# Triple Ratchet:
# A Bandwidth Efficient Hybrid-Secure Signal Protocol

Yevgeniy Dodis[1], Daniel Jost[1], Shuichi Katsumata[2,3], Thomas Prest[2], Rolfe Schmidt[4]

[1]New York University
{dodis, daniel.jost}@cs.nyu.edu
[2]PQShield
{shuichi.katsumata, thomas.prest}@pqshield.com
[3]AIST
[4]Signal Messenger
rolfe@signal.org

March 13, 2025

### Abstract

Secure Messaging apps have seen growing adoption, and are used by billions of people daily. However, due to imminent threat of a "Harvest Now, Decrypt Later" attack, secure messaging providers must react know in order to make their protocols *hybrid-secure*: at least as secure as before, but now also post-quantum (PQ) secure. Since many of these apps are internally based on the famous Signal's Double-Ratchet (DR) protocol, making Signal hybrid-secure is of great importance.

In fact, Signal and Apple already put in production various Signal-based variants with certain levels of hybrid security: PQXDH (only on the initial handshake), and PQ3 (on the entire protocol), by adding a *PQ-ratchet* to the DR protocol. Unfortunately, due to the large communication overheads of the Kyber scheme used by PQ3, real-world PQ3 performs this PQ-ratchet approximately every 50 messages. As we observe, the effectiveness of this amortization, while reasonable in the best-case communication scenario, quickly deteriorates in other still realistic scenarios; causing *many consecutive* (rather than 1 in 50) re-transmissions of the same Kyber public keys and ciphertexts (of combined size 2272 bytes!).

In this work we design a new Signal-based, hybrid-secure secure messaging protocol, which significantly reduces the communication complexity of PQ3. We call our protocol "the *Triple Ratchet*" (TR) protocol. First, TR uses *erasure codes* to make the communication inside the PQ-ratchet provably balanced. This results in much better *worst-case* communication guarantees of TR, as compared to PQ3. Second, we design a novel "variant" of Kyber, called Katana, with significantly smaller combined length of ciphertext and public key (which is the relevant efficiency measure for "PQ-secure ratchets"). For 192 bits of security, Katana improves this key efficiency measure by over 37%: from 2272 to 1416 bytes. In doing so, we identify a critical security flaw in prior suggestions to optimize communication complexity of lattice-based PQ-ratchets, and fix this flaw with a novel proof relying on the recently introduced hint-MLWE assumption.

During the development of this work we have been in discussion with the Signal team, and they are actively evaluating bringing a variant of it into production in a future iteration of the Signal protocol.

# Contents

**1 Introduction** **3**

1.1 Triple Ratchet Design Overview . . . 5

1.2 Lattice-based Katana RKEM Overview . . . 6

**2 Preliminary** **7**

2.1 Notation . . . 7

2.2 Lattices . . . 7

2.3 Symmetric Cryptographic Primitives . . . 9

2.4 Chunk Encoding . . . 10

2.5 Continuous Key Agreement . . . 11

**3 Hybrid Secure Messaging** **12**

3.1 Syntax . . . 13

3.2 Security . . . 14

**4 The Triple Ratchet** **16**

4.1 Construction . . . 16

4.2 Correctness and Security . . . 21

**5 From Ratcheting Key Encapsulation Mechanism to CKA** **28**

5.1 Definition of Forward-Secure Ratcheting KEM . . . 28

5.2 A Generic Construction of CKA from Ratcheting KEM . . . 31

5.3 Security . . . 31

**6 Katana: An Efficient Ratcheting KEM from Lattices** **37**

6.1 Construction of Katana . . . 37

6.2 Security of Katana . . . 39

6.3 Optimizing Katana with Bit-Dropping . . . 48

6.4 Concrete Parameter Selection . . . 50

**7 Efficiency Analysis of Triple Ratchet** **52**

7.1 Effect of Our RKEM on Communication Costs . . . 52

7.2 Effect of Chunk Encoding on Communication Costs . . . 53

**A Additional RKEM instantiation** **58**

A.1 Generic Construction . . . 58

A.2 Non-Forward-Secure Lattice-based Construction . . . 58

A.3 Diffie-Hellman Constructions . . . 59

**B Remark on Bad Randomness** **60**

# 1 Introduction

The Signal protocol, used by Signal, WhatsApp, Google RCS, and Facebook Messenger to protect the communications of billions of people worldwide, has widely been considered to be a benchmark for secure messaging. At its core, it uses the famous *Double Ratchet* protocol [MP16a] to provide important security properties called forward secrecy (FS) and post-compromise security (PCS). Signal (or more precisely the X3DH and Double Ratchet protocols) has been widely deployed with heavily scrutinized open source implementations, and has been formally analyzed in e.g., [CCD$^{+}$20, ACD19, BFG$^{+}$22a, CJSV22, KBB17, BBD$^{+}$21, CRT24], to show that it provides many desirable properties, including FS, PCS, but also mutual authentication and even certain form of deniability [VGIK20].

Post-Quantum Security. While this gives us confidence in the protocol today, these security guarantees are contingent on Diffie-Hellman (DH) assumptions for elliptic curves that can be broken by a quantum computer using Shor's algorithm [Sho94]. This is not only a future threat, since protocol transcripts collected today can be recorded and saved until a quantum computer is available, then decrypted in a Harvest Now, Decrypt Later (HNDL) attack. Motivated by these concerns, the work by Alwen et al. [ACD19] showed how to generalize the Double Ratchet protocol to work with any key encapsulation mechanism (KEM). There have been several works aiming to turn the X3DH protocol post-quantum secure [BFG$^{+}$20, DG22, BFG$^{+}$22b, HKKP21, HKKP22, CHN$^{+}$24], some of which relying on any KEMs and (ring) signatures. As a result, one could potentially replace the DH-based Signal with a post-quantum variant. Unfortunately, the resulting protocol is not sufficient for practical use, for two reasons. First, we do not want to lose the original DH-based security of Signal. Thus, practically relevant post-quantum extensions of Signal should provide what is called *hybrid* security, and meaningfully combine the DH-based Double Ratchet with some post-quantum variant. Second, the use of post-quantum KEMs like Kyber (i.e., ML-KEM) [SAB$^{+}$22] has noticeable costs in the communication complexity, making it often impractical in the real world.

PQXDH and PQ3. As a result, the industry transition to post-quantum Signal has been somewhat slower. First, Signal Messenger recently deployed PQXDH [KS23], an update to the X3DH [MP16b] handshake component of the Signal protocol, and formally verified and proven that the updated protocol provides HNDL protection without removing any of the previous DH-based security guarantees [BJKS24, FG, HKW]. Since this was only an update to the initial handshake protocol, it does not provide any post-quantum PCS, one of the key features of the original Double Ratchet protocol.

To address this issue, Apple recently deployed PQ3 [App24], — a protocol similar to Signal, — that continuously adds Kyber-768 freshly shared secrets to the "root secrets" of the Double Ratchet protocol. Simplifications of the resulting PQ3 protocol have been analyzed by [Ste24] and machine verified by [LSB24], but they do not fully capture what is done in the real world. Concretely, [Ste24] only models Kyber public keys and ciphertexts as being sent with *every* asymmetric ratchet message. As we mentioned above, this is quite expensive, and Apple decided to perform a post-quantum ratchet approximately every 50 messages (or whenever they have not sent a fresh Kyber public key within a week), in order to amortize the large communication cost of Kyber keys and ciphertexts [Jac]. Heuristically (and somewhat oversimplifying), this means that users have 50 "cheap" epochs (which do not help with post-quantum PCS), followed by 1 "expensive" epoch (which gives post-quantum PCS, but at a much slower rate than DH-based PCS).[1]

Communication Efficiency of PQ3. While the deployment of PQ3 was an amazing and greatly celebrated advance of post-quantum cryptography in the real-world, there are at least two avenues where it can be substantially improved in terms of its communication efficiency. (And we address these deficiencies in this work, as our main contribution.)

First, while PQ3's "amortization trick" might provide a reasonable trade-off in the best-case scenario, when the communication pattern between the users is roughly balanced, the effectiveness of this amortization quickly deteriorates in less balanced, but *still realistic* real-world scenarios. This is because each of Signal's sending epochs lasts roughly until the peer responds (and advances the public ratchet). So it might be possible — and certainly happens from time to time — that the "expensive epoch" happens exactly when

[1] This heuristic is related to "on-demand" ratcheting suggested by [CDV21].

one of the users is offline for an extended period of time,[2] resulting in *many consecutive re-transmissions repeating the same (long!) Kyber public keys and ciphertexts*. In particular, from a theoretical perspective one can easily define (adversarial) communication scenarios where the "expensive epochs" last for a long time, and PQ3's amortization heuristics do not offer any asymptotic saving, as compared to the simplified protocol analyzed by [Ste24].[3]

Second, we already mentioned that Kyber's public key and ciphertext (and each "expensive epoch" message in PQ3 sends both) is much larger than the single DH group element sent by classical Double Ratchet protocol. Concretely, (1088+1184=2272) bytes compared to 32 bytes, which is 71 times longer! Thus, any concrete efficiency improvement over using the generic (post-quantum) KEM advocated by [ACD19] will likely result in much faster PCS. For example, it allows reduction of the number 50 in PQ3's heuristic amortization, while maintaining similar communication complexity. In that regard, [ACD19, DG19, LKS23] already described lattice-based protocols (either directly for Kyber, or equivalent variants over other rings) which seemingly achieve this goal. Unfortunately, the protocol of [DG19] achieves almost no saving (less than 2%, as noticed by the authors) as compared to using the generic Kyber, while the protocols of [ACD19, LKS23] contain a critical subtle security flaw (as we show below) invalidating these analyses. Thus, prior to this work we did not have optimized variants of Kyber which would significantly reduce the communication complexity of post-quantum Double Ratchet protocol or its variants.

OUR CONTRIBUTIONS. In this work, we provide a practical hybrid-secure ratcheting protocol called the *Triple Ratchet* protocol.[4] Our name is taken from the fact that we use (1) a post-quantum public ratchet, (2) a classical public ratchet, and (3) symmetric ratchet. Compared to PQ3, it addresses both of the communication deficiencies mentioned above.

First, it uses *erasure codes* to evenly distribute the communication inside the post-quantum ratchet, without any amortization heuristics. At a high level, instead of sending one long message every 50 epochs, we encode the resulting message using an erasure code, and send a fresh chunk of this encoding with every message. For example, we could set parameters so that the long message will be decoded from *any* 50 chunks. Then, in a fully balanced setting we would still achieve PCS in 50 epochs and same communication as PQ3, but without any amortization. However, we start getting big savings in the unbalanced cases, when some epochs are long-lasting. For such epochs, PQ3's strategy could be viewed as using a hugely inefficient *repetition code*, leading to a big communication penalty; e.g., a factor of up to 50 in our "PQ3-inspired" example. We detail this in Section 7, and give an overview of some of the technical challenges we resolved in Section 1.1.

Second, we design a novel *Continuous Key Agreement* (CKA) protocol based on Kyber, which we call Katana-CKA, which can be used inside our Triple Ratchet protocol. Recall, CKA was a generic building block used by [ACD19] to abstract out the so-called *public* ratchet of the Double Ratchet Protocol. [ACD19] then presented a generic KEM-based CKA, where every message contained a KEM public key and ciphertext. When applied to Kyber at security level 192 bits, this gives CKA *messages of size 2272 bytes*. In contrast, for the same security level Katana-CKA uses *messages of size 1416 bytes*, saving over 37% over the generic construction.

We notice that Katana-CKA is closely related to what previous works called "optimized" lattice-based CKA [ACD19, LKS23], but instantiated with a carefully chosen variant of Kyber. As we mentioned, however, we identify a critical flaw in the previous analyses of this "optimized" KEM, and non-trivially fix it with a novel proof relying on the recently introduced hint-MLWE assumption [KLSS23, EEN+24].

In more detail, we first generalize the KEM-based CKA from [ACD19] to work with what we call a *Ratcheted* KEM (RKEM). On a high level, RKEM abstracts KEM properties in a way which allows a freshly sampled ciphertext also be used as part of a different KEM public key. In essence, this is precisely why the original DH-based CKA of Signal saved a factor of 2 in communication, when compared to the generic KEM-based DH construction. And this is why RKEM is precisely fitted for the use inside a CKA. Once we

[2] E.g., when using devices which are periodically turned off.

[3] [LSB24] explicitly models this optional sending behavior, but does not model the repetition of KEM public key and ciphertext messages required for immediate decryption [Jac].

[4] This should not be confused with the protocol by [BFG+22a] with the same name.

define RKEM and show that it generically implies CKA, it allows us to focus on a cleaner RKEM primitive, which we then construct from the hint-MLWE assumption. We call the resulting RKEM Katana,[5] which explains the name Katana-CKA for our new CKA. We expand on our technique in Section 1.2.

During the development of this work we have been in discussion with the Signal team, and they are actively evaluating bringing a variant of it into production in a future iteration of the Signal protocol.

### 1.1 Triple Ratchet Design Overview

As we mentioned, the *Triple Ratchet* protocol could be used as the generalization of the Double Ratchet paradigm from [MP16a, ACD19] to allow the use of a third "post-quantum CKA protocol (i.e., public ratchet)".[6] Formally, instead of composing a (classical) CKA protocol with the standard symmetric ratchet, we will compose the symmetric ratchet with two different CKA protocols. (In practice, we envision using the standard "optimized" DH-based CKA with our new Katana-CKA, but the composition is stated *generically.*) The key difference is that the second (post-quantum) CKA will use erasure codes to send its (potentially) long messages in "chunks". This seemingly simple optimization creates complications in the protocol, security model, modular choices of primitives/abstractions, and search for practical optimization.

First, the classical and post-quantum CKA protocols are no longer synchronized, and there are situations where one ratchet moves forward and the other does not. As the result, we can no longer use a single "root key" where we hash the new key material whenever one of the ratchets moves forward. We resolve it by having two root keys, carefully deriving two separate message keys (also using two separate[7] symmetric ratchets), and finally combine those message keys to encrypt the application message. In contrast, PQ3 could use a single root key, since the two ratchets were always synchronized.

Second, unlike the classical public ratchet protocol, the sender cannot immediately use the newly derived PQ key material (from CKA) to encrypt the message (although it will hash it to the appropriate "root key"). Indeed, since it could take several chunks for the recipient to get the new PQ CKA message, the recipient would not be able to immediately decrypt the message with just one chunk. Instead, the receiver now has to continuously acknowledge how many chunks it received so far. And the sender will only use the already updated root key to derive the message key *only if* it knows that the receiver is missing *at most one chuck* to decode the CKA message. This also creates other "book-keeping challenges", which are carefully resolved in our design. (For example, we need to remember the number of sent messages in the last *two* epochs, rather than only one.) An interested reader can fast-forward to Figs. 8 and 9 to look at our final Triple Ratchet protocol. The left column of both figures roughly corresponds to the DH ratchet, while the right figure — to the PQ ratchet with erasure codes. Despite the necessary extra complexity in the code, the actual protocol is quite fast and elegant, resulting in very efficient instantiations.

