USENIX

THE ADVANCED COMPUTING
SYSTEMS ASSOCIATION

Bundled Authenticated Key Exchange: A Concrete Treatment of Signal's Handshake Protocol and Post-Quantum Security

Keitaro Hashimoto, *National Institute of Advanced Industrial Science and Technology* (*AIST*)*;* Shuichi Katsumata, *National Institute of Advanced Industrial Science and Technology* (*AIST*) *and PQShield;* Thom Wiggers, *PQShield*

https://www.usenix.org/conference/usenixsecurity25/presentation/hashimoto-key-exchange

This paper is included in the Proceedings of the 34th USENIX Security Symposium.

August 13–15, 2025 • Seattle, WA, USA

978-1-939133-52-6

Open access to the Proceedings of the 34th USENIX Security Symposium is sponsored by USENIX.

Bundled Authenticated Key Exchange: A Concrete Treatment of Signal's Handshake Protocol and Post-Quantum Security

Keitaro Hashimoto
National Institute of Advanced Industrial Science and Technology (AIST)

Shuichi Katsumata
AIST & PQShield

Thom Wiggers
PQShield

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 analyze 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.

1 Introduction

The Signal protocol [36, 42] 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 [48] and Facebook Messenger [37]. 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 [42]. The Signal handshake protocol was initially X3DH [36], based on Triple Diffie–Hellman [32]. 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 [3, 6, 14, 15, 21, 29]. Proposals for fully post-quantum alternatives also devised protocol-specific models for analysis [9, 10, 16, 20, 25, 26], 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 indis-

tinguishability 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 [6]. 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 [36, 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 [29], 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 [6, 21] 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] Following the Signal source code and the specification for PQXDH.

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 Sec. 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 [9, 25, 26]. 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 [23].

[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 Sec. 3.3.2 for more detail.

By extrapolating from the runtime performance of the primitives, we 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. [14, 15] provided a tailored game-based security model, capturing both the X3DH and Double Ratchet protocols. Kobeissi, Bhargavan, and Blanchet [29] modeled the composition in the symbolic (namely ProVerif) and computational model (CryptoVerif). PQXDH has been recently developed alongside formal analysis by Bhargavan et al. [6]. Fiedler and Günther [21], building on the model of [14, 15] and [9], provided a tailored game-based security model for PQXDH and proved its security. Both [6] and [21] 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 [20], but SIDH famously was broken [13, 35]. Proposals based on lattices were put forward by Brendel et al. [9] and Hashimoto et al. [25, 26], 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 [1]. 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 [2, 47]. Collins et al. [16] proposed K-Waay using Split-KEMs, which were initially proposed for use in X3DH in an early paper by Brendel et al. [10]. 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.

Online Version. Due to space limitations, we defer some appendices to the full version. This version is freely available online at https://eprint.iacr.org/2025/040.

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., [9, 14, 15, 16, 21, 25, 26]), 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 Def. 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 $L \in \mathsf{poly}(\lambda)$.

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

$\mathsf{BAKE.PreKeyBundleGen}(\mathsf{isk}_u) \xrightarrow{\$} (\vec{\mathsf{prek}}_u, \mathsf{st}_u)$**:** 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. Sec. 2.1.2). The state may for example include the associated (ephemeral) secret keys to public keys in $\vec{\mathsf{prek}}_u$.

$\mathsf{BAKE.Send}(\mathsf{isk}_s, \mathsf{ik}_r, \mathsf{prek}_{r,t}) \xrightarrow{\$} (K, \rho)$**:** 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.

$\mathsf{BAKE.Receive}(\mathsf{isk}_r, \mathsf{st}_r, \mathsf{ik}_s, t, \rho) \rightarrow (K', \mathsf{st}_r)$**:** 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 Sec. 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 Sec. 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 [9, 16, 26, 28]. 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 Algs. 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 Sec. 2.1.1, each prekey bundle is assigned a value called epoch. The initial prekey bundle generated by user u has $\mathsf{epoch} = 1$, 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 $\mathsf{ctr} = 0$ as a normal AKE instance.

[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 Alg. 1 for more detail.

What is unique to a BAKE protocol is that receivers can reuse the last-resort prekey bundle, i.e., instances with $\mathsf{ctr} = \perp$. 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 $\mathsf{ctr} \in \mathbb{N}$ 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 i^{th} ($i \in \mathbb{N}$) 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 Alg. 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 $\mathsf{iID}_{\perp,j}$ for $j \neq i$. Moreover, letting the instance identifiers $\mathsf{iID}_{\perp,i}$ and $\mathsf{iID}_{\perp,j}$ share the same base instance identifier $\overline{\mathsf{iID}}_\perp$ allows to succinctly define security as we show in Sec. 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:

- $\mathsf{role}[\mathsf{iID}] \in \{\mathsf{sender}, \mathsf{receiver}\}$ records the instance's role, i.e., whether the instance acts as the sender or the receiver.
- $(\mathsf{Sender}[\mathsf{iID}], \mathsf{Receiver}[\mathsf{iID}]) \in (\mathcal{U} \cup \{\perp\}) \times \mathcal{U}$ records the identities of the sender and the receiver relative to the instance iID. $\mathsf{Sender}[\mathsf{iID}] = \perp$ captures the fact that the sender is undefined when the receiver creates the prekey bundles.
- $\mathsf{prek}[\mathsf{iID}]$ records the prekey used by the instance iID.
- $\mathsf{prekidx}[\mathsf{iID}]$ records the index of the prekey used by the *receiver* instance iID (i.e., $\mathsf{role}[\mathsf{iID}] = \mathsf{receiver}$) in the corresponding prekey bundle. When $\mathsf{role}[\mathsf{iID}] = \mathsf{sender}$, the sender is not assumed to know the index of the receiver's prekey bundle, i.e., $\mathsf{prekidx}[\mathsf{iID}] = \epsilon$.
- $\mathsf{epoch}[\mathsf{iID}]$ 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., $\mathsf{epoch}[\mathsf{iID}] = \epsilon$ when $\mathsf{role}[\mathsf{iID}] = \mathsf{sender}$.
- $\mathsf{prekreuse}[\mathsf{iID}]$ records the number of time a last-resort prekey of a receiver instance has being reused. Specifically, we have $\mathsf{prekreuse}[\mathsf{iID}] \neq \epsilon$ only for $\mathsf{iID} \in \mathbb{N} \times \{\perp\}$.
- $\rho[\mathsf{iID}]$ records the handshake message used by iID.
- $\mathsf{key}[\mathsf{iID}]$ 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 $\mathsf{key}[\mathsf{iID}] = \epsilon$ for $\mathsf{iID} \in \mathbb{N} \times \{\perp\}$.

For more detail, we refer the readers to Algs. 1 and 2. While the game records more information associated to iID, we postpone their explanations to Sec. 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 Alg. 1 and define the advantage of a *passive* adversary $\mathcal{P}$ as

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

We say a BAKE protocol is *correct* if $\mathsf{Adv}^{\mathsf{CORR}}_{\mathsf{BAKE},\mathcal{P}}(1^\lambda) = \mathsf{negl}(\lambda)$ 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 $\mathsf{key}[\mathsf{iID}]$ 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 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 Secs. 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 Secs. 3.4 and 3.5.

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

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

[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.

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. 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 and Partner, used internally by safe. 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.

