\text{prek}_{r,t}\|\text{epk}\boxed{\|\text{ct}}$
18:     $K\|\tau'_{\text{conf}} \coloneqq \text{KDF}(\text{ss}\boxed{\|\text{ss}_{\text{KEM}}}, \text{content})$
19:     **require** $[\![\tau_{\text{conf}} = \tau'_{\text{conf}}]\!]$
20:     ▷ *Delete prekey bundle if not last-resort* ◁
21:     **if** $[\![t \neq \perp]\!]$ **then** $D_{\text{prek}}[t] \leftarrow \perp$
22:     $\text{st}_r \leftarrow (D_{\text{prek}}, D_{\rho_\perp})$
23:     **return** (K, st_r)
---

the TLS 1.3 handshake does [Res18]. That means that the sender and receiver will each assume the other only supported X3DH if the KEM prekeys are just omitted. As X3DH was not designed with negotiation in mind, this issue can seemingly not be prevented without sacrificing backwards compatibility.

4.3. Security Overview

The correctness of X3DH and PQXDH follows from construction. Below, we state the match soundness and key indistinguishability of PQXDH. Due to its similarity with PQXDH, we focus on the security of the more complex PQXDH and explain how X3DH differs in Appendix C.2.

Match soundness. We prove match soundness of PQXDH (and X3DH) with respect to the following origin function.

Definition 10 (Origin Function for Signal Protocols). For any $\text{iID} \in \mathcal{S}_{\text{iID}}$ (i.e., the set of all instances created during the game) with $\text{prek}[\text{iID}] \neq \perp$, we define the origin function as $\Phi_{\text{origin}}^{\text{Signal}}(\text{iID}) \coloneqq \text{prek}[\text{iID}]$.

As mentioned in Section 3.3.2, this is one of the most common ways to define an origin instance in the AKE literature [CF12; CF15; Jag+21; PRZ24; PWZ23]. We then show the following which establishes the match soundness of PQXDH. As discussed, this entails security against replay and KEM re-encapsulation attacks.

Theorem 1 (Match Soundness of PQXDH). *Assume the KDF is collision resistant against a quantum adversary. Then, PQXDH is match sound against a harvest-now-decrypt-later adversary with respect to the predicate Match (cf. Definition 6) and origin function $\Phi_{\text{origin}}^{\text{Signal}}$ (cf. Definition 10).*

Proof. We defer the proof to Appendix C.1.2. At a high level, we first provide a helper lemma allowing to check whether two instances are partners only looking at the public transcripts. Note that the current definition of partnering is not publicly checkable as it compares the established keys. With this helper lemma, checking match soundness consists of a straightforward check. □

Key indistinguishability. We show key indistinguishability with respect to the predicate $\mathsf{safe}_{\mathsf{PQXDH}}$. PQXDH offers security against a class of HNDL adversaries. But, as explained in Section 3.5, if the classical adversary compromises the post-quantum KEM prekeys, then it cannot offer HNDL security as all the remaining security comes from classical primitives.

Theorem 2. PQXDH *is key indistinguishable against a harvest-now-decrypt-later adversary with respect to the predicate* $\mathsf{safe}_{\mathsf{PQXDH}}$ *(cf. Definition 9).*

Proof. We defer the proof to Appendix C.1.3. As explained in Section 3.4, we use predicate $\mathsf{safe}_{\mathsf{PQXDH}}$ to define the set of "avoidable" attacks. This translates to all the allowed adversary attack strategies (cf. Table 4). We prove the advantage is negligible for each of these strategies. $\square$

5. Our Post-Quantum RingXKEM

In this section, we propose a post-quantum BAKE protocol RingXKEM that is key indistinguishable with respect to the predicate $\mathsf{safe}_{\mathsf{BAKE}}$ (cf. Definition 8). The core design of RingXKEM is inspired from the *deniable* AKE protocol by Hashimoto et al. [Has+21; Has+22] based on ring signatures. We extend it to meet the syntax of a BAKE protocol and optimize it using Merkle trees to save on receiver bandwidth and server storage.

5.1. Description of RingXKEM

The description of RingXKEM is given in Algorithms 7 to 9. The construction is based on a KDF, Merkle tree, KEM, and a ring signature. If we ignore the Merkle tree for a moment, used only for optimization purposes, the construction is quite simple. The tth ($t \in [L] \cup \{\bot\}$) prekey bundle consists of a KEM public key $\widehat{\mathsf{ek}}_t$, a (ring) signature on the $\widehat{\mathsf{ek}}_t$, and a ring signature verification key rvk. Here, rvk is shared by all $L + 1$ prekey bundles and the associated signing key rsk is discarded. A sender, after checking validity of $\widehat{\mathsf{ek}}_t$, will generate two KEM ciphertexts ct and $\widehat{\mathsf{ct}}$: one associated to ek included in the receiver's identity key and the other to $\widehat{\mathsf{ek}}_t$. It then generates a ring signature σ with the ring $\{\mathsf{rvk}_s, \mathsf{rvk}\}$, where the message is ct and $\widehat{\mathsf{ct}}$ along with additional public information. Lastly, the sender derives a session key K and an SKE key K_{ske} from the KEM session keys ss and $\widehat{\mathsf{ss}}$, encrypts σ using K_{ske} as $\mathsf{ct}_{\mathsf{ske}}$, and sends the handshake message $\rho = (\mathsf{ct}, \widehat{\mathsf{ct}}, \mathsf{ct}_{\mathsf{ske}})$. The receiver can process ρ using the KEM secret keys.

Notice that this vanilla construction requires the users to upload $L + 1$ (ring) signatures to the server. While this is also the case for PQXDH, this becomes problematic in RingXKEM when targeting post-quantum security. The signatures can become an order of magnitude larger than in the classical setting, making the prekey bundles very large. The Merkle tree optimization allows to only upload a single signature: the users accumulate all the KEM public keys $(\widehat{\mathsf{ek}}_t)_{t \in [L] \cup \{\bot\}}$ and only sign the digest root. We provide concrete numbers for this optimization in Section 6.2. It is worth noting that this Merkle tree optimization is made possible owing to our new definition of BAKE protocols. Previous works on Signal's handshake protocols, e.g., [Bre+22; Coh+17; Coh+20; Col+24; FG24; Has+21; Has+22], are not able to handle such optimization as each prekey bundle prek_t was assumed to be generated *independently*.

One downside of our optimization is that prekey bundles become slightly larger. In particular, a sender is now required to download an extra Merkle tree path_t proving that $\widehat{\mathsf{ek}}_t$ was accumulated in root. Notice that in our construction, the users explicitly include path_t in each prekey bundle prek_t. However, in practice, we can simply let the server reconstruct them using the uploaded $(\widehat{\mathsf{ek}}_t)_{t \in [L] \cup \{\bot\}}$ without harming security. Namely, when a sender retrieves u's prekey bundle from the server, the server can compute path_t on the fly. Importantly, due to binding of the Merkle tree, the server cannot inject a prekey that u did not accumulate in the hash digest.

Lastly, we note that the usage of ring signatures and an SKE to encrypt the ring signature is purely for deniability reasons, similarly to what is done in the standard AKE protocol by Hashimoto et al.; Hashimoto et al. While our protocol plausibly satisfies deniability, we leave a formal proof for future work as we would first need to formalize deniability for BAKE protocols.

The formal security statements and proofs are given in Appendix D.

Algorithm 7 RingXKEM's identity key and prekey bundle generation algorithms.

1: **function** RingXKEM.IdKeyGen(1^λ)
2:     $(\mathsf{ek}, \mathsf{dk}) \xleftarrow{\$} \mathsf{KEM.KeyGen}(1^\lambda)$
3:     $(\mathsf{rvk}, \mathsf{rsk}) \xleftarrow{\$} \mathsf{RS.KeyGen}(1^\lambda)$
4:     **return** $(\mathsf{ik} := (\mathsf{ek}, \mathsf{rvk}), \mathsf{isk} := (\mathsf{dk}, \mathsf{rsk}))$

5: **function** RingXKEM.PreKeyBundleGen(isk_u)
6:     $(\mathsf{dk}_u, \mathsf{rsk}_u) \leftarrow \mathsf{isk}_u$
7:     $D_{\mathsf{kem}}, D_{\rho_\perp} := \emptyset$ ▷ *Initialize empty lists*
8:     **for** $t \in [L] \cup \{\perp\}$ **do**
9:        $(\widehat{\mathsf{ek}}_{u,t}, \widehat{\mathsf{dk}}_{u,t}) \xleftarrow{\$} \mathsf{KEM.KeyGen}(1^\lambda)$
10:    ▷ *Create and sign Merkle tree* ◁
11:     $(\mathsf{root}_u, \mathsf{tree}_u) \leftarrow \mathsf{MerkleTree}((\widehat{\mathsf{ek}}_{u,t})_{t \in [L] \cup \{\perp\}})$
12:     $\sigma_{u,\mathsf{root}} \xleftarrow{\$} \mathsf{RS.Sign}(\mathsf{rsk}_u, \mathsf{root}_u, \{\mathsf{rvk}_u\})$
13:     $(\mathsf{rvk}, \_) \xleftarrow{\$} \mathsf{RS.KeyGen}(1^\lambda)$ ▷ *Discard* rsk
14:     **for** $t \in [L]$ **do** ▷ *One-time prekey bundles*
15:        $\mathsf{path}_{u,t} \leftarrow \mathsf{getMerklePath}(\mathsf{tree}_u, t)$
16:        $\mathsf{prek}_{u,t} := (\widehat{\mathsf{ek}}_{u,t}, \mathsf{path}_{u,t}, \mathsf{root}_u, \sigma_{u,\mathsf{root}}, \mathsf{rvk})$
17:        $D_{\mathsf{kem}}[t] \leftarrow (\mathsf{prek}_{u,t}, \widehat{\mathsf{dk}}_{u,t})$
18:    ▷ *Last-resort prekey bundle* $t = \perp$ ◁
19:     $\mathsf{path}_{u,\perp} \leftarrow \mathsf{getMerklePath}(\mathsf{tree}_u, L+1)$
20:     $\mathsf{prek}_{u,\perp} := (\widehat{\mathsf{ek}}_{u,\perp}, \mathsf{path}_{u,\perp}, \mathsf{root}_u, \sigma_{u,\mathsf{root}}, \mathsf{rvk})$
21:     $D_{\mathsf{kem}}[t] \leftarrow (\mathsf{prek}_{u,\perp}, \widehat{\mathsf{dk}}_{u,\perp})$
22:     **return** $\begin{pmatrix} \overrightarrow{\mathsf{prek}}_u := (\mathsf{prek}_{u,t})_{t \in [L] \cup \{\perp\}}, \\ \mathsf{st}_u := (D_{\mathsf{kem}}, \mathsf{rvk}, D_{\rho_\perp}) \end{pmatrix}$

Algorithm 8 RingXKEM's sender algorithm. The prekey bundle index t is oblivious to the sender.

1: **function** RingXKEM.Send($\mathsf{isk}_s, \mathsf{ik}_r, \mathsf{prek}_r$)
2:     $(\mathsf{dk}_s, \mathsf{rsk}_s) \leftarrow \mathsf{isk}_s; (\mathsf{ek}_r, \mathsf{rvk}_r) \leftarrow \mathsf{ik}_r$
3:     $(\widehat{\mathsf{ek}}_r, \mathsf{path}_r, \mathsf{root}_r, \sigma_{r,\mathsf{root}}, \mathsf{rvk}) \leftarrow \mathsf{prek}_r$
4:     **require** $[\![\mathsf{ReconstructRoot}(\widehat{\mathsf{ek}}_r, \mathsf{path}_r) = \mathsf{root}_r]\!]$
5:     **require** $[\![\mathsf{RS.Verify}(\{\mathsf{rvk}_r\}, \widehat{\mathsf{ek}}_r, \sigma_{r,\mathsf{root}}) = 1]\!]$
6:     $(\mathsf{ss}_r, \mathsf{ct}_r) \xleftarrow{\$} \mathsf{KEM.Encaps}(\mathsf{ek}_r)$
7:     $(\widehat{\mathsf{ss}}_r, \widehat{\mathsf{ct}}_r) \xleftarrow{\$} \mathsf{KEM.Encaps}(\widehat{\mathsf{ek}}_r)$
8:     $\mathsf{content} := \mathsf{ik}_s \| \mathsf{ik}_r \| \mathsf{prek}_r \| \mathsf{ct}_r \| \widehat{\mathsf{ct}}_r$
9:     $K \| K_{\mathsf{ske}} := \mathsf{KDF}(\mathsf{ss}_r \| \widehat{\mathsf{ss}}_r, \mathsf{content})$
10:     $\sigma \xleftarrow{\$} \mathsf{RS.Sign}(\mathsf{rsk}_s, \mathsf{content}, \{\mathsf{rvk}_s, \mathsf{rvk}\})$
11:     $\mathsf{ct}_{\mathsf{ske}} \xleftarrow{\$} \mathsf{SKE.Enc}(K_{\mathsf{ske}}, \sigma)$ ▷ *Mask ring signature*
12:     $\rho := (\mathsf{ct}_r, \widehat{\mathsf{ct}}_r, \mathsf{ct}_{\mathsf{ske}})$
13:     **return** (K, ρ)

---

6. Comparison

In this section, we will first compare the security properties of the protocols that we discussed, followed by a comparison of the efficiency of the different schemes.

6.1. Security

By proving the security of Signal handshake protocols using the BAKE abstraction and security model, we can make a direct comparison of their security properties; we show an overview in Table 5. By setting the powers of the adversary and modeling unavoidable attacks, we were able to show that PQXDH is indeed secure against harvest now, decrypt later attacks, but that this requires that the adversary is not able to obtain the secrets for the post-quantum KEM prekeys. Additionally, receivers in both X3DH and PQXDH cannot avoid user state compromise impersonation attacks, while senders are only weakly forward secure. Our proposal, RingXKEM, is

Algorithm 9 RingXKEM's receiver algorithm.

1: **function** RingXKEM.Receive($\text{isk}_r, \text{st}_r, \text{ik}_s, t, \rho$)
2:    $(\text{dk}_r, \text{rsk}_r) \leftarrow \text{isk}_r$; $(\text{ek}_s, \text{rvk}_s) \leftarrow \text{ik}_s$
3:    $(D_{\text{kem}}, \text{rvk}, D_{\rho_\perp}) \leftarrow \text{st}_r$
4:    $(\text{ct}_r, \widehat{\text{ct}}_r, \text{ct}_{\text{ske}}) \leftarrow \rho$
5:    ▷ *Check t^{th} prekey bundle was not deleted.*   ◁
6:    **require** $[\![\, D_{\text{kem}}[t] \neq \perp \,]\!]$
7:    **if** $[\![\, t = \perp \,]\!]$ **then**
8:       **require** $[\![\, (\text{ct}_r, \widehat{\text{ct}}_r) \notin D_{\rho_\perp} \,]\!]$ ▷ *Check not replayed.*
9:       $D_{\rho_\perp} \leftarrow D_{\rho_\perp} \cup \{ (\text{ct}_r, \widehat{\text{ct}}_r) \}$
10:   $(\text{prek}_{r,t}, \widehat{\text{dk}}_{r,t}) \leftarrow D_{\text{kem}}[t]$
11:   $\text{ss}_r := \text{KEM.Decaps}(\text{dk}_r, \text{ct}_r)$
12:   $\widehat{\text{ss}}_r := \text{KEM.Decaps}(\widehat{\text{dk}}_{r,t}, \widehat{\text{ct}}_r)$
13:   $\text{content} := \text{ik}_s \| \text{ik}_r \| \text{prek}_{r,t} \| \text{ct}_r \| \widehat{\text{ct}}_r$
14:   $K \| K_{\text{ske}} := \text{KDF}(\text{ss}_r \| \widehat{\text{ss}}_r, \text{content})$
15:   $\sigma := \text{SKE.Dec}(K_{\text{ske}}, \text{ct}_{\text{ske}})$ ▷ *Unmask signature*
16:   **require** $[\![\, \text{RS.Verify}(\{\text{rvk}_s, \text{rvk}\}, \text{content}, \sigma) = 1 \,]\!]$
17:   **if** $[\![\, t \neq \perp \,]\!]$ **then**
18:      $D_{\text{kem}}[t] \leftarrow \perp$ ▷ *Delete prekey bundle*
19:   $\text{st}_r \leftarrow (D_{\text{kem}}, \text{rvk}, D_{\rho_\perp})$
20:   **return** (K, st_r)

Table 5: Security comparison of BAKE protocols.

Protocol	Adversary	Forward Secrecy	User-State Compromise Impersonation	Protocol-specific adversary restrictions
X3DH	Classical	Sender: weak Receiver: full	Receiver vulnerable	No quantum/HNDL adversaries.
PQXDH	HNDL	Sender: weak Receiver: full	Receiver vulnerable	KEM secret can not be revealed to HNDL adversary.
RingXKEM	Quantum	Full	Secure	No RingXKEM specific restrictions.

post-quantum, and proving its security does not require ruling out additional unavoidable attacks: it is secure against user-state compromise impersonation attacks and fully forwards secure.

6.2. Efficiency

In this section, we will instantiate the protocols described above and show how they perform. We will focus on the bandwidth and storage requirements; an overview is given in Table 7. The bandwidth costs of setting up a Signal conversation affect the network transmission times; storage requirements directly impact the cost of operating the Signal central servers. We will also approximate the computation time required.

For an overview of the primitives mentioned below, see Table 6. The algorithms used for post-quantum KEM, Diffie–Hellman and elliptic curve signatures follow Signal; for the ring signature scheme we choose the recently proposed Gandalf signature scheme [GJK24]. Gandalf only has parameters at NIST security level I, but we argue that authentication can afford more aggressive choices than confidentiality (for which Kyber-1024 gives NIST security level V), especially if Hybrid constructions are used.

Table 6: Primitives used to instantiate the BAKE protocols.

| | Algorithm | Sec. level | Size (bytes) | |
			pk	ct / sig
ECDH	X25519	Pre-Quantum	32	32
KEM	Kyber-1024	NIST V	1568	1568
Signature	XEd25519 [Per16]	Pre-Quantum	32	64
1-Ring Sig	Gandalf [GJK24]	NIST I	896	630
2-Ring Sig	Gandalf [GJK24]	NIST I	896	1236
	Tree size	Hash algorithm	|root|	|path|
Merkle Tree	L	SHA-256	32	$32 \lceil 1 + \log_2 L \rceil$
Merkle Tree	100	SHA-256	32	256

Table 7: Bandwidth and storage requirements (in bytes) of BAKE protocols. As in Signal, we use $L = 100$.

| Protocol | Identity public key | Prekey bundle | | Handshake message |
		Individual	L-key storage	
X3DH	32	128	3296	64
PQXDH	32	1696	166 496	1632
RingXKEM	2464	3350	158 326	4404
RingXKEM-noMT	2464	3094	220 696	4372

6.2.1. X3DH and PQXDH

The X3DH and PQXDH protocols, as deployed by Signal, use a single X25519 public key for both ECDH and signing. All prekey bundles contain a signed prekey: a 32-byte X25519 public key with 64-byte XEd25519 signature [Per16]. The one-time prekey bundles contain an additional 32-byte X25519 public key. This amounts to a 128 bytes download for the sender. PQXDH has an additional signed Kyber-1024 KEM prekey that is part of every prekey bundle for HNDL security. This adds 1536 bytes and a 64-byte XEd25519 signature.

The X3DH handshake message generated by the sender contains an ephemeral 32-byte X25519 public key, and a 32-byte confirmation tag. PQXDH senders include a 1536 byte ciphertext.

The computational overhead of adding KEM operations to X3DH is negligible; benchmarks of the Kyber-1024 reference implementation on ARM Cortex-A72 (as a stand in for a mobile CPU) show that the median time for decapsulation (the most expensive operation) is only 83 µs slower than X25519 computations.[9]

6.2.2. RingXKEM

The RingXKEM protocol uses a KEM encapsulation key and a ring signature verification key in its identity public key. Kyber-1024 encapsulation keys are 1536 bytes, while Gandalf verification keys are 896 bytes and the signatures are $606n + 24$ bytes, where n is the size of the ring. Prekey bundles always have the same size in RingXKEM, and consist of another KEM and ring signature key. During the generation of prekey bundles, a Merkle tree is constructed from the KEM encapsulation keys. Its root is signed using the identity key's ring signature key, which results in a 630 byte signature. To authenticate the KEM encapsulation key, a sender needs to also download a 256-byte Merkle tree path; the root of the tree can be reconstructed from the path and the KEM encapsulation key. Together, the download size is 2422 bytes per prekey bundle. Server-side storage requirements scale with KEM encapsulation key size, as the ring signature verification key is shared between all prekey bundles, there is only one signed Merkle tree root, and the server can re-compute the paths in the Merkle tree on-demand. The handshake message consists of two KEM ciphertexts, a symmetrically encrypted 2-ring signature of 1236 bytes, and a confirmation tag. Assuming no overhead from encryption, the message is 4340 bytes.