Third, we have to generalize the notion of "epochs" from [ACD19], as they do not necessarily correspond to a single "change of communication direction". In fact, there are separately evolving classical and post-quantum epochs, needed for the hybrid security guarantees. We resolve by providing a *single* protocol, but then parameterize its (either classical or post-quantum) security by a corresponding *epoch function*, which roughly models when the party fully communicated its fresh key material (e.g., a single CKA message, in a concrete instantiation) to its peer. And then providing concrete properties of the two resulting epoch functions, stating how quickly the (classical or PQ) epochs increase, based on the actual communication pattern.

With these important changes, our resulting protocol could be viewed as a natural, and still very modular, generalization of the Double Ratchet abstraction from [ACD19]. In particular, one can get many concrete instantiations by varying the two underlying CKA s, and the length of the "chunk" in the PQ-CKA.

[5] Similar to Kyber, Katana is a certain type of an ancient (Japanese) sword.

[6] We will interchangeably use the term CKA and public ratchet.

[7] In this sense we have *four* ratchets going on, but since the same symmetric ratchet is used twice, we stuck with the "Triple Ratchet" acronym.

## 1.2 Lattice-based Katana RKEM Overview

In theory, instantiating RKEMs from lattices is trivial, as standard KEMs are special cases of RKEMs. Indeed, the CKAs built from such RKEMs is exactly the generic construction of CKA based on KEMs by [ACD19]. The true strength of RKEM lies in enabling a more efficient CKA construction, like the Double Ratchet protocol used in Signal. Assume Alice holds $a \in \mathbb{Z}_p$ and Bob holds $g^a \in \mathbb{G}$. In Signal, Bob samples $b \xleftarrow{\$} \mathbb{Z}_p$ and sends $g^b$ to Alice. The shared key $\mathsf{K}$ is then updated by mixing $g^{ab}$ into $\mathsf{K}$, *ratcheting* the state forward. Importantly, $g^b$ holds two purposes: it acts as an "encryption/ciphertext" for the Diffie-Hellman key exchange while also serving to be a new "public key" for the next ratchet (i.e., Alice will generate $a' \xleftarrow{\$} \mathbb{Z}_p$ and update the state by $g^{a'b}$). While this reusing of $g^b$ for two purposes has an immediate benefit on efficiency, one downside compared to the KEM-based construction is that it achieves a weaker FS guarantee. Recently, [BFG$^+$22a] showed a simple trick to make it as secure, with almost no overhead.

There have been efforts to port the above efficient classical construction to the post-quantum setting [ACD19, DG19, LKS23]. Notably, [ACD19, LKS23] proposes a lattice-based equivalent to the Double Ratchet protocol used in Signal. At a high level, it goes as follows, where $R_q := \mathbb{Z}_q[X]/(X^n + 1)$ and $\mathbf{D} \in R_q^{k \times k}$ is a public matrix. Assume Alice holds $\mathbf{s}_\mathsf{A} \in R_q^k$ and Bob holds $\mathbf{u}_\mathsf{A} = \mathbf{D} \cdot \mathbf{s}_\mathsf{A} + \mathbf{e}_\mathsf{A} \in R_q^k$, where $\mathbf{s}_\mathsf{A}$ and $\mathbf{e}_\mathsf{A}$ are short. Bob samples short vectors $\mathbf{s}_\mathsf{B}, \mathbf{e}_\mathsf{B} \in R_q^k$ and $\tilde{e}_\mathsf{B} \in R_q$ from appropriate distributions, and a random $\mathsf{seed} \xleftarrow{\$} \{0,1\}^n \subset R_q$. It then sends $(\mathbf{u}_\mathsf{B}, v_\mathsf{A}) := (\mathbf{D}^\top \cdot \mathbf{s}_\mathsf{B} + \mathbf{e}_\mathsf{B}, \mathbf{u}_\mathsf{A}^\top \cdot \mathbf{s}_\mathsf{B} + \tilde{e}_\mathsf{B} + \mathsf{seed} \cdot \lfloor q/2 \rceil)$ to Alice.[8] Alice first interprets $(\mathbf{u}_\mathsf{B}, v_\mathsf{A})$ as a ciphertext and decrypts $\mathsf{seed}$ by rounding $v_\mathsf{A} - \mathbf{u}_\mathsf{B}^\top \cdot \mathbf{s}_\mathsf{A}$ to the nearest multiple of $\lfloor q/2 \rceil$. The $\mathsf{seed}$ is then mixed into the shared key $\mathsf{K}$ to ratchet the state forward. Alice then interprets part of the ciphertext $\mathbf{u}_\mathsf{B}$ as Bob's public key so that it can perform similar ratcheting. As $\mathbf{u}_\mathsf{B}$ is the dominant component in terms of size, this effectively almost halves the communication size, giving us the same benefit as Signal's Double Ratchet.

Flaw in Previous Analyses. While the construction is intuitive and simple, we observe that the security proof is subtle. Indeed, we identify that both previous works [ACD19, LKS23] contain the same flaw in the CKA security proof, rendering their scheme insecure for certain parameter regime. Recall that Signal's Double Ratchet was proven to satisfy PCS [ACD19] by arguing two things *even if* Alice's secret key $a \in \mathbb{Z}_p$ is compromised:

(C.1) $g^{ab}$ can be simulated without Bob's secret key $b$.

(C.2) $(g, g^{a'}, g^b, g^{a'b})$ is indistinguishable from $(g, g^{a'}, g^b, g^c)$ for $c \xleftarrow{\$} \mathbb{Z}_p$.

Item (C.2) stipulates that once Alice updates its key to $g^{a'}$, while Bob's key $g^b$ is uncompromised, then the state *heals* since $g^{a'b}$ is mixed into the shared key. While seemingly unimportant, Item (C.1) is a vital property to formally invoke the DDH assumption in Item (C.2) — if not for Item (C.1), the reduction cannot embed $g^b$ given by the DDH challenge into the CKA protocol. To imitate this proof for the aforementioned lattice-based scheme, we have to argue the following, even if Alice's secret key $\mathbf{s}_\mathsf{A} \in R_q^k$ is compromised:

(L.1) $v_\mathsf{A} := \mathbf{u}_\mathsf{A}^\top \cdot \mathbf{s}_\mathsf{B} + \tilde{e}_\mathsf{B} + \mathsf{seed} \cdot \lfloor q/2 \rceil$ can be simulated without Bob's secret key $\mathbf{s}_\mathsf{B}$.

(L.2) $(\mathbf{u}'_\mathsf{A}, \mathbf{u}_\mathsf{B}, v'_\mathsf{B}) = (\mathbf{D} \cdot \mathbf{s}'_\mathsf{A} + \mathbf{e}'_\mathsf{A}, \mathbf{D}^\top \cdot \mathbf{s}_\mathsf{B} + \mathbf{e}_\mathsf{B}, \mathbf{u}_\mathsf{B}^\top \cdot \mathbf{s}'_\mathsf{A} + \tilde{e}'_\mathsf{A} + \mathsf{seed}' \cdot \lfloor q/2 \rceil)$ is indistinguishable from $(\mathbf{u}'_\mathsf{A}, \mathbf{u}_\mathsf{B}, v)$ for $v \xleftarrow{\$} R_q$.

It turns out that this Item (L.1) is where the subtlety lies. Unlike in the classical setting, we no longer have clear symmetry. Indeed, observe that $v_\mathsf{A}$ is identically expressible as $v_\mathsf{A} = (\mathbf{D} \cdot \mathbf{s}_\mathsf{A} + \mathbf{e}_\mathsf{A})^\top \cdot \mathbf{s}_\mathsf{B} + \tilde{e}_\mathsf{B} + \mathsf{seed} \cdot \lfloor q/2 \rceil = \mathbf{u}_\mathsf{B}^\top \cdot \mathbf{s}_\mathsf{A} \underline{- \mathbf{e}_\mathsf{B}^\top \cdot \mathbf{s}_\mathsf{A} + \mathbf{e}_\mathsf{A}^\top \cdot \mathbf{s}_\mathsf{B} + \tilde{e}_\mathsf{B}} + \mathsf{seed} \cdot \lfloor q/2 \rceil$, where we plug in $\mathbf{u}_\mathsf{B} = \mathbf{D}^\top \cdot \mathbf{s}_\mathsf{B} + \mathbf{e}_\mathsf{B}$. Denoting the underlined value as $h$, it is clear that $h$ *cannot* be simulated only using Alice's secret $\mathbf{s}_\mathsf{A}$ (and $\mathbf{e}_\mathsf{A}$). In fact, an adversary with $\mathbf{s}_\mathsf{A}$ can directly compute $h$ to infer statistical knowledge of $\mathbf{s}_\mathsf{B}$ and $\mathbf{e}_\mathsf{B}$. Even worse, since the adversary learns slight information about $\mathbf{s}_\mathsf{B}$, we can inductively see that the adversary may also learn some information even on the *updated* key $\mathbf{s}'_\mathsf{A}$, creating a vicious cycle.

[8] Note that while [ACD19] bases their construction on FrodoKEM, our explanation is based on a Kyber-like KEM as in [LKS23]. These differences will have no importance to our argument.

Previous work has overlooked this issue and falsely invoked Item (L.2). We note that technically, we can *statistically* prove Item (L.1) by sampling $\tilde{e}_{\mathsf{B}}$ from a distribution super-polynomially larger than $-\mathbf{e}_{\mathsf{B}}^{\top} \cdot \mathbf{s}_{\mathsf{A}} + \mathbf{e}_{\mathsf{A}}^{\top} \cdot \mathbf{s}_{\mathsf{B}}$ (i.e., noise flooding). However, this renders the scheme unusable in practice, and defeats the purpose of using the optimization.

Our Solution. At the core of our technical contribution, we use the recent hint-MLWE problem by [KLSS23, EEN+24] to *computationally* prove Item (L.1), and carefully argue Item (L.2). hint-MLWE in essence stipulates that the standard MLWE remains hard even if some *noisy* linear leakage of the secret is given to the adversary. In fact, we go one step further and show that our new proof strategy is essential to make the recent trick by Bienstock et al. [BFG+22a] improving FS to work in the lattice-setting. The main idea of [BFG+22a] was for Alice to run the same Signal's original Double Ratchet protocol, but to store $\widehat{a} := a' + \mathsf{H}(g^{a'b})$ as opposed to $a'$. The intuition is that even if $\widehat{a}$ is compromised, $g^{a'b}$ remains secure assuming $\mathsf{H}$ is a random oracle (or ElGamal encryption is circular secure), hence offering better FS. In the lattice-setting however, the updated $\widehat{\mathsf{s}}_{\mathsf{A}} := \mathsf{s}'_{\mathsf{A}} + \mathsf{H}(\mathsf{seed}')$ must still remain *short* for decryption to work, and as such, leaking $\widehat{\mathsf{s}}_{\mathsf{A}}$ again statistically leaks information on $\mathbf{s}'_{\mathsf{A}}$. For more detail of the proof, we refer to Section 6.2.

While hint-MLWE reduces from MLWE, this is not without a slight degradation in the parameters. We wrap up everything by performing cryptanalysis on hint-MLWE based on the reduction from hint-MLWE to MLWE, and set concrete parameters for our RKEM called Katana. We conclude that the size of the CKA message is ≈40 % better than naively using Kyber as the KEM-based CKA [ACD19].

# 2 Preliminary

## 2.1 Notation

**Sets and distributions.** When $S$ is a finite set, we let $\mathcal{U}(S)$ denote the uniform distribution over $S$, and abbreviate $x \xleftarrow{\$} S$ for $x \xleftarrow{\$} \mathcal{U}(S)$. If $x$ is a set, we use $x \xleftarrow{+} y$ as a shorthand for $x \leftarrow x \cup \{y\}$ and, conversely, $x \xleftarrow{-} y$ to denote removing $y$. Given a positive integer $N$ and a distribution $D$ of support included in an additive group, we let $[N] \cdot D$ denote the convolution of $N$ independent copies of $D$. In other words, $[N] \cdot D$ is the distribution of $x = \sum_{i \in [N]} x_i$, where $\forall i \in [N], x_i \xleftarrow{\$} D$. Given two distributions $X, Y$ over a multiplicative group, we also let $X \cdot Y$ denote the product distribution of $X$ and $Y$. Lastly, we may write $x \xleftarrow{\$} D\{\mathsf{rand}\}$ to make explicit the randomness used to sample from the distribution $D$. In protocol descriptions, whenever a **req** statement fails or an **error** statement is output by an algorithm, all changes to the algorithm state is assumed to be discarded and undone. With an overload in notations, in security game descriptions, **req** means restricting the class of valid adversaries to those not violating the condition.

**Cyclotomic rings.** Let $n$ be a power-of-two integer, which we leave undefined unless explicitly specified otherwise. Let $\mathcal{R} = \mathbb{Z}[x]/(x^n + 1)$ the cyclotomic ring of degree $n$ and $\mathcal{K} = \mathbb{R}[x]/(x^n + 1)$. For a real matrix $\mathbf{M} \in \mathbb{R}^{k \times \ell}$, we note $s_1(\mathbf{M})$ and call spectral norm of $\mathbf{M}$ the value $\max_{\mathbf{x} \neq \mathbf{0}} \frac{\|\mathbf{M} \cdot \mathbf{x}\|}{\|\mathbf{x}\|}$, where $\|\cdot\|$ denotes the $L_2$-norm. The spectral norm of $\mathbf{M}$ is also the (unique non-negative) square root of the largest eigenvalue of $\mathbf{M}^{\top} \cdot \mathbf{M}$. We recall that if $\mathbf{M}$ is symmetric, then its singular values are the square roots of its eigenvalues. If $\mathbf{B} \in \mathcal{R}^{k \times \ell}$ has its entries in $\mathcal{R}$, we identify $\mathbf{B}$ with its associated anti-circulant matrix $\mathbf{M} \in \mathbb{Z}^{nk \times n\ell}$ and abusively say that the spectral norm of $\mathbf{B}$ is the spectral norm of $\mathbf{M}$.

## 2.2 Lattices

### 2.2.1 Hardness Assumption

In this work, we rely on the standard module learning with errors (MLWE) problem along with (a generalization of) the recent *hint* MLWE problem by Kim et al. [KLSS23], stating that MLWE remains hard even if some leakage of the secret is provided.