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 $\mathsf{key}[\mathsf{iID}] \neq \bot$ 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 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 iID' is an *origin instance* to the sender instance iID, 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 iID to be the receiver instance iID' such that $\mathsf{prek}[\mathsf{iID}] = \mathsf{prek}[\mathsf{iID}']$; in the AKE literature, this corresponds to setting the origin instance identifier as the first message of a two-round AKE protocol [17, 18, 28, 39, 40]. More generally, we use an *origin function* Φ_{origin} and say that iID' is the origin instance of iID if $\Phi_{\mathsf{origin}}(\mathsf{iID}) = \Phi_{\mathsf{origin}}(\mathsf{iID}')$. In some cases where $\mathsf{prek}[\mathsf{iID}]$ contains malleable components unnecessary for the secrecy of the session key (e.g., $\mathsf{prek}[\mathsf{iID}]$ contains a non-strongly unforgeable signature), this general definition captures security more appropriately (see Li and Schäge [34] 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 $\mathsf{iID}' \in \mathbb{N} \times (\{0, \bot\} \cup \mathbb{N})$ is an *origin instance* of $\mathsf{iID} \in \mathbb{N} \times (\{0, \bot\} \cup \mathbb{N})$ if the predicate $\mathsf{Origin}(\mathsf{iID}, \mathsf{iID}')$ defined below holds true:

$$\begin{aligned} &[\![\mathsf{Receiver}[\mathsf{iID}] = \mathsf{Receiver}[\mathsf{iID}']]\!] \\ \wedge\ &[\![(\mathsf{role}[\mathsf{iID}], \mathsf{role}[\mathsf{iID}']) = (\mathsf{sender}, \mathsf{receiver})]\!] \\ \wedge\ &[\![\Phi_{\mathsf{origin}}(\mathsf{iID}) = \Phi_{\mathsf{origin}}(\mathsf{iID}')]\!]. \end{aligned}$$

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 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 $\mathsf{iID}, \mathsf{iID}' \in \mathbb{N} \times (\{0, \bot\} \cup \mathbb{N})$ are *partners* if the predicate $\mathsf{Partner}(\mathsf{iID}, \mathsf{iID}')$ defined below holds true:

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

We note that the partnering definition captures unknown key share attacks [8]. 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 $\mathsf{iID}_\bot = (\overline{\mathsf{iID}}, \bot)$ (i.e., a receiver instance associated with a last-resort prekey bundle) cannot be partnered with any other instance as $\mathsf{key}[\mathsf{iID}_\bot] = \epsilon$ 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 [11], and as such, we opt to use the simpler definition. This comes with the benefit of the partner definition being much more

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, $\mathcal{O}^\star$ denotes the set $\{ \mathcal{O}_{\mathsf{PubNewPrekeyBundle}}, \mathcal{O}_{\mathsf{Send}}, \mathcal{O}_{\mathsf{Receive}} \}$, and $\mathcal{O}$ denotes the set of all oracles defined in Alg. 2. Additionally, $\mathsf{mode} \in \{ \mathsf{KIND}, \mathsf{MATCH} \}$.

function $\mathsf{Game}^{\mathsf{CORR}}_{\mathsf{BAKE},\mathcal{P}}(1^\lambda)$
  $\mathcal{S}_{\mathsf{ilD}} := \emptyset$ ▷ *Admin variable for $\mathcal{O}^\star$: Set of existing instances.*
  $\mathsf{NumilD} := 0$ ▷ *Admin variable: Number of instances.*
  for user $u \in \mathcal{U}$ **do**
    ▷ *Initialize epoch and counter.* ◁
    $(\mathsf{epoch}_u, \mathsf{ctr}_u) := (0, 0)$
    $(\mathsf{ik}_u, \mathsf{isk}_u) \xleftarrow{\$} \mathsf{BAKE.IdKeyGen}(1^\lambda)$
  $1 \leftarrow \mathcal{P}^{\mathcal{O}^\star}((\mathsf{ik}_u)_{u \in \mathcal{U}})$ ▷ *$\mathcal{P}$ always terminates with 1*
  for $(\mathsf{ilD}, \mathsf{ilD}') \in \mathcal{S}_{\mathsf{ilD}} \times \mathcal{S}_{\mathsf{ilD}}$ **do**
    $\mathsf{cond} := [\![\mathsf{role}[\mathsf{ilD}] \neq \mathsf{role}[\mathsf{ilD}']]\!]$
      $\wedge\ [\![\mathsf{sender}[\mathsf{ilD}] = \mathsf{sender}[\mathsf{ilD}']]\!]$
      $\wedge\ [\![\mathsf{receiver}[\mathsf{ilD}] = \mathsf{receiver}[\mathsf{ilD}']]\!]$
      $\wedge\ [\![\mathsf{prekidx}[\mathsf{ilD}] = \mathsf{prekidx}[\mathsf{ilD}']]\!]$
      $\wedge\ [\![\rho[\mathsf{ilD}] = \rho[\mathsf{ilD}']]\!] \wedge [\![\mathsf{key}[\mathsf{ilD}] \neq \mathsf{key}[\mathsf{ilD}']]\!]$
    if cond **then**
      return 1
  return 0

function $\mathsf{Game}^{\mathsf{mode}}_{\mathsf{BAKE},\mathcal{A}}(1^\lambda)$
  $b \xleftarrow{\$} \{0, 1\}$
  $\mathcal{S}_{\mathsf{ilD}} := \emptyset$ ▷ *Admin variable: Set of existing instances.*
  $\mathsf{NumilD} := 0$ ▷ *Admin variable: Number of instances.*
  $\mathsf{ilD}^* := \bot$ ▷ *Tested instance.*
  for user $u \in \mathcal{U}$ **do**
    ▷ *Initialize epoch and counter.* ◁
    $(\mathsf{epoch}_u, \mathsf{ctr}_u) := (0, 0)$
    $(\mathsf{ik}_u, \mathsf{isk}_u) \xleftarrow{\$} \mathsf{BAKE.IdKeyGen}(1^\lambda)$
  $\mathsf{st} \xleftarrow{\$} \mathcal{A}_1^{\mathcal{O}}((\mathsf{ik}_u)_{u \in \mathcal{U}})$
  $b' \xleftarrow{\$} \mathcal{A}_2(\mathsf{st})$
  if $[\![\mathsf{mode} = \mathsf{KIND}]\!]$ **then** ▷ *Key ind. game.*
    if $[\![\mathsf{ilD}^* = \bot]\!] \vee [\![\mathsf{safe}(\mathsf{ilD}^*) = \mathsf{false}]\!]$ **then**
      $b' \xleftarrow{\$} \{0, 1\}$
    return $[\![b = b']\!]$
  else ▷ mode = MATCH, *Match soundness game*
    return $[\![\mathsf{Match}(\mathcal{S}_{\mathsf{ilD}}) = \mathsf{false}]\!]$

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 $\Phi_{\mathsf{origin}}(\mathsf{ilD})$ will have implications when defining what an "optimally" secure BAKE protocol is. See Sec. 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 $\Phi_{\mathsf{origin}}(\mathsf{ilD}) = \bot$ for any instance ilD, 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 $\mathcal{S}_{\mathsf{ilD}} \subset \mathbb{N} \times (\{0, \bot\} \cup \mathbb{N})$ be the set of instances generated in the game (cf. Alg. 1). The predicate $\mathsf{Match}(\mathcal{S}_{\mathsf{ilD}})$ holds true if and only if for any $\mathsf{ilD}, \mathsf{ilD}', \mathsf{ilD}'' \in \mathcal{S}_{\mathsf{ilD}}$, we have the following.