The Merkle tree approach saves a significant amount of data on the server, at the cost of a small increase in download size per prekey bundle. For comparison, Table 7 row RingXKEM-noMT shows a variant of RingXKEM

that signs each KEM prekey instead of using a Merkle tree. If the server wishes to avoid the computation costs of reconstructing the Merkle Tree, paths to each KEM encapsulation key can be stored at time of upload at the additional cost of 256 bytes per prekey bundle; this is still more efficient than using a 630-byte Gandalf signature per prekey bundle. Note that for PQXDH, the savings are much less pronounced, as the signature on the KEM that is replaced by the Merkle tree approach is only 64 bytes (and storage savings is thus only $64(L - 1)$ bytes).

We expect computational performance of RingXKEM to be competitive with PQXDH. As above, the KEM operations are not noticeably slower than comparable ECDH operations. Though Gandalf does not report concrete performance numbers, they write that signing (the most expensive operation, by far) is linear in the size of the ring and expect that it should be faster than the comparable Falcon signature scheme [Pre+22]. On ARM Cortex-A72, Falcon needs 1 ms to sign.[9] Assuming that the two-ring Gandalf signature takes twice as long to compute, this is still much less than typical network latency. Finally, computing the Merkle Tree uses only hash operations. On the same chip, which runs at 1.5 GHz, hashing a Kyber-1024 public key using SHA-256 takes 18325 cycles.[9] Computing the full Merkle Tree requires $\lceil L \log_2 L \rceil$ hash operations, so for $L = 100$ this should take approximately 9 ms.

Acknowledgments

We would like to thank all anonymous reviewers who helped improve our paper. We also thank Daniel Collins for his helpful input during the initial phase of this project and Rolfe Schmidt for helping us understand the implementation and requirements of Signal. This paper is based on results obtained from a project, JPNP24003, commissioned by the Japanese New Energy and Industrial Technology Development Organization (NEDO).

Ethic Considerations

In this work, we set out to analyze the security of implemented and deployed cryptographic protocols. The security of Signal's handshake protocol is relied on by very large numbers of users, which makes better understanding the security of Signal's handshake protocol and proposals for new security protocols with better security guarantees highly relevant. We based our analysis on publicly available documentation and open-source implementations of Signal's protocols.

Risks and Risk Mitigation. As part of analyzing the security of Signal's X3DH and PQXDH protocols, it was a possibility that we might find new security flaws that could be used in real-world attacks on users of Signal. As prior work has thoroughly investigated the security of both of these protocols, we deemed this risk exceedingly unlikely. During the development of this work, we were in constant discussion with Signal's developers; if any significant issues had been found, we would have coordinated with them on how to best protect Signal's users, both of the Signal app itself, and other users of Signal including Facebook Messenger, WhatsApp, and others. Although we found that certain features of the deployed Signal handshake protocol made analysis more difficult, and Signal have indicated that they will be making changes in response to our findings, these changes only increase the robustness of the protocol and do not affect security or privacy of Signal's users or other applications that use the X3DH or PQXDH protocols. If we would have had findings that affected the security of users, we would have followed standard responsible disclosure practices with suitable embargo periods before disclosure.

Benefits. Signal is in the process of a transition towards full post-quantum security. We aim to contribute to this discussion by providing new models and results that can help developers using Signal's handshake protocol evaluate how to proceed with this transition. We view that these benefits are well worth the (in our view, negligible) risks. The work was done while in open communication with Signal developers. They have received and reviewed our findings before submission.

---

[9]Based on `supercop-20240425` [BL] results for hostname `pi4b` (latest measurements: DH, KEM, sign, hash).

Open Science

The formalization and security model for Bundled AKE protocols and the RingXKEM protocol that we developed are documented in this paper. We do not have any other artifacts (e.g., datasets, scripts, or binaries) related to this paper. We have shared our results with Signal developers, and they are considering changes to their implementations in response to our results.

References

[App24]   Apple Security Engineering and Architecture. *iMessage with PQ3: The new state of the art in quantum-secure messaging at scale*. Feb. 21, 2024. URL: https://security.apple.com/blog/imessage-pq3/ (visited on 08/27/2024) (cit. on p. 5).

[Beg+24]   Hugo Beguinet, Céline Chevalier, Thomas Ricosset, and Hugo Senet. "Formal Verification of a Post-quantum Signal Protocol with Tamarin." In: *Verification and Evaluation of Computer and Communication Systems*. Ed. by Belgacem Ben Hedia, Yassine Maleh, and Moez Krichen. Springer, 2024, pp. 105–121. DOI: 10.1007/978-3-031-49737-7_8. URL: https://hal.science/hal-04361766/document (cit. on p. 3).

[Bha+23]   Karthikeyan Bhargavan, Charlie Jacomme, Franziskus Kiefer, and Rolfe Schmidt. *An Analysis of Signal's PQXDH*. Cryspen Blog. Oct. 20, 2023. URL: https://cryspen.com/post/pqxdh/ (visited on 08/27/2024) (cit. on p. 13).

[Bha+24]   Karthikeyan Bhargavan, Charlie Jacomme, Franziskus Kiefer, and Rolfe Schmidt. "Formal verification of the PQXDH Post-Quantum key agreement protocol for end-to-end secure messaging." In: *USENIX Security 2024*. Ed. by Davide Balzarotti and Wenyuan Xu. USENIX Association, Aug. 2024 (cit. on pp. 3–5, 13).

[BJM97]   Simon Blake-Wilson, Don Johnson, and Alfred Menezes. "Key Agreement Protocols and Their Security Analysis." In: *6th IMA International Conference on Cryptography and Coding*. Ed. by Michael Darnell. Vol. 1355. LNCS. Springer, Berlin, Heidelberg, Dec. 1997, pp. 30–45. DOI: 10.1007/bfb0024447 (cit. on p. 16).

[BKP20]   Ward Beullens, Shuichi Katsumata, and Federico Pintore. "Calamari and Falafl: Logarithmic (Linkable) Ring Signatures from Isogenies and Lattices." In: *ASIACRYPT 2020, Part II*. Ed. by Shiho Moriai and Huaxiong Wang. Vol. 12492. LNCS. Springer, Cham, Dec. 2020, pp. 464–492. DOI: 10.1007/978-3-030-64834-3_16 (cit. on pp. 33, 34).

[BL]   Daniel J. Bernstein and Tanja Lange. *eBACS: ECRYPT Benchmarking of Cryptographic Systems*. URL: https://bench.cr.yp.to (visited on 08/19/2024) (cit. on p. 26).

[BLS24]   David Basin, Felix Linker, and Ralf Sasse. *A Formal Analysis of the iMessage PQ3 Messaging Protocol*. Technical Report. Feb. 2024. URL: https://security.apple.com/assets/files/A_Formal_Analysis_of_the_iMessage_PQ3_Messaging_Protocol_Basin_et_al.pdf (cit. on p. 5).

[BM99]   Simon Blake-Wilson and Alfred Menezes. "Unknown Key-Share Attacks on the Station-to-Station (STS) Protocol." In: *PKC'99*. Ed. by Hideki Imai and Yuliang Zheng. Vol. 1560. LNCS. Springer, Berlin, Heidelberg, Mar. 1999, pp. 154–170. DOI: 10.1007/3-540-49162-7_12 (cit. on p. 12).

[BR06]   Mihir Bellare and Phillip Rogaway. "The Security of Triple Encryption and a Framework for Code-Based Game-Playing Proofs." In: *EUROCRYPT 2006*. Ed. by Serge Vaudenay. Vol. 4004. LNCS. Springer, Berlin, Heidelberg, May 2006, pp. 409–426. DOI: 10.1007/11761679_25 (cit. on p. 45).

[Bre+17]   Jacqueline Brendel, Marc Fischlin, Felix Günther, and Christian Janson. "PRF-ODH: Relations, Instantiations, and Impossibility Results." In: *CRYPTO 2017, Part III*. Ed. by Jonathan Katz and Hovav Shacham. Vol. 10403. LNCS. Springer, Cham, Aug. 2017, pp. 651–681. DOI: 10.1007/978-3-319-63697-9_22 (cit. on p. 34).

[Bre+20] Jacqueline Brendel, Marc Fischlin, Felix Günther, Christian Janson, and Douglas Stebila. "Towards Post-Quantum Security for Signal's X3DH Handshake." In: *SAC 2020*. Ed. by Orr Dunkelman, Michael J. Jacobson Jr., and Colin O'Flynn. Vol. 12804. LNCS. Springer, Cham, Oct. 2020, pp. 404–430. DOI: 10.1007/978-3-030-81652-0_16 (cit. on pp. 3, 5).

[Bre+22] Jacqueline Brendel, Rune Fiedler, Felix Günther, Christian Janson, and Douglas Stebila. "Post-quantum Asynchronous Deniable Key Exchange and the Signal Handshake." In: *PKC 2022, Part II*. Ed. by Goichiro Hanaoka, Junji Shikata, and Yohei Watanabe. Vol. 13178. LNCS. Springer, Cham, Mar. 2022, pp. 3–34. DOI: 10.1007/978-3-030-97131-1_1 (cit. on pp. 3–5, 7, 22, 32).

[Brz+24] Chris Brzuska, Cas Cremers, Håkon Jacobsen, Douglas Stebila, and Bogdan Warinschi. *Falsifiability, Composability, and Comparability of Game-based Security Models for Key Exchange Protocols*. Cryptology ePrint Archive, Report 2024/1215. 2024. URL: https://eprint.iacr.org/2024/1215 (cit. on p. 12).

[CD23] Wouter Castryck and Thomas Decru. "An Efficient Key Recovery Attack on SIDH." In: *EURO-CRYPT 2023, Part V*. Ed. by Carmit Hazay and Martijn Stam. Vol. 14008. LNCS. Springer, Cham, Apr. 2023, pp. 423–447. DOI: 10.1007/978-3-031-30589-4_15 (cit. on p. 5).

[CF12] Cas J. F. Cremers and Michele Feltz. "Beyond eCK: Perfect Forward Secrecy under Actor Compromise and Ephemeral-Key Reveal." In: *ESORICS 2012*. Ed. by Sara Foresti, Moti Yung, and Fabio Martinelli. Vol. 7459. LNCS. Springer, Berlin, Heidelberg, Sept. 2012, pp. 734–751. DOI: 10.1007/978-3-642-33167-1_42 (cit. on pp. 10, 21).

[CF15] Cas Cremers and Michèle Feltz. "Beyond eCK: perfect forward secrecy under actor compromise and ephemeral-key reveal." In: *DCC* 74.1 (2015), pp. 183–218. DOI: 10.1007/s10623-013-9852-1 (cit. on pp. 10, 21).

[CK01] Ran Canetti and Hugo Krawczyk. "Analysis of Key-Exchange Protocols and Their Use for Building Secure Channels." In: *EUROCRYPT 2001*. Ed. by Birgit Pfitzmann. Vol. 2045. LNCS. Springer, Berlin, Heidelberg, May 2001, pp. 453–474. DOI: 10.1007/3-540-44987-6_28 (cit. on p. 16).

[Coh+17] Katriel Cohn-Gordon, Cas Cremers, Benjamin Dowling, Luke Garratt, and Douglas Stebila. "A Formal Security Analysis of the Signal Messaging Protocol." In: *2017 IEEE European Symposium on Security and Privacy*. IEEE Computer Society Press, Apr. 2017, pp. 451–466. DOI: 10.1109/EuroSP.2017.27 (cit. on pp. 3, 5, 19, 22).

[Coh+20] Katriel Cohn-Gordon, Cas Cremers, Benjamin Dowling, Luke Garratt, and Douglas Stebila. "A Formal Security Analysis of the Signal Messaging Protocol." In: *Journal of Cryptology* 33.4 (Oct. 2020), pp. 1914–1983. DOI: 10.1007/s00145-020-09360-1 (cit. on pp. 3, 5, 22).

[Col+24] Daniel Collins, Loïs Huguenin-Dumittan, Ngoc Khanh Nguyen, Nicolas Rolin, and Serge Vaudenay. "K-Waay: Fast and Deniable Post-Quantum X3DH without Ring Signatures." In: *USENIX Security 2024*. Ed. by Davide Balzarotti and Wenyuan Xu. USENIX Association, Aug. 2024 (cit. on pp. 3, 5, 7, 22).

[Cra+07] Ronald Cramer, Goichiro Hanaoka, Dennis Hofheinz, Hideki Imai, Eike Kiltz, Rafael Pass, abhi shelat, and Vinod Vaikuntanathan. "Bounded CCA2-Secure Encryption." In: *ASIACRYPT 2007*. Ed. by Kaoru Kurosawa. Vol. 4833. LNCS. Springer, Berlin, Heidelberg, Dec. 2007, pp. 502–518. DOI: 10.1007/978-3-540-76900-2_31 (cit. on pp. 43, 51).

[DG22] Samuel Dobson and Steven D. Galbraith. "Post-Quantum Signal Key Agreement from SIDH." In: *Post-Quantum Cryptography - 13th International Workshop, PQCrypto 2022*. Ed. by Jung Hee Cheon and Thomas Johansson. Springer, Cham, Sept. 2022, pp. 422–450. DOI: 10.1007/978-3-031-17234-2_20 (cit. on pp. 3, 5).

[DOW92] Whitfield Diffie, Paul C. van Oorschot, and Michael J. Wiener. "Authentication and Authenticated Key Exchanges." In: *DCC* 2.2 (1992), pp. 107–125. DOI: 10.1007/BF00124891 (cit. on p. 16).

[FG24]   Rune Fiedler and Felix Günther. *Security Analysis of Signal's PQXDH Handshake*. Cryptology ePrint Archive, Report 2024/702. 2024. URL: https://eprint.iacr.org/2024/702 (cit. on pp. 3–5, 13, 22).

[Fuj+12]   Atsushi Fujioka, Koutarou Suzuki, Keita Xagawa, and Kazuki Yoneyama. "Strongly Secure Authenticated Key Exchange from Factoring, Codes, and Lattices." In: *PKC 2012*. Ed. by Marc Fischlin, Johannes Buchmann, and Mark Manulis. Vol. 7293. LNCS. Springer, Berlin, Heidelberg, May 2012, pp. 467–484. DOI: 10.1007/978-3-642-30057-8_28 (cit. on p. 16).

[GJK24]   Phillip Gajland, Jonas Janneck, and Eike Kiltz. "Ring Signatures for Deniable AKEM: Gandalf's Fellowship." In: *CRYPTO 2024, Part I*. Ed. by Leonid Reyzin and Douglas Stebila. Vol. 14920. LNCS. Springer, Cham, Aug. 2024, pp. 305–338. DOI: 10.1007/978-3-031-68376-3_10 (cit. on pp. 4, 24, 25).

[Han+21]   Shuai Han, Tibor Jager, Eike Kiltz, Shengli Liu, Jiaxin Pan, Doreen Riepel, and Sven Schäge. "Authenticated Key Exchange and Signatures with Tight Security in the Standard Model." In: *CRYPTO 2021, Part IV*. Ed. by Tal Malkin and Chris Peikert. Vol. 12828. LNCS. Virtual Event: Springer, Cham, Aug. 2021, pp. 670–700. DOI: 10.1007/978-3-030-84259-8_23 (cit. on p. 16).

[Has+21]   Keitaro Hashimoto, Shuichi Katsumata, Kris Kwiatkowski, and Thomas Prest. "An Efficient and Generic Construction for Signal's Handshake (X3DH): Post-Quantum, State Leakage Secure, and Deniable." In: *PKC 2021, Part II*. Ed. by Juan Garay. Vol. 12711. LNCS. Springer, Cham, May 2021, pp. 410–440. DOI: 10.1007/978-3-030-75248-4_15 (cit. on pp. 3–5, 16, 22, 32).

[Has+22]   Keitaro Hashimoto, Shuichi Katsumata, Kris Kwiatkowski, and Thomas Prest. "An Efficient and Generic Construction for Signal's Handshake (X3DH): Post-quantum, State Leakage Secure, and Deniable." In: *Journal of Cryptology* 35.3 (July 2022), p. 17. DOI: 10.1007/s00145-022-09427-1 (cit. on pp. 3–5, 7, 16, 22, 32).

[Höv+20]   Kathrin Hövelmanns, Eike Kiltz, Sven Schäge, and Dominique Unruh. "Generic Authenticated Key Exchange in the Quantum Random Oracle Model." In: *PKC 2020, Part II*. Ed. by Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas. Vol. 12111. LNCS. Springer, Cham, May 2020, pp. 389–422. DOI: 10.1007/978-3-030-45388-6_14 (cit. on p. 16).

[Jag+12]   Tibor Jager, Florian Kohlar, Sven Schäge, and Jörg Schwenk. "On the Security of TLS-DHE in the Standard Model." In: *CRYPTO 2012*. Ed. by Reihaneh Safavi-Naini and Ran Canetti. Vol. 7417. LNCS. Springer, Berlin, Heidelberg, Aug. 2012, pp. 273–293. DOI: 10.1007/978-3-642-32009-5_17 (cit. on p. 34).

[Jag+21]   Tibor Jager, Eike Kiltz, Doreen Riepel, and Sven Schäge. "Tightly-Secure Authenticated Key Exchange, Revisited." In: *EUROCRYPT 2021, Part I*. Ed. by Anne Canteaut and François-Xavier Standaert. Vol. 12696. LNCS. Springer, Cham, Oct. 2021, pp. 117–146. DOI: 10.1007/978-3-030-77870-5_5 (cit. on pp. 7, 10, 16, 21).

[KBB17]   Nadim Kobeissi, Karthikeyan Bhargavan, and Bruno Blanchet. "Automated Verification for Secure Messaging Protocols and Their Implementations: A Symbolic and Computational Approach." In: *2017 IEEE European Symposium on Security and Privacy*. IEEE Computer Society Press, Apr. 2017, pp. 435–450. DOI: 10.1109/EuroSP.2017.38 (cit. on pp. 3–5, 13, 20).

[KP05]   Caroline Kudla and Kenneth G. Paterson. "Modular Security Proofs for Key Agreement Protocols." In: *ASIACRYPT 2005*. Ed. by Bimal K. Roy. Vol. 3788. LNCS. Springer, Berlin, Heidelberg, Dec. 2005, pp. 549–565. DOI: 10.1007/11593447_30 (cit. on pp. 3, 18).

[Kra05]   Hugo Krawczyk. "HMQV: A High-Performance Secure Diffie-Hellman Protocol." In: *CRYPTO 2005*. Ed. by Victor Shoup. Vol. 3621. LNCS. Springer, Berlin, Heidelberg, Aug. 2005, pp. 546–566. DOI: 10.1007/11535218_33 (cit. on p. 16).

[Kra10]   Hugo Krawczyk. "Cryptographic Extraction and Key Derivation: The HKDF Scheme." In: *CRYPTO 2010*. Ed. by Tal Rabin. Vol. 6223. LNCS. Springer, Berlin, Heidelberg, Aug. 2010, pp. 631–648. DOI: 10.1007/978-3-642-14623-7_34 (cit. on p. 34).

[KS23]   Ehren Kret and Rolfe Schmidt. *The PQXDH Key Agreement Protocol*. Protocol documentation. Oct. 18, 2023. URL: https://signal.org/docs/specifications/pqxdh/ (cit. on pp. 13, 17–20).

[LLM07]   Brian A. LaMacchia, Kristin Lauter, and Anton Mityagin. "Stronger Security of Authenticated Key Exchange." In: *ProvSec 2007*. Ed. by Willy Susilo, Joseph K. Liu, and Yi Mu. Vol. 4784. LNCS. Springer, Berlin, Heidelberg, Nov. 2007, pp. 1–16. DOI: 10.1007/978-3-540-75670-5_1 (cit. on p. 16).

[LS17]   Yong Li and Sven Schäge. "No-Match Attacks and Robust Partnering Definitions: Defining Trivial Attacks for Security Protocols is Not Trivial." In: *ACM CCS 2017*. Ed. by Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu. ACM Press, Oct. 2017, pp. 1343–1360. DOI: 10.1145/3133956.3134006 (cit. on p. 10).

[Mai+23]   Luciano Maino, Chloe Martindale, Lorenz Panny, Giacomo Pope, and Benjamin Wesolowski. "A Direct Key Recovery Attack on SIDH." In: *EUROCRYPT 2023, Part V*. Ed. by Carmit Hazay and Martijn Stam. Vol. 14008. LNCS. Springer, Cham, Apr. 2023, pp. 448–471. DOI: 10.1007/978-3-031-30589-4_16 (cit. on p. 5).

[Mer87]   Ralph C. Merkle. "A Digital Signature Based on a Conventional Encryption Function." In: *CRYPTO'87*. Ed. by Carl Pomerance. Vol. 293. LNCS. Springer, Berlin, Heidelberg, Aug. 1987, pp. 369–378. DOI: 10.1007/3-540-48184-2_32 (cit. on p. 33).

[Met23]   Meta, Inc. *Messenger End-to-End Encryption Overview*. Technical white paper. Dec. 6, 2023. URL: https://engineering.fb.com/wp-content/uploads/2023/12/MessengerEnd-to-EndEncryptionOverview_12-6-2023.pdf (cit. on p. 3).