**Definition 2.1** (MLWE). *Let $k, q$ be integers and $\chi$ be a probability distribution over $\mathcal{R}_q^k$. The advantage of an adversary $\mathcal{A}$ against the* Module Learning with Errors $\mathsf{MLWE}_{q,k,\chi}$ *problem is defined as:*

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{MLWE}}(1^\lambda) = \left|\Pr\left[\mathcal{A}(\mathbf{D}, \mathbf{D}\cdot\mathbf{s}+\mathbf{e}) = 1\right] - \Pr\left[\mathcal{A}(\mathbf{D}, \mathbf{b}) = 1\right]\right|,$$

*where $(\mathbf{D}, \mathbf{b}, \mathbf{s}, \mathbf{e}) \xleftarrow{\$} \mathcal{R}_q^{k\times k} \times \mathcal{R}_q^k \times \chi \times \chi$. The $\mathsf{MLWE}_{q,k,\chi}$ assumption states that any efficient adversary $\mathcal{A}$ has negligible advantage.*

The following is a slight generalization of the original $\mathsf{hint}\text{-}\mathsf{MLWE}$ problem [KLSS23], formally introduced by [EEN+24]. In the original definition, only hints of the form $(c\cdot\mathbf{s}+\mathbf{z_s}, c\cdot\mathbf{e}+\mathbf{z_e})$ for a randomly sampled coefficient $c \in \mathcal{R}_q$ and noise $(\mathbf{z_s}, \mathbf{z_e})$ leaked. This is easily generalized to the setting where hints can be any noisy linear combination of $(\mathbf{s}, \mathbf{e})$, that is, $\mathbf{M}\begin{bmatrix}\mathbf{s}\\ \mathbf{e}\end{bmatrix} + \mathbf{z}$. By setting $\mathbf{M} = \begin{bmatrix} c & 0 \\ 0 & c\end{bmatrix}$, we recover the original definition.

**Definition 2.2** (hint-MLWE). *Let $k, \ell, q$ be integers, $\chi$ and $\tilde{\chi}$ be probability distributions over $\mathcal{R}_q^k$ and $\mathcal{R}_q^\ell$, respectively, and $\mathcal{F}$ be a probability distribution over $\mathcal{R}_q^{\ell\times 2k}$. The advantage of an adversary $\mathcal{A}$ against the* Hint Module Learning with Errors $\mathsf{hint}\text{-}\mathsf{MLWE}_{q,k,\ell,\chi,\tilde{\chi},\mathcal{F}}$ *problem is defined as:*

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{hint}\text{-}\mathsf{MLWE}}(1^\lambda) = \left|\Pr\left[\mathcal{A}\Big(\mathbf{D}, \mathbf{D}\cdot\mathbf{s}+\mathbf{e}, \mathbf{M}, \mathbf{h}\Big) = 1\right] - \Pr\left[\mathcal{A}\Big(\mathbf{D}, \mathbf{b}, \mathbf{M}, \mathbf{h}\Big) = 1\right]\right|,$$

*where $(\mathbf{D}, \mathbf{b}, \mathbf{s}, \mathbf{e}, \mathbf{M}) \xleftarrow{\$} R_q^{k\times k} \times \mathcal{R}_q^k \times \chi \times \chi \times \mathcal{F}$. Moreover, the* hint *is defined as $\mathbf{h} = \mathbf{M}\begin{bmatrix}\mathbf{s}\\ \mathbf{e}\end{bmatrix} + \mathbf{z}$ where $\mathbf{z} \xleftarrow{\$} \tilde{\chi}$. The $\mathsf{hint}\text{-}\mathsf{MLWE}_{q,k,\ell,\chi,\tilde{\chi},\mathcal{F}}$ assumption states that any efficient adversary $\mathcal{A}$ has negligible advantage.*

Lastly, we recall the following result which establishes the hardness of the $\mathsf{hint}\text{-}\mathsf{MLWE}$ problem based on the $\mathsf{MLWE}$ problem. This is a simple adaptation of the original proof [KLSS23], formally appearing in [EEN+24]. Below, we denote $\mathcal{D}_\sigma$ as a discrete Gaussian distribution with standard deviation $\sigma$.

**Theorem 2.3 (Hardness of hint-MLWE).** *For any integers $k$, $\ell$, $q$, $n$, let $\mathcal{F}$ be a probability distribution over $\mathcal{R}_q^{\ell\times 2k}$, $\chi$ and $\tilde{\chi}$ be discrete Gaussian distributions $\mathcal{D}_{\sigma_1}$ and $\mathcal{D}_{\sigma_2}$, respectively, and $\mathcal{B}, \sigma$ a positive real such that*

$$\Pr\left[s_1\left(\mathbf{M}\mathbf{M}^\top\right) < \mathcal{B} : \mathbf{M} \xleftarrow{\$} \mathcal{F}\right] \geqslant 1 - \mathsf{negl}(\lambda),$$

*and $\sigma = \omega(\sqrt{\log n})$ and $\frac{1}{\sigma^2} = 2\cdot\left(\frac{1}{\sigma_1^2} + \frac{\mathcal{B}}{\sigma_2^2}\right)$. Under these conditions, the $\mathsf{hint}\text{-}\mathsf{MLWE}_{q,k,\ell,\mathcal{D}_{\sigma_1},\mathcal{D}_{\sigma_2},\mathcal{F}}$ problem is as hard as the $\mathsf{MLWE}_{q,k,\mathcal{D}_\sigma}$ problem.*

### 2.2.2 Rounding

In our work, we use the rounding definition used by Kyber [SAB+22]. Below, we briefly recall their definition.

For an even (resp. odd) positive integer $q$, we define $x' = x \bmod {}^{\pm}q$ to be the unique element $x'$ in the range $\frac{-q}{2} < x' \leqslant \frac{q}{2}$ (resp. $-\frac{q-1}{2} < x' \leqslant \frac{q-1}{2}$) such that $x' = x \bmod q$. For any positive integer $q$, we define $x' = x \bmod {}^{+}q$ to be the unique element $x'$ in the range $0 \leqslant x' < q$ such that $x' = x \bmod q$. We simply write $x \bmod q$ when the representation is not important. Also, for an element in $x \in \mathbb{Q}$, $\lfloor x \rceil$ denotes the rounding to the nearest integer, where in case of a tie, we take the larger integer.

**Compression and Decompression.** We define the following compression and decompression algorithms for positive integers $d$ and $q$ such that $d < \lceil \log_2(q) \rceil$:

$$\begin{aligned}\mathsf{Compress}_q : \mathbb{Z}_q &\longrightarrow \mathbb{Z}_{2^d}\\ x &\longmapsto \left\lfloor \frac{2^d}{q}\cdot x \right\rceil \mod {}^{+}2^d. \end{aligned} \tag{1}$$

$$\begin{aligned}\mathsf{Decompress}_q : \mathbb{Z}_{2^d} &\longrightarrow \mathbb{Z}_q \\ y &\longmapsto \left\lfloor \frac{q}{2^d} \cdot y \right\rceil . \end{aligned} \tag{2}$$

For these functions, we have the following:

**Lemma 2.4.** *Let $d$ and $q$ be positive integers such that $d < \lceil \log_2(q) \rceil$. Then, for any $x \in \mathbb{Z}_q$, we have*

$$\left| x' - x \bmod {}^{\pm} q \right| \leqslant \left\lfloor \frac{q}{2^{d+1}} \right\rceil ,$$

*where $x' = \mathsf{Decompress}_q(\mathsf{Compress}_q(x, d), d)$.*

When $\mathsf{Compress}_q$ or $\mathsf{Decompress}_q$ is used with $x \in R_q$ or $\mathbf{x} \in R_q^k$, the procedure is applied to each coefficient individually.

## 2.3 Symmetric Cryptographic Primitives

### 2.3.1 Authenticated Encryption with Associated Data

We use an *authenticated encryption with associated data* (AEAD) scheme $\mathsf{AEAD} = (\mathsf{Enc}, \mathsf{Dec})$.

**Definition 2.5 (Authenticated Encryption).** *An* authenticated encryption with associated data *(AEAD) scheme is a pair of algorithms* $\mathsf{AEAD} := (\mathsf{Enc}, \mathsf{Dec})$ *with the following syntax:*

$\mathsf{Enc}(\mathsf{K}, \mathsf{h}, \mathsf{M}) \to \mathsf{e}$*: It takes a key* $\mathsf{K}$*, authenticated data* $\mathsf{h}$*, and a message* $\mathsf{M}$*, and produces a ciphertext* $\mathsf{e}$*.*

$\mathsf{Dec}(\mathsf{K}, \mathsf{h}, \mathsf{e}) \to \mathsf{M}'$*: It takes a key* $\mathsf{K}$*, authenticated data* $\mathsf{h}$*, and a ciphertext* $\mathsf{e}$*, and outputs a plaintext* $\mathsf{M}'$*.*

*We assume all algorithms to be deterministic, i.e., all randomness to be based off the key.*

*We say that an* $\mathsf{AEAD}$ *scheme is* correct*, if for all keys* $\mathsf{K}$ *and all pairs* $(\mathsf{h}, \mathsf{M})$*,*

$$\mathsf{Dec}\big(\mathsf{K}, \mathsf{h}, \mathsf{Enc}(\mathsf{K}, \mathsf{h}, \mathsf{M})\big) = \mathsf{M}.$$

*We require* $\mathsf{AEAD}$ *to be* one-time IND-CCA *secure, formalized by the game in Fig. 1, and define the following advantage*

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

*and say the scheme to be secure iff every PPT $\mathcal{A}$ has negligible advantage.*

```
$\mathsf{Game}_{\mathcal{A}}^{\mathsf{AEAD}}(1^\lambda)$
$b \xleftarrow{\$} \{0,1\}$
$\mathsf{K} \xleftarrow{\$} \{0,1\}^\lambda$
$\mathsf{e}^* \leftarrow \perp$
$b' \xleftarrow{\$} \mathcal{A}(1^\lambda)^{\mathsf{encrypt}(),\mathsf{decrypt}()}$
return $[\![b = b']\!]$
```

```
$\mathsf{encrypt}(\mathsf{h}, \mathsf{M})$
if $[\![b = 0]\!]$ then
    $\mathsf{e}^* \leftarrow \mathsf{Enc}(\mathsf{K}, \mathsf{h}, \mathsf{M})$
else
    $\mathsf{e}^* \xleftarrow{\$} \mathcal{E}$
return $\mathsf{e}^*$
```

```
$\mathsf{decrypt}(\mathsf{h}, \mathsf{e})$
if $[\![\mathsf{e} = \mathsf{e}^*]\!] \vee [\![b = 1]\!]$ then
    return $\perp$
return $\mathsf{Dec}(\mathsf{K}, \mathsf{h}, \mathsf{e})$
```

Figure 1: The one-time IND-CCA game of an $\mathsf{AEAD}$ scheme $(\mathsf{Enc}, \mathsf{Dec})$ with ciphertext space $\mathcal{E}$, where encrypt is a one-time oracle.

### 2.3.2 Key Derivation Functions

We use several $\mathsf{KDF}$s in our work. Syntax wise, $\mathsf{KDF}$ is a deterministic algorithm taking one or more inputs and producing one or more (uniform) values. While we fix the number of inputs for each concrete $\mathsf{KDF}$, in slight abuse of notation we overload the same function to output a variable number of arguments. In practice, each $\mathsf{KDF}$ would be instantiated by a hash-based construction such as HKDF. In the following we outline the different security assumptions we need.

**Definition 2.6 (Pseudorandom generator (PRG)).** *A* $\mathsf{KDF}$ *with one input argument is said to behave like a PRG, with domain* $\{0,1\}^\lambda$ *and codomain* $\mathcal{Y}$*, if*

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{PRG}}(1^\lambda) := \left|\Pr[x \xleftarrow{\$} \{0,1\}^\lambda, y \leftarrow \mathsf{KDF}(x), b' \xleftarrow{\$} \mathcal{A}(y) : b' = 1] - \Pr[y \xleftarrow{\$} \mathcal{Y}, b' \xleftarrow{\$} \mathcal{A}(y) : b' = 1]\right|$$

*is negligible for every PPT* $\mathcal{A}$*.*

**Definition 2.7 ((Dual)-PRF).** *A* $\mathsf{KDF}$ *with two input arguments is said to behave like a* pseudo-random function *(PRF), if*

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

*is negligible for every PPT* $\mathcal{A}$*, for the game from Fig. 2. Moreover, it is said to be a* dual-PRF *if the advantage is also negligible in a variant of the game where the roles of* $\sigma$ *and* $I$ *are swapped, i.e., where initially* $I$ *is sampled and* $\mathsf{chall}$, $\mathsf{eval}$ *take* $\sigma$ *as input, and* $\mathcal{F}$ *is indexed by* $\sigma$*.*

```
$\mathsf{Game}_{\mathcal{A}}^{\mathsf{PRF}}(1^\lambda)$
$b \xleftarrow{\$} \{0,1\}$
$\sigma \xleftarrow{\$} \{0,1\}^\lambda$
$\mathcal{F}[\cdot] \leftarrow \bot$
$b' \xleftarrow{\$} \mathcal{A}(1^\lambda)^{\mathsf{chall}}$
return $[\![b = b']\!]$
```

```
$\mathsf{chall}(I)$
$(\sigma', R) \leftarrow \mathsf{KDF}(\sigma, I)$
if $[\![b = 1]\!]$ then
    $R \leftarrow \mathsf{eval}(I)$
return $(\sigma', R)$
```

```
$\mathsf{eval}(I)$
if $[\![\mathcal{F}[I] = \bot]\!]$ then
    $\mathcal{F}[I] \xleftarrow{\$} \mathcal{R}$
return $\mathcal{F}[I]$
```

Figure 2: $\mathsf{PRF}$ security of a two-input $\mathsf{KDF}$. If $\mathsf{KDF}$ expands to more than two return values, then the first one is $\sigma$ and the remaining outputs should be considered as $R$ over an appropriate composite space $\mathcal{R}$.

**Definition 2.8 (PRF-PRNG).** *A* $\mathsf{KDF}$ *with two input arguments is said to have* PRF-PRNG *security, if*

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{PRF\text{-}PRNG}}(1^\lambda) := \left|\Pr[\mathsf{Game}_{\mathcal{A}}^{\mathsf{PRF\text{-}PRNG}}(1^\lambda)] - \frac{1}{2}\right|$$

*is negligible for every PPT* $\mathcal{A}$*, for the game from Fig. 3.*

## 2.4 Chunk Encoding

We use a standard erasure code instantiated using Reed-Solomon erasure codes to implement our "chunking" strategy of post-quantum $\mathsf{CKA}$ messages.