1. If $\mathsf{Partner}(\mathsf{ilD}, \mathsf{ilD}') = \mathsf{true}$, then either $\mathsf{Origin}(\mathsf{ilD}, \mathsf{ilD}') = \mathsf{true}$ or $\mathsf{Origin}(\mathsf{ilD}', \mathsf{ilD}) = \mathsf{true}$.
2. If $\mathsf{Partner}(\mathsf{ilD}, \mathsf{ilD}') = \mathsf{Partner}(\mathsf{ilD}, \mathsf{ilD}'') = \mathsf{true}$, then $\mathsf{ilD}' = \mathsf{ilD}''$.
3. If $\mathsf{Origin}(\mathsf{ilD}, \mathsf{ilD}') = \mathsf{Origin}(\mathsf{ilD}, \mathsf{ilD}'') = \mathsf{true}$, then $\mathsf{base}(\mathsf{ilD}') = \mathsf{base}(\mathsf{ilD}'')$. Moreover, we have two cases:
   (a) If $\mathsf{ilD}' \in \mathbb{N} \times \{0\}$ (i.e., a receiver instance associated with a one-time prekey bundle), then $\mathsf{ilD}' = \mathsf{ilD}''$.
   (b) Otherwise, if $\mathsf{ilD}' \in \mathbb{N} \times (\{\bot\} \cup \mathbb{N}^*)$ (i.e., a receiver instance associated with a last-resort prekey bundle), then there exists a unique instance $\mathsf{ilD}_\bot = (\mathsf{base}(\mathsf{ilD}'), \bot) \in \mathcal{S}_{\mathsf{ilD}}$, and we have

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

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

Items 3a and 3b add additional checks to Item 3. The first, Item 3a, demands that if ilD′ 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 ilD′ used a last-resort key bundle (i.e., $\mathsf{ctr}' \in \{\bot\} \cup \mathbb{N}$), 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 $\mathsf{ilD}_\bot$ that must have been generated during BAKE.PreKeyBundleGen and all other instances are of the form $\{(\mathsf{base}(\mathsf{ilD}'), i)\}_{i \in [\mathsf{prekreuse}[\mathsf{ilD}_\bot]]}$, where recall $\mathsf{prekreuse}[\mathsf{ilD}_\bot]$ 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.

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., $u, s, r \in \mathcal{U}$.

function $O_{\mathsf{PubNewPrekeyBundle}}(u)$
  $\mathsf{epoch}_u \leftarrow \mathsf{epoch}_u + 1$ ▷ *Move to next epoch*
  $\mathsf{ctr}_u \leftarrow 0$ ▷ *Reset counter*
  $(\vec{\mathsf{prek}}_u, \mathsf{st}_u) \xleftarrow{\$} \mathsf{BAKE.PreKeyBundleGen}(\mathsf{isk}_u)$
  ▷ *Assign instances to prekeys* ◁
  for $t \in [L] \cup \{\bot\}$ **do**
    ▷ *Create new base instance* ◁
    $\mathsf{NumIID} \leftarrow \mathsf{NumIID} + 1$
    if $[\![t \neq \bot]\!]$ **then** ▷ *One-time prekey bundle*
      $\mathsf{iID} \coloneqq (\mathsf{NumIID}, 0)$
    else ▷ *Last-resort prekey bundle*
      $\mathsf{iID} \coloneqq (\mathsf{NumIID}, \bot)$
      $\mathsf{prekreuse}[\mathsf{iID}] \leftarrow 0$ ▷ *Record number of reuses*
    $\mathcal{S}_{\mathsf{iID}} \leftarrow \mathcal{S}_{\mathsf{iID}} \cup \{\mathsf{iID}\}$
    $(\mathsf{role}[\mathsf{iID}], \mathsf{Sender}[\mathsf{iID}], \mathsf{Receiver}[\mathsf{iID}]) \leftarrow (\mathsf{receiver}, \bot, u)$
    $(\mathsf{prek}[\mathsf{iID}], \mathsf{prekidx}[\mathsf{iID}]) \leftarrow (\vec{\mathsf{prek}}_u[t], t)$
    $\mathsf{epoch}[\mathsf{iID}] \leftarrow \mathsf{epoch}_u$
  return $\vec{\mathsf{prek}}_u$

function $O_{\mathsf{Send}}(s, r, \mathsf{prek})$
  $\mathsf{NumIID} \leftarrow \mathsf{NumIID} + 1$ ▷ *Create new base instance*
  $\mathsf{iID} \coloneqq (\mathsf{NumIID}, 0)$
  $\mathcal{S}_{\mathsf{iID}} \leftarrow \mathcal{S}_{\mathsf{iID}} \cup \{\mathsf{iID}\}$
  $(K, \rho) \xleftarrow{\$} \mathsf{BAKE.Send}(\mathsf{isk}_s, \mathsf{ik}_r, \mathsf{prek})$
  $(\mathsf{role}[\mathsf{iID}], \mathsf{Sender}[\mathsf{iID}], \mathsf{Receiver}[\mathsf{iID}]) \leftarrow (\mathsf{sender}, s, r)$
  $(\mathsf{prek}[\mathsf{iID}], \rho[\mathsf{iID}]) \leftarrow (\mathsf{prek}, \rho)$
  $\mathsf{key}[\mathsf{iID}] \leftarrow K$
  $\mathsf{PeerCorr}[\mathsf{iID}] \leftarrow \mathsf{RevIK}[r]$ ▷ *Check if peer's* isk *is corrupted*
  return ρ

function $O_{\mathsf{Receive}}(r, s, \rho)$
  $\mathsf{ctr}_r \leftarrow \mathsf{ctr}_r + 1$
  if $[\![\mathsf{ctr}_r \leq L]\!]$ **then** ▷ *One-time prekey exists*
    $t \coloneqq \mathsf{ctr}_r$
    Fetch iID s.t. $(\mathsf{epoch}[\mathsf{iID}], \mathsf{prekidx}[\mathsf{iID}]) = (\mathsf{epoch}_u, t)$
    ▷ *Unique*
  else ▷ *One-time prekey depleted*
    $t \coloneqq \bot$
    Fetch $\mathsf{iID}_\bot$ s.t. $(\mathsf{epoch}[\mathsf{iID}_\bot], \mathsf{prekidx}[\mathsf{iID}_\bot]) = (\mathsf{epoch}_u, t)$
    ▷ *Unique*
  $(K', \mathsf{st}_r) \leftarrow \mathsf{BAKE.Receive}(\mathsf{isk}_r, \mathsf{st}_r, \mathsf{ik}_s, t, \rho)$
  if $[\![t = \bot]\!]$ **then**
    ▷ *Create new completed last-resort instance* ◁
    $\mathsf{prekreuse}[\mathsf{iID}] \leftarrow \mathsf{prekreuse}[\mathsf{iID}] + 1$ ▷ $\mathsf{prekreuse}[\mathsf{iID}] = \mathsf{ctr}_r - L$
    $\mathsf{iID} \coloneqq (\mathsf{base}(\mathsf{iID}_\bot), \mathsf{prekreuse}[\mathsf{iID}])$
    $\mathcal{S}_{\mathsf{iID}} \leftarrow \mathcal{S}_{\mathsf{iID}} \cup \{\mathsf{iID}\}$
    ▷ *Copy information into new* iID ◁
    $\mathsf{role}[\mathsf{iID}] \leftarrow \mathsf{role}[\mathsf{iID}_\bot]$
    $\mathsf{Receiver}[\mathsf{iID}] \leftarrow \mathsf{Receiver}[\mathsf{iID}_\bot]$
    $\mathsf{prek}[\mathsf{iID}] \leftarrow \mathsf{prek}[\mathsf{iID}_\bot]$
    $\mathsf{prekidx}[\mathsf{iID}] \leftarrow \mathsf{prekidx}[\mathsf{iID}_\bot]$
    $\mathsf{epoch}[\mathsf{iID}] \leftarrow \mathsf{epoch}[\mathsf{iID}_\bot]$
  ▷ *Record completed instance* ◁
  $(\mathsf{Sender}[\mathsf{iID}], \rho[\mathsf{iID}], \mathsf{key}[\mathsf{iID}]) \leftarrow (s, \rho, K')$
  ▷ *Check if peer's* isk *is corrupted* ◁
  $\mathsf{PeerCorr}[\mathsf{iID}] \leftarrow \mathsf{RevIK}[s]$
  ▷ *Check if own* st *is corrupted* ◁
  $\mathsf{StateRev}[\mathsf{iID}] \leftarrow \mathsf{RevUserSt}[r]$
  ▷ *Inform success of* BAKE.Receive ◁
  return $[\![K' \neq \bot]\!]$