[MP16]   Moxie Marlinspike and Trevor Perrin. *The X3DH Key Agreement Protocol*. Protocol documentation. Nov. 4, 2016. URL: https://signal.org/docs/specifications/x3dh/ (cit. on pp. 3, 17, 18, 35, 36).

[Per16]   Trevor Perrin. *The XEdDSA and VXEdDSA Signature Schemes*. documentation. Oct. 20, 2016. URL: https://signal.org/docs/specifications/xeddsa/ (cit. on pp. 20, 25).

[PM16]   Trevor Perrin and Moxie Marlinspike. *The Double Ratchet Algorithm*. Protocol documentation. Nov. 20, 2016. URL: https://signal.org/docs/specifications/doubleratchet/ (cit. on p. 3).

[PQR21]   Jiaxin Pan, Chen Qian, and Magnus Ringerud. "Signed Diffie-Hellman Key Exchange with Tight Security." In: *CT-RSA 2021*. Ed. by Kenneth G. Paterson. Vol. 12704. LNCS. Springer, Cham, May 2021, pp. 201–226. DOI: 10.1007/978-3-030-75539-3_9 (cit. on p. 16).

[Pre+22]   Thomas Prest, Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Ricosset, Gregor Seiler, William Whyte, and Zhenfei Zhang. *FALCON*. Tech. rep. National Institute of Standards and Technology, 2022. URL: https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022 (cit. on p. 26).

[PRZ24]   Jiaxin Pan, Doreen Riepel, and Runzhi Zeng. "Key Exchange with Tight (Full) Forward Secrecy via Key Confirmation." In: *EUROCRYPT 2024, Part VII*. Ed. by Marc Joye and Gregor Leander. Vol. 14657. LNCS. Springer, Cham, May 2024, pp. 59–89. DOI: 10.1007/978-3-031-58754-2_3 (cit. on pp. 10, 21).

[PWZ23]   Jiaxin Pan, Benedikt Wagner, and Runzhi Zeng. "Lattice-Based Authenticated Key Exchange with Tight Security." In: *CRYPTO 2023, Part V*. Ed. by Helena Handschuh and Anna Lysyanskaya. Vol. 14085. LNCS. Springer, Cham, Aug. 2023, pp. 616–647. DOI: 10.1007/978-3-031-38554-4_20 (cit. on pp. 10, 21).

[Res18]   Eric Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.3*. RFC 8446. Aug. 2018. DOI: 10.17487/RFC8446 (cit. on p. 21).

[Sch24]   Rolfe Schmidt. "Private communications." 2024 (cit. on pp. 18, 20, 35).

[Sig]   Signal foundation. *libsignal*. URL: https://github.com/signalapp/libsignal (cit. on pp. 20, 35, 36).

[Ste24]   Douglas Stebila. *Security analysis of the iMessage PQ3 protocol*. Cryptology ePrint Archive, Report 2024/357. 2024. URL: https://eprint.iacr.org/2024/357 (cit. on p. 5).

[Wha23]   WhatsApp. *WhatsApp Encryption Overview*. Technical white paper. Sept. 27, 2023. URL: https://www.whatsapp.com/security/WhatsApp-Security-Whitepaper.pdf (cit. on p. 3).

A. Preliminaries

In this section, we will describe standard cryptographic notions and definitions.

A.1. Notation

Let $\mathbb{N}$ denote the natural numbers and $\mathbb{Z}$ denote the integers. $\mathbb{N}^*$ denotes the positive natural numbers, and $\mathbb{Z}_N$ denotes the integers modulo N. If P is a point on an elliptic curve, we denote multiplication of P by scalar k as $[k]P$. Let λ be the security parameter. For $n \in \mathbb{N}$, we define $[n] = \{1, 2, \ldots, n-1, n\} \subset \mathbb{N}$.

A.2. Key Derivation Functions

We recall the definition of a (multi-key) key derivation function KDF. To satisfy the below properties, we can use for instance HKDF where the hash function is instantiated by SHA-384. By modeling SHA-384 as a (quantum) random oracle, the following properties hold even against quantum adversaries.

Definition 11 (Pseudorandom). Let $\mathcal{K}$ and $\mathcal{X}$ denote the key and output spaces, respectively and let $\ell = \mathsf{poly}(\lambda)$. We say a (multi-key) key derivation function $\mathsf{KDF} : \mathcal{K}^\ell \times \{0,1\}^* \to \mathcal{X}$ is *pseudorandom* if for any efficient adversary $\mathcal{A}$ and all $i = 1, \ldots, \ell$, we have

$$\left| \Pr\left[1 \xleftarrow{\$} \mathcal{A}^{O_0(\cdot)}(1^\lambda) : s_i \xleftarrow{\$} \mathcal{K}_i\right] - \Pr\left[1 \xleftarrow{\$} \mathcal{A}^{O_1(\cdot)}(1^\lambda)\right] \right| = \mathsf{negl}(\lambda),$$

where O_0 and O_1 take the keys s_j for $j \in 1, \ldots, \ell$ but excluding $j = i$ and the input label m as input, and O_0 returns $\mathsf{KDF}(s_1, \ldots, s_\ell, m)$ and O_1 returns $\mathsf{RF}(m)$ for a random function $\mathsf{RF} : \{0,1\}^* \to \mathcal{X}$.

Note that when $\mathcal{A}$ is a quantum adversary, we assume it can access the KDF in superposition.

Definition 12 (Collision Resistance). Let $\ell = \mathsf{poly}(\lambda)$. Let $\mathsf{KDF} : \mathcal{K}^\ell \times \{0,1\}^* \to \mathcal{X}$ be a key derivation function defined as in Definition 11. We say a KDF is *collision resistant* if for any efficient adversary $\mathcal{A}$, we have

$$\Pr\left[\begin{array}{c} \mathsf{KDF}(s_1, \ldots, s_\ell, m) = \mathsf{KDF}(s_1', \ldots, s_\ell', m') \\ \wedge (s_1, \ldots, s_\ell, m) \neq (s_1', \ldots, s_\ell', m') : \\ (s_1, \ldots, s_\ell, s_1', \ldots, s_\ell', m, m') \xleftarrow{\$} \mathcal{A}(1^\lambda) \end{array} \right]$$
$$= \mathsf{negl}(\lambda).$$

A.3. Signature Schemes

We recall the definition of a signature scheme.

Definition 13 (Signature Schemes). A signature scheme with message space $\mathcal{M}$ consists of the following PPT algorithms (Sig.KeyGen, Sig.Sign, Sig.Verify):

Sig.KeyGen $(1^\lambda) \xrightarrow{\$} (\mathsf{vk}, \mathsf{sk})$: Takes the security parameter 1^λ as input and outputs a pair of keys $(\mathsf{vk}, \mathsf{sk})$.

Sig.Sign $(\mathsf{sk}, \mathsf{M}) \xrightarrow{\$} \sigma$: Takes a signing key sk and a message $\mathsf{M} \in \mathcal{M}$ as input and outputs a signature σ.

Sig.Verify $(\mathsf{vk}, \mathsf{M}, \sigma) \to 1/0$: Takes a verification key vk, a message M and a signature σ as input and outputs a bit.

Definition 14 (Correctness). We say a signature scheme is correct if for all messages $\mathsf{M} \in \mathcal{M}$, we have Sig.Verify $(\mathsf{vk}, \mathsf{M}, \sigma) = 1$, where we generate $(\mathsf{vk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{Sig.KeyGen}(1^\lambda)$ and $\sigma \xleftarrow{\$} \mathsf{Sig.Sign}(\mathsf{sk}, \mathsf{M})$.

Definition 15 (EUF-CMA Security)**.** We define the advantage of adversary $\mathcal{A}$ against the EUF-CMA security game as follows:

$$\mathsf{Adv}_{\mathsf{SIG},\mathcal{A}}^{\mathsf{EUF\text{-}CMA}}(\lambda) := \Pr \left[\begin{array}{c} \mathsf{Sig}.\mathsf{Verify}\,(\mathsf{vk}, \mathsf{M}^*, \sigma^*) = 1 \\ \wedge\ \mathsf{M}^* \notin Q : \\ (\mathsf{vk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{Sig}.\mathsf{KeyGen}\,(1^\lambda); \\ (\mathsf{M}^*, \sigma^*) \xleftarrow{\$} \mathcal{A}^{O_{\mathsf{Sign}}(\mathsf{sk},\cdot)}(\mathsf{vk}) \end{array} \right]$$

where O_{Sign} is the signing oracle and Q is the set of messages that $\mathcal{A}$ submitted to the signing oracle. We say a signature scheme is EUF-CMA secure if $\mathsf{Adv}_{\mathsf{SIG},\mathcal{A}}^{\mathsf{EUF\text{-}CMA}}(\lambda) = \mathsf{negl}(\lambda)$ for any efficient $\mathcal{A}$.

A.4. Ring Signatures

We recall the definition of ring signatures. We note that we omit anonymity, a standard security requirement for ring signatures. This is typically used to show *deniability* of the bundled AKE protocol (cf. [Bre+22; Has+21; Has+22]), a property which we do not focus in this work.

Definition 16 (Ring Signatures)**.** A ring signature scheme with a message space $\mathcal{M}$ consists of three PPT algorithms (RS.KeyGen, RS.Sign, RS.Verify):

RS.KeyGen(1^λ) $\xrightarrow{\$}$ (rvk, rsk) : It takes the security parameter 1^λ as input and outputs a pair of keys (rvk, rsk).

RS.Sign(rsk, M, RL) $\xrightarrow{\$}$ σ : It takes a secret key rsk, a message $\mathsf{M} \in \mathcal{M}$, and a list of verification keys, i.e., a *ring*, $\mathsf{RL} = \{\mathsf{rvk}_1, \dots, \mathsf{rvk}_N\}$ as inputs and outputs signature σ.

RS.Verify(RL, M, σ) $\rightarrow$ 1/0 : It takes a ring of keys $\mathsf{RL} = \{\mathsf{rvk}_1, \dots, \mathsf{rvk}_N\}$, a message M, and a signature σ as inputs and outputs a bit.

Definition 17 (Correctness)**.** We say a ring signature is correct if for $N = \mathsf{poly}(\lambda)$, $j \in [N]$, and message $\mathsf{M} \in \mathcal{M}$, we have RS.Verify(RL, M, σ) = 1, where $(\mathsf{rvk}_i, \mathsf{rsk}_i) \xleftarrow{\$} \mathsf{RS}.\mathsf{KeyGen}(1^\lambda)$ for all $i \in [N]$, $\mathsf{RL} := (\mathsf{rvk}_1, \cdots, \mathsf{rvk}_N)$, and $\sigma \xleftarrow{\$} \mathsf{RS}.\mathsf{Sign}(\mathsf{rsk}_j, \mathsf{M}, \mathsf{RL})$.

Definition 18 (Unforgeability)**.** We define the unforgeability game between an adversary and a challenger as follows:

1. The challenger generates $(\mathsf{rvk}_i, \mathsf{rsk}_i) \xleftarrow{\$} \mathsf{RS}.\mathsf{KeyGen}(1^\lambda)$ for all $i \in [N]$ and sets $\mathsf{VK} := \{\,\mathsf{rvk}_i\,\}_{i \in [N]}$, and initializes two empty sets Q_{sign} and Q_{cor}. It then provides VK to $\mathcal{A}$.

2. $\mathcal{A}$ can arbitrarily make polynomially many signing and corruption queries:
   - (sign, i, M, RL): The challenger checks if key $\mathsf{rvk}_i \in \mathsf{RL}$. If so, it creates signature $\sigma \xleftarrow{\$} \mathsf{RS}.\mathsf{Sign}(\mathsf{rsk}_i, \mathsf{M}, \mathsf{RL})$, returns σ to $\mathcal{A}$, and adds $(i, \mathsf{M}, \mathsf{RL})$ to Q_{sign};
   - (corrupt, i): The challenger adds rvk_i to Q_{cor} and returns rsk_i to $\mathcal{A}$.

3. $\mathcal{A}$ outputs $(\mathsf{RL}^*, \mathsf{M}^*, \sigma^*)$. If $\mathsf{RL}^* \subset \mathsf{VK} \backslash Q_{\mathsf{cor}}$, as well as $(\cdot, \mathsf{M}^*, \mathsf{RL}^*) \notin Q_{\mathsf{sign}}$, and RS.Verify($\mathsf{RL}^*, \mathsf{M}^*, \sigma^*$) = 1, then we say the adversary $\mathcal{A}$ wins.

The advantage of $\mathcal{A}$ is defined as $\mathsf{Adv}_{\mathsf{RS},\mathcal{A}}^{\mathsf{UNF}}(1^\lambda) = \Pr[\mathcal{A}\ \mathsf{wins}]$. We say a ring signature is *unforgeable* if $\mathsf{Adv}_{\mathsf{RS},\mathcal{A}}^{\mathsf{UNF}}(\lambda) = \mathsf{negl}(\lambda)$ for all $N = \mathsf{poly}(\lambda)$ and any efficient adversary $\mathcal{A}$.

A.5. Key Encapsulation Mechanisms

We recall the definition of a key encapsulation mechanism KEM. We define both IND-CPA and IND-CCA security.

Definition 19 (KEM)**.** A key encapsulation mechanism KEM scheme with session key space $\mathcal{K}$ consists of three PPT algorithms (KEM.KeyGen, KEM.Encaps, KEM.Decaps):

KEM.KeyGen (1^λ) $\xrightarrow{\$}$ (ek, dk)**:** It takes the security parameter 1^λ as input and outputs a pair of keys (ek, dk).

KEM.Encaps $(\mathsf{ek}) \xrightarrow{\$} (\mathsf{ss}, \mathsf{ct})$**:** It takes an encapsulation key ek as input and outputs a session key $\mathsf{ss} \in \mathcal{K}$ and a ciphertext ct.

KEM.Decaps $(\mathsf{dk}, \mathsf{ct}) \rightarrow \mathsf{ss}$**:** It takes a decapsulation key dk and a ciphertext ct as input and outputs a session key $\mathsf{ss} \in \mathcal{K}$.

Definition 20 (Correctness)**.** We say a KEM is correct if we have

$$
\Pr \left[\begin{array}{c} \mathsf{KEM.Decaps}\,(\mathsf{dk}, \mathsf{ct}) = \mathsf{ss} : \\ (\mathsf{ek}, \mathsf{dk}) \xleftarrow{\$} \mathsf{KEM.KeyGen}(1^\lambda); \\ (\mathsf{ss}, \mathsf{ct}) \xleftarrow{\$} \mathsf{KEM.Encaps}\,(\mathsf{ek}) \end{array} \right]
$$
$$
= 1 - \mathsf{negl}(\lambda).
$$

Definition 21 (IND-CCA Security)**.** We define the advantage of $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ against the IND-CCA security game as follows:

$$
\mathsf{Adv}_{\mathsf{KEM}, \mathcal{A}}^{\mathsf{IND\text{-}CCA}}(\lambda) :=
$$
$$
\left| \Pr \left[\begin{array}{c} b = b' : \\ (\mathsf{ek}^*, \mathsf{dk}^*) \xleftarrow{\$} \mathsf{KEM.KeyGen}\,(1^\lambda); \\ \mathsf{state} \xleftarrow{\$} \mathcal{A}_1^{O_{\mathsf{Decaps}}(\mathsf{dk}^*, \cdot)}(\mathsf{ek}^*); \\ b \xleftarrow{\$} \{0, 1\}; \\ (\mathsf{ss}_0^*, \mathsf{ct}_0^*) \xleftarrow{\$} \mathsf{KEM.Encaps}\,(\mathsf{ek}^*); \\ \mathsf{ss}_1^* \xleftarrow{\$} \mathcal{K}; \\ b' \xleftarrow{\$} \mathcal{A}_2^{O_{\mathsf{Decaps}}(\mathsf{dk}^*, \cdot)}(\mathsf{ek}^*, (\mathsf{ss}_b^*, \mathsf{ct}_0^*), \mathsf{state}) \end{array} \right] - \frac{1}{2} \right|.
$$

$\mathcal{A}_2$ is not allowed to query the challenge ciphertext ct_0^* to O_{Decaps}. We say KEM is IND-CCA secure if $\mathsf{Adv}_{\mathsf{KEM}, \mathcal{A}}^{\mathsf{IND\text{-}CCA}}(\lambda) = \mathsf{negl}(\lambda)$ for any efficient adversary $\mathcal{A}$.

The following two notions are natural properties that most KEM will have. It states that when using proper randomness, the min-entropy of an encapsulation key and ciphertext are high. Below, note that $\omega(f)$ is an asymptotic notation capturing functions that grow strictly faster than f.

Definition 22 (Min-Entropy of KEM Encapsulation Key)**.** We say that encapsulation keys of KEM have high min-entropy if for all $\lambda \in \mathbb{N}$,

$$
\omega(\log(\lambda)) \leq -\log_2 \left(\max_{\mathsf{ek}^*} \Pr \left[\begin{array}{c} \mathsf{ek} = \mathsf{ek}^* : \\ (\mathsf{ek}, \mathsf{dk}) \leftarrow \mathsf{KEM.KeyGen}(1^\lambda) \end{array} \right] \right).
$$

Definition 23 (Min-Entropy of KEM Ciphertext)**.** We say ciphertexts of KEM have high min-entropy if for all $\lambda \in \mathbb{N}$,

$$
\omega(\log(\lambda)) \leq -\log_2 \left(\mathbb{E} \left[\max_{\mathsf{ct}^*} \Pr \left[\begin{array}{c} \mathsf{ct} = \mathsf{ct}^* : \\ (\mathsf{ss}, \mathsf{ct}) \leftarrow \mathsf{KEM.Encaps}(\mathsf{ek}) \end{array} \right] \right] \right),
$$

where the expectation is taken over the randomness used to sample $(\mathsf{ek}, \mathsf{dk}) \leftarrow \mathsf{KEM.KeyGen}(1^\lambda)$.

A.6. Merkle Trees

Merkle trees [Mer87] allow hashing a list of elements $A = (a_0, \cdots, a_N)$ into one hash value root. At a later point, one can efficiently prove to a third party that an element a_i was included in the list A using the hash digest. In the following, we rely on a specific Merkle tree construction based on [BKP20] allowing to prove that a single element a_i was included in the tree without revealing its position in the list. Looking ahead, this allows one to hide the index of the prekey bundle being used in RingXKEM, that is, to hide how many times the prekeys has been used.

Definition 24. A Merkle tree consists of PPT algorithms (MerkleTree, getMerklePath, ReconstructRoot) with access to a hash function $\mathcal{H}_{\mathsf{Coll}} : \{0, 1\}^\star \rightarrow \{0, 1\}^{2\lambda}$.

MerkleTree(A) → (root, tree): On input a list of (at most) 2^k elements $A = (a_1, \cdots, a_{2^k})$, with $k \in \mathbb{N}$, it constructs a binary tree of height k with $\{ \ell_i = \mathcal{H}_{\mathsf{Coll}}(a_i) \}_{i \in [2^k]}$ as its leaf nodes, and where every internal node h with children h_{left} and h_{right} equals the hash digest of $(h_{\mathsf{left}}, h_{\mathsf{right}})_{\mathsf{lex}}$, where the subscript lex indicates the lexicographical order (or any other total order on binary strings).[10] It then outputs the root root of the Merkle tree, as well as a description of the entire tree tree.

getMerklePath(tree, i) → path: On input the description of a Merkle tree tree and an index $i \in [2^k]$, it outputs the list path, which contains the sibling of ℓ_i, as well as the sibling of any ancestor of ℓ_i, ordered by decreasing height.

ReconstructRoot(a, path) → root: On input an element a in the list of elements $A = (a_1, \cdots, a_{2^k})$ and path = $(n_1, \cdots, n_k)$, it outputs a reconstructed root root$' = h_k$, which is calculated by putting $h_0 = \mathcal{H}_{\mathsf{Coll}}(a)$ and defining h_i for $i \in [k]$ recursively as $h_i = \mathcal{H}_{\mathsf{Coll}}((h_{i-1}, n_i)_{\mathsf{lex}})$.

If the hash function $\mathcal{H}_{\mathsf{Coll}}$ that is used in the Merkle tree is collision resistant, then we have that the Merkle tree construction is binding. Formally, we have the following.

Lemma 1 (Binding). *There is an efficient extractor algorithm that, given the description* tree *of a Merkle tree (having* root root *and constructed using the list of elements A) and $(b,$ path$)$ such that $b \notin A$ and* ReconstructRoot(b, path) = root, *outputs a collision for the hash function $\mathcal{H}_{\mathsf{Coll}}$.*

Lastly, the use of the lexicographical order to concatenate two children nodes in the Merkle tree construction implies that the output path of the algorithm getMerklePath hides the index $i \in [N]$ given as input. As we do not formally use this in our work, we refer to [BKP20, Lemma 2.10] for more details.