**Definition 2.9.** *An* erasure code *for a set of symbols* $\Sigma$*, a block length* $N$*, and a message size* $n_{\mathsf{chunk}}$ *consists of PPT algorithms* $\mathsf{Encode}, \mathsf{Decode}$ *defined as follows:*

$\mathsf{Encode}(M, i) \xrightarrow{\$} c$ : *It takes as input a message* $M \in \Sigma^{n_{\mathsf{chunk}}}$*, and an integer* $i \in \mathbb{Z}_N$ *and outputs symbol* $c \in \Sigma$*.*

```
$\mathsf{Game}^{\mathsf{PRF\text{-}PRNG}}_{\mathcal{A}}(1^\lambda)$
$b \xleftarrow{\$} \{0,1\}$
$\sigma \xleftarrow{\$} \{0,1\}^\lambda$
$\mathsf{corr}, \mathsf{prng}, \mathsf{prf} \leftarrow \mathsf{false}$
$\mathcal{F}[\cdot] \leftarrow \bot$
$b' \xleftarrow{\$} \mathcal{A}(1^\lambda)^{\mathsf{process},\mathsf{chall\text{-}prf},\mathsf{chall\text{-}prng},\mathsf{corr}}$
return $[\![b = b']\!]$
```

```
$\mathsf{corr}()$
req $[\![\neg\mathsf{prf}]\!]$
$\mathsf{corr} \leftarrow \mathit{true}$
return $\sigma$
```

```
$\mathsf{process}(I)$
$I \leftarrow \mathsf{sample\text{-}if\text{-}nec}(I)$
$(\sigma, R) \leftarrow \mathsf{KDF}(\sigma, I)$
return $R$
```

```
$\mathsf{chall\text{-}prf}(I)$
req $[\![\neg\mathsf{corr}]\!] \wedge [\![\neg\mathsf{prng}]\!]$
$\mathsf{prf} \leftarrow \mathit{true}$
$(\sigma', R) \leftarrow \mathsf{KDF}(\sigma, I)$
if $[\![b = 1]\!]$ then
    $R \leftarrow \mathsf{eval}(I)$
return $(\sigma', R)$
```

```
$\mathsf{chall\text{-}prng}(I)$
$I \leftarrow \mathsf{sample\text{-}if\text{-}nec}(I)$
req $[\![\neg\mathsf{corr}]\!] \wedge [\![\neg\mathsf{prf}]\!]$
$\mathsf{prng} \leftarrow \mathit{true}$
$(\sigma, R) \leftarrow \mathsf{KDF}(\sigma, I)$
if $[\![b = 1]\!]$ then
    $R \xleftarrow{\$} \mathcal{R}$
return $R$
```

```
$\mathsf{sample\text{-}if\text{-}nec}(I)$
if $[\![I = \bot]\!]$ then
    $I \xleftarrow{\$} \mathcal{I}$
    $\mathsf{corr} \leftarrow \mathsf{false}$
return $I$
```

```
$\mathsf{eval}(I)$
if $[\![\mathcal{F}[I] = \bot]\!]$ then
    $\mathcal{F}[I] \xleftarrow{\$} \mathcal{R}$
return $\mathcal{F}[I]$
```

Figure 3: PRF-PRNG security of a two-input KDF. If KDF expands to more than two return values, then the first one is $\sigma$ and the remaining tuple of outputs should be considered $R$ over an appropriate composite space $\mathcal{R}$.

$\mathsf{Decode}(\mathsf{L}) \xrightarrow{\$} M$ : *It takes as input a set* $\mathsf{L} \subset \mathbb{Z}_N \times \Sigma$ *such that* $|\mathsf{L}| \geqslant n_{\mathsf{chunk}}$ *and outputs a message* $M \in \Sigma^{n_{\mathsf{chunk}}}$.

An erasure code is said to be correct if for all messages $M \in \Sigma^{n_{\mathsf{chunk}}}$, for all $I \subset \mathbb{Z}_N$ such that $|I| = n_{\mathsf{chunk}}$, if $\mathsf{L} = \{(i, \mathsf{Encode}(M, i, n_{\mathsf{chunk}}) \mid i \in I)\}$ then $\mathsf{Decode}(\mathsf{L}, n_{\mathsf{chunk}}) = M$.

A correct erasure code can be instantiated using systematic Reed-Solomon codes, allowing an implementation to avoid decoding overhead in a typical case when no messages are dropped. Furthermore we note that using Reed-Solomon erasure codes over a finite field whose size is much larger than $n_{\mathsf{chunk}}$ makes the encoding effectively rateless, similar to a fountain code. Unlike fountain codes, however, we do not require linear time decoding but do require reconstruction with exactly $n_{\mathsf{chunk}}$ symbols.

## 2.5 Continuous Key Agreement

We follow the abstraction of continuous key agreement (CKA) put forth by Alwen, Coretti, and Dodis [ACD19]. A CKA is a two-party protocol between parties A and B that enables them to exchange a sequence of shared symmetric keys — roughly abstracting the public-ratchet of the Signal protocol. A CKA is a two-party protocol between parties A and B, where without loss of generality we assume A to be the initiating party of the communication.

**Definition 2.10.** *A* continuous key agreement (CKA) *protocol* $\Pi_{\mathsf{CKA}}$ *with initial key space* $\mathcal{I}_{\mathsf{CKA}}$*, key space* $\mathcal{K}$ *consists of PPT algorithms* $\big(\mathsf{CKA\text{-}Init\text{-}KeyGen}, (\mathsf{CKA\text{-}Init\text{-}P}, \mathsf{CKA\text{-}Send\text{-}P}, \mathsf{CKA\text{-}Rec\text{-}P})_{\mathsf{P} \in \{\mathsf{A},\mathsf{B}\}}\big)$ *defined as follows:*

$\mathsf{CKA\text{-}Init\text{-}KeyGen}(1^\lambda) \xrightarrow{\$} \mathsf{I}_{\mathsf{CKA}}$ : *It takes as input the security parameter* $1^\lambda$ *and outputs an* initial key $\mathsf{I}_{\mathsf{CKA}} \in \mathcal{I}_{\mathsf{CKA}}$.

$\mathsf{CKA\text{-}Init\text{-}A}(\mathsf{I}_{\mathsf{CKA}}) \xrightarrow{\$} \mathsf{st}_{\mathsf{A}}$ : *It takes as input an initial key* $\mathsf{I}_{\mathsf{CKA}} \in \mathcal{I}_{\mathsf{CKA}}$ *and outputs an initial state* $\mathsf{st}_{\mathsf{A}}$ *for party* $\mathsf{A}$.

$\mathsf{CKA\text{-}Send\text{-}A}(\mathsf{st}_{\mathsf{A}}) \xrightarrow{\$} (\mathsf{K}, \rho, \mathsf{st}_{\mathsf{A}})$ : *It takes as input a state* $\mathsf{st}_{\mathsf{A}}$ *of party* $\mathsf{A}$ *and outputs a key* $\mathsf{K} \in \mathcal{K}$, *a message* $\rho$ *and an updated state* $\mathsf{st}_{\mathsf{A}}$.

$\mathsf{CKA\text{-}Rec\text{-}A}(\mathsf{st}_{\mathsf{A}}, \rho) \to (\mathsf{K}, \mathsf{st}_{\mathsf{A}})$ : *It takes as input a state* $\mathsf{st}_{\mathsf{A}}$ *of party* $\mathsf{A}$ *and a message* $\rho$, *and outputs a key* $\mathsf{K} \in \mathcal{K} \cup \{\bot\}$, *and an updated state* $\mathsf{st}_{\mathsf{A}}$. *This algorithm is assumed to be deterministic.*

*In the above, we define algorithms* CKA-Init-B, CKA-Send-B, *and* CKA-Rec-B *analogously with roles of parties* A *and* B *swapped.*

*Remark* 2.11 (Alternating Communication). Following Alwen, Coretti, and Dodis [ACD19], we always assume parties A and B execute the sending and receiving algorithms in an alternating order. That is, CKA-Send-A → CKA-Rec-B → CKA-Send-B → CKA-Rec-A → $\cdots$. For instance, this restriction suffices to capture Signal's double ratchet protocol. Moreover, we assume without loss of generality that party A is always the first to send a message.

**Security.** A CKA scheme's correctness and security are formalized as in [ACD19] with the latter phrased as a real-or-random experiment for a (fixed) challenge epoch $\widehat{\mathsf{t}}^*$. For this epoch, the attacker is either given the real key output by the protocol, or an independent and fresh key. The game considers *passive* attacker that cannot modify or reorder the messages being delivered. The adversary can leak a party's protocol state as long as the party's epoch is not too close to the challenge epoch $\widehat{\mathsf{t}}^*$. More concretely, a party must recover from a state compromise within $\Delta_{\mathsf{PCS}}$ epochs and a state compromise must not endanger keys more than $\Delta_{\mathsf{FS}}$ epochs from the past.

More formally, we have the following, where note that we use separate parameters $\Delta_{\mathsf{FS}}$ and $\Delta_{\mathsf{PCS}}$ for FS and PCS, respectively, whereas [ACD19] hardcoded $\Delta_{\mathsf{PCS}} = 2$.

**Definition 2.12 (Key Indistinguishability).** *Let* $\Delta_{\mathsf{FS}}$ *and* $\Delta_{\mathsf{PCS}}$ *be positive integers, dictating how fast* forward secrecy *and* post-compromise security *come into effect. For a* CKA *protocol* $\Pi_{\mathsf{CKA}}$, *the advantage of an adversary* $\mathcal{A}$ *against* key indistinguishability *is defined as*

$$\mathsf{Adv}^{\mathsf{CKA}}_{\mathcal{A},\Delta_{\mathsf{FS}},\Delta_{\mathsf{PCS}}}(1^\lambda) := \max_{\widehat{\mathsf{t}}^*}\left(\Pr[\mathsf{Game}^{\mathsf{CKA}}_{\mathcal{A},\Delta_{\mathsf{FS}},\Delta_{\mathsf{PCS}},\widehat{\mathsf{t}}^*}(1^\lambda) = 1] - \frac{1}{2}\right),$$

*where* $\mathsf{Game}^{\mathsf{CKA}}_{\mathcal{A},\Delta_{\mathsf{FS}},\Delta_{\mathsf{PCS}},\widehat{\mathsf{t}}^*}(1^\lambda)$ *for any challenge epoch* $\widehat{\mathsf{t}}^* \in \mathbb{N}$ *is described in Fig. 4.*

*We say* $\Pi_{\mathsf{CKA}}$ *is* $(\Delta_{\mathsf{FS}}, \Delta_{\mathsf{PCS}})$*-key indistinguishable if for any efficient* $\mathcal{A}$ *that respects alternating communications (cf. Remark 2.11), we have* $\mathsf{Adv}^{\mathsf{CKA}}_{\mathcal{A},\Delta_{\mathsf{FS}},\Delta_{\mathsf{PCS}}}(1^\lambda) = \mathsf{negl}(\lambda)$. *In the context of alternating communications, it is understood that a call to* Chall-P *has the same effect as a call to* Send-P.

*Remark* 2.13 (Bad randomness). We deviate from [ACD19] (and other works on secure messaging) by not considering adversarially chosen randomness. Instead, we consider a slightly weaker model in which randomness is always honestly sampled but might be leaked to the adversary instead. We discuss mitigations against adversarially influenced randomness in Appendix B.

# 3 Hybrid Secure Messaging

In this work, we consider two-party secure messaging (SM) schemes that allow parties A and B to communicate securely. We will first recap the notion of a secure messaging protocol introduced by [ACD19], a two-party asynchronous interactive protocol allowing to securely exchange messages. This was originally used to formally model the Double Ratchet protocol by Signal. Later, in Section 4, we will instantiate this primitive with a hybrid secure messaging protocol.

## 3.1 Syntax

To define the syntax of a secure messaging scheme, we mostly follow [ACD19]. However, since we will consider a hybrid secure messaging protocol, we slightly generalize the syntax. Instead of having the receive algorithm output the epoch number and period of the message, we allow it to output a general message index that establishes an order on the received messages. (Recall that we generalize the Double Ratchet protocol which supports *immediate decryption*, i.e., the out-of-order receiving of messages.) In the following we only make the minimal assumption on the index set to have a partial order that allows to totally order all messages sent by each party — more expressive information encoding like causality between send and receive events can be supported as studied in [CF24].

**Definition 3.1.** *A* secure messaging (SM) *protocol* $\Pi_{\mathsf{TR}}$ *with initial key space* $\mathcal{I}_{\mathsf{K}}$*, message space* $\mathcal{M}$*, and index space* $(\mathcal{I}dx, \leqslant)$ *consists of PPT algorithms* $\big(\mathsf{SM\text{-}Init\text{-}KeyGen}, (\mathsf{SM\text{-}Init\text{-}P}, \mathsf{SM\text{-}Send\text{-}P}, \mathsf{SM\text{-}Rec\text{-}P})_{\mathsf{P}\in\{\mathsf{A},\mathsf{B}\}}\big)$ *defined as follows:*

$\mathsf{SM\text{-}Init\text{-}KeyGen}(1^\lambda) \xrightarrow{\$} \mathsf{I_K}$ : *It takes as input the security parameter* $1^\lambda$ *and outputs an* initial key $\mathsf{I_K} \in \mathcal{I}_{\mathsf{K}}$.

$\mathsf{SM\text{-}Init\text{-}A}(\mathsf{I_K}) \xrightarrow{\$} \mathsf{st_A}$ : *It takes as input an initial key* $\mathsf{I_K} \in \mathcal{I}_{\mathsf{K}}$ *and outputs an initial state* $\mathsf{st_A}$ *for party* $\mathsf{A}$.