function $O_{\mathsf{RevSessKey}}(\mathsf{iID})$
  require $[\![\mathsf{base}(\mathsf{iID}) \leq \mathsf{NumIID}]\!]$ ▷ *Existing instance*
  $\mathsf{RevSessKey}[\mathsf{iID}] \leftarrow \mathsf{true}$
  return $\mathsf{key}[\mathsf{iID}]$

function $O_{\mathsf{RevIK}}(u)$
  $\mathsf{RevIK}[u] \leftarrow \mathsf{true}$
  return isk_u

function $O_{\mathsf{RevState}}(u)$
  ▷ st_u *for* epoch_u *can't have been corrupted before* ◁
  require $[\![\mathsf{UserStCtr}[u, \mathsf{epoch}_u] = \epsilon]\!]$
  $\mathsf{RevUserSt}[u, \mathsf{epoch}_u] \leftarrow \mathsf{true}$
  $\mathsf{UserStCtr}[u, \mathsf{epoch}_u] \leftarrow \mathsf{ctr}_u$
  return st_u

function $O_{\mathsf{Test}}(\mathsf{iID})$
  require $[\![\mathsf{base}(\mathsf{iID}) \leq \mathsf{NumIID}]\!]$ ▷ *Existing instance*
  require $[\![\mathsf{iID}^* = \bot]\!] \wedge [\![\mathsf{key}[\mathsf{iID}] \neq \bot]\!]$
  $\mathsf{iID}^* \leftarrow \mathsf{iID}$
  $K_0 \coloneqq \mathsf{key}[\mathsf{iID}]$; $K_1 \xleftarrow{\$} \mathcal{K}$
  return K_b

Definition 7 (Match Soundness)**.** We define the match soundness game in Alg. 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

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

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 [5, 6, 21, 31].*[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 Sec. 4), seemingly never covered by any game-based security model. The one exception being [29], covering this using symbolic analysis.*

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

As discussed in Sec. 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 $\mathsf{safe}_{\mathsf{BAKE}}$. This allows us to define the "optimal" key indistinguishability security as it provides the *maximum* attack freedom to the adversary.

Keeping track of adversary's knowledge. To define predicate $\mathsf{safe}_{\mathsf{BAKE}}$, 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] $\in \{\mathsf{true}, \mathsf{false}\}$ records whether the session key of the instance iID is revealed.

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

RevUserSt[u, epoch] $\in \mathbb{N} \cup \{\mathsf{false}\}$ 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 = RevUserSt[u, 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.

PeerCorr[iID] $\in \{\mathsf{true}, \mathsf{false}\}$ 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] $\in \{\mathsf{true}, \mathsf{false}\}$ 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 Rem. 2 for more details).

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

```
1:  function safe_protocol(iID*)
2:      (s*, r*) ← (Sender[iID*], Receiver[iID*])
3:      ▷ Origin instances ◁
4:      𝔒(iID*) ← { iID ∈ S_iID | Origin(iID*, iID) = true }
5:      ▷ Partner instances ◁
6:      𝔓(iID*) ← { iID ∈ S_iID | Partner(iID*, iID) = true }
7:      if ⟦∀ Attack ∈ Tab. 1 : Attack(iID*) = false⟧ then
8:          ▷ A did not execute any unavoidable attacks ◁
9:          return true
10:     else if ⟦protocol ∈ { X3DH, PQXDH }⟧ ∧ ⟦∀ Attack ∈ Tab. 3 \
        { Attack-6&7 } : Attack(iID*) = false⟧ then
11:         ▷ X3DH/PQXDH A does not execute classical attacks ◁
12:         return true
13:     if ⟦protocol = PQXDH⟧ ∧ ⟦Attack-6(iID*) = false⟧ ∧
        ⟦Attack-7(iID*) = false⟧ then
14:         ▷ A does not execute a specific HNDL attack in Tab. 3 ◁
15:         return true
16:     return false
```

Predicate $\mathsf{safe}_{\mathsf{BAKE}}$ ⇐ Unavoidable attacks against any BAKE protocol. We first specify the set of unavoidable attacks that no BAKE protocol can prevent in Tab. 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. Def. 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

[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. See [31, Sec. 4.2] and Sec. 4 for more details.

Table 1: Minimal set of unavoidable attacks against any BAKE protocol. Each row denotes the predicate Attack-xx(ilD*) returning the logical AND of the conditions specified in each column. Variables s^* = Sender[ilD*] and r^* = Receiver[ilD*] denote the sender and receiver relative to tested instance ilD*; one of them is the identity of the user in ilD* 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 ilD*					Adversary's activities							
Attack	role[ilD*]	$\lvert\mathfrak{O}(\text{ilD}^*)\rvert$	$\lvert\mathfrak{P}(\text{ilD}^*)\rvert$	$\exists \text{ilD} \in \mathfrak{O}(\text{ilD}^*)$: prekidx[ilD]	prekidx[ilD*]	PeerCorr[ilD*]	StateRev[ilD*]	RevSessKey[ilD*]	$\exists \text{ilD} \in \mathfrak{P}(\text{ilD}^*)$: RevSessKey[ilD]	RevIK[s^*]	RevIK[r^*]	RevUserSt[r^*, ep*]	**Explanation**
1	—	—	—	—	—	—	—	true	—	—	—	—	key[ilD*] is revealed.
2	—	—	1	—	—	—	—	—	true	—	—	—	The session key key[ilD] of the partner instance of ilD* is revealed.
3-1	sender	1	—	$\leq L$	—	—	—	—	—	—	true	< prekidx[ilD]	When the same one-time prekey is used by the sender and the receiver, both isk_{r^*} and $\text{st}_{r^*,\text{ep}^*}$ are revealed before the one-time prekey is used by r^*.
3-2	receiver	—	1	—	$\leq L$	—	—	—	—	—	true	< prekidx[ilD*]	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 $\text{st}_{r^*,\text{ep}^*}$ 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: $\mathfrak{O}$(ilD) and $\mathfrak{P}$(ilD) give the set of origin and partner sessions, respectively, for ilD (see Alg. 3).

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 ilD* is owned by a sender (resp. receiver), it has an origin (resp. a partner) instance ilD, 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 ilD* has no origin or partner instance. Consider an adversary that corrupts the identity secret key of the peer of ilD* before ilD* 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).