A.7. Computational Assumptions

Lastly, we define the PRF-ODH problem for multi-key PRF. This is a natural extension of the PRF-ODH problem for single-key PRF [Bre+17; Jag+12]. Brendel et al. [Bre+17] proved that the PRF-ODH problem for single-key PRF is hard assuming that the Gap-CDH problem is hard and PRF is a programmable random oracle. The proof can be extended to the PRF-ODH problem for multi-key PRF, i.e., it is also hard under the same assumptions.

In the protocols discussed in this paper, we use a (multi-key) key derivation function (KDF) as PRF. More specifically, Signal's implementations use HKDF [Kra10] as the KDF. Brendel et al. showed that the PRF-ODH problem holds for HKDF, both as a PRF and a dual-PRF [Bre+17, Sec. 6.3].

Definition 25 (PRF-ODH problem for multi-key PRF). Let $\ell = \mathsf{poly}(\lambda)$ and i be an index such that $i \in [\ell]$. Let GenG be an algorithm that on input 1^λ, outputs a tuple $(\mathbb{G}, p, G)$, where G is a generator of cyclic group $\mathbb{G}$ of prime order p and let $\mathsf{PRF} : \mathcal{K}^\ell \times \{0,1\}^* \to \mathcal{X}$ be a pseudorandom function such that $\mathcal{K}_i = \mathbb{G}$. The advantage of an adversary $\mathcal{A}$ against the PRF-ODH problem w.r.t. the ith key of PRF is defined as:

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{PRF\text{-}ODH}}(\lambda) :=$$

$$\left| \Pr \left[\begin{array}{c} b = b' : \\[4pt] (\mathbb{G}, p, G) \xleftarrow{\$} \mathsf{GenG}(1^\lambda); \\ u, v \xleftarrow{\$} \mathbb{Z}_p; \\ \left(\begin{array}{c} \{ s_j^* \}_{j \in [\ell] \setminus \{i\}}, \\ m^*, \mathsf{state} \end{array} \right) \xleftarrow{\$} \mathcal{A}_1^{O_u, O_v} \left(\begin{array}{c} \mathbb{G}, p, G, \\ [u]G, [v]G \end{array} \right); \\ b \xleftarrow{\$} \{0, 1\}; \\ x_0 \leftarrow \mathsf{PRF} \left(\begin{array}{c} s_1^*, \ldots, s_{i-1}^*, [uv]G, \\ s_{i+1}^*, \ldots, s_\ell^*, m^* \end{array} \right); \\ x_1 \xleftarrow{\$} \mathcal{X}; \\ b' \xleftarrow{\$} \mathcal{A}_2^{O_u, O_v}(x_b, \mathsf{state}) \end{array} \right] - \frac{1}{2} \right|,$$

---

[10]While it is standard to consider the concatenation $h_{\mathsf{left}} || h_{\mathsf{right}}$, this slight modification allows to show index hiding (cf. [BKP20, Lemma 2.10]).

where $O_u(W, \{ s_j \}_{j \in [\ell] \setminus \{ i \}}, m)$ returns

$$\mathsf{PRF}(s_1, \ldots, s_{i-1}, [u]W, s_{i+1}, \ldots, s_\ell, m)$$

and $O_v(W, \{ s_j \}_{j \in [\ell] \setminus \{ i \}}, m)$ returns

$$\mathsf{PRF}(s_1, \ldots, s_{i-1}, [v]W, s_{i+1}, \ldots, s_\ell, m).$$

$\mathcal{A}$ is not allowed to query $O_u([v]G, \{ s_j^* \}_{j \in [\ell] \setminus \{ i \}}, m^*)$ and $O_v([u]G, \{ s_j^* \}_{j \in [\ell] \setminus \{ i \}}, m^*)$, which output x_0. We say the PRF-ODH problem w.r.t. the ith key of PRF is hard if the advantage is negligible for any efficient classical adversary $\mathcal{A}$.

B. Different "Versions" of X3DH and PQXDH

As discussed in Section 4.1, we have added a confirmation tag τ_{conf} derived from the KDF to allow for a more modular analysis of the X3DH and PQXDH Signal handshake protocols. In Signal's protocol documentation [MP16] and implementation [Sig], the KDF takes no input. Instead, the protocol documentation states that X3DH.Send takes an additional message as input. This message is then encrypted using some unspecified AEAD algorithm. At a high level, our confirmation tag τ_{conf} can be viewed as a one-time MAC, serving a similar purpose as the AEAD. In the implementation, this message is actually sent using the Double Ratchet algorithm, which is initialized using the key K that is the output of the X3DH handshake. The Double Ratchet AEAD encryption is instantiated by AES in CBC mode paired with HMAC-SHA256.

We show the differences between the implementation, the X3DH documentation, and our specified version of both X3DH.Send and X3DH.Receive in Algorithm 10. The documentation's handling of the message input is marked like this. We show (a simplified version of) how `libsignal` implements X3DH marked thusly. We remove the processing of an application-layer message and the AEAD, and generate a confirmation tag, which are marked as such. Removing the AEAD that reuses the key K that is also an output of the protocol (as is specified in the documentation) simplifies the analysis; but it is also closer to the real implementation's use of the Double Ratchet key schedule in some aspects. Removing the AEAD and simply adding a confirmation tag also makes the X3DH protocol more modular and its output keys easier to use in subsequent protocols that are not Double Ratchet. In response, Signal developers have indicated that they are considering ways to increase the separation between the Double Ratchet and the handshake protocols, and will investigate adding our findings [Sch24].

C. Security of X3DH and PQXDH

In this section, we provide the formal proofs for the security of X3DH and PQXDH. Since X3DH shares most of its protocol design with PQXDH, we mainly focus on the security of PQXDH and explain how X3DH differs. Recall the only difference between X3DH and PQXDH is that PQXDH includes an extra post-quantum KEM key in the prekey bundles to provide security against HNDL adversaries.

C.1. Security Proofs of PQXDH

The correctness of PQXDH follows from construction. Below, we show match soundness and key indistinguishability of PQXDH. To this end, we first provide a helper lemma and then prove match soundness in Appendix C.1.2 and key indistinguishability in Appendix C.1.3.

C.1.1. Helper Lemma

To prove the match soundness and key indistinguishability of PQXDH, we consider the security game that uses the following partnering predicate $\mathsf{Partner}_{\mathsf{PQXDH}}$ instead of the predicate $\mathsf{Partner}$ in Definition 5. Here content[iID] is

Algorithm 10 Key confirmation and the (not) sending of a message in X3DH.Send and X3DH.Receive as in Algorithms 5 and 6 vs documentation [MP16] vs the implementation [Sig].

1: **function** X3DH.Send(isk$_s$, ik$_r$, prek$_r$, msg, msg)
2:    (ik$_r$, vk$_r$) ← ik$_r$
3:    vk$_r$:= ik$_r$ / vk$_r$:= ik$_r$
4:    (spk$_r$, σ_{spk_r}, opk$_r$) ← prek$_r$ ▷ opk$_r$ = ⊥ if prek *is a last-resort prekey bundle.*
5:    **require** ⟦Sig.Verify(vk$_r$, spk$_r$, σ_{spk_r}) = 1⟧
6:    esk $\overset{\$}{\leftarrow}$ $\mathbb{Z}_p$; epk := [esk]G
7:    ss$_1$:= [isk$_s$]spk$_r$
8:    ss$_2$:= [esk]ik$_r$
9:    ss$_3$:= [esk]spk$_r$
10:    ss := ss$_1$∥ss$_2$∥ss$_3$
11:    **if** ⟦opk$_r$ ≠ ⊥⟧ **then** ▷ *One-time key bundle*
12:      ss$_4$:= [esk]opk$_r$
13:      ss := ss$_1$∥ss$_2$∥ss$_3$∥ss$_4$
14:    content := ik$_s$∥ik$_r$∥prek$_r$∥epk
15:    K∥τ_{conf} := KDF(ss, content)
16:    ρ := (epk, τ_{conf})
17:    enc $\overset{\$}{\leftarrow}$ AEAD.Enc(K, msg, ik$_s$∥ik$_r$)
18:    ρ := (epk, enc)
19:    ▷ *Initialize Double Ratchet* ◁
20:    sk $\overset{\$}{\leftarrow}$ $\mathbb{Z}_p$; pk := [sk]G ▷ *Additional ephemeral key*
21:    ctr := 1 ▷ *Message counters*
22:    ▷ *We simplify the Double Ratchet key schedule here.* ◁
23:    K_{send}∥K_{recv} := KDF(K∥[sk]spk$_r$∥ctr)
24:    ▷ *AES-CBC encrypt with key derived from* send *chain* ◁
25:    enc ← Enc(KDF(K_{send}∥'enc'), msg)
26:    ctr′ := ctr; ctr ← ctr + 1
27:    ρ' := (pk, ctr, ctr′, enc)
28:    ▷ *Compute MAC with key derived from* send *chain* ◁
29:    τ ← MAC.Tag(KDF(K_{send}∥'mac'), (ik$_r$∥ik$_s$∥ρ'))
30:    K_{send} ← KDF(K_{send}∥ctr) ▷ *Simplified ratcheting*
31:    ρ := (epk, ρ', τ)
32:    **return** (K, K, (K_{send}, K_{recv}, ctr), ρ)

33: **function** X3DH.Receive(isk$_r$, st$_r$, ik$_s$, t, ρ)
34:    (D_{prek}, $D_{\rho_\perp}$) ← st$_r$
35:    **if** ⟦t ≠ ⊥⟧ **then** ▷ *One-time prekey bundle*
36:      **require** ⟦$D_{\mathsf{prek}}[t]$ ≠ ⊥⟧ ▷ *Check if unused.*
37:      (prek$_{r,t}$, (spksec$_r$, osk$_{r,t}$)) ← $D_{\mathsf{prek}}[t]$
38:    **else** ▷ *Last-resort prekey bundle (i.e., t = ⊥)*
39:      **require** ⟦$\rho \notin D_{\rho_\perp}$⟧ ▷ *Check ρ is not replayed.*
40:      $D_{\rho_\perp}$ ← $D_{\rho_\perp} \cup \{\rho\}$
41:      (prek$_{r,t}$, (spksec$_r$, ⊥)) ← $D_{\mathsf{prek}}[t]$
42:    (epk, τ_{conf}) ← ρ
43:    (epk, enc) ← ρ
44:    (epk, pk, ctr, ctr′, enc, τ) ← ρ
45:    ss$_1$:= [spksec$_r$]ik$_s$
46:    ss$_2$:= [isk$_r$]epk
47:    ss$_3$:= [spksec$_r$]epk
48:    ss := ss$_1$∥ss$_2$∥ss$_3$
49:    **if** ⟦t ≠ ⊥⟧ **then**
50:      ss$_4$:= [osk$_{r,t}$]epk
51:      ss := ss$_1$∥ss$_2$∥ss$_3$∥ss$_4$
52:    content := ik$_s$∥ik$_r$∥prek$_{r,t}$∥epk
53:    K∥τ'_{conf} := KDF(ss, content)
54:    **require** ⟦τ_{conf} = τ'_{conf}⟧
55:    ▷ *Verify and decrypt* enc ◁
56:    **require** msg := AEAD.Dec(K, enc)
57:    ▷ *Initialize Double Ratchet* ◁
58:    ▷ *We simplify the key schedule here. Note K_{recv}∥K_{send} is swapped compared to* Line 23. ◁
59:    K_{recv}∥K_{send} := KDF(K∥[spksec$_r$]pk∥ctr′)
60:    ▷ *Decrypt with key derived from* receive *chain* ◁
61:    msg′ ← Dec(KDF(K_{recv}∥'enc'), enc)
62:    **require** MAC.Verify(KDF(K_{recv}∥'mac'), τ, (ik$_r$∥ik$_s$∥ρ'))
63:    K_{recv} ← KDF(K_{recv}∥ctr) ▷ *Simplified ratcheting*
64:    ▷ *Delete prekey bundle if not last-resort* ◁
65:    **if** ⟦t ≠ ⊥⟧ **then** $D_{\mathsf{prek}}[t]$ ← ⊥
66:    st$_r$ ← (D_{prek}, $D_{\rho_\perp}$)
67:    **return** (K, K, msg, (K'_{recv}, K_{send}, ctr), msg, st$_r$)

the content, defined in the protocol description, used by iID, which is the input of KDF. Note that content can be determined from public information (i.e., identity public keys, prekey bundles and handshake messages).

$$\mathsf{Partner}_{\mathsf{PQXDH}}(\mathsf{iID}, \mathsf{iID}')$$
$$\Longleftrightarrow$$
$$[\![\,\mathsf{Sender}[\mathsf{iID}] = \mathsf{Sender}[\mathsf{iID}']\,]\!]$$
$$\wedge \quad [\![\,\mathsf{Receiver}[\mathsf{iID}] = \mathsf{Receiver}[\mathsf{iID}']\,]\!]$$
$$\wedge \quad [\![\,\mathsf{role}[\mathsf{iID}] \neq \mathsf{role}[\mathsf{iID}']\,]\!]$$
$$\wedge \quad [\![\,\mathsf{content}[\mathsf{iID}] = \mathsf{content}[\mathsf{iID}']\,]\!]$$
$$\wedge \quad [\![\,\mathsf{key}[\mathsf{iID}] \neq \epsilon\,]\!]$$
$$\wedge \quad [\![\,\mathsf{key}[\mathsf{iID}'] \neq \epsilon\,]\!].$$

The predicate $\mathsf{Partner}_{\mathsf{PQXDH}}$ checks if the exchanged messages in PQXDH match instead of equality of the derived session key. We can show that using $\mathsf{Partner}_{\mathsf{PQXDH}}$ instead of $\mathsf{Partner}$ is indistinguishable to the adversary up to negligible error.

Lemma 2. *Assume* KDF *is collision resistant and* KEM *is correct. For* mode $\in \{\,\mathsf{KIND}, \mathsf{MATCH}\,\}$, *the game* $\mathsf{Game}^{\mathsf{mode}}_{\mathsf{PQXDH}, \mathcal{A}}$ *using* $\mathsf{Partner}$ *are indistinguishable from the same game using* $\mathsf{Partner}_{\mathsf{PQXDH}}$.

Proof. We will show that equivalence $\mathsf{Partner}(\mathsf{iID}, \mathsf{iID}') \Leftrightarrow \mathsf{Partner}_{\mathsf{PQXDH}}(\mathsf{iID}, \mathsf{iID}')$ holds except with negligible errors.

$\mathsf{Partner}(\mathsf{iID}, \mathsf{iID}') \Rightarrow \mathsf{Partner}_{\mathsf{PQXDH}}(\mathsf{iID}, \mathsf{iID}')$**:** Since sender instances always compute a session key, $\mathsf{key}[\mathsf{iID}] = \mathsf{key}[\mathsf{iID}']$ means that the receiver also computes a session key. Thus, having $\mathsf{key}[\mathsf{iID}] = \mathsf{key}[\mathsf{iID}']$ implies that the output of the KDF was equal given its inputs: $\mathsf{KDF}(\mathsf{ss}\|\mathsf{ss}_{\mathsf{KEM}}, \mathsf{content}) = \mathsf{KDF}(\mathsf{ss}'\|\mathsf{ss}'_{\mathsf{KEM}}, \mathsf{content})$. Since we assume KDF is collision resistant, $(\mathsf{ss}\|\mathsf{ss}_{\mathsf{KEM}}, \mathsf{content}) = (\mathsf{ss}'\|\mathsf{ss}'_{\mathsf{KEM}}, \mathsf{content}')$, that is, $\mathsf{content}[\mathsf{iID}] = \mathsf{content}[\mathsf{iID}']$ (and of course $\mathsf{key}[\mathsf{iID}] \neq \epsilon$ and $\mathsf{key}[\mathsf{iID}'] \neq \epsilon$).

$\mathsf{Partner}_{\mathsf{PQXDH}}(\mathsf{iID}, \mathsf{iID}') \Rightarrow \mathsf{Partner}(\mathsf{iID}, \mathsf{iID}')$**:** Having that $\mathsf{Partner}_{\mathsf{PQXDH}}(\mathsf{iID}, \mathsf{iID}') = \mathsf{true}$ implies that $\mathsf{content}[\mathsf{iID}] = \mathsf{content}[\mathsf{iID}']$ and $\mathsf{key}[\mathsf{iID}] \neq \epsilon \wedge \mathsf{key}[\mathsf{iID}'] \neq \epsilon$. Due to the latter condition, the receiver instance (iID or iID') accepts the received confirmation tag and computes its session key following the protocol description. In addition, since $\mathsf{content}[\mathsf{iID}] = \mathsf{content}[\mathsf{iID}']$ holds, both the sender and the receiver share $(\mathsf{ik}_s, \mathsf{ik}_r, \mathsf{ek}_r, \mathsf{ct}_r, \mathsf{spk}_r, \mathsf{opk}_r)$. Thus, the deterministic nature of DH computations and the correctness of the KEM ensures that $\mathsf{ss}\|\mathsf{ss}_{\mathsf{KEM}} = \mathsf{ss}'\|\mathsf{ss}'_{\mathsf{KEM}}$. Summarizing, we conclude that we must have $\mathsf{key}[\mathsf{iID}] = \mathsf{key}[\mathsf{iID}']$. □

This lemma allows us to check whether two instances are partner or not from the exchanged messages.

C.1.2. Match Soundness of PQXDH

We provide the full proof of PQXDH's match soundness. For reference, we restate Theorem 1 more formally below.

Theorem 3 (Match Soundness of PQXDH)**.** *Assume the* KDF *is collision resistant against quantum adversaries and* KEM *encapsulation keys and ciphertexts have high min-entropy. Then,* PQXDH *is match sound against a harvest-now-decrypt-later adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *with respect to the predicate* Match *(cf. Definition 6) and the origin function* $\Phi^{\mathsf{Signal}}_{\mathsf{origin}}$ *(cf. Definition 10).*

Proof. Thanks to Lemma 2, it is sufficient that we prove the match soundness in $\mathsf{Game}^{\mathsf{MATCH}}_{\mathsf{PQXDH}, \mathcal{A}}$ w.r.t. $\mathsf{Partner}_{\mathsf{PQXDH}}$. We will show that each condition in the Match predicate is satisfied for any $\mathsf{iID}, \mathsf{iID}', \mathsf{iID}'' \in \mathcal{S}_{\mathsf{iID}}$.

1. If $\mathsf{Partner}_{\mathsf{PQXDH}}(\mathsf{iID}, \mathsf{iID}') = \mathsf{true}$, we then have

    a) $\mathsf{Receiver}[\mathsf{iID}] = \mathsf{Receiver}[\mathsf{iID}']$,

    b) $\mathsf{role}[\mathsf{iID}] \neq \mathsf{role}[\mathsf{iID}']$, and

    c) $\mathsf{content}[\mathsf{iID}] = \mathsf{content}[\mathsf{iID}']$.

Item 1b implies one of (iID, iID′) is the sender and the other is the receiver. Item 1c implies that by content being equal, prek[iID] = prek[iID′]. Together, Items 1a to 1c concludes Origin(iID, iID′) = true or Origin(iID′, iID) = true.

2. If $\mathsf{Partner}_{\mathsf{PQXDH}}(\mathsf{iID}, \mathsf{iID}') = \mathsf{Partner}_{\mathsf{PQXDH}}(\mathsf{iID}, \mathsf{iID}'') = \mathsf{true}$, then we have

$$
\begin{aligned}
& \mathsf{Sender}[\mathsf{iID}'] = \mathsf{Sender}[\mathsf{iID}''] \\
\wedge\ & \mathsf{Receiver}[\mathsf{iID}'] = \mathsf{Receiver}[\mathsf{iID}''] \\
\wedge\ & \mathsf{role}[\mathsf{iID}'] = \mathsf{role}[\mathsf{iID}''] \\
\wedge\ & \mathsf{content}[\mathsf{iID}'] = \mathsf{content}[\mathsf{iID}''].
\end{aligned}
$$

Thus, if content is unique during the game, it implies iID = iID′. We show this holds in the following:

- Case 1: role[iID] = receiver and role[iID′] = role[iID″] = sender: Notice that iID′ and iID″ use independent randomness esk to generate ephemeral public key epk. Since esk is chosen uniformly at random from $\mathbb{Z}_p$, iID′ and iID″ generate different epk = [esk]G with overwhelming probability. Since epk is a part of the content by definition, if content[iID′] = content[iID″], then iID′ = iID″ with overwhelming probability.

- Case 2: role[iID] = sender and role[iID′] = role[iID″] = receiver: Notice that the game assigns a different iID ∈ $\mathbb{N} \times \{0, \bot\}$ to each one-time/last-resort prekey bundle. Since prekey bundles are generated from fresh and independent randomness, the probability the two different iIDs are assigned to the same $(\mathsf{spk}_r, \mathsf{opk}_r)$ is negligible. Thus, iID′ is the only instance that is assigned to $(\mathsf{spk}_r, \mathsf{opk}_r)$. If iID′ ∈ $\mathbb{N} \times \{0\}$, such an iID is assigned one-time prekey bundle. Thus, content[iID′] = content[iID″] implies iID′ = iID″. Otherwise, if iID′ ∈ $\mathbb{N} \times \mathbb{N}$, iID′ and iID″ uses the same last-resort prekey bundle. In this case, the confirmation tag is unique for each (epk, ct). Thus, each iID ∈ $\mathbb{N} \times \mathbb{N}$ uses a different (epk, ct) and a different content if it accepts a session key. Therefore, in this case, if content[iID′] = content[iID″], then iID′ = iID″.