$\mathsf{SM\text{-}Send\text{-}A}(\mathsf{st_A}, \mathsf{M}) \xrightarrow{\$} (\mathsf{ct}, \mathsf{st}'_{\mathsf{A}})$ : *It takes as input a state* $\mathsf{st}$ *of party* $\mathsf{A}$ *and a message* $\mathsf{M} \in \mathcal{M}$*, and outputs a ciphertext* $\mathsf{ct}$ *and an updated state* $\mathsf{st}'_{\mathsf{A}}$.

```
$\mathsf{Game}^{\mathsf{CKA}}_{\mathcal{A},\Delta_{\mathsf{FS}},\Delta_{\mathsf{PCS}},\hat{\mathsf{t}}^*}(1^\lambda)$
$b \xleftarrow{\$} \{0,1\}$
$\mathsf{I_{CKA}} \xleftarrow{\$} \mathsf{CKA\text{-}Init\text{-}KeyGen}(1^\lambda)$   // Initial key
for $\mathsf{P} \in \{\mathsf{A},\mathsf{B}\}$
    $\mathsf{st_P} \xleftarrow{\$} \mathsf{CKA\text{-}Init\text{-}P}(\mathsf{I_{CKA}})$
    $\hat{\mathsf{t}}_{\mathsf{P}} \leftarrow 0$
$b' \xleftarrow{\$} \mathcal{A}(\hat{\mathsf{t}}^*)^{\mathsf{Send\text{-}P}(),\mathsf{Receive\text{-}P}(),\mathsf{Chall\text{-}P}(),\mathsf{Corr\text{-}P}()}$
return $[\![b = b']\!]$
```

```
$\mathsf{Send\text{-}P}(\mathsf{rleak})$
$\hat{\mathsf{t}}_{\mathsf{P}} \leftarrow \hat{\mathsf{t}}_{\mathsf{P}} + 1$
if $[\![\mathsf{rleak}]\!]$   // Leak randomness
    // Allow leaking rand. $\Delta_{\mathsf{PCS}}$-epoch before $\hat{\mathsf{t}}^*$
    req $[\![\hat{\mathsf{t}}_{\mathsf{A}}, \hat{\mathsf{t}}_{\mathsf{B}} \leqslant \hat{\mathsf{t}}^* - \Delta_{\mathsf{PCS}}]\!]$
    $\mathsf{rand} \xleftarrow{\$} \mathcal{R}$
    $(\mathsf{K}_{\hat{\mathsf{t}}_{\mathsf{P}}}, \rho_{\hat{\mathsf{t}}_{\mathsf{P}}}, \mathsf{st_P}) \leftarrow \mathsf{CKA\text{-}Send\text{-}P}(\mathsf{st_P}; \mathsf{rand})$
else   // Secure randomness
    $\mathsf{rand} \leftarrow \bot$
    $(\mathsf{K}_{\hat{\mathsf{t}}_{\mathsf{P}}}, \rho_{\hat{\mathsf{t}}_{\mathsf{P}}}, \mathsf{st_P}) \xleftarrow{\$} \mathsf{CKA\text{-}Send\text{-}P}(\mathsf{st_P})$
return $(\mathsf{K}_{\hat{\mathsf{t}}_{\mathsf{P}}}, \rho_{\hat{\mathsf{t}}_{\mathsf{P}}}, \mathsf{rand})$
```

```
$\mathsf{Chall\text{-}P}()$
$\hat{\mathsf{t}}_{\mathsf{P}} \leftarrow \hat{\mathsf{t}}_{\mathsf{P}} + 1$
req $[\![\hat{\mathsf{t}}_{\mathsf{P}} = \hat{\mathsf{t}}^*]\!]$   // Challenge epoch $\hat{\mathsf{t}}^*$
$(\mathsf{K}_{\hat{\mathsf{t}}_{\mathsf{P}}}, \rho_{\hat{\mathsf{t}}_{\mathsf{P}}}, \mathsf{st_P}) \xleftarrow{\$} \mathsf{CKA\text{-}Send\text{-}P}(\mathsf{st_P})$
if $[\![b = 0]\!]$ then
    $\mathsf{K} \leftarrow \mathsf{K}_{\hat{\mathsf{t}}_{\mathsf{P}}}$
else
    $\mathsf{K} \xleftarrow{\$} \mathcal{K}$   // Replace with random key
return $(\mathsf{K}, \rho_{\hat{\mathsf{t}}_{\mathsf{P}}})$
```

```
$\mathsf{Receive\text{-}P}()$
$\hat{\mathsf{t}}_{\mathsf{P}} \leftarrow \hat{\mathsf{t}}_{\mathsf{P}} + 1$
$(\mathsf{K}, \mathsf{st_P}) \xleftarrow{\$} \mathsf{CKA\text{-}Rec\text{-}P}(\mathsf{st_P}, \rho_{\hat{\mathsf{t}}_{\mathsf{P}}})$
assert $[\![\mathsf{K} = \mathsf{K}_{\hat{\mathsf{t}}_{\mathsf{P}}}]\!]$   // Correctness
```

```
$\mathsf{Corr\text{-}P}()$
// Allow corrupting $\Delta_{\mathsf{PCS}}$-epoch before $\hat{\mathsf{t}}^*$
req $[\![\hat{\mathsf{t}}_{\mathsf{A}}, \hat{\mathsf{t}}_{\mathsf{B}} \leqslant \hat{\mathsf{t}}^* - \Delta_{\mathsf{PCS}}]\!]$
// Allow corrupting $\Delta_{\mathsf{FS}}$-epoch after $\hat{\mathsf{t}}^*$
req $[\![\hat{\mathsf{t}}_{\mathsf{P}} \geqslant \hat{\mathsf{t}}^* + \Delta_{\mathsf{FS}}]\!]$
return $\mathsf{st_P}$
```

Figure 4: Security game for continuous key agreement (CKA) protocol. With an overload of notation, in the above P denotes the variable that can be either A or B. For instance, it is understood that $\mathcal{A}$ is given oracle access to both Send-A and Send-B with the shorthand Send-P.

$\mathsf{SM\text{-}Rec\text{-}A}(\mathsf{st_A}, \mathsf{ct}) \xrightarrow{\$} (\mathsf{M}, \mathsf{idx}, \mathsf{st'_A})$ : *It takes as input a state* $\mathsf{st_A}$ *of party* $\mathsf{A}$ *and a ciphertext* $\mathsf{ct}$*, and outputs a message* $\mathsf{M} \in \mathcal{M}$*, a message index* $\mathsf{idx} \in \mathcal{I}dx$*, and an updated state* $\mathsf{st'_A}$*.*

*We define algorithms* SM-Init-B*,* SM-Send-B*, and* SM-Rec-B *analogously with roles of parties* A *and* B *swapped. For simplicity, we assume the state* $\mathsf{st_A}$ *to store* A*'s current index* $\mathsf{st_A.idx}$*, and analogously for user* B*.*[9]

## 3.2 Security

We formalize correctness and security as part of a combined security game as shown in Fig. 5, a generalization of the game from Alwen et al. [ACD19]. On a high level, the game allows the adversary to execute a protocol session by issuing send and receive commands. Furthermore, the attacker can try to break confidentiality by issuing challenges where either message $\mathsf{M}_0$ or $\mathsf{M}_1$ is sent depending on the game's challenge bit $b$, and can try to break authenticity by injecting their own ciphertexts. The game ensures the following properties:

**Correctness.** In the absence of an active attacker, B must output the message sent by A (and vice versa). Importantly, we require the protocol to support *immediate decryption* of incoming ciphertexts, even if ciphertexts are reordered on the network and some ciphertexts are dropped altogether. In addition, we require B to output the message index that matches the one stored as A's state just after the send operation, and we require the index stored in each party's state to strictly increase with each operation. Jointly, those properties allow the receiver to put all received messages into correct order.

**Authenticity.** The attacker cannot make a party accept ciphertexts that have not been sent, as long as neither party has been corrupted. After a state compromise, authenticity restores as long as the attacker remains passive and the compromised party has access to fresh randomness. We refer to this property as *post-compromise security* (PCS) and the game requires for PCS to restore security within $\Delta_{\mathsf{PCS}}$ epochs. Here, we measure epochs using an (efficiently computable) *epoch function* $\tau(\mathsf{idx})$ of the message indices. The epoch function is a parameter of the security game, and looking ahead, will depend on whether the classical or the post-quantum part of the protocol will be assumed secure.

**Privacy.** While the parties' states are uncompromised, the attacker obtains no information about the messages sent. Analogously to authenticity, privacy is required to restore after $\Delta_{\mathsf{PCS}}$ epochs after a state compromise. Furthermore, *forward secrecy* (FS) dictates that messages sent at least $\Delta_{\mathsf{FS}}$ epochs prior to a state compromise also remain secure. In other words, a state compromise may reveal messages of the last $\Delta_{\mathsf{FS}}$ and the next $\Delta_{\mathsf{PCS}}$ epochs.

In a bit more detail, the security game allows an adversary to control a messaging session, where either party can send and receive messages. Security is defined using a special challenge oracle that takes two messages, with the adversary's goal to guess which one was encrypted. Parties can moreover be corrupted where two predicates safe-corr and safe-chall rule out trivial attacks by challenging before PCS kicked in after a corruption, or corrupting before FS kicked in after a challenge, respectively. The game uses "semi-active" adversaries as in [ACD19], where the adversary has to behave passively after a corruption until PCS restores security, but can otherwise try to break authenticity by injecting other ciphertexts. We discuss authenticity more below.

Recall that for a hybrid SM scheme the receive algorithm SM-Rec-P returns the index idx of the received message, while the sender stores the index of the last sent message as part of their protocol state. In the security game, correctness thus enforces that the recipient outputs the correct message index (in addition to the correct message) as part of the respective oracle. The game moreover uses an *epoch function* $\tau(\mathsf{idx})$ defined on those message indices to abstract the handling of epochs, which can advance at different velocities depending on whether we consider classical or post-quantum security. FS and PCS are then defined in the number of epochs $\Delta_{\mathsf{FS}}$ and $\Delta_{\mathsf{PCS}}$, respectively, describing the corruption window. The game uses the