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 $\text{safe}_{\text{BAKE}}$)**.** We define the *optimal* predicate $\text{safe}_{\text{BAKE}}$ for any BAKE protocol in Alg. 3 based on the set of unavoidable attacks in Tab. 1.

The rows of Tab. 1 work as predicates that return the logical AND of the conditions specified in each column. Predicate $\text{safe}_{\text{BAKE}}$ 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 $\text{safe}_{\text{BAKE}}$ is parameterized by the predicates Origin and Partner. As mentioned in Sec. 3.3.2, while predicate Partner is defined unambiguously by Def. 5 between different protocols, predicate Origin has some ambiguity due to our usage of the origin function Φ_{origin} (see Def. 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 Tab. 1. Taking the counter-positive, we consider every attack strategy for which predicate $\text{safe}_{\text{BAKE}}$ 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

Tab. 1 is false. As this is a useful tool for any security proof, we formally depict this in Tab. 2. Specifically, if the predicate $\mathsf{safe}_{\mathsf{BAKE}}$ evaluates to true, then the adversary must take one of the attack strategies shown in Tab. 2. Note that all standard AKE security proofs either implicitly or explicitly follow this proof strategy [22, 24, 25, 26, 27, 28, 38].

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

Maximal exposure attack [12, 22, 30, 33]: 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. Rem. 2), we always allow the adversary to reveal the sender's identity secret and its user-state.

Key-compromise impersonation (KCI) attack [7, 30]: 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 [12, 19]: 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 Tab. 3.

The first attack assumes a harvest-now-decrypt-later (HNDL) adversary 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 [36, Sec. 4.6] and [31, 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 $\mathsf{safe}_{\mathsf{X3DH}}$ and $\mathsf{safe}_{\mathsf{PQXDH}}$.

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

3.5.1 Avoidable attacks.

Similarly to $\mathsf{safe}_{\mathsf{BAKE}}$, 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 Tab. 4.

Notice that Types 2 and 3 are identical to the allowed attack strategies for the optimal BAKE protocol (cf. Tab. 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.

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

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

	Status of the tested ilD*					Adversary's activities							
Attack	role[ilD*]	$\lvert\Sigma(\text{ilD}^*)\rvert$	$\lvert\mathfrak{P}(\text{ilD}^*)\rvert$	$\exists \text{ilD} \in \Sigma(\text{ilD}^*) :$ prekidx[ilD]	prekidx[ilD*]	PeerCorr[ilD*]	StateRev[ilD*]	RevSessKey[ilD*]	$\exists \text{ilD} \in \mathfrak{P}(\text{ilD}^*) :$ RevSessKey[ilD]	RevIK[$\cdot^*$]	RevIK[r^*]	RevUserSt[r^*, ep*]	Explanation
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	—	$\leq L$	—	—	—	false	false	—	—	$\geq$ prekidx[ilD]	Reveal $\mathsf{st}_{r^*,\mathsf{ep}^*}$ *after* the one-time prekey is used by r^* and reveal isk_{r^*}.
2-2	receiver	—	1	—	$\leq L$	—	—	false	false	—	—	$\geq$ prekidx[ilD*]	Same as 2-1
3-1	sender	1	—	$\perp$	—	—	—	false	false	—	—	false	Only reveal isk_{r^*} if the last-resort key is used.
3-2	receiver	—	1	—	$\perp$	—	—	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

4 Signal's X3DH and PQXDH

The X3DH protocol [36] was proposed in 2016 by Marlinspike and Perrin based on the Triple Diffie–Hellman AKE protocol [32]. In 2023, Signal introduced PQXDH to protect the Signal handshake protocol against harvest now, decrypt later attacks [31]. 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 Algs. 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. Alg. 5, Lns. 6 to 13).

While our description of X3DH and PQXDH closely follows Signal's documentation [31, 36], we incorporated several minor modifications based on discussions with Signal developers that may be included in future updates [45].

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" [31], 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 output from the AKE. 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 (DR) algorithm with the PQXDH handshake, using DR 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 DR, 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 DR. 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. Rem. 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 Alg. 6, Ln. 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 replay at-

[7] This was also noted by [14], who also heavily refer to source code.

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 Tab. 1 for the notation used in this table.

	Status of the tested ilD*					Adversary's activities								
Attack	role[ilD*]	$\lvert\mathfrak{S}(\text{ilD}^*)\rvert$	$\lvert\mathfrak{P}(\text{ilD}^*)\rvert$	$\exists \text{ilD} \in \mathfrak{S}(\text{ilD}^*) : \text{prekidx[ilD]}$	prekidx[ilD*]	$\mathcal{A}_2$ is quantum	PeerCorr[ilD*]	StateRev[ilD*]	RevSessKey[ilD*]	$\exists \text{ilD} \in \mathfrak{P}(\text{ilD}^*) : \text{RevSessKey[ilD]}$	RevIK[ι^*]	RevIK[r^*]	RevUserSt[r^*, ep*]	Explanation
6-1	sender	1	—	$\leq L$	—	true	—	—	—	—	—	—	< prekidx[ilD]	**PQXDH**: The KEM prekey is known to *quantum* adversaries.
6-2	receiver	—	1	—	$\leq L$	true	—	—	—	—	—	—	< prekidx[ilD]	Same as 6-1
7-1	sender	1	—	$\perp$	—	true	—	—	—	—	—	—	$\neq$ false	**PQXDH**: The KEM prekey is known to *quantum* adversaries.
7-2	receiver	—	1	—	$\perp$	true	—	—	—	—	—	—	$\neq$ 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*.

tack 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. Def. 6, Item 2). Replay attacks appear to have been overlooked in prior analyses (although mentioned in the documentation [31, Sec. 4.2], and covered by a symbolic analysis [29]), 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 Algs. 5 and 6, the Signal X3DH protocol description, and the `libsignal` implementation [46] in App. B in the full version. 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 [41].

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.

[8] This DH-only mode will eventually be disabled [45].

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., the TLS 1.3 handshake does [44]. 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 App. C.2 in the full version.

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 $\text{ilD} \in \mathcal{S}_{\text{ilD}}$ (i.e., the set of all instances created during the game) with $\text{prek[ilD]} \neq \perp$, we define the origin function as $\Phi^{\text{Signal}}_{\text{origin}}(\text{ilD}) := \text{prek[ilD]}$.

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

	Status of the tested ilD*					Adversary's activities								
Attack	role[ilD*]	$\lvert\Sigma(\text{ilD}^*)\rvert$	$\lvert\mathfrak{P}(\text{ilD}^*)\rvert$	$\exists \text{ilD} \in \Sigma(\text{ilD}^*)$: prekidx[ilD]	prekidx[ilD*]	$\mathcal{A}_2$ is quantum	PeerCorr[ilD*]	StateRev[ilD*]	RevSessKey[ilD*]	$\exists \text{ilD} \in \mathfrak{P}(\text{ilD}^*)$: RevSessKey[ilD]	RevIK[s^*]	RevIK[r^*]	RevUserSt[r^*, ep*]	**Explanation**
1-1	sender	1	—	—	—	false	—	—	false	false	—	false	—	Only for classical adversary, reveal user-state $\mathsf{st}_{r^*,\mathsf{ep}^*}$ but not secret key isk_{r^*}.
1-2	receiver	—	1	—	—	false	—	—	false	false	—	false	—	Same as 1-1
2-1	sender	1	—	$\leq L$	—	—	—	—	false	false	—	—	$\geq$ prekidx[ilD]	Reveal $\mathsf{st}_{r^*,\mathsf{ep}^*}$ *after* the one-time prekey is used by r^* and reveal isk_{r^*}.
2-2	receiver	—	1	—	$\leq L$	—	—	—	false	false	—	—	$\geq$ prekidx[ilD*]	Same as 2-1
3-1	sender	1	—	$\perp$	—	—	—	—	false	false	—	—	false	Only reveal isk_{r^*} if the last-resort key is used.
3-2	receiver	—	1	—	$\perp$	—	—	—	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.