3. If Origin(iID, iID′) = Origin(iID, iID″) = true, then we have

$$
\begin{aligned}
& \mathsf{Sender}[\mathsf{iID}'] = \mathsf{Sender}[\mathsf{iID}''] \\
\wedge\ & \mathsf{Receiver}[\mathsf{iID}'] = \mathsf{Receiver}[\mathsf{iID}''] \\
\wedge\ & \mathsf{role}[\mathsf{iID}'] = \mathsf{role}[\mathsf{iID}''] = \mathsf{receiver} \\
\wedge\ & \Phi_{\mathsf{origin}}^{\mathsf{PQXDH}}(\mathsf{iID}') = \Phi_{\mathsf{origin}}^{\mathsf{PQXDH}}(\mathsf{iID}'').
\end{aligned}
$$

Since prekey bundles are generated from fresh and independent randomness, the probability the same signed prekey and one-time prekey is generated via $O_{\mathsf{PubNewPrekeyBundle}}$ queries is negligible. Furthermore, prekey bundle is assigned to a unique iID ∈ $\mathbb{N} \times \{0, \bot\}$ by definition of the game. Thus, if $\Phi_{\mathsf{origin}}^{\mathsf{PQXDH}}(\mathsf{iID}') = \Phi_{\mathsf{origin}}^{\mathsf{PQXDH}}(\mathsf{iID}'')$, then base(iID′) = base(iID″). Moreover, the following hold:

(a) One-time prekey bundles are assigned to iID′ ∈ $\mathbb{N} \times \{0\}$ by definition of the game. PQXDH uses one-time prekey bundles only once (cf. Line 8 of PQXDH . Receive). Therefore, iID′ is used only once. Due to the uniqueness of one-time prekeys, explained above, we have iID′ = iID″.

(b) Last-resort prekey bundles are assigned to some iID$_\bot$ ∈ $\mathbb{N} \times \{\bot\}$ when they are generated. Then, each time the same last-resort prekey is used, the game assigns $(\mathsf{base}(\mathsf{iID}_\bot), i)$ for $i \in [\mathsf{prekreuse}[\mathsf{iID}_\bot]]$. Therefore, if iID′ ∈ $\mathbb{N} \times (\{\bot\} \cup \mathbb{N})$, then there exists a unique instance iID$_\bot$ = (base(iID′), ⊥) ∈ $\mathcal{S}_{\mathsf{iID}}$, and we have

$$
\begin{aligned}
& \left\{ \mathsf{iID}'' \ \middle| \ \begin{array}{c} \mathsf{Origin}(\mathsf{iID}, \mathsf{iID}'') = \mathsf{true} \\ \wedge\ \mathsf{iID}'' \neq \mathsf{iID}_\bot \end{array} \right\} \\
& = \{(\mathsf{base}(\mathsf{iID}_\bot), i)\}_{i \in [\mathsf{prekreuse}[\mathsf{iID}_\bot]]}.
\end{aligned}
$$

$\square$

C.1.3. Key Indistinguishability of PQXDH

We provide the full proof of PQXDH's key indistinguishability. For reference, we restate Theorem 2 more formally below.

Theorem 4 (Key Indistinguishability of PQXDH). *Assume the* PRF-ODH *problem w.r.t.* KDF *is hard against classical adversaries and* KDF *is pseudorandom and the* KEM *is* IND-CCA *secure against quantum adversaries. Then,* PQXDH *is key indistinguishable against a harvest-now-decrypt-later adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *with respect to the predicate* $\mathsf{safe}_{\mathsf{PQXDH}}$ *(cf. Definition 9).*

Proof. Let $\mathcal{A}$ be an adversary that plays the security game $\mathsf{Game}_{\mathcal{A}}^{\mathsf{KIND}}(1^\lambda)$ and let $\epsilon = \mathsf{Adv}_{\mathcal{A}}^{\mathsf{KIND}}(\lambda)$ be its advantage. In order to prove the theorem, we show that PQXDH is secure against all the attack strategies listed in Table 4.

For each strategy taken by $\mathcal{A}$, we construct an algorithm that breaks one of the underlying assumptions by using $\mathcal{A}$ as a subroutine. Let $N = |\mathcal{U}|$ be the number of users in the system and M be the upper bound of NumiID (i.e., the maximum number of base iID generated by $\mathcal{A}$). We construct algorithms $\mathcal{B}_1, \mathcal{B}_2, \mathcal{B}_3, \mathcal{B}_4$, and $\mathcal{B}_5$ satisfying the following:

1. If $\mathcal{A}$ follows the Type-{1-1, 1-2} strategy, then $\mathcal{B}_1$ succeeds in solving the PRF-ODH problem with advantage $\approx \frac{1}{NM}\epsilon$.

2. If $\mathcal{A}$ follows the Type-{2-1, 2-2} strategy, then $\mathcal{B}_2$ succeeds in breaking the IND-CCA security of KEM with advantage $\approx \frac{1}{M^2}\epsilon$.

3. If $\mathcal{A}$ follows the Type-{3-1, 3-2} strategy, then $\mathcal{B}_3$ succeeds in breaking the IND-CCA security of KEM with advantage $\approx \frac{1}{M^2}\epsilon$.

4. If $\mathcal{A}$ follows the Type-4 strategy, then $\mathcal{B}_4$ succeeds in solving the PRF-ODH problem with advantage $\approx \frac{1}{NM}\epsilon$.

5. If $\mathcal{A}$ follows the Type-5 strategy, then $\mathcal{B}_5$ succeeds in solving the PRF-ODH problem and correctly guessing the confirmation tag τ_{conf} with advantage $\approx \frac{1}{NM}\epsilon$.

We present a security proof structured as a sequence of games. Without loss of generality, we assume that $\mathcal{A}$ always issues O_{Test}-query. In the following, let

$$\epsilon_i := \left| \Pr\left[\mathsf{Game}_i(1^\lambda) = 1\right] - 1/2 \right|$$

denote the advantage of the adversary in Game_i. Regardless of the strategy taken by $\mathcal{A}$, all proofs share a common game sequence Game_0–Game_2 as described below.

Game_0. This game is identical to the original security game. We thus have

$$\epsilon_0 = \epsilon.$$

Game_1. This game uses $\mathsf{Partner}_{\mathsf{PQXDH}}$ instead of $\mathsf{Partner}$. Due to Lemma 2, we have

$$|\epsilon_0 - \epsilon_1| \leq \mathsf{negl}(\lambda).$$

Game_2. This game is identical to Game_1, except that it aborts if the predicate Match returns false. Since we proved that PQXDH is match-sound w.r.t. $\mathsf{Partner}_{\mathsf{PQXDH}}$, we have

$$|\epsilon_1 - \epsilon_2| \leq \mathsf{Adv}_{\mathsf{PQXDH}, \mathcal{A}}^{\mathsf{MATCH}}(\lambda).$$

We now divide the game sequence depending on the strategy taken by the adversary $\mathcal{A}$. Regardless of $\mathcal{A}$'s strategy, we prove that ϵ_2 is negligible, which in particular implies that ϵ_0 is also negligible. Formally, this is

shown in Lemmas 3 to 7 provided below. We first complete the proof of the theorem. Specifically, by combining all the lemmas together, we obtain the following bound:

$$\mathsf{Adv}^{\mathsf{KIND}}_{\mathsf{PQXDH},\mathcal{A}}(\lambda) \leq \mathsf{negl}(\lambda) + \mathsf{Adv}^{\mathsf{MATCH}}_{\mathsf{PQXDH},\mathcal{A}}(\lambda)$$

$$+ \left\{ \begin{array}{c} 2NM \cdot \mathsf{Adv}^{\mathsf{PRF\text{-}ODH}}_{\mathcal{B}_1}(\lambda) \\ 2M^2 \cdot \mathsf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathsf{KEM},\mathcal{B}_2}(\lambda) \\ 2M^2 \cdot \mathsf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathsf{KEM},\mathcal{B}_3}(\lambda) \\ 2NM \cdot \mathsf{Adv}^{\mathsf{PRF\text{-}ODH}}_{\mathcal{B}_4}(\lambda) \\ 2NM \cdot \mathsf{Adv}^{\mathsf{PRF\text{-}ODH}}_{\mathcal{B}_5}(\lambda) \end{array} \right\}$$

$\square$

It remains to prove Lemmas 3 to 7.

Security against Type-{1-1, 1-2} strategy.

Lemma 3. *For any classical adversary $\mathcal{A}$ following the Type-{1-1, 1-2} strategy, there exists a reduction $\mathcal{B}_1$ that solves the* PRF-ODH *problem w.r.t. the 2nd key of* KDF *such that*

$$\epsilon_2 \leq 2NM \cdot \mathsf{Adv}^{\mathsf{PRF\text{-}ODH}}_{\mathcal{B}_1}(\lambda) + \mathsf{negl}(\lambda).$$

Note that the Type-{1-1, 1-2} second-stage adversary $\mathcal{A}_2$ is not allowed to be quantum, as this would lead to an unavoidable attack.

Proof of Lemma 3. We present the rest of the sequence of games from game Game_3.

Game_3. This game guesses the tested instance and its peer. At the beginning of the game, it chooses an index i at random from $[M]$ and $\widetilde{u}$ at random from $\mathcal{U}$. Let Tested be the event that the index i is used for a sender instance $\widetilde{\mathsf{iID}}_s = (i, 0)$ such that $\mathsf{role}[\widetilde{\mathsf{iID}}_s] = \mathsf{sender}$ and the tested instance iID^* satisfies $\mathsf{Receiver}[\mathsf{iID}^*] = \widetilde{u}$ and either

- $\mathsf{iID}^* = \widetilde{\mathsf{iID}}_s$ if $\mathsf{role}[\mathsf{iID}^*] = \mathsf{sender}$, or

- $\mathsf{Partner}_{\mathsf{PQXDH}}(\widetilde{\mathsf{iID}}_s, \mathsf{iID}^*)$ if $\mathsf{role}[\mathsf{iID}^*] = \mathsf{receiver}$.

Since the Tested event can be efficiently checked, the game aborts as soon as it detects that Tested does not occur. The probability that the tested instance and its peer are correctly guessed is $1/(NM)$, so we have

$$\epsilon_3 = \frac{\epsilon_2}{NM}.$$

Game_4. In this game, we change how the user $\widetilde{u}$ processes received handshake messages from $\widetilde{\mathsf{iID}}_s$. When a receiver $\widetilde{u}$ is invoked on a sender's identity s and a handshake message, the game checks if

$$\mathsf{Sender}[\widetilde{\mathsf{iID}}_s] = s$$
$$\wedge \; \mathsf{Receiver}[\widetilde{\mathsf{iID}}_s] = \widetilde{u}$$
$$\wedge \; \mathsf{content}[\widetilde{\mathsf{iID}}_s] = \mathsf{content},$$

where $\mathsf{content} = \mathsf{ik}_s \|\mathsf{ik}_r\|\mathsf{prek}_r\|\mathsf{epk}\|\mathsf{ct}$, which can be determined by public information. If so, the game uses the KDF output $K\|\tau_{\mathsf{conf}}$ derived by $\widetilde{\mathsf{iID}}_s$ for this receiver instance, and use it to verify the confirmation tag. Because DH values are symmetric and by the correctness of KEM, we have

$$|\epsilon_3 - \epsilon_4| \leq \mathsf{negl}(\lambda).$$

40

Game$_5$. We change how $\widetilde{\mathsf{iID}}_s$ computes the session key. In this game, we sample a random function RF and $\widetilde{\mathsf{iID}}_s$ computes the session key as $\mathsf{RF}(\mathsf{content})$ instead of $\mathsf{KDF}(\mathsf{ss}_1\|\mathsf{ss}_2\|\mathsf{ss}_3\|\mathsf{ss}_4\|\hat{\mathsf{ss}}, \mathsf{content})$. We will show that this change is indistinguishable from $\mathcal{A}$ assuming the PRF-ODH problem is hard w.r.t. the 2nd key ss_2. To this end, we show an algorithm $\mathcal{B}_1$ that solves the PRF-ODH problem by using $\mathcal{A}$.

$\mathcal{B}_1$ receives two DH value $(U, V) = ([u]G, [v]G)$ and is given oracle access to O_u and O_v. Then, $\mathcal{B}_1$ simulates the security game as follows: $\mathcal{B}_1$ first generates the users' identity key pair. For the user $\widetilde{u}$, it sets U as the identity DH public key and generates its signature key pair. For the other users, $\mathcal{B}_1$ generates their DH key and signature key pair by itself. Then, $\mathcal{B}_1$ invokes $\mathcal{A}$ on input 1^λ and $\{\mathsf{ik}_u\}_{u\in\mathcal{U}}$, and answers the queries by $\mathcal{A}$ as follows:

- $O_{\mathsf{Send}}(s, r, \mathsf{prek})$: If this is the $\widetilde{\mathsf{iID}}_s$ sender instance, $\mathcal{B}_1$ answers as in Game$_4$ except that it sets $\mathsf{epk} \coloneqq [v]G$ and computes the keys $K\|\tau_{\mathsf{conf}}$ using the challenge phase of the PRF-ODH game; it outputs $(\mathsf{ss}_1\|\mathsf{ss}_3\|\mathsf{ss}_4\|\hat{\mathsf{ss}}, \mathsf{content})$ and receives $K\|\tau_{\mathsf{conf}}$, which is

$$x_0 = \mathsf{KDF}(\mathsf{ss}_1\|[uv]G\|\mathsf{ss}_3\|\mathsf{ss}_4\|\hat{\mathsf{ss}}, \mathsf{content})$$

  or $x_1 = \mathsf{RF}(\mathsf{content})$ depending on the PRF-ODH game's challenge bit d. Otherwise, $\mathcal{B}_1$ responds as in Game$_4$.

- $O_{\mathsf{Receive}}(r, s, \rho)$: If $r = \widetilde{u}$ and $\widetilde{\mathsf{iID}}_s$ satisfies

$$\begin{aligned} &\mathsf{Sender}[\widetilde{\mathsf{iID}}_s] = s \\ \wedge\ &\mathsf{Receiver}[\widetilde{\mathsf{iID}}_s] = r \\ \wedge\ &\mathsf{content}[\widetilde{\mathsf{iID}}_s] = \mathsf{content} \end{aligned}$$

  for $\mathsf{content} = \mathsf{ik}_s\|\mathsf{ik}_r\|\mathsf{spk}_r\|\mathsf{opk}_r\|\mathsf{ek}_r\|\mathsf{ct}$, the KDF output is determined as in the previous game. Else if $r = \widetilde{u}$, $\mathcal{B}_1$ answers as in Game$_4$ except that it evaluates KDF by using its O_u oracle; it sends $(\mathsf{epk}, \mathsf{ss}_1\|\mathsf{ss}_3\|\mathsf{ss}_4\|\hat{\mathsf{ss}}, \mathsf{content})$ and receives $K\|\tau_{\mathsf{conf}}$. Otherwise, if $r \neq \widetilde{u}$, $\mathcal{B}_1$ responds as in Game$_4$.

- The other oracles: $\mathcal{B}_1$ responds as in Game$_4$.

Finally, $\mathcal{A}$ outputs its guess b'. Let b be the challenge bit of the key indistinguishability game. If $\mathcal{A}$ breaks the key indistinguishability of PQXDH (i.e., $b = b'$), $\mathcal{B}_1$ outputs $d' = 0$ to the PRF-ODH game; otherwise it outputs 1.

We can see if the challenge bit of the PRF-ODH game is $d = 0$, $\mathcal{B}_1$ simulates Game$_4$; otherwise it simulates Game$_5$. In addition, we have

$$\epsilon_4 = \left| \Pr\left[b = b' \mid d = 0 \right] - \frac{1}{2} \right|$$

$$\epsilon_5 = \left| \Pr\left[b = b' \mid d = 1 \right] - \frac{1}{2} \right|.$$

Since $d' = 0$ if $b = b'$, we have

$$\begin{aligned} \mathsf{Adv}^{\mathsf{PRF\text{-}ODH}}_{\mathcal{B}_1}(\lambda) &= \left| \Pr\left[d = d' \right] - \frac{1}{2} \right| \\ &= \frac{1}{2} \left| \Pr\left[d' = 0 \mid d = 0 \right] - \Pr\left[d' = 0 \mid d = 1 \right] \right| \\ &= \frac{1}{2} \left| \Pr\left[b = b' \mid d = 0 \right] - \Pr\left[b = b' \mid d = 1 \right] \right| \\ &\geq \frac{1}{2} \left| \epsilon_4 - \epsilon_5 \right|. \end{aligned}$$

Therefore we have,

$$|\epsilon_4 - \epsilon_5| \leq 2\mathsf{Adv}^{\mathsf{PRF\text{-}ODH}}_{\mathcal{B}_1}(\lambda).$$

It remains to show that the session key output by the tested instance in Game$_5$ is uniformly random regardless of the challenge bit $b \in \{0, 1\}$ chosen by the game. We consider the case where $b = 0$ and prove that the session key

honestly generated by iID^* is distributed uniformly random. First, conditioning on the event Tested occurring, it must be the case that the tested instance (and its partner) computes the session key as $K^* \| \tau^*_{\text{conf}} \leftarrow \text{RF}(\text{content}^*)$ for some content^*. Next, by PQXDH satisfying Match soundness, the only instances that share the same content^* must be the tested instance (and its partner, who is unique by Match soundness). Therefore, we conclude that K^* is the session key of only the tested instance and its partner. Since the output of RF is distributed uniformly at random for different inputs, it must be that $\Pr[\text{Game}_5 = 1] = 1/2$, i.e., $\epsilon_5 = 0$. Combining all the arguments together, we obtain

$$\epsilon_2 \leq 2NM \cdot \text{Adv}^{\text{PRF-ODH}}_{\mathcal{B}_1}(\lambda) + \text{negl}(\lambda).$$

$\square$

Security against Type-{2-1, 2-2} strategy.

Lemma 4. *Assume that* KDF *is pseudorandom. For any HNDL adversary $\mathcal{A}$ following the Type-{2-1, 2-2} strategy, there exists a reduction $\mathcal{B}_2$ that breaks the* IND-CCA *security of* KEM *such that*

$$\epsilon_2 \leq 2M^2 \cdot \text{Adv}^{\text{IND-CCA}}_{\text{KEM}, \mathcal{B}_2}(\lambda) + \text{negl}(\lambda).$$

Proof of Lemma 4. We present the rest of the sequence of games from game Game_3.

Game_3. This game guesses the tested instance and its origin/partner instance. At the beginning of the game, it chose values i and j at random from $[M]$. Let Tested be the event that the index i is assigned for some sender instance $\widetilde{\text{iID}}_s$ such that $\widetilde{\text{iID}}_s = (i, 0)$, the index j is assigned for some receiver instance $\widetilde{\text{iID}}_r$ such that $\widetilde{\text{iID}}_r = (j, k)$ for $k \in \mathbb{N}$ and the tested instance iID^* satisfies either

- if $\text{role}[\text{iID}^*] = \text{sender}$, then $\text{iID}^* = \widetilde{\text{iID}}_s$ and we have $\text{Origin}(\widetilde{\text{iID}}_s, \widetilde{\text{iID}}_r) = \text{true}$, or

- if $\text{role}[\text{iID}^*] = \text{receiver}$, then $\text{iID}^* = \widetilde{\text{iID}}_r$ and we have $\text{Partner}_{\text{PQXDH}}(\widetilde{\text{iID}}_s, \widetilde{\text{iID}}_r) = \text{true}$.