[9] Alternatively, SM-Init-A, SM-Send-A, and SM-Rec-A could each output this index.

```
$\mathsf{Game}^{\mathsf{SM}}_{\mathcal{A},\Delta_{\mathsf{PCS}},\Delta_{\mathsf{FS}},\tau}(1^\lambda)$
$b \xleftarrow{\$} \{0,1\}$
$\mathsf{I_K} \xleftarrow{\$} \mathsf{SM\text{-}Init\text{-}KeyGen}(1^\lambda)$    // Sample initial key
for $\mathsf{P} \in \{\mathsf{A},\mathsf{B}\}$
    $\mathsf{st_P} \xleftarrow{\$} \mathsf{SM\text{-}Init\text{-}P}(\mathsf{I_K})$
    $(\mathsf{t_{Chall\text{-}P}}, \mathsf{idx_P}) \leftarrow (0, -\infty)$
$\mathsf{t_L} \leftarrow -\infty$
$L_{\mathsf{trans}}, L_{\mathsf{chall}}, L_{\mathsf{comp}} \leftarrow \varnothing$
$b' \xleftarrow{\$} \mathcal{A}(1^\lambda)^{\mathsf{Send\text{-}P}(),\mathsf{Receive\text{-}P}(),\mathsf{Chall\text{-}P}(),\mathsf{Corr\text{-}P}()}$
return $\llbracket b = b' \rrbracket$
```

```
$\mathsf{Chall\text{-}A}(\mathsf{M}_0, \mathsf{M}_1, \mathsf{rleak})$
$\mathsf{rand} \xleftarrow{\$} \mathcal{R}$
req $\llbracket |\mathsf{M}_0| = |\mathsf{M}_1| \rrbracket$
$(\mathsf{ct}, \mathsf{st_A}) \leftarrow \mathsf{SM\text{-}Send\text{-}A}(\mathsf{st_A}, \mathsf{M}_b; \mathsf{rand})$
$\mathsf{epoch\text{-}mgmt}(\mathsf{A}, \mathsf{chall}, \mathsf{rleak})$
req $\llbracket \mathsf{safe\text{-}chall}(\mathsf{A}) \rrbracket$
$\mathsf{record} := (\mathsf{A}, \mathsf{M}_b, \mathsf{idx_A}, \mathsf{ct})$
$L_{\mathsf{trans}}, L_{\mathsf{chall}}, L_{\mathsf{comp}} \xleftarrow{+} \mathsf{record}$
if $\llbracket \neg\mathsf{rleak} \rrbracket$ then $\mathsf{rand} \leftarrow \bot$
return $(\mathsf{ct}, \mathsf{idx_A}, \mathsf{rand})$
```

```
$\mathsf{Inject\text{-}A}(\mathsf{ct})$
req $\llbracket (\mathsf{B}, \_, \_, \mathsf{ct}) \notin L_{\mathsf{trans}} \rrbracket \wedge \llbracket \mathsf{safe\text{-}inj}() \rrbracket$
$(\mathsf{M}', \mathsf{idx}', \mathsf{st_A}) \xleftarrow{\$} \mathsf{SM\text{-}Rec\text{-}A}(\mathsf{st_A}, \mathsf{ct})$
$\mathsf{epoch\text{-}mgmt}(\mathsf{A}, \mathsf{receive}, \mathsf{false})$
// Authenticity guarantee
if $\llbracket \mathsf{M}' \neq \bot \rrbracket$ then
    assert $\exists \mathsf{idx}'' : \llbracket \mathsf{equiv}(\mathsf{idx}', \mathsf{idx}'') \rrbracket$
                $\wedge \llbracket (\mathsf{B}, \_, \mathsf{idx}'', \_) \in L_{\mathsf{comp}} \rrbracket$
foreach $\mathsf{idx}'' : \mathsf{equiv}(\mathsf{idx}', \mathsf{idx}'')$
    if $\llbracket (\mathsf{B}, \_, \mathsf{idx}'', \_) \in L_{\mathsf{trans}} \rrbracket$ then
        $L_{\mathsf{trans}}, L_{\mathsf{chall}}, L_{\mathsf{comp}} \xleftarrow{-} (\mathsf{B}, \_, \mathsf{idx}'', \_)$
return $(\mathsf{M}', \mathsf{idx}')$
```

```
$\mathsf{Send\text{-}A}(\mathsf{M}, \mathsf{rleak})$
$\mathsf{rand} \xleftarrow{\$} \mathcal{R}$
$(\mathsf{ct}, \mathsf{st_A}) \leftarrow \mathsf{SM\text{-}Send\text{-}A}(\mathsf{st_A}, \mathsf{M}; \mathsf{rand})$
$\mathsf{epoch\text{-}mgmt}(\mathsf{A}, \mathsf{send}, \mathsf{rleak})$
$\mathsf{record} := (\mathsf{A}, \mathsf{M}, \mathsf{idx_A}, \mathsf{ct})$
$L_{\mathsf{trans}} \xleftarrow{+} \mathsf{record}$
if $\llbracket \neg\mathsf{safe\text{-}chall}(\mathsf{A}) \rrbracket$ then
    $L_{\mathsf{comp}} \xleftarrow{+} \mathsf{record}$
if $\llbracket \neg\mathsf{rleak} \rrbracket$ then $\mathsf{rand} \leftarrow \bot$
return $(\mathsf{ct}, \mathsf{idx_A}, \mathsf{rand})$
```

```
$\mathsf{Receive\text{-}A}(\mathsf{ct})$
req $\llbracket (\mathsf{B}, \_, \_, \mathsf{ct}) \in L_{\mathsf{trans}} \rrbracket$
$(\mathsf{M}', \mathsf{idx}', \mathsf{st_A}) \xleftarrow{\$} \mathsf{SM\text{-}Rec\text{-}A}(\mathsf{st_A}, \mathsf{ct})$
$\mathsf{epoch\text{-}mgmt}(\mathsf{A}, \mathsf{receive}, \mathsf{false})$
$\mathsf{record} := (\mathsf{B}, \mathsf{M}', \mathsf{idx}', \mathsf{ct})$
// Correctness guarantee
assert $\llbracket \mathsf{record} \in L_{\mathsf{trans}} \rrbracket$
if $\llbracket \mathsf{record} \in L_{\mathsf{chall}} \rrbracket$ then
    $\mathsf{M}' \leftarrow \bot$
$L_{\mathsf{trans}}, L_{\mathsf{chall}}, L_{\mathsf{comp}} \xleftarrow{-} \mathsf{record}$
return $(\mathsf{M}', \mathsf{idx}')$
```

```
$\mathsf{Corr\text{-}A}()$
req $\llbracket (\mathsf{B}, \_, \_, \_) \notin L_{\mathsf{chall}} \rrbracket \wedge \llbracket \mathsf{safe\text{-}corr}(\mathsf{A}) \rrbracket$
foreach $(\mathsf{B}, \mathsf{M}', \mathsf{idx}', \mathsf{ct}') \in L_{\mathsf{trans}}$
    $L_{\mathsf{comp}} \xleftarrow{+} (\mathsf{B}, \mathsf{M}', \mathsf{idx}', \mathsf{ct}')$
$\mathsf{t_L} \leftarrow \max(\mathsf{t_A}, \mathsf{t_B})$
return $\mathsf{st_A}$
```

```
$\mathsf{epoch\text{-}mgmt}(\mathsf{P}, \mathsf{act}, \mathsf{rleak})$
$\mathsf{idx} \leftarrow \mathsf{st_P}.\mathsf{idx}$
$(\mathsf{t}, \mathsf{t_P}) \leftarrow (\tau(\mathsf{idx}), \tau(\mathsf{idx_P}))$
$(\mathsf{i}, \mathsf{i_P}) \leftarrow (\imath(\mathsf{idx}), \imath(\mathsf{idx_P}))$
if $\llbracket \mathsf{act} \in \{\mathsf{send}, \mathsf{chall}\} \rrbracket$ then
    assert $\llbracket \mathsf{idx} > \mathsf{idx_P} \rrbracket$
else assert $\llbracket \mathsf{idx} \geqslant \mathsf{idx_P} \rrbracket$
assert $\llbracket \mathsf{t} \geqslant \mathsf{t_P} \rrbracket$
if $\llbracket \mathsf{t} > \mathsf{t_P} \rrbracket$ then
    assert $\llbracket \mathsf{i} = 1 \rrbracket$
elseif $\llbracket \mathsf{act} \in \{\mathsf{send}, \mathsf{chall}\} \rrbracket$ then
    assert $\llbracket \mathsf{i} = \mathsf{i_P} + 1 \rrbracket$
else assert $\llbracket \mathsf{i} = \mathsf{i_P} \rrbracket$
$\mathsf{corr\text{-}mgmt}(\mathsf{P}, \mathsf{act}, \mathsf{rleak}, \mathsf{t}, \mathsf{t_P})$
$\mathsf{idx_P} \leftarrow \mathsf{idx}$
```

```
$\mathsf{corr\text{-}mgmt}(\mathsf{P}, \mathsf{act}, \mathsf{rleak}, \mathsf{t}, \mathsf{t_P})$
if $\llbracket \mathsf{act} = \mathsf{chall} \rrbracket$ then $\mathsf{t_{Chall\text{-}P}} \leftarrow \mathsf{t}$
if $\llbracket \mathsf{rleak} \rrbracket$ then $\mathsf{bad\text{-}rand_P} \leftarrow \mathsf{true}$
if $\llbracket \mathsf{t} > \mathsf{t_P} \rrbracket \wedge \llbracket \mathsf{bad\text{-}rand_P} \rrbracket$ then
    while $\llbracket \mathsf{t_P} < \mathsf{t} \rrbracket$
        $\mathsf{t_P} \mathrel{+}= 1$
        if $\llbracket \mathsf{sending\text{-}ep}(\mathsf{P}, \mathsf{t}) \rrbracket$
                $\wedge \llbracket \neg\mathsf{safe\text{-}chall}(\mathsf{P}) \rrbracket$ then
            $\mathsf{t_L} \leftarrow \max(\mathsf{t_L}, \mathsf{t})$
    $\mathsf{bad\text{-}rand_P} \leftarrow \mathsf{false}$
```

```
$\mathsf{equiv}(\mathsf{idx}_1, \mathsf{idx}_2)$
return $\llbracket \tau(\mathsf{idx}_1) = \tau(\mathsf{idx}_2) \rrbracket$
        $\wedge \llbracket \imath(\mathsf{idx}_1) = \imath(\mathsf{idx}_2) \rrbracket$
```

```
$\mathsf{sending\text{-}ep}(\mathsf{P}, \mathsf{t})$
return $\llbracket \mathsf{P} = \mathsf{A}$ and $\mathsf{t}$ is odd$\rrbracket$
        $\vee \llbracket \mathsf{P} = \mathsf{B}$ and $\mathsf{t}$ is even$\rrbracket$
```

```
$\mathsf{safe\text{-}chall}(\mathsf{P})$    // $\Delta_{\mathsf{PCS}}$ after last corruption
return $\llbracket \mathsf{t_P} \geqslant \mathsf{t_L} + \Delta_{\mathsf{PCS}} \rrbracket$
```

```
$\mathsf{safe\text{-}inj}()$    // Once both parties healed
return $\llbracket \min(\mathsf{t_A}, \mathsf{t_B}) \geqslant \mathsf{t_L} + \Delta_{\mathsf{PCS}} \rrbracket$
```

```
$\mathsf{safe\text{-}corr}(\mathsf{P})$    // $\Delta_{\mathsf{FS}}$ epochs after last challenge
return $\llbracket \mathsf{t_P} \geqslant \mathsf{t_{Chall\text{-}P}} + \Delta_{\mathsf{FS}} \rrbracket$
```

Figure 5: The SM security game parametrized in the epoch function $\tau$, the number of epochs $\Delta_{\mathsf{PCS}}$ for PCS, and the number of epochs $\Delta_{\mathsf{FS}}$ for FS. The period function $\imath$ is used for bookkeeping purposes and not security relevant. The oracles for $\mathsf{B}$ are defined analogously.

helper algorithm $\mathsf{epoch\text{-}mgmt}$ to ensure consistency of the indices and the epoch function: For each operation, indices must strictly increase while the associated epoch must be monotonic. The additional helper algorithm $\mathsf{corr\text{-}mgmt}$ moreover keeps track of corrupted epochs — updating the last corrupted epoch $\mathsf{t_L}$ in case the party does not have access to good randomness — and the last epoch each party has been challenged.

Finally, the game also uses a *period function* $\imath(\mathsf{idx})$ for bookkeeping. Within each epoch, periods have to start at 1 and then increment on each send operation. Periods are then used to formalize the precise authenticity guarantees. Recall that we said that the attacker may try to inject messages as long as neither party is currently compromised (as formalized by $\mathsf{safe\text{-}inj}$.) If all messages have been delivered, then we expect that no injections can be performed outside such a window of compromise. This is, however not necessarily true if delayed messages for which the keys where compromised have not been delivered. The game keeps track of those messages using $L_{\mathsf{comp}}$ and then permits injecting the *same number* of messages without being counted as a compromise. In other words if Alice sent ten messages not yet delivered while being compromised, then the attacker may substitute those ten messages but must not be able to inject an eleventh. This is checked by ensuring that for each message in $L_{\mathsf{comp}}$ only one injection happens with the same epoch-period pair. (The overall message index, however, may differ.)

**Definition 3.2.** *For a* SM *protocol* $\Pi_{\mathsf{SM}}$ *with message index space* $\mathcal{I}dx$*, let* $\tau\colon \mathcal{I}dx \to \mathbb{N}$ *be an* epoch function *that dictates how fast* forward secrecy *and* post-compromise security *come into effect, measured as positive integers* $\Delta_{\mathsf{FS}}$ *and* $\Delta_{\mathsf{PCS}}$*, respectively. The advantage of an adversary* $\mathcal{A}$ *is defined as*

$$\mathsf{Adv}^{\mathsf{SM}}_{\mathcal{A},\Delta_{\mathsf{PCS}},\Delta_{\mathsf{FS}},\tau}(1^\lambda) := \left|\Pr[\mathsf{Game}^{\mathsf{SM}}_{\mathcal{A},\Delta_{\mathsf{PCS}},\Delta_{\mathsf{FS}},\tau}(1^\lambda) = 1] - \frac{1}{2}\right|$$

*where the game is described in Fig. 5. We say* $\Pi_{\mathsf{SM}}$ *is* $(\Delta_{\mathsf{FS}}, \Delta_{\mathsf{PCS}}, \tau)$*-secure if for any efficient* $\mathcal{A}$ *we have* $Adv^{\mathsf{SM}}_{\mathcal{A},\Delta_{\mathsf{PCS}},\Delta_{\mathsf{FS}},\tau}(1^\lambda) = \mathsf{negl}(\lambda)$.

Note that our game generalizes the one by Alwen et al. [ACD19] to hybrid messaging and deviates in the following ways:

Message indices: Whereas the game in [ACD19] kept track of epochs and periods in a predetermined manner — with epochs changing on every change in communication direction and periods incrementing for each message within an epoch — we use the more general message indices to formalize correctness.

Epoch function: Along the same line, our game makes use of the abstract epoch function $\tau$ to formalize FS and PCS. We remark that for our concrete scheme the two choices of $\tau$, for classical and post-quantum security, will be unambiguous. Intuitively, the post-quantum part will utilize a slower incrementing epoch function translating into slower FS and PCS.

Randomness leakage: Whereas [ACD19] considered adversarially chosen randomness, we consider honestly sampled but leaked randomness only (cf. Appendix B).

# 4 The Triple Ratchet

## 4.1 Construction

We now present the Triple Ratchet protocol, building on the seminal Double Ratchet protocol for secure messaging. The Triple Ratchet protocol combines a classically secure CKA with a post-quantum secure CKA protocol. The classically secure part of the protocol directly follows the modularization of the Double Ratchet put forth by [ACD19]. Since the post-quantum CKA messages are significantly larger than their classical counterparts, however, each post-quantum CKA message is split into $n_{\mathsf{chunk}}$ many chunks and sent alongside multiple (application) messages. To retain immediate decryption, i.e., to ensure functioning of the protocol even if individual messages are dropped, an *erasure code* is used. The protocol is presented in Figs. 6 to 9. To ease presentation, we use the following conventions: We depict the classical part in the left column with

| | | |
|---|---|---|
| Epochs | $\mathsf{tR}$ | The epoch under which received messages are encrypted. |
| | $\mathsf{tS}$ | The key epoch under which sent messages are encrypted.<br>Invariant: $\mathsf{tS} \in \{\mathsf{tR}, \mathsf{tR}+1\}$ |
| | $\mathsf{tCurr}$ | The epoch for which key material is being exchanged.<br>Invariant: $\mathsf{tCurr} \in \{\mathsf{tS}, \mathsf{tS}+1\}$ |
| Periods | $\mathsf{iS}$ | The number of messages sent in epoch $\mathsf{tS}$. |
| | $\mathsf{iS}_{\text{-}1}, \mathsf{iS}_{\text{-}2}$ | The number of messages sent in epochs $\mathsf{tS}-1$ and $\mathsf{tS}-2$. |
| | $\mathsf{iR}$ | The number of messages received in epoch $\mathsf{tR}$. |
| Keys | $\mathsf{K}_{\mathsf{root}}$ | The current root key of the asymmetric ratchet. |
| | $\mathsf{KS}$ | The current sending key for epoch $\mathsf{tS}$. |
| | $\mathsf{KS}_{+1}$ | The sending key for epoch $\mathsf{tS}+1$ (if already known) |
| | $\mathsf{KR}$ | The current receiving key for epoch $\mathsf{tR}$. |
| | $\mathsf{KR}_{+1}, \mathsf{KR}_{+2}$ | The receiving keys for epoch $\mathsf{tR}+1$ and $\mathsf{tR}+2$ (if already known). |
| | $\mathsf{StoredKeys}[\mathsf{t}, \mathsf{i}]$ | Stored keys for processing out-of-order messages. |
| Chunks | $\mathsf{c}_{\mathsf{R}}$ | The number of chunks received for $\mathsf{tCurr}$ (for receiving epochs). |
| | $\mathsf{c}_{\mathsf{S}}$ | The number of chunks sent for $\mathsf{tCurr}$ (for sending epochs). |
| | $\mathsf{c}_{\mathsf{Ack}}$ | The number of chunks acknowledged for $\mathsf{tCurr}$ (for receiving epochs). Invariant: $\mathsf{c}_{\mathsf{Ack}} \leqslant \mathsf{c}_{\mathsf{S}}$. |
| | $\mathsf{L}$ | The set of chunk-period pairs received of the next $\mathsf{CKA}$ message. |

Table 1: Protocol variables used by the post-quantum part of the Triple Ratchet protocol, by each party. For simplicity $\mathsf{Q}$ superscripts have been omitted.

the post-quantum part in the right column.[10] Shared parts run before and after the two sub-protocols are depicted in the center. The two sub-protocols are independent of each other and can, in principle, be run in parallel. In particular, they use disjoint sets of variables, with corresponding variables either denoted with a superscript $\mathsf{C}$ (for classical) or $\mathsf{Q}$ (for post-quantum) — for example, $\mathsf{tCurr}^{\mathsf{C}}$ and $\mathsf{tCurr}^{\mathsf{Q}}$ denote the two independent epoch counters of the two CKAs. To reduce clutter we omit those superscripts whenever clear from the context which protocol part they refer to. Finally, the protocol maintains implicit state. In the following, we mainly describe the post-quantum part of the protocol, referring to [ACD19] for an in-depth discussion of the classical part.

**Exchanging CKA messages.** Analogous to the original Double Ratchet, parties take turns in exchanging CKA messages. If a party wants to send an application message while the other party is distributing chunks of their CKA message, the party will simply acknowledge the number of chunks they already received without sending their own chunks. As such, we still say that A acts as the sender in odd epochs and as the receiver in even epochs. More concretely,

- In TR-Send-A, on line 17 the party A checks whether they are currently in a sender or receiver epoch.
- In the former case (lines 18-25) A sends an additional CKA chunk $\rho_{\mathsf{enc}}$ to the receiver. It keeps track of the number of chunks sent for the current CKA message $\rho$ using $\mathsf{c}_{\mathsf{S}}$. (Note that $\mathsf{c}_{\mathsf{S}}$ is not necessarily equal to the sending period $\mathsf{iS}$, as the sending epoch can change while sending $\rho$, resetting $\mathsf{iS}$.)
- In the latter case (lines 27 and 28) A simply acknowledges the number of chunks received $\mathsf{c}_{\mathsf{R}}$ of $\rho$ sent by the other party. The other party B then uses this information (as stored in $\mathsf{c}_{\mathsf{Ack}}$) to deduce when the new CKA key becomes usable in TR-Rec-B and TR-Send-B (as discussed later).