As mentioned in Sec. 3.3.2, this is one of the most common ways to define an origin instance in the AKE literature [17, 18, 28, 39, 40]. 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. PQXDH *is match sound against a harvest-now-decrypt-later adversary with respect to the predicate* Match *(cf. Def. 6) and origin function* $\Phi^{\mathsf{Signal}}_{\mathsf{origin}}$ *(cf. Def. 10).*

Proof. We defer the proof to App. C.1.2 in the full version. 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 Sec. 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. Def. 9).*

Proof. We defer the proof to App. C.1.3 in the full version. As explained in Sec. 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. Tab. 4). We prove the advantage is negligible for each of these strategies. □

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. Def. 8). The core design of RingXKEM is inspired from the *deniable* AKE protocol by Hashimoto et al. [25, 26] 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 Algs. 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 t^{th} ($t \in [L] \cup \{\perp\}$) 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

Algorithm 4 PQX3DH identity key and prekey bundle generation algorithms.

function PQX3DH.IdKeyGen(1^λ)
  $\overline{\mathsf{isk}} \xleftarrow{\$} \mathbb{Z}_p$; $\overline{\mathsf{ik}} := [\overline{\mathsf{isk}}]G$
  $(\mathsf{vk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{Sig.KeyGen}(1^\lambda)$
  return $(\mathsf{ik} := (\overline{\mathsf{ik}}, \mathsf{vk}), \mathsf{isk} := (\overline{\mathsf{isk}}, \mathsf{sk}))$

function PQX3DH.PreKeyBundleGen(isk_u)
  $(\overline{\mathsf{isk}}_u, \mathsf{sk}_u) \leftarrow \mathsf{isk}_u$
  $D_{\mathsf{prek}}, D_{\rho_\perp} := \emptyset$ ▷ *Initialize empty lists*
  ▷ *Generate what Signal calls the* signed prekey ◁
  $\mathsf{spksec}_u \xleftarrow{\$} \mathbb{Z}_p$; $\mathsf{spk}_u := [\mathsf{spksec}]G$
  $\sigma_{\mathsf{spk}_u} \xleftarrow{\$} \mathsf{Sig.Sign}(\mathsf{sk}_u, \mathsf{spk}_u)$
  ▷ *Create the* L one-time *prekey bundles* ◁
  for $t \in [L]$ **do**
    $\mathsf{osk}_{u,t} \xleftarrow{\$} \mathbb{Z}_p$; $\mathsf{opk}_{u,t} := [\mathsf{osk}_{u,t}]G$
    $(\mathsf{ek}_{u,t}, \mathsf{dk}_{u,t}) \xleftarrow{\$} \mathsf{KEM.KeyGen}(1^\lambda)$
    $\sigma_{\mathsf{ek}_{u,t}} \xleftarrow{\$} \mathsf{Sig.Sign}(\mathsf{sk}_u, \mathsf{ek}_{u,t})$
    $\mathsf{prek}_{u,t} := (\mathsf{spk}_u, \sigma_{\mathsf{spk}_u}, \mathsf{opk}_{u,t}, \mathsf{ek}_{u,t}, \sigma_{\mathsf{ek}_{u,t}})$
    $D_{\mathsf{prek}}[t] \leftarrow (\mathsf{prek}_{u,t}, (\mathsf{spksec}_u, \mathsf{osk}_{u,t}, \mathsf{dk}_{u,t}))$
  ▷ *Set up the* last-resort *prekey bundle* ◁
  $(\mathsf{ek}_{u,\perp}, \mathsf{dk}_{u,\perp}) \xleftarrow{\$} \mathsf{KEM.KeyGen}(1^\lambda)$
  $\sigma_{\mathsf{ek}_{u,\perp}} \xleftarrow{\$} \mathsf{Sig.Sign}(\mathsf{sk}_u, \mathsf{ek}_{u,\perp})$
  $\mathsf{prek}_{u,\perp} := (\mathsf{spk}_u, \sigma_{\mathsf{spk}_u}, \perp, \mathsf{ek}_{u,\perp}, \sigma_{\mathsf{ek}_{u,\perp}})$
  $D_{\mathsf{prek}}[\perp] \leftarrow (\mathsf{prek}_{u,\perp}, (\mathsf{spksec}_u, \perp, \mathsf{dk}_{u,\perp}))$
  return $(\vec{\mathsf{prek}}_u, \mathsf{st}_u := (D_{\mathsf{prek}}, D_{\rho_\perp}))$

Algorithm 5 PQX3DH sender algorithms. prek is not indexed by $t \in [L] \cup \{\perp\}$ as they are oblivious to the sender.

function PQX3DH.Send($\mathsf{isk}_s, \mathsf{ik}_r, \mathsf{prek}_r$)
  $(\overline{\mathsf{isk}}_s, \mathsf{sk}_s) \leftarrow \mathsf{isk}_s$; $(\overline{\mathsf{ik}}_r, \mathsf{vk}_r) \leftarrow \mathsf{ik}_r$
  $\left(\mathsf{spk}_r, \sigma_{\mathsf{spk}_r}, \mathsf{opk}_r, \mathsf{ek}_r, \sigma_{\mathsf{ek}_r}\right) \leftarrow \mathsf{prek}_r$ ▷ $\mathsf{opk}_r = \perp$ *if* prek_r *is a last-resort key bundle*
  require $[\![\mathsf{Sig.Verify}(\mathsf{vk}_r, \mathsf{spk}_r, \sigma_{\mathsf{spk}_r}) = 1]\!]$
  require $[\![\mathsf{Sig.Verify}(\mathsf{vk}_r, \mathsf{ek}_r, \sigma_{\mathsf{ek}_r}) = 1]\!]$
  $\mathsf{esk} \xleftarrow{\$} \mathbb{Z}_p$, $\mathsf{epk} := [\mathsf{esk}]G$
  $\mathsf{ss}_1 := [\overline{\mathsf{isk}}_s]\mathsf{spk}_r$
  $\mathsf{ss}_2 := [\mathsf{esk}]\overline{\mathsf{ik}}_r$
  $\mathsf{ss}_3 := [\mathsf{esk}]\mathsf{spk}_r$
  $\mathsf{ss} := \mathsf{ss}_1 \| \mathsf{ss}_2 \| \mathsf{ss}_3$
  if $[\![\mathsf{opk}_r \neq \perp]\!]$ **then** ▷ *One-time prekey bundle*
    $\mathsf{ss}_4 := [\mathsf{esk}]\mathsf{opk}_r$
    $\mathsf{ss} := \mathsf{ss}_1 \| \mathsf{ss}_2 \| \mathsf{ss}_3 \| \mathsf{ss}_4$
  $(\mathsf{ss}_{\mathsf{KEM}}, \mathsf{ct}) \xleftarrow{\$} \mathsf{KEM.Encaps}(\mathsf{ek}_r)$
  $\mathsf{content} := \mathsf{ik}_s \| \mathsf{ik}_r \| \mathsf{prek}_r \| \mathsf{epk} \| \mathsf{ct}$
  $K \| \tau_{\mathsf{conf}} := \mathsf{KDF}(\mathsf{ss} \| \mathsf{ss}_{\mathsf{KEM}}, \mathsf{content})$
  $\rho := (\mathsf{epk}, \mathsf{ct}, \tau_{\mathsf{conf}})$
  return (K, ρ)

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 $(\widehat{\mathsf{ek}}_t)_{t \in [L] \cup \{\perp\}}$ and only sign the digest root. We provide concrete numbers for this optimization in Sec. 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., [9, 14, 15, 16, 21, 25, 26], 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 $(\widehat{\mathsf{ek}}_t)_{t \in [L] \cup \{\perp\}}$ 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. 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 App. D in the full version.