Since event Tested can be efficiently checked, the game aborts as soon as it detects that event Tested does not occur. The probability that $\mathcal{A}$ guesses correctly is $1/M^2$, so we have

$$\epsilon_3 = \frac{\epsilon_2}{M^2}.$$

Game_4. In this game, we modify how $\widetilde{\text{iID}}_s$ and $\widetilde{\text{iID}}_r$ compute their session key. Let $\widetilde{\text{ek}}$ be the one-time KEM prekey assigned to $\widetilde{\text{iID}}_r$, and let $(\widetilde{\text{ss}}, \widetilde{\text{ct}}) \leftarrow \text{KEM.Encaps}(\widetilde{\text{ek}})$ be the KEM key-ciphertext pair generated by $\widetilde{\text{iID}}_s$ with $\widetilde{\text{ek}}$. (Due to the change we made in Game_3, $\widetilde{\text{iID}}_s$ and $\widetilde{\text{iID}}_r$ are origin/partner instance, so they share $\widetilde{\text{ek}}$.) Then, when $\widetilde{\text{iID}}_r$ is invoked on input $\rho = (\text{epk}, \text{ct}, \tau)$, it first checks if $\text{ct} = \widetilde{\text{ct}}$. If so, it uses the key $\widetilde{\text{ss}}$ that was generated by $\widetilde{\text{iID}}_s$ instead of decrypting $\widetilde{\text{ct}}$. Otherwise, if $\text{ct} \neq \widetilde{\text{ct}}$, then it proceeds exactly as in the previous game. Conditioning on decryption errors are not occurring on KEM, the two games Game_3 and Game_4 are identical. Hence,

$$|\epsilon_3 - \epsilon_4| \leq \text{negl}(\lambda).$$

Game_5. In this game, we modify the way the sender instance $\widetilde{\text{iID}}_s$ computes the handshake message and session key. When $\widetilde{\text{iID}}_s$ is invoked on input the prekey bundle $\widetilde{\text{prek}}_r$ assigned to $\widetilde{\text{iID}}_r$, it samples a random KEM secret $\widetilde{\text{ss}}$ instead of computing the real shared secret by $(\widetilde{\text{ss}}, \widetilde{\text{ct}}) \xleftarrow{\$} \text{KEM.Encaps}(\widetilde{\text{ek}})$. Note that due to the modification we made in the previous game, when $\widetilde{\text{iID}}_r$ receives $\widetilde{\text{ct}}$, it also uses $\widetilde{\text{ss}}$ generated by $\widetilde{\text{iID}}_s$. Since $\mathcal{A}$ follows the Type-2 strategy, the user state of the receiver r^* is leaked after r^* has erased the one-time KEM decryption key corresponding to $\widetilde{\text{ek}}$ from its state. Thus, $\mathcal{A}$ does not know the KEM decryption key. Due to the IND-CCA security of KEM, Game_4 and Game_5 are indistinguishable. Thus, we have

$$|\epsilon_4 - \epsilon_5| \leq 2\text{Adv}^{\text{IND-CCA}}_{\text{KEM}, \mathcal{B}_2}(\lambda).$$

We would like to note that why we assume IND-CCA security of KEM. When the adversary follows the Type-2-1 strategy, $\widetilde{\text{iID}}_s$ and $\widetilde{\text{iID}}_r$ may not be partners. That is, $\widetilde{\text{iID}}_r$, which has the challenge KEM encapsulation key embedded,

42

may decrypt a ciphertext that is not generated by $\widetilde{\mathsf{iID}}_s$, which has the embedded challenge KEM ciphertext. To correctly answer in such a situation, a decapsulation oracle is needed.[11]

Game_6. We change how $\widetilde{\mathsf{iID}}_s$ computes the session key. In this game, we sample a random function RF and $\widetilde{\mathsf{iID}}_s$ computes the session key using the random function as $\mathsf{RF}(\mathsf{content})$ instead of by $\mathsf{KDF}(\mathsf{ss}\|\widetilde{\mathsf{ss}}, \mathsf{content})$. Due to the modification we made in the previous game, $\widetilde{\mathsf{ss}}$ is chosen uniformly at random so the KDF key has sufficiently large min-entropy. Therefore, assuming the pseudorandomness of KDF, the two games are indistinguishable. Thus, we have

$$|\epsilon_5 - \epsilon_6| \leq \mathsf{negl}(\lambda).$$

It remains to show that the session key output by the tested instance in Game_6 is uniformly random regardless of the challenge bit $b \in \{0, 1\}$ chosen by the game. We consider the case where $b = 0$ and prove that the honestly generated session key by iID^* is distributed uniformly random. First, by event Tested occurring, it must be the case that the tested instance (and its partner) prepares the session key as $K^* \| \tau_{\mathsf{conf}}^* \leftarrow \mathsf{RF}(\mathsf{content}^*)$ for some $\mathsf{content}^*$. Next, by PQXDH satisfying Match soundness, the only instances that share the same $\mathsf{content}^*$ must be the tested instance and its partner. Therefore, we conclude that K^* is the session key of only the tested instance and its partner. Since the output of RF is distributed uniformly at random for different inputs, we conclude that $\Pr[\mathsf{Game}_6 = 1] = 1/2$, i.e., $\epsilon_6 = 0$. Combining all the arguments together, we obtain

$$\epsilon_2 \leq 2M^2 \cdot \mathsf{Adv}_{\mathsf{KEM}, \mathcal{B}_2}^{\mathsf{IND\text{-}CCA}}(\lambda) + \mathsf{negl}(\lambda).$$

$\square$

Security against Type-{3-1, 3-2} strategy.

Lemma 5. *Assume* KDF *is pseudorandom. For any HNDL adversary $\mathcal{A}$ following the Type-{3-1, 3-2} strategy, there exists a reduction $\mathcal{B}_3$ that breaks the* IND-CCA *security of* KEM *such that*

$$\epsilon_2 \leq 2M^2 \cdot \mathsf{Adv}_{\mathsf{KEM}, \mathcal{B}_3}^{\mathsf{IND\text{-}CCA}}(\lambda) + \mathsf{negl}(\lambda).$$

Proof of Lemma 5. The proof is identical to the proof of Lemma 4. The difference is the tested instance (and its partner) uses a last-resort prekey bundle, not a one-time prekey bundle. Since $\mathcal{A}$ follows the Type-3 strategy, it never leaks the user state of the user r^*, who is the receiver of the tested instance. This ensures that $\mathcal{A}$ does not know the last-resort KEM decryption key, and thus IND-CCA security of KEM ensures that the randomness of the KEM shared secret. As a result, the real session key K seems random from $\mathcal{A}$. Note that the last-resort KEM key is used multiple time, so the challenged KEM decryption key is used to decrypt various KEM ciphertexts. Therefore, a decapsulation oracle (i.e., IND-CCA security) is required. $\square$

Security against Type-4 strategy.

Lemma 6. *Assume* KDF *is a random oracle. For any classic adversary $\mathcal{A}$ following the Type-4 strategy, there exists a reduction $\mathcal{B}_4$ that solves the* PRF-ODH *problem such that*

$$\epsilon_2 \leq 2NM \cdot \mathsf{Adv}_{\mathcal{B}_4}^{\mathsf{PRF\text{-}ODH}}(\lambda) + \mathsf{negl}(\lambda).$$

Proof of Lemma 6. The proof is identical to the proof of Lemma 3. The Type-4 adversary tests a sender instance, and it does not know the identity secret key of the receiver r^* of the tested instance. This situation is identical to the situation where the Type-1-1 adversary attacks (the difference is whether the tested instance has origin instances or not). Thus, similarly to the Type-1-1 adversary, the 2nd DH value $\mathsf{ss}_2 = [\mathsf{esk} \cdot \mathsf{isk}_{r^*}]G$ cannot be computed by the Type-4 adversary, and thus the session key K^* of the tested instance looks random from the Type-4 adversary assuming the PRF-ODH problem is hard. $\square$

---

[11]More precisely, IND-1-CCA security [Cra+07] is sufficient in this case because $\widetilde{\mathsf{iID}}_s$ uses the challenge KEM decapsulation key at most once. For simplicity, we assume standard IND-CCA security.

Security against Type-5 strategy.

Lemma 7. *Assume* KDF *is a random oracle. For any HNDL adversary $\mathcal{A}$ following the Type-5 strategy, there exists a reduction $\mathcal{B}_5$ that solves the* PRF-ODH *problem such that*

$$\epsilon_2 \leq 2NM \cdot \mathsf{Adv}_{\mathcal{B}_5}^{\mathsf{PRF\text{-}ODH}}(\lambda) + \mathsf{negl}(\lambda).$$

Proof of Lemma 7. We present the rest of the sequence of games from game Game_3.

Game_3. This game guesses the tested instance and its peer. At the beginning of the game, it chose an index i at random from $[M]$ and $\widetilde{u}$ at random from $\mathcal{U}$. Let Tested be the event that the tested instance iID^* satisfies $\mathsf{iID}^* = (i, k)$ for some $k \in \{0\} \cup \mathbb{N}$, $\mathsf{role}[\mathsf{iID}^*] = \mathsf{receiver}$ and $\mathsf{Sender}[\mathsf{iID}^*] = \widetilde{u}$. Since Tested is an efficiently checkable event, the game aborts as soon as it detects that event Tested does not occur. The probability the choice made by $\mathcal{A}$ is correctly guessed is probability $1/(NM)$, so we have

$$\epsilon_3 = \frac{\epsilon_2}{NM}.$$

In the following, we assume Tested occurs.

Game_4. We change how $\widetilde{\mathsf{iID}}_r$ computes the session key. In this game, we sample a random function RF and $\widetilde{\mathsf{iID}}_r$ computes the session key as $\mathsf{RF}(\mathsf{content})$ instead of evaluating $\mathsf{KDF}(\mathsf{ss}_1\|\mathsf{ss}_2\|\mathsf{ss}_3\|\mathsf{ss}_4\|\hat{\mathsf{ss}}, \mathsf{content})$. We will show that this change is indistinguishable from $\mathcal{A}$ assuming the PRF-ODH problem is hard w.r.t. the 1st key ss_1. To this end, we show an algorithm $\mathcal{B}_5$ that solves the PRF-ODH problem by using $\mathcal{A}$.

$\mathcal{B}_5$ receives two DH value $(U, V) = ([u]G, [v]G)$ and is given oracle access to O_u and O_v. Then, $\mathcal{B}_5$ simulates the security game as follows: $\mathcal{B}_5$ first generates the users' identity key pair. For the user $\widetilde{u}$, it sets U as the identity DH public key and generates its signature key pair. For the other users, $\mathcal{B}_5$ generates their DH key and signature key pair by itself. Then, $\mathcal{B}_5$ invokes $\mathcal{A}$ on input 1^λ and $\{\mathsf{ik}_u\}_{u \in \mathcal{U}}$, and answers the queries by $\mathcal{A}$ as follows:

- $O_{\mathsf{PubNewPrekeyBundle}}(u)$: If this is the ith instance, $\mathcal{B}_5$ sets the signed prekey $\mathsf{spk} = V$ and generates the others prekey following the protocol description. Otherwise, $\mathcal{B}_5$ responds as in Game_3.

- $O_{\mathsf{Receive}}(r, s, \rho)$: If this is the $\widetilde{\mathsf{iID}}_r$ sender instance, $\mathcal{B}_5$ answers as in Game_3 except that it computes the keys $K\|\tau_{\mathsf{conf}}$ using the challenge phase of the PRF-ODH game; it outputs $(\mathsf{ss}_2\|\mathsf{ss}_3\|\mathsf{ss}_4\|\hat{\mathsf{ss}}, \mathsf{content})$ and receives $K\|\tau_{\mathsf{conf}}$, which is
  $$x_0 = \mathsf{KDF}([uv]G\|\mathsf{ss}_2\|\mathsf{ss}_3\|\mathsf{ss}_4\|\hat{\mathsf{ss}}, \mathsf{content})$$
  or $x_1 = \mathsf{RF}(\mathsf{content})$ depending on the PRF-ODH game's challenge bit d. Otherwise, $\mathcal{B}_5$ responds as in Game_3.

- The other oracles: $\mathcal{B}_5$ responds as in Game_4.

Finally, $\mathcal{A}$ outputs its guess b'. Let b be the challenge bit of the key indistinguishability game. If $\mathcal{A}$ breaks the key indistinguishability of PQXDH (i.e., $b = b'$), $\mathcal{B}_5$ outputs $d' = 0$ to the PRF-ODH game; otherwise it outputs 1.

We can see if the challenge bit of the PRF-ODH game is $d = 0$, $\mathcal{B}_5$ simulates Game_4; Otherwise it simulates Game_5. In addition, we have

$$\epsilon_4 = \left| \Pr[b = b' | d = 0] - \frac{1}{2} \right|$$

$$\epsilon_5 = \left| \Pr[b = b' | d = 1] - \frac{1}{2} \right|.$$

Since $d' = 0$ if $b = b'$, we have

$$\mathsf{Adv}_{\mathcal{B}_1}^{\mathsf{PRF\text{-}ODH}}(\lambda) = \left| \Pr\left[d = d' \right] - \frac{1}{2} \right|$$

$$= \frac{1}{2} \left| \Pr\left[d' = 0 | d = 0 \right] - \Pr\left[d' = 0 | d = 1 \right] \right|$$

$$= \frac{1}{2} \left| \Pr\left[b = b' | d = 0 \right] - \Pr\left[b = b' | d = 1 \right] \right|$$

$$\geq \frac{1}{2} \left| \epsilon_4 - \epsilon_5 \right|.$$

Therefore, we have,

$$\left| \epsilon_4 - \epsilon_5 \right| \leq 2\mathsf{Adv}_{\mathcal{B}_5}^{\mathsf{PRF\text{-}ODH}}(\lambda).$$

Game_5. This game is identical to Game_4, except that we add an abort condition. If $\widetilde{\mathsf{iID}}_r$ accepts the received confirmation tag τ_{conf}, the game aborts. We call the event that abort occurs $\mathsf{TAGForge}$. Since the two games are identical-until-abort [BR06], we have

$$\left| \epsilon_4 - \epsilon_5 \right| \leq \Pr\left[\mathsf{TAGForge} \right].$$

Before bounding $\Pr\left[\mathsf{TAGForge} \right]$, we finish the proof of the lemma. We show that no adversary $\mathcal{A}$ following the Type-5 strategy has any remaining advantage in game Game_5, i.e., $\Pr\left[\mathsf{Game}_5(1^\lambda) = 1 \right] = 1/2$ ($\epsilon_5 = 0$). To see this, first let us assume $\mathcal{A}$ issued $\mathsf{Test}(\mathsf{iID}^*)$ and received a key that is not $\bot$. By assumption, Tested occurs in Game_5, i.e., $\mathsf{iID}^* = \widetilde{\mathsf{iID}}_r$. In addition, if $\mathsf{key}[\mathsf{iID}^*] \neq \bot$, then event $\mathsf{TAGForge}$ must have not triggered. On the other hand, iID^* accepts the receivedtag only if $\mathsf{TAGForge}$ occurs. This causes contradiction, so $\mathcal{A}$ can only receive $\bot$ when it issues $\mathsf{Test}(\mathsf{iID}^*)$. Thus, challenge bit b is statistically hidden from $\mathcal{A}$.

It remains to bound $\Pr\left[\mathsf{TAGForge} \right]$. Due to the changes we made in the previous games, the confirmation tag τ_{conf}^* computed by iID^* is random from the adversary. This concludes that if $\mathcal{A}$ triggers $\mathsf{TAGForge}$, it correctly guesses the tag. Since the tag is information-theoretically hidden from $\mathcal{A}$, the probability of correct guess is negligible, i.e., $\Pr[\mathsf{TAGForge}] \leq \mathsf{negl}(\lambda)$.

Combining all arguments together, we obtain the bound in Lemma 7. $\qquad\square$

C.2. Security of X3DH

The correctness of X3DH follows from construction. Below, we show match soundness and key indistinguishability. Their proofs are mostly inherited from the respective proofs of PQXDH, which are formally given in Appendix C.1.

Match soundness of X3DH. We prove match soundness with respect to the origin function $\Phi_{\mathsf{origin}}^{\mathsf{Signal}}$ as follows.

Theorem 5 (Match Soundness of X3DH). *Assume the KDF is collision resistance against a quantum adversary. Then,* X3DH *is match sound against a classical adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *with respect to the predicate* Match *(cf. Definition 6) and the origin function* $\Phi_{\mathsf{origin}}^{\mathsf{Signal}}$ *(cf. Definition 10).*

Proof. The proof of Theorem 5 is identical to the proof of Theorem 1. The proof of Theorem 1 relies on the fact that DH keys (spk, opk, epk) have high entropy and receivers reject replayed handshake messages. X3DH also satisfies these properties. Therefore, the same proof works to prove Theorem 5. $\qquad\square$

Key indistinguishability of X3DH. We show key indistinguishability with respect to the predicate $\mathsf{safe}_{\mathsf{X3DH}}$ (cf. Definition 9). Compared to the optimal predicate $\mathsf{safe}_{\mathsf{BAKE}}$, it is prone to an attack against full forward secrecy of the *sender* and user-state compromise impersonation security of the *receiver* (see Section 3.5). Formally, we have the following.

Theorem 6 (Key Indistinguishability of X3DH). *Assume the KDF is pseudorandom and the PRF-ODH problem w.r.t. KDF is hard against classical adversary. Then,* X3DH *is key indistinguishable against a classical adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *with respect to the predicate* $\mathsf{safe}_{\mathsf{X3DH}}$ *(cf. Definition 9).*

Proof. (Sketch) The security proofs of X3DH against the Type-{1,4,5} adversaries are identical to the proofs of PQXDH against the same adversaries because they rely on the secrecy of the common components of PQXDH and X3DH. On the other hand, the security proofs of X3DH against the Type-2 and the Type-3 adversaries are different from that of PQXDH. In X3DH, the security against the Type-2 (resp. Type-3) adversary is guaranteed by the secrecy of the DH value ss_4 (resp. ss_3). Intuitively, when the adversary follows the Type-2 (resp. Type-3) adversary, it does not know the one-time prekey (resp. last-resort prekey) generated by the tested receiver and the ephemeral secret key generated by the tested sender. Also, the tested instance has the origin/partner instance. Thus, assuming the PRF-ODH is hard w.r.t. ss_4 (resp. ss_3), the session key is indistinguishable from random against the Type-2 (resp. Type-3) adversary. The proof strategy is identical to the proof of Lemma 3. □

D. Security of RingXKEM

In this section, we provide the formal proofs of RingXKEM. The correctness of RingXKEM follows directly from the correctness of the underlying primitives. Below, we show match soundness and key indistinguishability of PQXDH. To this end, we first provide a helper lemma and then prove match soundness in Appendix D.2 and key indistinguishability in Appendix D.3.

D.1. Helper Lemma

To prove the match soundness and key indistinguishability of RingXKEM, we consider the security game that uses the following partnering predicate $\mathsf{Partner}_{\mathsf{RingXKEM}}$ instead of the predicate $\mathsf{Partner}$ in Definition 5. In the following, we use content[iID] to denote the content content used by the instance iID.

$$\mathsf{Partner}_{\mathsf{RingXKEM}}(\mathsf{iID}, \mathsf{iID}')$$
$$\Longleftrightarrow$$
$$[\![\,\mathsf{Sender[iID]} = \mathsf{Sender[iID']}\,]\!]$$
$$\wedge \quad [\![\,\mathsf{Receiver[iID]} = \mathsf{Receiver[iID']}\,]\!]$$
$$\wedge \quad [\![\,\mathsf{role[iID]} \neq \mathsf{role[iID']}\,]\!]$$
$$\wedge \quad [\![\,\mathsf{content[iID]} = \mathsf{content[iID']}\,]\!]$$
$$\wedge \quad [\![\,\mathsf{key[iID]} \neq \epsilon\,]\!]$$
$$\wedge \quad [\![\,\mathsf{key[iID']} \neq \epsilon\,]\!].$$

The predicate $\mathsf{Partner}_{\mathsf{RingXKEM}}$ checks the equality of content used in RingXKEM instead of the equality of the derived session key.

We can show that using $\mathsf{Partner}_{\mathsf{RingXKEM}}$ instead of $\mathsf{Partner}$ is indistinguishable to the adversary up to negligible error.

Lemma 8. *Assume that* KDF *is collision resistant and* KEM *is correct. For* $\mathsf{mode} \in \{\,\mathsf{KIND}, \mathsf{MATCH}\,\}$, $\mathsf{Game}^{\mathsf{mode}}_{\mathsf{RingXKEM}, \mathcal{A}}$ *using* $\mathsf{Partner}(\mathsf{iID}, \mathsf{iID}')$ *is indistinguishable from the same game using the* RingXKEM*-specific predicate* $\mathsf{Partner}_{\mathsf{RingXKEM}}(\mathsf{iID}, \mathsf{iID}')$.

Proof. We will show that
$$\mathsf{Partner}(\mathsf{iID}, \mathsf{iID}') \Leftrightarrow \mathsf{Partner}_{\mathsf{RingXKEM}}(\mathsf{iID}, \mathsf{iID}')$$
holds except with negligible errors.

$\mathsf{Partner}(\mathsf{iID}, \mathsf{iID}') \Rightarrow \mathsf{Partner}_{\mathsf{RingXKEM}}(\mathsf{iID}, \mathsf{iID}')$**:** If we have $\mathsf{Partner}(\mathsf{iID}, \mathsf{iID}') = \mathsf{true}$, then key[iID] = key[iID'] holds. This implies that the output of the KDF was equal given its inputs: $\mathsf{KDF}(\mathsf{ss}_r\|\widehat{\mathsf{ss}_r}, \mathsf{content[iID]}) = \mathsf{KDF}(\mathsf{ss}'_r\|\widehat{\mathsf{ss}_r}', \mathsf{content[iID']})$, where $\mathsf{ss}_r\|\widehat{\mathsf{ss}_r}$ and $\mathsf{ss}'_r\|\widehat{\mathsf{ss}_r}'$ are KEM shared secrets used by iID and iID', respectively. Since we assume KDF is collision resistant, we have that the inputs to KDF must be equal: $(\mathsf{ss}_r\|\widehat{\mathsf{ss}_r}, \mathsf{content[iID]}) = (\mathsf{ss}'_r\|\widehat{\mathsf{ss}_r}', \mathsf{content[iID']})$, and thus content[iID] = content[iID'] (and of course key[iID] $\neq \epsilon$ and key[iID'] $\neq \epsilon$).