[10] Note that the classical part technically can be seen as a simplification of the post-quantum protocol for $n_{\mathsf{chunk}} = 1$ with certain optimizations applied.

```
$\mathsf{TR\text{-}Init\text{-}KeyGen}(1^\lambda)$
$\mathsf{I_{CKA}} \xleftarrow{\$} \mathsf{CKA\text{-}Init\text{-}KeyGen}^{\mathsf{C}}(1^\lambda)$
$(\mathsf{K_{root}}, \mathsf{K_{CKA}}) \xleftarrow{\$} \mathcal{K}_{\mathsf{PP}} \times \mathcal{K}^{\mathsf{C}}_{\mathsf{CKA}}$
$\mathsf{I}^{\mathsf{C}}_{\mathsf{K}} \leftarrow (\mathsf{I_{CKA}}, \mathsf{K_{root}}, \mathsf{K_{CKA}})$
$\mathsf{I_{CKA}} \xleftarrow{\$} \mathsf{CKA\text{-}Init\text{-}KeyGen}^{\mathsf{Q}}(1^\lambda)$
$(\mathsf{K_{root}}, \mathsf{K_{CKA}}) \xleftarrow{\$} \mathcal{K}_{\mathsf{PP}} \times \mathcal{K}^{\mathsf{Q}}_{\mathsf{CKA}}$
$\mathsf{I}^{\mathsf{Q}}_{\mathsf{K}} \leftarrow (\mathsf{I_{CKA}}, \mathsf{K_{root}}, \mathsf{K_{CKA}})$
$\textbf{return } \mathsf{I_K} := (\mathsf{I}^{\mathsf{C}}_{\mathsf{K}}, \mathsf{I}^{\mathsf{Q}}_{\mathsf{K}})$
```

```
$\mathsf{TR\text{-}Init\text{-}A}(\mathsf{I_K})$
$\textbf{parse } (\mathsf{I}^{\mathsf{C}}_{\mathsf{K}}, \mathsf{I}^{\mathsf{Q}}_{\mathsf{K}}) \leftarrow \mathsf{I_K}$
    $\textbf{parse } (\mathsf{I_{CKA}}, \mathsf{K_{root}}, \mathsf{K_{CKA}}) \leftarrow \mathsf{I}^{\mathsf{C}}_{\mathsf{K}}$
    $(\mathsf{K_{root}}, \mathsf{KR}) \leftarrow \mathsf{KDF}_1(\mathsf{K_{root}}, \mathsf{K_{CKA}})$
    $\mathsf{KS} \leftarrow \bot$
    $(\mathsf{tCurr}, \mathsf{iR}, \mathsf{iS}, \mathsf{iS}_{\text{-}2}) \leftarrow 0$
    $\overline{\mathsf{st}}_{\mathsf{A}} \xleftarrow{\$} \mathsf{CKA\text{-}Init\text{-}A}(\mathsf{I_{CKA}})$
    $\rho \leftarrow \bot$
    $\mathsf{StoredKeys}^{\mathsf{C}}[\cdot] := \bot$
    $\textbf{parse } (\mathsf{I_{CKA}}, \mathsf{K_{root}}, \mathsf{K_{CKA}}) \leftarrow \mathsf{I}^{\mathsf{Q}}_{\mathsf{K}}$
    $(\mathsf{K_{root}}, \mathsf{KS}, \mathsf{KR}_{+1}) \leftarrow \mathsf{KDF}_1(\mathsf{K_{root}}, \mathsf{K_{CKA}})$
    $(\mathsf{KS}_{+1}, \mathsf{KR}, \mathsf{KR}_{+2}) \leftarrow \bot$
    $(\mathsf{tCurr}, \mathsf{tS}, \mathsf{iR}, \mathsf{iS}, \mathsf{iS}_{\text{-}1}, \mathsf{iS}_{\text{-}2}) \leftarrow 0$
    $\mathsf{tR} \leftarrow -1$
    $(\mathsf{c_S}, \mathsf{c_{Ack}}, \mathsf{c_R}) \leftarrow (0, 0, n_{\mathsf{chunk}})$
    $\overline{\mathsf{st}}_{\mathsf{A}} \xleftarrow{\$} \mathsf{CKA\text{-}Init\text{-}A}(\mathsf{I_{CKA}})$
    $\rho \leftarrow \bot$
    $\mathsf{StoredKeys}^{\mathsf{Q}}[\cdot] := \bot$
```

```
$\mathsf{TR\text{-}Init\text{-}B}(\mathsf{I_K})$
$\textbf{parse } (\mathsf{I}^{\mathsf{C}}_{\mathsf{K}}, \mathsf{I}^{\mathsf{Q}}_{\mathsf{K}}) \leftarrow \mathsf{I_K}$
    $\textbf{parse } (\mathsf{I_{CKA}}, \mathsf{K_{root}}, \mathsf{K_{CKA}}) \leftarrow \mathsf{I}^{\mathsf{C}}_{\mathsf{K}}$
    $(\mathsf{K_{root}}, \mathsf{KS}) \leftarrow \mathsf{KDF}_1(\mathsf{K_{root}}, \mathsf{K_{CKA}})$
    $\mathsf{KR} \leftarrow \bot$
    $(\mathsf{tCurr}, \mathsf{iR}, \mathsf{iS}, \mathsf{iS}_{\text{-}2}) \leftarrow 0$
    $\overline{\mathsf{st}}_{\mathsf{B}} \xleftarrow{\$} \mathsf{CKA\text{-}Init\text{-}B}(\mathsf{I_{CKA}})$
    $\rho \leftarrow \bot$
    $\mathsf{StoredKeys}^{\mathsf{C}}[\cdot] := \bot$
    $\textbf{parse } (\mathsf{I_{CKA}}, \mathsf{K_{root}}, \mathsf{K_{CKA}}) \leftarrow \mathsf{I}^{\mathsf{Q}}_{\mathsf{K}}$
    $(\mathsf{K_{root}}, \mathsf{KR}_{+1}, \mathsf{KS}) \leftarrow \mathsf{KDF}_1(\mathsf{K_{root}}, \mathsf{K_{CKA}})$
    $(\mathsf{KS}_{+1}, \mathsf{KR}, \mathsf{KR}_{+2}) \leftarrow \bot$
    $(\mathsf{tCurr}, \mathsf{tS}, \mathsf{iR}, \mathsf{iS}, \mathsf{iS}_{\text{-}1}, \mathsf{iS}_{\text{-}2}) \leftarrow 0$
    $\mathsf{tR} \leftarrow -1$
    $(\mathsf{c_S}, \mathsf{c_{Ack}}, \mathsf{c_R}) \leftarrow (n_{\mathsf{chunk}}, n_{\mathsf{chunk}}, 0)$
    $\overline{\mathsf{st}}_{\mathsf{B}} \xleftarrow{\$} \mathsf{CKA\text{-}Init\text{-}B}(\mathsf{I_{CKA}})$
    $\rho \leftarrow \bot$
    $\mathsf{StoredKeys}^{\mathsf{Q}}[\cdot] := \bot$
```

Figure 6: Setup algorithms of the Triple Ratchet protocol. The classical part (left-hand side) and the post-quantum part (right-hand side) use disjoint variables, indicated by superscripts $\mathsf{C}$ and $\mathsf{Q}$, respectively. For ease of reading, we omit those superscripts whenever clear from the context.

```
$\mathsf{skip}^{\mathsf{X}}(\mathsf{t}, \mathsf{iR}')$    // for $\mathsf{X} \in \{\mathsf{C}, \mathsf{Q}\}$
$\textbf{while } \mathsf{iR}^{\mathsf{X}} < \mathsf{iR}'$
    $\mathsf{iR}^{\mathsf{X}} \mathrel{+}= 1$
    $(\mathsf{KR}^{\mathsf{X}}, \mathsf{K}^{\mathsf{X}}_{\mathsf{aead}}) \leftarrow \mathsf{KDF}_2(\mathsf{KR}^{\mathsf{X}})$
    $\mathsf{StoredKeys}^{\mathsf{X}}[\mathsf{t}, \mathsf{iR}^{\mathsf{X}}] \leftarrow \mathsf{K_{aead}}$
```

```
$\mathsf{try\text{-}skipped}^{\mathsf{X}}(\mathsf{t}, \mathsf{i})$    // for $\mathsf{X} \in \{\mathsf{C}, \mathsf{Q}\}$
$\mathsf{K}^{\mathsf{X}}_{\mathsf{aead}} \leftarrow \mathsf{StoredKeys}[\mathsf{t}, \mathsf{i}]$
$\mathsf{StoredKeys}^{\mathsf{X}}[\mathsf{t}, \mathsf{i}] \leftarrow \bot$
$\textbf{return } \mathsf{K}^{\mathsf{X}}_{\mathsf{aead}}$
```

Figure 7: Helper algorithms of the Triple Ratchet protocol.

```
TR-Send-A(M)
1 :  if $[\![\mathsf{tCurr}\text{ is even}]\!]$ then
2 :     $\mathsf{tCurr} \mathrel{+}= 1$
3 :     $(\mathsf{K}_{\mathsf{CKA}}, \rho, \overline{\mathsf{st}}_{\mathsf{A}}) \xleftarrow{\$} \mathsf{CKA\text{-}Send\text{-}A}(\overline{\mathsf{st}}_{\mathsf{A}})$
4 :     $(\mathsf{K}_{\mathsf{root}}, \mathsf{KS}) \leftarrow \mathsf{KDF}_1(\mathsf{K}_{\mathsf{root}}, \mathsf{K}_{\mathsf{CKA}})$
5 :     $(\mathsf{iS}, \mathsf{iS}_{\text{-}2}) \leftarrow (0, \mathsf{iS})$
6 :  endif
7 :  $\mathsf{iS} \mathrel{+}= 1$
8 :  $\mathsf{h}^{\mathsf{C}} := (\mathsf{tCurr}, \mathsf{iS}, \rho, \mathsf{iS}_{\text{-}2})$
9 :  $(\mathsf{KS}, \mathsf{K}^{\mathsf{C}}_{\mathsf{aead}}) \leftarrow \mathsf{KDF}_2(\mathsf{KS})$

10 : if $[\![\mathsf{tS} = \mathsf{tCurr}]\!] \wedge [\![\mathsf{tCurr}\text{ is even}]\!]$ then
11 :    $\mathsf{tCurr} \mathrel{+}= 1$   // start sending next key
12 :    $(\mathsf{K}_{\mathsf{CKA}}, \rho, \overline{\mathsf{st}}_{\mathsf{A}}) \xleftarrow{\$} \mathsf{CKA\text{-}Send\text{-}A}(\overline{\mathsf{st}}_{\mathsf{A}})$
13 :    $(\mathsf{K}_{\mathsf{root}}, \mathsf{KS}_{+1}, \mathsf{KR}_{+2}) \leftarrow \mathsf{KDF}_1(\mathsf{K}_{\mathsf{root}}, \mathsf{K}_{\mathsf{CKA}})$
14 :    if $[\![\mathsf{tR} = \mathsf{tS}]\!]$ then
15 :       $(\mathsf{KR}_{+1}, \mathsf{KR}_{+2}) \leftarrow (\mathsf{KR}_{+2}, \perp)$
16 :    $(\mathsf{c}_{\mathsf{S}}, \mathsf{c}_{\mathsf{Ack}}) \leftarrow (0, 0)$
17 : if $[\![\mathsf{tCurr}\text{ is odd}]\!]$ then   // sending chunks
18 :    $\rho_{\mathsf{enc}} \leftarrow \mathsf{Encode}(\rho, \mathsf{c}_{\mathsf{S}})$
19 :    $\mathsf{c}_{\mathsf{S}} \mathrel{+}= 1$
20 :    if $[\![\mathsf{tS} < \mathsf{tCurr}]\!] \wedge [\![\mathsf{c}_{\mathsf{Ack}} + 1 \geqslant n_{\mathsf{chunk}}]\!]$ then
21 :       $\mathsf{tS} \leftarrow \mathsf{tCurr}$   // start using next key
22 :       $(\mathsf{iS}, \mathsf{iS}_{\text{-}1}, \mathsf{iS}_{\text{-}2}) \leftarrow (0, \mathsf{iS}, \mathsf{iS}_{\text{-}1})$
23 :       $(\mathsf{KS}, \mathsf{KS}_{+1}) \leftarrow (\mathsf{KS}_{+1}, \perp)$
24 :    $\mathsf{iS} \mathrel{+}= 1$
25 :    $\mathsf{h}^{\mathsf{Q}} := (\mathsf{tS}, \mathsf{iS}, \mathsf{tCurr}, \rho_{\mathsf{enc}}, \perp, \mathsf{iS}_{\text{-}1}, \mathsf{iS}_{\text{-}2})$
26 : else   // acknowledging chunks
27 :    $\mathsf{iS} \mathrel{+}= 1$
28 :    $\mathsf{h}^{\mathsf{Q}} := (\mathsf{tS}, \mathsf{iS}, \mathsf{tCurr}, \perp, \mathsf{c}_{\mathsf{R}}, \mathsf{iS}_{\text{-}1}, \mathsf{iS}_{\text{-}2})$
29 : $(\mathsf{KS}, \mathsf{K}^{\mathsf{Q}}_{\mathsf{aead}}) \leftarrow \mathsf{KDF}_2(\mathsf{KS})$

30 : $\mathsf{K}_{\mathsf{aead}} \leftarrow \mathsf{KDF}_3(\mathsf{K}^{\mathsf{C}}_{\mathsf{aead}}, \mathsf{K}^{\mathsf{Q}}_{\mathsf{aead}})$
31 : $\mathsf{h} \leftarrow (\mathsf{h}^{\mathsf{C}}, \mathsf{h}^{\mathsf{Q}})$
32 : $\mathsf{e} \xleftarrow{\$} \mathsf{AEAD.Enc}(\mathsf{K}_{\mathsf{aead}}, \mathsf{h}, \mathsf{M})$
33 : return $\mathsf{ct} := (\mathsf{h}, \mathsf{e})$
```

Figure 8: The send algorithm of A. The TR-Send-B algorithm is defined analogously, except for (1) even and odd exchanged and (2) in the post-quantum part the output order of $\mathsf{KDF}_1$ swapped with the output becoming $(\mathsf{K}_{\mathsf{root}}, \mathsf{KR}, \mathsf{KS})$ for consistency.