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 Tab. 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 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.

Algorithm 6 PQX3DH receiver algorithms.

function PQX3DH.Receive($\text{isk}_r, \text{st}_r, \text{ik}_s, t, \rho$)
  $(\overline{\text{isk}}_r, \text{sk}_r) \leftarrow \text{isk}_r$; $(\overline{\text{ik}}_s, \text{vk}_s) \leftarrow \text{ik}_s$
  $(D_{\text{prek}}, D_{\rho_\perp}) \leftarrow \text{st}_r$
  if $[\![t \neq \perp]\!]$ **then** ▷ *One-time prekey bundle*
    require $[\![D_{\text{prek}}[t] \neq \perp]\!]$ ▷ *Check if unused.*
    $(\text{prek}_{r,t}, (\text{spksec}_r, \text{osk}_{r,t}, \widehat{\text{dk}}_{r,t})) \leftarrow D_{\text{prek}}[t]$
  else ▷ *Last-resort prekey bundle (i.e., $t = \perp$)*
    require $[\![\rho \notin D_{\rho_\perp}]\!]$ ▷ *Check ρ is not replayed.*
    $D_{\rho_\perp} \leftarrow D_{\rho_\perp} \cup \{ \rho \}$
    $(\text{prek}_{r,t}, (\text{spksec}_r, \perp, \widehat{\text{dk}}_{r,t})) \leftarrow D_{\text{prek}}[t]$
  $(\text{epk}, \widehat{\text{ct}}, \tau_{\text{conf}}) \leftarrow \rho$
  $\text{ss}_1 \coloneqq [\text{spksec}_r]\overline{\text{ik}}_s$; $\text{ss}_2 \coloneqq [\overline{\text{isk}}_r]\text{epk}$
  $\text{ss}_3 \coloneqq [\text{spksec}_r]\text{epk}$; $\text{ss} \coloneqq \text{ss}_1 \| \text{ss}_2 \| \text{ss}_3$
  if $[\![t \neq \perp]\!]$ **then** ▷ *One-time prekey bundle*
    $\text{ss}_4 \coloneqq [\text{osk}_{r,t}]\text{epk}$; $\text{ss} \coloneqq \text{ss}_1 \| \text{ss}_2 \| \text{ss}_3 \| \text{ss}_4$
  $\widehat{\text{ss}}_{\text{KEM}} \leftarrow \text{KEM.Decaps}(\widehat{\text{dk}}_{r,t}, \widehat{\text{ct}})$
  $\text{content} \coloneqq \text{ik}_s \| \text{ik}_r \| \text{prek}_{r,t} \| \text{epk} \| \widehat{\text{ct}}$
  $K \| \tau'_{\text{conf}} \coloneqq \text{KDF}(\text{ss} \| \widehat{\text{ss}}_{\text{KEM}}, \text{content})$
  require $[\![\tau_{\text{conf}} = \tau'_{\text{conf}}]\!]$
  ▷ *Delete prekey bundle if not last-resort* ◁
  if $[\![t \neq \perp]\!]$ **then** $D_{\text{prek}}[t] \leftarrow \perp$
  $\text{st}_r \leftarrow (D_{\text{prek}}, D_{\rho_\perp})$
  return (K, st_r)

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

function RingXKEM.IdKeyGen(1^λ)
  $(\text{ek}, \text{dk}) \xleftarrow{\$} \text{KEM.KeyGen}(1^\lambda)$
  $(\text{rvk}, \text{rsk}) \xleftarrow{\$} \text{RS.KeyGen}(1^\lambda)$
  return $(\text{ik} \coloneqq (\text{ek}, \text{rvk}), \text{isk} \coloneqq (\text{dk}, \text{rsk}))$

function RingXKEM.PreKeyBundleGen(isk_u)
  $(\text{dk}_u, \text{rsk}_u) \leftarrow \text{isk}_u$
  $D_{\text{kem}}, D_{\rho_\perp} \coloneqq \emptyset$ ▷ *Initialize empty lists*
  for $t \in [L] \cup \{\perp\}$ **do**
    $(\widehat{\text{ek}}_{u,t}, \widehat{\text{dk}}_{u,t}) \xleftarrow{\$} \text{KEM.KeyGen}(1^\lambda)$
  ▷ *Create and sign Merkle tree* ◁
  $(\text{root}_u, \text{tree}_u) \leftarrow \text{MerkleTree}((\widehat{\text{ek}}_{u,t})_{t \in [L] \cup \{\perp\}})$
  $\sigma_{u,\text{root}} \xleftarrow{\$} \text{RS.Sign}(\text{rsk}_u, \text{root}_u, \{ \text{rvk}_u \})$
  $(\text{rvk}, \_) \xleftarrow{\$} \text{RS.KeyGen}(1^\lambda)$ ▷ *Discard* rsk
  for $t \in [L]$ **do** ▷ *One-time prekey bundles*
    $\text{path}_{u,t} \leftarrow \text{getMerklePath}(\text{tree}_u, t)$
    $\text{prek}_{u,t} \coloneqq (\widehat{\text{ek}}_{u,t}, \text{path}_{u,t}, \text{root}_u, \sigma_{u,\text{root}}, \text{rvk})$
    $D_{\text{kem}}[t] \leftarrow (\text{prek}_{u,t}, \widehat{\text{dk}}_{u,t})$
  ▷ *Last-resort prekey bundle $t = \perp$* ◁
  $\text{path}_{u,\perp} \leftarrow \text{getMerklePath}(\text{tree}_u, L+1)$
  $\text{prek}_{u,\perp} \coloneqq (\widehat{\text{ek}}_{u,\perp}, \text{path}_{u,\perp}, \text{root}_u, \sigma_{u,\text{root}}, \text{rvk})$
  $D_{\text{kem}}[t] \leftarrow (\text{prek}_{u,\perp}, \widehat{\text{dk}}_{u,\perp})$
  return $\begin{pmatrix} \vec{\text{prek}}_u \coloneqq (\text{prek}_{u,t})_{t \in [L] \cup \{\perp\}}, \\ \text{st}_u \coloneqq (D_{\text{kem}}, \text{rvk}, D_{\rho_\perp}) \end{pmatrix}$