$\mathsf{Partner}_{\mathsf{RingXKEM}}(\mathsf{iID}, \mathsf{iID}') \Rightarrow \mathsf{Partner}(\mathsf{iID}, \mathsf{iID}')$: By its definition, $\mathsf{Partner}_{\mathsf{RingXKEM}}(\mathsf{iID}, \mathsf{iID}') = \mathsf{true}$ implies that we have $\mathsf{content}[\mathsf{iID}] = \mathsf{content}[\mathsf{iID}']$ and $\mathsf{key}[\mathsf{iID}] \neq \epsilon \wedge \mathsf{key}[\mathsf{iID}'] \neq \epsilon$. Due to the latter condition, the receiver instance (iID or iID') accepts the received signature and computes its session key following the protocol description. In addition, since $\mathsf{content}[\mathsf{iID}] = \mathsf{content}[\mathsf{iID}']$ holds, both the sender and the receiver share the KEM public keys and ciphertexts used for their session key computation. Thus, by correctness of the KEM, both share the same KEM shared secrets ss and $\hat{\mathsf{ss}}$. Thus, we conclude $\mathsf{key}[\mathsf{iID}] = \mathsf{key}[\mathsf{iID}']$. □

This lemma allows us to check whether two instances are partnered or not from the exchanged messages.

D.2. Match Soundness of RingXKEM

We provide the proof of RingXKEM's match soundness with respect to the following origin function.

Definition 26 (Origin Function for RingXKEM)**.** For any $\mathsf{iID} \in \mathcal{S}_{\mathsf{iID}}$ (i.e., the set of all instances created during the game) with $\mathsf{prek}[\mathsf{iID}] = (\widehat{\mathsf{ek}}_r, \mathsf{path}_r, \mathsf{root}_r, \sigma_{r,\mathsf{root}}, \mathsf{rvk}) \neq \bot$, we define the origin function as $\Phi_{\mathsf{origin}}^{\mathsf{RingXKEM}}(\mathsf{iID}) := \widehat{\mathsf{ek}}_r$.

Notice the origin function only looks at the ephemeral KEM encapsulation key included in the prekey bundle. Namely, even if the sender uses a prekey bundle with different, say rvk, from the receiver, the receiver instance is deemed an origin instance of the sender instance as long as they share the same $\widehat{\mathsf{ek}}_r$. Such choice is validated by the following Theorem 7, establishing that match soundness holds for this origin function. More intuitively though, notice that among the elements in the prekey bundle, $\widehat{\mathsf{ek}}_r$ is the only one that critically affects the session key as the users derive the input to the KDF using the KEMs. Therefore, even if the adversary modifies, say rvk, a receiver instance using $\widehat{\mathsf{ek}}_r$ should still be considered an origin instance to the sender instance.

It is worth highlighting that the same can be said for the origin function used by the Signal protocols (cf. Definition 10). For instance, we could have defined $\Phi_{\mathsf{origin}}^{\mathsf{Signal}}(\mathsf{iID}) := (\mathsf{spk}_r, \mathsf{opk}_r, \mathsf{ek}_r)$, excluding the signatures in the prekey bundle. We would have arrived at the same security properties and proofs. We chose the one in Definition 10, as it was the most common one used in the literature.

Formally, we have the following.

Theorem 7 (Match Soundness of RingXKEM)**.** *Assume the KDF is collision resistance against a quantum adversary and assume KEM's encapsulation keys and ciphertexts have high min-entropy. Then, RingXKEM is match sound against a quantum adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ with respect to the predicate Match (cf. Definition 6) and the origin function $\Phi_{\mathsf{origin}}^{\mathsf{RingXKEM}}$ (cf. Definition 26).*

Proof. Thanks to Lemma 8, it is sufficient that we prove the match soundness in $\mathsf{Game}_{\mathsf{RingXKEM}, \mathcal{A}}^{\mathsf{MATCH}}$ w.r.t. proposition $\mathsf{Partner}_{\mathsf{RingXKEM}}(\mathsf{iID}, \mathsf{iID}')$. We need so show that each condition in the Match predicate is satisfied for any $\mathsf{iID}, \mathsf{iID}', \mathsf{iID}'' \in \mathcal{S}_{\mathsf{iID}}$.

1. If $\mathsf{Partner}_{\mathsf{RingXKEM}}(\mathsf{iID}, \mathsf{iID}') = \mathsf{true}$, we have

   a) $\mathsf{Receiver}[\mathsf{iID}] = \mathsf{Receiver}[\mathsf{iID}']$,

   b) $\mathsf{role}[\mathsf{iID}] \neq \mathsf{role}[\mathsf{iID}']$, and

   c) $\mathsf{content}[\mathsf{iID}] = \mathsf{content}[\mathsf{iID}']$.

Item 1b implies one of $(\mathsf{iID}, \mathsf{iID}')$ is the sender and the other the receiver, and Item 1c implies $\widehat{\mathsf{ek}} = \widehat{\mathsf{ek}}'$. Since the origin function $\Phi_{\mathsf{origin}}^{\mathsf{RingXKEM}}(\mathsf{iID}) := \widehat{\mathsf{ek}}_r$, it must be that $\mathsf{Origin}(\mathsf{iID}, \mathsf{iID}') = \mathsf{true}$ or $\mathsf{Origin}(\mathsf{iID}', \mathsf{iID}) = \mathsf{true}$ holds.

2. If both proposition $\mathsf{Partner}_{\mathsf{RingXKEM}}(\mathsf{iID}, \mathsf{iID}') = \mathsf{true}$ and proposition $\mathsf{Partner}_{\mathsf{RingXKEM}}(\mathsf{iID}, \mathsf{iID}'') = \mathsf{true}$, then we have

$$\mathsf{Sender}[\mathsf{iID}'] = \mathsf{Sender}[\mathsf{iID}'']$$
$$\wedge \; \mathsf{Receiver}[\mathsf{iID}'] = \mathsf{Receiver}[\mathsf{iID}'']$$
$$\wedge \; \mathsf{role}[\mathsf{iID}'] = \mathsf{role}[\mathsf{iID}'']$$
$$\wedge \; \mathsf{content}[\mathsf{iID}'] = \mathsf{content}[\mathsf{iID}''].$$

Thus, if $\mathsf{content}$ is unique during the game, it implies $\mathsf{iID} = \mathsf{iID}'$. So, we will prove the uniqueness of $\mathsf{content}$.

- Case 1: role[iID] = receiver and role[iID′] = role[iID″] = sender: Notice that iID′ and iID″ use independent randomness to generate KEM ciphertexts (ct, $\widehat{\text{ct}}$). Since we assume KEM ciphertexts have high min-entropy, even if iID′ and iID″ use the same prekey bundle, they generate different KEM ciphertexts with overwhelming probability. Thus, each sender instance has different (ct, $\widehat{\text{ct}}$). In other words, content[iID′] = content[iID″] implies iID′ = iID″.

- Case 2: role[iID] = sender and role[iID′] = role[iID″] = receiver: Notice that the game assigns iID ∈ $\mathbb{N} \times \{0, \perp\}$ to each prekey bundle. Since prekey bundles are generated from fresh and independent randomness, and we assume KEM encapsulation key have high min-entropy, the probability the same encapsulation key is generated via $O_{\text{PubNewPrekeyBundle}}$ queries is negligible. Thus, each iID′ ∈ $\mathbb{N} \times \{0, \perp\}$ is the only instance that is assigned the encapsulation key. Thus, if iID′ ∈ $\mathbb{N} \times \{0\}$, content[iID′] = content[iID″] implies iID′ = iID″, Otherwise, if iID′ ∈ $\mathbb{N} \times \mathbb{N}$ and iID″ ∈ $\mathbb{N} \times \mathbb{N}$, iID′ and iID″ uses the same last-resort prekey bundle (and thus the same encapsulation key). On the other hand, by the protocol description, reject replayed KEM ciphertexts (cf. Line 8 of the RingXKEM. Receive algorithm). Thus, each iID ∈ $\mathbb{N} \times \mathbb{N}$ uses different KEM ciphertexts (ct, $\widehat{\text{ct}}$) and thus different content. Therefore, in this case, iID′ and iID″ satisfy content[iID′] = content[iID″] only when iID′ = iID″.

3. If Origin(iID, iID′) = Origin(iID, iID″) = true, then we have

$$
\begin{aligned}
& \text{Sender[iID′]} = \text{Sender[iID″]} \\
\wedge\ & \text{Receiver[iID′]} = \text{Receiver[iID″]} \\
\wedge\ & \text{role[iID′]} = \text{role[iID″]} = \text{receiver} \\
\wedge\ & \Phi_{\text{origin}}^{\text{RingXKEM}}(\text{iID′}) = \Phi_{\text{origin}}^{\text{RingXKEM}}(\text{iID″}).
\end{aligned}
$$

Since prekey bundles are generated from fresh and independent randomness, and we assume KEM encapsulation key have high min-entropy, the probability the same encapsulation key is generated via $O_{\text{PubNewPrekeyBundle}}$ queries is negligible. Also, each prekey bundle is assigned to a unique iID ∈ $\mathbb{N} \times \{0, \perp\}$ by definition of the game. Thus, if $\Phi_{\text{origin}}^{\text{RingXKEM}}(\text{iID′}) = \Phi_{\text{origin}}^{\text{RingXKEM}}(\text{iID″})$, then base(iID′) = base(iID″). Moreover, the following holds.

(a) One-time prekey bundles are assigned to iID′ ∈ $\mathbb{N} \times \{0\}$ by definition of the game. RingXKEM uses one-time prekey bundles only once (cf. Line 6 and Lines 17 and 18 of RingXKEM.Receive (Algorithm 9)). Therefore, iID′ is used only once. Also, due to the uniqueness of the KEM encapsulation key, explained above, we have iID′ = iID″.

(a) Last-resort prekey bundles are assigned to some $\text{iID}_\perp$ ∈ $\mathbb{N} \times \{\perp\}$ when they are generated. Then, each time the same last-resort prekey is used, the game assigns (base($\text{iID}_\perp$), i) for i ∈ [prekreuse[$\text{iID}_\perp$]]. Therefore, if iID′ ∈ $\mathbb{N} \times (\{\perp\} \cup \mathbb{N})$, then there exists a unique instance $\text{iID}_\perp$ = (base(iID′), $\perp$) ∈ $\mathcal{S}_{\text{iID}}$, and we have

$$
\left\{ \text{iID″} \ \middle|\ \begin{array}{c} \text{Origin(iID, iID″)} = \text{true} \\ \wedge\ \text{iID″} \neq \text{iID}_\perp \end{array} \right\}
$$
$$
= \{(\text{base}(\text{iID}_\perp), i)\}_{i \in [\text{prekreuse}[\text{iID}_\perp]]}.
$$

$\square$

D.3. Key Indistinguishability of RingXKEM

Here, we provide the proof of RingXKEM's key indistinguishability.

Theorem 8 (Key Indistinguishability of RingXKEM). *Assume the KDF is pseudorandom, the KEM is IND-CCA secure, and the ring signature is unforgeable against quantum adversaries. Then, RingXKEM is key indistinguishable against a quantum adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ with respect to the predicate $\text{safe}_{\text{BAKE}}$ (cf. Definition 8).*

Proof. Let $\mathcal{A}$ be an adversary that plays the security game $\mathsf{Game}_{\mathcal{A}}^{\mathsf{KIND}}(1^\lambda)$ and let $\epsilon = \mathsf{Adv}_{\mathcal{A}}^{\mathsf{KIND}}(\lambda)$ be its advantage. In order to prove the theorem, we show that RingXKEM is secure against all the attack strategies listed in Table 2.

For each strategy taken by $\mathcal{A}$, we construct an algorithm that breaks one of the underlying assumptions by using $\mathcal{A}$ as a subroutine. Let $N = |\mathcal{U}|$ be the number of users in the system and M be the upper bound of NumiID (i.e., the maximum number of base iID generated by $\mathcal{A}$). We construct algorithms $\mathcal{B}_1$, $\mathcal{B}_2$, $\mathcal{B}_3$, $\mathcal{B}_{4,1}$, $\mathcal{B}'_{4,1}$ and $\mathcal{B}_{4,2}$ satisfying the following:

1. If $\mathcal{A}$ follows the Type-{1-1, 1-2} strategy, then $\mathcal{B}_1$ succeeds in breaking the IND-CCA security of KEM with advantage $\approx \frac{1}{NM}\epsilon$.

2. If $\mathcal{A}$ follows the Type-{2-1, 2-2} strategy, then $\mathcal{B}_2$ succeeds in breaking the IND-CCA security of KEM with advantage $\approx \frac{1}{M^2}\epsilon$.

3. If $\mathcal{A}$ follows the Type-Type-{3-1, 3-2} strategy, then $\mathcal{B}_3$ succeeds in breaking the IND-CCA security of KEM with advantage $\approx \frac{1}{M^2}\epsilon$.

4. If $\mathcal{A}$ follows the Type-4-1 strategy, then $\mathcal{B}_{4,1}$ succeeds in breaking the unforgeability of RS or $\mathcal{B}_{4,2}$ succeeds in breaking the binding of Merkle Tree with advantage $\approx \epsilon$.

5. If $\mathcal{A}$ follows the Type-4-2 strategy, then $\mathcal{B}_{4,2}$ succeeds in breaking the unforgeability of RS with advantage $\approx \epsilon$.

We present a security proof structured as a sequence of games. Without loss of generality, we assume that $\mathcal{A}$ always issues O_{Test}-query. In the following, let us denote the advantage of $\mathcal{A}$ in Game_i by $\epsilon_i := \left| \Pr\left[\mathsf{Game}_i(1^\lambda) = 1 \right] - 1/2 \right|$. Regardless of the strategy taken by $\mathcal{A}$, all proofs share a common game sequence Game_0–Game_2 as described below.

Game_0. This game is identical to the original security game. We thus have

$$\epsilon_0 = \epsilon.$$

Game_1. This game uses $\mathsf{Partner}_{\mathsf{RingXKEM}}$ instead of $\mathsf{Partner}$. Due to Lemma 8, we have

$$|\epsilon_0 - \epsilon_1| \leq \mathsf{negl}(\lambda).$$

Game_2. This game is identical to Game_1, except that it aborts if the predicate Match returns false. Since we proved that RingXKEM is match-sound w.r.t. $\mathsf{Partner}_{\mathsf{RingXKEM}}$, we have

$$|\epsilon_1 - \epsilon_2| \leq \mathsf{Adv}_{\mathsf{RingXKEM},\mathcal{A}}^{\mathsf{MATCH}}(\lambda).$$

We now divide the game sequence depending on the strategy taken by the adversary $\mathcal{A}$. Regardless of $\mathcal{A}$'s strategy, we prove that ϵ_2 is negligible, which in particular implies that ϵ_0 is also negligible. Formally, this is shown in Lemmas 9 to 13 provided below. We first complete the proof of the theorem. Specifically, by combining all the lemmas together and folding $\mathcal{B}_{4,1}$ and $\mathcal{B}_{4,2}$ into one algorithm $\mathcal{B}_4$, we obtain the following bound:

$$\mathsf{Adv}_{\mathsf{RingXKEM},\mathcal{A}}^{\mathsf{KIND}}(\lambda) \leq +\mathsf{Adv}_{\mathsf{RingXKEM},\mathcal{A}}^{\mathsf{MATCH}}(\lambda) + \mathsf{negl}(\lambda)$$

$$+ \begin{cases} 2NM \cdot \mathsf{Adv}_{\mathsf{KEM},\mathcal{B}_1}^{\mathsf{IND\text{-}CCA}}(\lambda) \\ 2M^2 \cdot \mathsf{Adv}_{\mathsf{KEM},\mathcal{B}_2}^{\mathsf{IND\text{-}CCA}}(\lambda) \\ 2M^2 \cdot \mathsf{Adv}_{\mathsf{KEM},\mathcal{B}_3}^{\mathsf{IND\text{-}CCA}}(\lambda) \\ \mathsf{Adv}_{\mathsf{RS},\mathcal{B}_4}^{\mathsf{Unf}}(\lambda) \end{cases}.$$

$\square$

It remains to prove Lemmas 9 to 13.

Security against Type-{1-1, 1-2} strategy.

Lemma 9. *Let $N = |\mathcal{U}|$ be the number of users in the system and M be the upper bound of NumiID (i.e., the maximum number of base iID generated by $\mathcal{A}$). Assume KDF is pseudorandom. For any quantum adversary $\mathcal{A}$ following the Type-{1-1, 1-2} strategy, there exists a reduction $\mathcal{B}_1$ that breaks the IND-CCA security of KEM such that*

$$\epsilon_2 \leq 2NM \cdot \mathsf{Adv}_{\mathsf{KEM}, \mathcal{B}_1}^{\mathsf{IND\text{-}CCA}}(\lambda) + \mathsf{negl}(\lambda).$$

Proof of Lemma 9. We present the rest of the sequence of games from game Game_3.

Game_3. This game guesses the tested instance and its peer. At the beginning of the game, it chose i at random from $[M]$ and a user $\widetilde{u}$ at random from $\mathcal{U}$. Let Tested be the event that the index i is used for a sender instance $\widetilde{\mathsf{iID}}_s := (i, 0)$ and the tested instance iID^* satisfies $\mathsf{Receiver}[\mathsf{iID}^*] = \widetilde{u}$ and either

- $\mathsf{iID}^* = \widetilde{\mathsf{iID}}_s$ if $\mathsf{role}[\mathsf{iID}^*] = \mathsf{sender}$, or

- $\mathsf{Partner}_{\mathsf{RingXKEM}}(\widetilde{\mathsf{iID}}_s, \mathsf{iID}^*) = \mathsf{true}$ if $\mathsf{role}[\mathsf{iID}^*] = \mathsf{receiver}$.

Since the Tested event can be efficiently checked, the game aborts as soon as it detects that Tested does not occur. The probability that the tested instance and its peer are correctly guessed is $1/(NM)$, so we have

$$\epsilon_3 = \frac{\epsilon_2}{NM}.$$

Game_4. In this game, we modify how the user $\widetilde{u}$ computes session keys if it receives the handshake message generated by $\widetilde{\mathsf{iID}}_s$. Let $(\mathsf{ss}_{\widetilde{u}}, \mathsf{ct}_{\widetilde{u}}) \xleftarrow{\$} \mathsf{KEM.Encaps}(\mathsf{ek}_{\widetilde{u}})$ be the KEM key-ciphertext pair generated by the instance $\widetilde{\mathsf{iID}}_s$ with $\widetilde{u}$'s identity key. Then, when some iID' such that $\mathsf{Receiver}[\mathsf{iID}'] = \widetilde{u}$ is invoked on input $\rho = (\mathsf{ct}, \hat{\mathsf{ct}}, \mathsf{ct}_{\mathsf{ske}})$, it first checks if $\mathsf{ct} = \mathsf{ct}_{\widetilde{u}}$. If so, it uses the key $\mathsf{ss}_{\widetilde{u}}$ that was generated by $\widetilde{\mathsf{iID}}_s$ instead of decrypting ct. Otherwise, if $\mathsf{ct} \neq \mathsf{ct}_{\widetilde{u}}$, then it proceeds exactly as in the previous game. If decryption errors do not occur in KEM, the two games Game_3 and Game_4 are identical. Hence,

$$|\epsilon_3 - \epsilon_4| \leq \mathsf{negl}(\lambda).$$

Game_5. In this game, we modify the way the sender instance $\widetilde{\mathsf{iID}}_s$ computes the handshake message and session key. When $\widetilde{\mathsf{iID}}_s$ is created, it samples a random KEM secret $\mathsf{ss}_{\widetilde{u}}$ instead of computing $(\mathsf{ss}_{\widetilde{u}}, \mathsf{ct}_{\widetilde{u}}) \xleftarrow{\$} \mathsf{KEM.Encaps}(\mathsf{ek}_{\widetilde{u}})$. Note that due to the modification we made in the previous game, when $\widetilde{u}$ receives $\mathsf{ct}_{\widetilde{u}}$, it also uses the random key $\mathsf{ss}_{\widetilde{u}}$ generated by $\widetilde{\mathsf{iID}}_s$. Since $\mathcal{A}$ follows the Type-1 strategy, the identity secret key of the user $\widetilde{u}$ if Tested occurs, who is the receiver of the tested instance, is never leaked. Thus, due to the IND-CCA security of KEM, Game_4 and Game_5 are indistinguishable. Thus, we have

$$|\epsilon_4 - \epsilon_5| \leq 2\mathsf{Adv}_{\mathsf{KEM}, \mathcal{B}_1}^{\mathsf{IND\text{-}CCA}}(\lambda).$$

Note that the user $\widetilde{u}$ decrypts multiple ciphertexts with its identity decapsulation key $\mathsf{dk}_{\widetilde{u}}$. To ensure that such decapsulations do not leak any information, IND-CCA security is required.