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

function RingXKEM.Send($\text{isk}_s, \text{ik}_r, \text{prek}_r$)
  $(\text{dk}_s, \text{rsk}_s) \leftarrow \text{isk}_s$; $(\text{ek}_r, \text{rvk}_r) \leftarrow \text{ik}_r$
  $(\widehat{\text{ek}}_r, \text{path}_r, \text{root}_r, \sigma_{r,\text{root}}, \text{rvk}) \leftarrow \text{prek}_r$
  require $[\![\text{ReconstructRoot}(\widehat{\text{ek}}_r, \text{path}_r) = \text{root}_r]\!]$
  require $[\![\text{RS.Verify}(\{ \text{rvk}_r \}, \widehat{\text{ek}}_r, \sigma_{r,\text{root}}) = 1]\!]$
  $(\text{ss}_r, \text{ct}_r) \xleftarrow{\$} \text{KEM.Encaps}(\text{ek}_r)$
  $(\widehat{\text{ss}}_r, \widehat{\text{ct}}_r) \xleftarrow{\$} \text{KEM.Encaps}(\widehat{\text{ek}}_r)$
  $\text{content} \coloneqq \text{ik}_s \| \text{ik}_r \| \text{prek}_r \| \text{ct}_r \| \widehat{\text{ct}}_r$
  $K \| K_{\text{ske}} \coloneqq \text{KDF}(\text{ss}_r \| \widehat{\text{ss}}_r, \text{content})$
  $\sigma \xleftarrow{\$} \text{RS.Sign}(\text{rsk}_s, \text{content}, \{ \text{rvk}_s, \text{rvk} \})$
  $\text{ct}_{\text{ske}} \xleftarrow{\$} \text{SKE.Enc}(K_{\text{ske}}, \sigma)$ ▷ *Mask ring signature*
  $\rho \coloneqq (\text{ct}_r, \widehat{\text{ct}}_r, \text{ct}_{\text{ske}})$
  return (K, ρ)

Algorithm 9 RingXKEM's receiver algorithm.

function RingXKEM.Receive($\text{isk}_r, \text{st}_r, \text{ik}_s, t, \rho$)
  $(\text{dk}_r, \text{rsk}_r) \leftarrow \text{isk}_r$; $(\text{ek}_s, \text{rvk}_s) \leftarrow \text{ik}_s$
  $(D_{\text{kem}}, \text{rvk}, D_{\rho_\perp}) \leftarrow \text{st}_r$
  $(\text{ct}_r, \widehat{\text{ct}}_r, \text{ct}_{\text{ske}}) \leftarrow \rho$
  ▷ *Check t^{th} prekey bundle was not deleted.* ◁
  require $[\![D_{\text{kem}}[t] \neq \perp]\!]$
  if $[\![t = \perp]\!]$ **then**
    require $[\![(\text{ct}_r, \widehat{\text{ct}}_r) \notin D_{\rho_\perp}]\!]$ ▷ *Check not replayed.*
    $D_{\rho_\perp} \leftarrow D_{\rho_\perp} \cup \{ (\text{ct}_r, \widehat{\text{ct}}_r) \}$
  $(\text{prek}_{r,t}, \widehat{\text{dk}}_{r,t}) \leftarrow D_{\text{kem}}[t]$
  $\text{ss}_r \coloneqq \text{KEM.Decaps}(\text{dk}_r, \text{ct}_r)$
  $\widehat{\text{ss}}_r \coloneqq \text{KEM.Decaps}(\widehat{\text{dk}}_{r,t}, \widehat{\text{ct}}_r)$
  $\text{content} \coloneqq \text{ik}_s \| \text{ik}_r \| \text{prek}_{r,t} \| \text{ct}_r \| \widehat{\text{ct}}_r$
  $K \| K_{\text{ske}} \coloneqq \text{KDF}(\text{ss}_r \| \widehat{\text{ss}}_r, \text{content})$
  $\sigma \coloneqq \text{SKE.Dec}(K_{\text{ske}}, \text{ct}_{\text{ske}})$ ▷ *Unmask signature*
  require $[\![\text{RS.Verify}(\{\text{rvk}_s, \text{rvk}\}, \text{content}, \sigma) = 1]\!]$
  if $[\![t \neq \perp]\!]$ **then**
    $D_{\text{kem}}[t] \leftarrow \perp$ ▷ *Delete prekey bundle*
  $\text{st}_r \leftarrow (D_{\text{kem}}, \text{rvk}, D_{\rho_\perp})$
  return (K, st_r)

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 Tab. 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 Tab. 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 [23].

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

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.

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-512	NIST I	800	768
KEM	Kyber-1024	NIST V	1568	1568
Signature	XEd25519 [41]	Pre-Quantum	32	64
1-Ring Sig	Gandalf [23]	NIST I	896	630
2-Ring Sig	Gandalf [23]	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

prekey: a 32-byte X25519 public key with 64-byte XEd25519 signature [41]. 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 prekey in every prekey bundle for HNDL security. This adds 1536 bytes and a 64-byte XEd25519 signature.

The X3DH handshake message 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 show that the median time for decapsulation (the most expensive operation) is only 83 µs slower than X25519 computations.[9]

RingXKEM. The RingXKEM protocol uses a KEM encapsulation key and a ring signature verification key in its identity public key. Kyber-512 encapsulation keys are 800 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 2582 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 2772 bytes. We also give the sizes for RingXKEM with Kyber-1024 like PQXDH.

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

Protocol	KEX	Identity public key	Prekey bundle size		Handshake message
			Individual	L keys	
X3DH	ECDH	32	128	3296	64
PQXDH	DH+K-1024	32	1696	166 496	1632
RingXKEM	Kyber-512	1696	2582	81 526	2772
RingXKEM-noMT	Kyber-512	1696	2326	143 896	2772
RingXKEM	Kyber-1024	2464	3350	158 326	4372
RingXKEM-noMT	Kyber-1024	2464	3094	220 696	4372

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, Tab. 7 row RingXKEM-noMT shows a variant of RingXKEM that signs each KEM prekey instead of using a Merkle tree. 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 [43]. 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).

[9] Based on `supercop-20240425` [4] results for hostname `pi4b` (latest measurements: DH, KEM, sign, hash).

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.

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

[1] 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).

[2] 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.

[3] 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.

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

[5] 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).

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] Wouter Castryck and Thomas Decru. "An Efficient 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. 423–447. DOI: 10.1007/978-3-031-30589-4_15.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

[29] 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.

[30] 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.

[31] Ehren Kret and Rolfe Schmidt. *The PQXDH Key Agreement Protocol*. Protocol documentation. Oct. 18, 2023. URL: https://signal.org/docs/specifications/pqxdh/.

[32] 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.

[33] 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.

[34] 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.

[35] 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.

[36] Moxie Marlinspike and Trevor Perrin. *The X3DH Key Agreement Protocol*. Protocol documentation. Nov. 4, 2016. URL: https://signal.org/docs/specifications/x3dh/.

[37] 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.

[38] 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.

[39] 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.

[40] 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.

[41] Trevor Perrin. *The XEdDSA and VXEdDSA Signature Schemes*. documentation. Oct. 20, 2016. URL: https://signal.org/docs/specifications/xeddsa/.

[42] Trevor Perrin and Moxie Marlinspike. *The Double Ratchet Algorithm*. Protocol documentation. Nov. 20, 2016. URL: https://signal.org/docs/specifications/doubleratchet/.

[43] 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. available at https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022. National Institute of Standards and Technology, 2022.

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

[45] Rolfe Schmidt. "Private communications." 2024.

[46] Signal foundation. *libsignal*. URL: https://github.com/signalapp/libsignal.

[47] Douglas Stebila. *Security analysis of the iMessage PQ3 protocol*. Cryptology ePrint Archive, Report 2024/357. 2024. URL: https://eprint.iacr.org/2024/357.

[48] WhatsApp. *WhatsApp Encryption Overview*. Technical white paper. Sept. 27, 2023. URL: https://www.whatsapp.com/security/WhatsApp-Security-Whitepaper.pdf.