Game_6. We change how $\widetilde{\mathsf{iID}}_s$ computes the session key. In this game, we sample a random function RF and $\widetilde{\mathsf{iID}}_s$ computes the session key as $\mathsf{RF}(\mathsf{content})$ instead of computing $\mathsf{KDF}(\mathsf{ss}\|\widetilde{\mathsf{ss}}, \mathsf{content})$. Due to the modification we made in the previous game, $\widetilde{\mathsf{ss}}$ is chosen uniformly at random. Therefore, by the pseudorandomness of KDF, the two games are indistinguishable. Thus, we have

$$|\epsilon_5 - \epsilon_6| \leq \mathsf{negl}(\lambda).$$

It remains to show that the session key output by the tested instance in Game_6 is uniformly random regardless of the challenge bit $b \in \{0, 1\}$ chosen by the game. We consider the case where $b = 0$ and prove that the honestly generated session key by iID^* is distributed uniformly random. First, by the event Tested occurring, it must be

the case that the tested instance (and its partner) computes the session key as $K^* \| K^*_{\text{ske}} \leftarrow \text{RF}(\text{content}^*)$ for some content*. Next, by RingXKEM satisfying Match soundness, the only instances that share the same content* must be the tested instance and its partner. Therefore, we conclude that K^* is the session key of only the tested instance and its partner. Since the output of RF is distributed uniformly at random for different inputs, we conclude that $\Pr[\text{Game}_6 = 1] = 1/2$, i.e., $\epsilon_6 = 0$. Combining all arguments together, we obtain

$$\epsilon_2 \leq 2NM \cdot \text{Adv}^{\text{IND-CCA}}_{\text{KEM}, \mathcal{B}_1}(\lambda) + \text{negl}(\lambda).$$

$\square$

Security against Type-{2-1, 2-2} adversary.

Lemma 10. *Let M be the upper bound of* NumiID *(i.e., the maximum number of base* iID *generated by $\mathcal{A}$). Assume* KDF *is pseudorandom. For any quantum adversary $\mathcal{A}$ following the Type-{2-1, 2-2} strategy, there exists a reduction $\mathcal{B}_2$ that breaks the* IND-CCA *security of* KEM *such that*

$$\epsilon_2 \leq 2M^2 \cdot \text{Adv}^{\text{IND-CCA}}_{\text{KEM}, \mathcal{B}_2}(\lambda) + \text{negl}(\lambda).$$

Proof of Lemma 10. We present the rest of the sequence of games from game Game_3.

Game_3. This game guesses the tested instance and its origin/partner instance. At the beginning of the game, it chose values i and j at random from $[M]$. Let Tested be the event that the index i is assigned for some sender instance $\widetilde{\text{iID}}_s$ such that $\widetilde{\text{iID}}_s = (i, 0)$, the index j is assigned for some receiver instance $\widetilde{\text{iID}}_r$ such that $\widetilde{\text{iID}}_r = (j, k)$ for $k \in \mathbb{N}$ and the tested instance iID* satisfies either

- if role[iID*] = sender, then iID* = $\widetilde{\text{iID}}_s$ and we have $\text{Origin}(\widetilde{\text{iID}}_s, \widetilde{\text{iID}}_r)$ = true, or

- if role[iID*] = receiver, then iID* = $\widetilde{\text{iID}}_r$ and we have $\text{Partner}_{\text{RingXKEM}}(\widetilde{\text{iID}}_s, \widetilde{\text{iID}}_r)$ = true.

Since Tested is an efficiently checkable event, the game aborts as soon as it detects that event Tested does not occur. The probability the choice made by $\mathcal{A}$ is correctly guessed is probability $1/M^2$, so we have

$$\epsilon_3 = \frac{\epsilon_2}{M^2}.$$

Game_4. In this game, we modify how $\widetilde{\text{iID}}_s$ and $\widetilde{\text{iID}}_r$ compute their session key. Let $\widetilde{\text{ek}}$ be the one-time KEM prekey assigned to $\widetilde{\text{iID}}_r$, and let $(\widetilde{\text{ss}}, \widetilde{\text{ct}}) \leftarrow \text{KEM.Encaps}(\widetilde{\text{ek}})$ be the KEM key-ciphertext pair generated by $\widetilde{\text{iID}}_s$ with $\widetilde{\text{ek}}$. (Due to the change we made in Game_3, $\widetilde{\text{iID}}_s$ and $\widetilde{\text{iID}}_r$ are origin/partner instance, so they share $\widetilde{\text{ek}}$.) Then, when $\widetilde{\text{iID}}_r$ is invoked on input $\rho = (\text{ct}, \hat{\text{ct}}, \text{ct}_{\text{ske}})$, it first checks if $\hat{\text{ct}} = \widetilde{\text{ct}}$. If so, it uses the key $\widetilde{\text{ss}}$ that was generated by $\widetilde{\text{iID}}_s$ instead of decrypting $\widetilde{\text{ct}}$. Otherwise, if $\hat{\text{ct}} \neq \widetilde{\text{ct}}$, then it proceeds exactly as in the previous game. Conditioning on decryption errors are not occurring on KEM, the two games Game_3 and Game_4 are identical. Hence,

$$|\epsilon_3 - \epsilon_4| \leq \text{negl}(\lambda).$$

Game_5. In this game, we modify the way the sender instance $\widetilde{\text{iID}}_s$ computes the handshake message and session key. When $\widetilde{\text{iID}}_s$ is invoked on input the prekey bundle $\widetilde{\text{prek}}_r$ assigned to $\widetilde{\text{iID}}_r$, it samples a random KEM secret $\widetilde{\text{ss}}$ instead of computing the real shared secret by $(\widetilde{\text{ss}}, \widetilde{\text{ct}}) \xleftarrow{\$} \text{KEM.Encaps}(\widetilde{\text{ek}})$. Note that due to the modification we made in the previous game, when $\widetilde{\text{iID}}_r$ receives $\widetilde{\text{ct}}$, it also uses $\widetilde{\text{ss}}$ generated by $\widetilde{\text{iID}}_s$. Since $\mathcal{A}$ follows the Type-2 strategy, the user state of the receiver r^* is leaked after r^* has erased the one-time KEM decryption key corresponding to $\widetilde{\text{ek}}$ from its state. Thus, $\mathcal{A}$ does not know the KEM decryption key. Due to the IND-CCA security of KEM, Game_4 and Game_5 are indistinguishable. Thus, we have

$$|\epsilon_4 - \epsilon_5| \leq 2\text{Adv}^{\text{IND-CCA}}_{\text{KEM}, \mathcal{B}_2}(\lambda).$$

As in the proof of PQXDH (Lemma 4, Game_5), we rely on the IND-CCA security of KEM for simplicity. To correctly answer if $\widetilde{\text{iID}}_r$ receives a KEM ciphertext that is not the challenge ciphertext, a decapsulation oracle is needed. As this may happen at most once, IND-1-CCA security [Cra+07] would be sufficient for this proof step.

Game$_6$. We change how $\widetilde{\text{iID}}_s$ computes the session key. In this game, we sample a random function RF and $\widetilde{\text{iID}}_s$ computes the session key using the random function as RF(content) instead of KDF($\text{ss}\|\widetilde{\text{ss}}$, content). Due to the modification we made in the previous game, $\widetilde{\text{ss}}$ is chosen uniformly at random so the KDF key has sufficiently large min-entropy. Therefore, assuming the pseudorandomness of KDF, the two games are indistinguishable. Thus, we have

$$|\epsilon_5 - \epsilon_6| \le \text{negl}(\lambda).$$

It remains to show that the session key output by the tested instance in Game$_6$ is uniformly random regardless of the challenge bit $b \in \{0, 1\}$ chosen by the game. We consider the case where $b = 0$ and prove that the honestly generated session key by iID* is distributed uniformly random. First, by the event Tested occurring, it must be the case that the tested instance (and its partner) computes the session key as $K^*\|K^*_{\text{ske}} \leftarrow \text{RF}(\text{content}^*)$ for some content*. Next, as RingXKEM satisfies Match soundness, the only instances that share the same content* must be the tested instance and its partner. Therefore, we conclude that K^* is the session key of only the tested instance and its partner. Since the output of RF is distributed uniformly at random for different inputs, we conclude that $\Pr[\text{Game}_6 = 1] = 1/2$, i.e., $\epsilon_6 = 0$. Combining all the arguments together, we obtain

$$\epsilon_2 \le 2M^2 \cdot \text{Adv}^{\text{IND-CCA}}_{\text{KEM}, \mathcal{B}_2}(\lambda) + \text{negl}(\lambda).$$

$\square$

Security against Type-{3-1, 3-2} adversary.

Lemma 11. *Let M be the upper bound of* NumiID *(i.e., the maximum number of base iID generated by $\mathcal{A}$). Assume* KDF *is pseudorandom. For any quantum adversary $\mathcal{A}$ following the Type-{3-1, 3-2} strategy, there exists a reduction $\mathcal{B}_3$ that breaks the* IND-CCA *security of* KEM *such that*

$$\epsilon_2 \le 2M^2 \cdot \text{Adv}^{\text{IND-CCA}}_{\text{KEM}, \mathcal{B}_3}(\lambda) + \text{negl}(\lambda).$$

Proof of Lemma 11. The proof is identical to the proof of Lemma 10. The difference is the tested instance (and its partner) uses a last-resort prekey bundle, not a one-time prekey bundle. Since $\mathcal{A}$ follows the Type-3 strategy, it never leaks the user state of the user r^*, who is the receiver of the tested instance. This ensures that $\mathcal{A}$ does not know the last-resort KEM decryption key, and thus IND-CCA security of KEM ensures that the randomness of the KEM shared secret generated by the last-resort KEM encapsulation key. This also ensures the session key K^* of the tested instance is indistinguishable from a random key against the Type-3 $\mathcal{A}$. Note that the last-resort KEM key is used multiple time, its decryption key is used to decrypt various KEM ciphertexts. Therefore, a decapsulation oracle (i.e., IND-CCA security) is required. $\square$

Security against Type-4-1 adversary.

Lemma 12. *Assume the hash function used in the Merkle Tree is collision resistant. For any quantum adversaries $\mathcal{A}$ following the Type-4-1 strategy, there exists a $\mathcal{B}_{4,1}$ that breaks the unforgeability of* RS *such that*

$$\epsilon_2 \le \text{Adv}^{\text{Unf}}_{\text{RS}, \mathcal{B}_{4,1}}(\lambda) + \text{negl}(\lambda).$$

Proof of Lemma 12. We present the continuation of the sequence of games starting from Game$_2$. In the following, we assume the game keeps a list $\mathcal{L}_u$ that stores all values $(\widehat{\text{ek}}_{u,t}, \text{path}_{u,t}, \text{root}_u, \sigma_{u, \text{root}_u})$ generated by user u. That is, every time $O_{\text{PubNewPrekeyBundle}}(u)$ is queried, $\mathcal{L}_u$ is updated by appending the resulting $(\widehat{\text{ek}}_{u,t}, \text{path}_{u,t}, \text{root}_u, \sigma_{u, \text{root}_u})$ for $t \in [L] \cup \{\bot\}$ into $\mathcal{L}_u$.

Game$_3$. This game is identical to Game$_2$, except that we add abort conditions. When a sender instance iID receives a prekey $\text{prek} = (\widehat{\text{ek}}, \text{path}, \text{root}, \sigma_{\text{root}}, \text{rvk})$ from a user $r = \text{Receiver}[\text{iID}]$, the game additionally checks if the following event occurs:

Event SIGForge: we have a Merkle tree root that is not in $\mathcal{L}_r$ $((*, *, \text{root}) \notin \mathcal{L}_r)$, an uncorrupted peer (PeerCorr[iID] = false) and RS.Verify($\{\text{rvk}_r\}, \text{root}, \sigma_{\text{root}}) = 1$.

Event MTColl: we have $(\widehat{ek}', path', root) \in \mathcal{L}_r$ for some $\widehat{ek}'$, $path'$, such that $\widehat{ek} \neq \widehat{ek}'$ and the root matches such that ReconstructRoot$(\widehat{ek}, path) = root$.

If either of them occurs, the game aborts. If the game does not abort, two games proceeds identically. Since the two events are mutually exclusive, we have

$$|\epsilon_2 - \epsilon_3| \leq \Pr[\mathsf{SIGForge}] + \Pr[\mathsf{MTColl}].$$

Before bounding $\Pr[\mathsf{SIGForge}]$ and $\Pr[\mathsf{MTColl}]$, we finish the proof of the lemma. We show that no adversary $\mathcal{A}$ following the Type-4-1 strategy has any remaining advantage in game Game_3, i.e., $\Pr[\mathsf{Game}_3(1^\lambda) = 1] = 1/2$, i.e., $\epsilon_3 = 0$. To see this, first let us assume $\mathcal{A}$ received a key that is not a $\perp$ when it issued $\mathsf{Test}(\mathsf{iID}^*)$. Let $\mathsf{prek}^* = (\widehat{ek}^*, path^*, root^*, \sigma_{root^*}, rvk^*)$ be the prekey received by iID^*. If $\mathsf{key}[\mathsf{iID}^*] \neq \perp$ in Game_3, then events MTColl and SIGForge must have not triggered, especially for iID^*. This implies that $(\widehat{ek}^*, *, root^*) \in \mathcal{L}_{r^*}$. Consequently, there exists some instance $\widetilde{\mathsf{iID}}$ that is assigned prek^*, implying that $\widetilde{\mathsf{iID}}$ is the origin instance of iID^*. On the other hand, by the definition of the Type-4-1 strategy, iID^* has no origin instance. Since this forms a contradiction, the assumption that $\mathcal{A}$ received a key that is not a $\perp$ when it issued $\mathsf{Test}(\mathsf{iID}^*)$ is false. Therefore, $\mathcal{A}$ can only receive $\perp$ when it issues $\mathsf{Test}(\mathsf{iID}^*)$. As a result, the challenge bit b is statistically hidden from $\mathcal{A}$ we have $\Pr[\mathsf{Game}_3(1^\lambda) = 1] = 1/2$.

It remains to bound $\Pr[\mathsf{SIGForge}]$ and $\Pr[\mathsf{MTColl}]$.

Bounding $\Pr[\mathsf{SIGForge}]$. Assume SIGForge occurs for some sender instance iID. Then, its input prekey bundle $\mathsf{prek} = (\widehat{ek}, path, root, \sigma_{root}, rvk)$ satisfies having $(*, *, root) \notin \mathcal{L}_r$, RS.Verify$(\{rvk_r\}, root, \sigma_{root}) = 1$ and PeerCorr$[\mathsf{iID}] = \mathsf{false}$. This implies that $root$ is not signed by the user r, but σ_{root} is a valid signature and the identity secret key of the user u has not been revealed when iID accepts the signature. Thus, $\mathcal{A}$ created a valid forgery $(root, \sigma_{root})$ for the verification key rvk_u without knowing the corresponding secret key. With $\mathcal{A}$ we can construct an algorithm $\mathcal{B}_{4,1}$ that breaks the unforgeability of RS, such that $\Pr[\mathsf{SIGForge}] = \mathsf{Adv}_{\mathsf{RS}}^{\mathsf{Unf}}(\mathcal{B}_{4,1})$.

Bounding $\Pr[\mathsf{MTColl}]$. Assume MTColl occurs for some sender iID. Then, its input prekey bundle $\mathsf{prek} = (\widehat{ek}, path, root, \sigma_{root}, rvk)$ satisfies $(\widehat{ek}', path', root) \in \mathcal{L}_r$ for some $\widehat{ek}'$, $path'$ and $\widehat{ek} \neq \widehat{ek}'$. These conditions imply that $\mathcal{A}$ finds a pair $(\widehat{ek}, path)$ such that $\widehat{ek}$ is not a committed element into $root$ and the pair satisfies ReconstructRoot$(\widehat{ek}, path) = root$. Due to Lemma 1, by using such $\mathcal{A}$, we can construct an efficient extractor $\mathcal{B}'_{4,1}$ outputs a collision for the hash function used in the Merkle Tree. However, this contradicts the assumption that the hash function is collision resistant. Thus, we have

$$\Pr[\mathsf{MTColl}] \leq \mathsf{negl}(\lambda).$$

Combining everything together, we conclude

$$\epsilon_2 \leq \mathsf{Adv}_{\mathsf{RS}, \mathcal{B}_{4,1}}^{\mathsf{Unf}}(\lambda) + \mathsf{negl}(\lambda).$$

$\square$

Security against Type-4-2 adversary.

Lemma 13. *For any quantum adversaries $\mathcal{A}$ following the Type-4-2 strategy, there exists a reduction $\mathcal{B}_{4,2}$ that breaks the unforgeability of RS such that*

$$\epsilon_2 \leq \mathsf{Adv}_{\mathsf{RS}, \mathcal{B}_{4,2}}^{\mathsf{Unf}}(\lambda).$$

Proof of Lemma 13. We present the rest of the sequence of games started from Game_3.

Game$_3$. This game is identical to Game$_2$, except that we add an abort condition. Let $\mathcal{L}_u$ be a list of message-signature pairs that the user u generates as being a sender. That is, every time an instance iID$'$ owned by u is created as a sender, it updates the list $\mathcal{L}_u$ by appending the message-signature pair (content, σ) that u generates. Then, when a receiver instance iID is invoked on input (ct, ĉt, ct$_{\mathsf{ske}}$) from a sender s, it checks if RS.Verify($\{$ rvk$_s$, rvk $\}$, content, σ) = 1, (content, $*$) $\notin \mathcal{L}_s$, and PeerCorr[iID] = false. If so, the game aborts. (Recall that σ is the signature decrypted from ct$_{\mathsf{ske}}$) We call the event that abort occurs as SIGForge. Since the two games are identical until abort, we have

$$|\epsilon_2 - \epsilon_3| \leq \Pr\left[\mathsf{SIGForge}\right].$$

Before bounding $\Pr\left[\mathsf{SIGForge}\right]$, we finish the proof of the lemma. We show that no adversary $\mathcal{A}$ following the Type-4-2 strategy has winning advantage in game Game$_3$, i.e., $\Pr\left[\mathsf{Game}_3(1^\lambda) = 1\right] = 1/2$ ($\epsilon_3 = 0$). To see this, first let us assume $\mathcal{A}$ issued Test(iID*) and received a key that is not a $\perp$. If key[iID*] $\neq \perp$, then event SIGForge must have not triggered, especially for iID*. Since key[iID*] $\neq \perp$ and $\mathcal{A}$ follows the Type-4-2 strategy, RS.Verify($\{$ rvk$_s$, rvk $\}$, content, σ) = 1 and PeerCorr[iID] = false hold. Thus, if SIGForge does not trigger, (content, $*$) $\in \mathcal{L}_s$ is satisfied. That is, there exists an instance $\widetilde{\mathsf{iID}}$ that signed content*. Also, this implies that $\widetilde{\mathsf{iID}}$ and iID* are partners w.r.t. Partner$_{\mathsf{RingXKEM}}$. On the other hand, by the definition of the Type-4-2 strategy, iID* has no partner instance. This contradicts the fact that iID* has no partner, so $\mathcal{A}$ can only receive $\perp$ when it issues Test(iID*). Hence, since the challenge bit b is statistically hidden from $\mathcal{A}$, we have $\Pr\left[\mathsf{Game}_3(1^\lambda) = 1\right] = 1/2$.

It remains to bound $\Pr\left[\mathsf{SIGForge}\right]$. Assume SIGForge occurs for some receiver instance iID. Then, its content[iID] satisfies (content[iID], $*$) $\notin \mathcal{L}_{\mathsf{Sender[iID]}}$: the message content[iID] was never signed by the user s. Also, PeerCorr[iID] = false means the identity secret key of the user s has not been revealed when iID accepts the signature. Moreover, it implies we have some σ for which RS.Verify($\{$ rvk$_s$, rvk $\}$, content[iID], σ) = 1 holds. Note that the signing key corresponding to rvk is not corrupted, because RingXKEM deletes ephemeral ring signing keys by definition. This concludes that if $\mathcal{A}$ triggers SIGForge, it created a valid forgery (content[iID], σ) for the ring $\{$ rvk$_s$, rvk $\}$ that consists of uncorrupted verification keys. This concludes we can construct an algorithm $\mathcal{B}_{4,2}$ that breaks the unforgeability of RS, so that $\Pr[\mathsf{SIGForge}] = \mathsf{Adv}^{\mathsf{Unf}}_{\mathsf{RS}}(\mathcal{B}_{4,2})$.

Combining everything together, we conclude

$$\epsilon_2 \leq \mathsf{Adv}^{\mathsf{Unf}}_{\mathsf{RS}, \mathcal{B}_{4,2}}(\lambda).$$

$\square$