**Key schedule.** Splitting the post-quantum CKA messages has several implications. One of them is that the classical and the post-quantum CKA advance at different speeds. In particular, there a some subtle cases where the switch to the next epoch on the classical and the post-quantum protocol can happen in swapped order for the two parties A and B. As a result, the Triple Ratchet uses two separate root keys into which the corresponding CKA keys are mixed, and a symmetric ratchet is applied to each one. Only then, the two keys get combined to use the combined key to encrypt the application message, and authenticate the header, using AEAD. Concretely, TR-Send-A derives separate AEAD keys $\mathsf{K}^{\mathsf{C}}_{\mathsf{aead}}$ and $\mathsf{K}^{\mathsf{Q}}_{\mathsf{aead}}$ in lines 9 and 29, respectively, before combining them on line 30. TR-Rec-A proceeds analogously with the algorithm determining the two separate keys before attempting to decrypt under the combined key.

**Epoch handling.** Another implication of sending CKA messages in chunks is that there is no longer a unique protocol epoch. Whereas in the classical Double Ratchet each message is encrypted under the key derived from the current epoch's CKA key while simultaneously sending the CKA message for that epoch t, the *sending epoch* and the *CKA epoch* now typically differ. Only once the sender is sure the other party will have sufficiently many chunks, they can start using the corresponding CKA key. A bit more concretely, the protocol maintains separate epoch counters tCurr, the epoch for which CKA messages are currently being exchanged, and tS, the epoch under which they currently encrypt messages. We refer to Table 1 for an overview of the variables used by the protocol. Analogously, each party keeps track of a receiving epoch tR,

```
TR-Rec-A(ct)

parse $(\mathsf{h}, \mathsf{e}) \leftarrow \mathsf{ct}$
parse $(\mathsf{h}^{\mathsf{C}}, \mathsf{h}^{\mathsf{Q}}) \leftarrow \mathsf{h}$

parse $(\mathsf{t}, \mathsf{i}, \rho, \mathsf{i}_{\text{-}2}) \leftarrow \mathsf{h}^{\mathsf{C}}$
req $[\![\mathsf{t} \leqslant \mathsf{tCurr} + 1]\!]$
if $[\![\mathsf{t} = \mathsf{tCurr} + 1]\!]$ then
    $\mathsf{skip}(\mathsf{t} - 2, \mathsf{i}_{\text{-}2})$
    $(\mathsf{tCurr}, \mathsf{iR}) \leftarrow (\mathsf{t}, 0)$
    $(\mathsf{K}_{\mathsf{CKA}}, \overline{\mathsf{st}}_{\mathsf{A}}) \leftarrow \mathsf{CKA\text{-}Rec\text{-}A}(\overline{\mathsf{st}}_{\mathsf{A}}, \rho)$
    $(\mathsf{K}_{\mathsf{root}}, \mathsf{KR}) \leftarrow \mathsf{KDF}_1(\mathsf{K}_{\mathsf{root}}, \mathsf{K}_{\mathsf{CKA}})$
endif
$\mathsf{K}_{\mathsf{aead}} \leftarrow \mathsf{try\text{-}skipped}(\mathsf{t}, i)$
if $\mathsf{K}_{\mathsf{aead}} = \perp$ then
    $\mathsf{skip}(\mathsf{t}, \mathsf{i} - 1)$
    $\mathsf{iR} \mathrel{+}= 1$
    $(\mathsf{KR}, \mathsf{K}^{\mathsf{C}}_{\mathsf{aead}}) \leftarrow \mathsf{KDF}_2(\mathsf{KR})$

parse $(\mathsf{t}, \mathsf{i}, \mathsf{tCurr}', \rho_{\mathsf{enc}}, \mathsf{c}'_{\mathsf{Ack}}, \mathsf{i}_{\text{-}1}, \mathsf{i}_{\text{-}2}) \leftarrow \mathsf{h}^{\mathsf{Q}}$
req $[\![\mathsf{tCurr}' \leqslant \mathsf{tCurr} + 1]\!] \wedge [\![\mathsf{t} \leqslant \mathsf{tR} + 2]\!]$
        $\wedge\ [\![\mathsf{tCurr}' - 1 \leqslant \mathsf{t} \leqslant \mathsf{tCurr}']\!]$
if $[\![\mathsf{t} = \mathsf{tR} + 2]\!]$ then
    $\mathsf{skip}(\mathsf{t} - 2, \mathsf{i}_{\text{-}2})$
    $(\mathsf{KR}, \mathsf{KR}_{+1}, \mathsf{KR}_{+2}) \leftarrow (\mathsf{KR}_{+1}, \mathsf{KR}_{+2}, \perp)$
if $[\![\mathsf{t} > \mathsf{tR}]\!]$ then
    $\mathsf{skip}(\mathsf{t} - 1, \mathsf{i}_{\text{-}1})$
    $(\mathsf{KR}, \mathsf{KR}_{+1}, \mathsf{KR}_{+2}) \leftarrow (\mathsf{KR}_{+1}, \mathsf{KR}_{+2}, \perp)$
    $(\mathsf{tR}, \mathsf{iR}) \leftarrow (\mathsf{t}, 0)$
if $[\![\mathsf{tCurr}' = \mathsf{tCurr} + 1]\!]$ then
    if $[\![\mathsf{tS} < \mathsf{tCurr}]\!]$ then
        $\mathsf{tS} \leftarrow \mathsf{tCurr}$
        $(\mathsf{iS}, \mathsf{iS}_{\text{-}1}, \mathsf{iS}_{\text{-}2}) \leftarrow (0, \mathsf{iS}, \mathsf{iS}_{\text{-}1})$
        $(\mathsf{KS}, \mathsf{KS}_{+1}) \leftarrow (\mathsf{KS}_{+1}, \perp)$
    $(\mathsf{tCurr}, \mathsf{c}_{\mathsf{R}}) \leftarrow (\mathsf{tCurr}', 0)$
if $[\![\mathsf{tCurr}' = \mathsf{tCurr}]\!] \wedge [\![\mathsf{tCurr}$ is even$]\!]$ then
    $\mathsf{c}_{\mathsf{R}} \mathrel{+}= 1$
    $\mathsf{L} \xleftarrow{+} (i, \rho_{\mathsf{enc}})$
    if $[\![\mathsf{c}_{\mathsf{R}} \geqslant n_{\mathsf{chunk}}]\!] \wedge [\![\mathsf{tCurr} > \mathsf{tS}]\!]$ then
        $\mathsf{tS} \leftarrow \mathsf{tCurr}$
        $(\mathsf{iS}, \mathsf{iS}_{\text{-}1}, \mathsf{iS}_{\text{-}2}) \leftarrow (0, \mathsf{iS}, \mathsf{iS}_{\text{-}1})$
        $\rho \leftarrow \mathsf{Decode}(\mathsf{L})$
        $(\mathsf{K}_{\mathsf{CKA}}, \overline{\mathsf{st}}_{\mathsf{A}}) \leftarrow \mathsf{CKA\text{-}Rec\text{-}A}(\overline{\mathsf{st}}_{\mathsf{A}}, \rho)$
        if $[\![\mathsf{tR} = \mathsf{tS}]\!]$ then
            $(\mathsf{K}_{\mathsf{root}}, \mathsf{KS}, \mathsf{KR}) \leftarrow \mathsf{KDF}_1(\mathsf{K}_{\mathsf{root}}, \mathsf{K}_{\mathsf{CKA}})$
        else
            $(\mathsf{K}_{\mathsf{root}}, \mathsf{KS}, \mathsf{KR}_{+1}) \leftarrow \mathsf{KDF}_1(\mathsf{K}_{\mathsf{root}}, \mathsf{K}_{\mathsf{CKA}})$
        $\mathsf{L} \leftarrow \varnothing$
elseif $[\![\mathsf{tCurr}' = \mathsf{tCurr}]\!]$ then
    $\mathsf{c}_{\mathsf{Ack}} \leftarrow \mathsf{c}'_{\mathsf{Ack}}$
$\mathsf{K}_{\mathsf{aead}} \leftarrow \mathsf{try\text{-}skipped}(\mathsf{t}, \mathsf{i})$
if $\mathsf{K}_{\mathsf{aead}} = \perp$ then
    $\mathsf{skip}(\mathsf{t}, \mathsf{i} - 1)$
    $\mathsf{iR} \mathrel{+}= 1$
    $(\mathsf{KR}, \mathsf{K}_{\mathsf{aead}}) \leftarrow \mathsf{KDF}_2(\mathsf{KR})$

$\mathsf{K}_{\mathsf{aead}} \leftarrow \mathsf{KDF}_3(\mathsf{K}^{\mathsf{C}}_{\mathsf{aead}}, \mathsf{K}^{\mathsf{Q}}_{\mathsf{aead}})$
$\mathsf{M} \leftarrow \mathsf{AEAD.Dec}(\mathsf{K}_{\mathsf{aead}}, \mathsf{h}, \mathsf{e})$
if $\mathsf{M} = \perp$ then error
return $(\mathsf{M}, (\mathsf{tCurr}^{\mathsf{C}}, \mathsf{iS}^{\mathsf{C}}, \mathsf{tS}^{\mathsf{Q}}, \mathsf{iS}^{\mathsf{Q}}))$
```

Figure 9: The receive algorithm of A. TR-Rec-B is defined analogously with the roles of even and odd swapped and in the post-quantum part the output order of $\mathsf{KDF}_1$ swapped with the output becoming $(\mathsf{K}_{\mathsf{root}}, \mathsf{KR}, \mathsf{KS})$ for consistency. skip and try-skipped are defined in Fig. 7.

which is the epoch they last received a message encrypted under. Observe that since epochs advance more slowly, each party may act both as a sender and a receiver during each given sending epoch. In more detail,

- Whenever entering a "sending epoch" a party generates a fresh $\mathsf{CKA}$ message and its corresponding key. This key is then immediately mixed into the post-quantum root key, deriving three keys: the updated root key $\mathsf{K}_{\mathsf{root}}$, a sending key, and a receiving key. See lines 12 and 13 of $\mathsf{TR\text{-}Send\text{-}A}$.

- However, unless $n_{\mathsf{chunk}} = 1$, those keys cannot be immediately used. Instead, $\mathsf{A}$ at this point simply schedules the keys for further use. The new receiving key will be used once the other party advances to the respective epoch. The new sending key will be used once $\mathsf{A}$ knows that $\mathsf{B}$ has sufficient information for immediate decryption. There are two cases for this to happen. First, once $\mathsf{A}$ knows that $\mathsf{B}$ received at least $n_{\mathsf{chunk}}$ many chunks and thus reconstructed the key. In our protocol, this happens implicitly by $\mathsf{B}$ sending chunks for the next key (lines 25-30 in $\mathsf{TR\text{-}Rec\text{-}A}$). Alternatively, once $\mathsf{B}$ acknowledged exactly $n_{\mathsf{chunk}} - 1$ many chunks, $\mathsf{A}$ knows that with any further message enough chunks will be received and therefore can start using the key as well (lines 20-23 of $\mathsf{TR\text{-}Send\text{-}A}$).

- Similarly, during a "receiving epoch" the user reconstructs $\rho$ once they received sufficiently many chunks. The party can then immediately start using the new sending key (lines 34-38 in $\mathsf{TR\text{-}Rec\text{-}A}$) while the receiving key may have to be scheduled for later use unless the other party already uses it (lines 39-42 in $\mathsf{TR\text{-}Rec\text{-}A}$). In either case, for the next sending operation $\mathsf{A}$ then can initiate the next sending epoch.

## 4.2 Correctness and Security

This section establishes $\mathsf{SM}$ security of the $\mathsf{TR}$ protocol from Section 4.1. Recall that the FS and PCS properties of $\mathsf{SM}$ security are defined with respect to an epoch function $\tau$, abstracting that FS and PCS progress at different speed for the sub-protocol secure against classical adversaries and the sub-protocol secure against quantum adversaries. We first discuss the respective epoch functions for both cases.

*Remark* 4.1 (Epoch functions). For the protocol $\mathsf{TR}$, we define $\tau^{\mathsf{C}} := \mathsf{tCurr}^{\mathsf{C}}$ and $\tau^{\mathsf{Q}} := \mathsf{tS}^{\mathsf{Q}}$. Observe that for the classical $\mathsf{CKA}$ the epoch function directly corresponds to epochs as introduced in [ACD19] and increment on every change in direction. For the post-quantum protocol, once $\mathsf{A}$ enters an even epoch, it takes the following for $\mathsf{A}$ to advance to the next odd epoch:

1. $\mathsf{A}$ needs to send at least $n_{\mathsf{chunk}} - 1$ many messages that need to be received by $\mathsf{B}$ (any subset of $n_{\mathsf{chunk}} - 1$ many does, in case more are sent)

2. $\mathsf{B}$ sends a message that is received by $\mathsf{A}$.

3. If $\mathsf{B}$ received at least $n_{\mathsf{chunk}}$ many messages before (2), then $\mathsf{A}$ immediately increments the epoch; otherwise $\mathsf{A}$ increments the epoch upon the next send action.

$\mathsf{B}$ on the other hand increments from an even to an odd epoch after receiving $n_{\mathsf{chunk}}$ many messages from $\mathsf{A}$. In particular, this implies that once $\mathsf{A}$ moves to an odd epoch and any further message is received by $\mathsf{B}$, $\mathsf{B}$ advances as well. The parties then advance from the odd to the next even epoch upon the same steps happening with the roles reversed.

**Theorem 4.2 (Security of $\mathsf{TR}$).** *For the $\mathsf{TR}$ protocol, let $\tau^{\mathsf{C}}$ and $\tau^{\mathsf{Q}}$ denote the respective epoch functions as discussed in Remark 4.1. Assume that*

- $\mathsf{CKA}^{\mathsf{C}}$ *is* $(\Delta^{\mathsf{C}}_{\mathsf{FS}}, \Delta^{\mathsf{C}}_{\mathsf{PCS}})$*-secure* $\mathsf{CKA}$ *scheme or* $\mathsf{CKA}^{\mathsf{Q}}$ *is* $(\Delta^{\mathsf{Q}}_{\mathsf{FS}}, \Delta^{\mathsf{Q}}_{\mathsf{PCS}})$*-secure* $\mathsf{CKA}$ *scheme;*

- $\mathsf{CKA}^{\mathsf{C}}$ *and* $\mathsf{CKA}^{\mathsf{Q}}$ *are both correct;*

- $\mathsf{KDF}_1$ *is a secure PRF-PRNG,* $\mathsf{KDF}_2$ *is a secure PRG, and* $\mathsf{KDF}_3$ *is a secure dual-PRF;*

- $\mathsf{AEAD}$ *is a secure authenticated encryption scheme with associated data.*

*Then, the* $\mathsf{TR}$ *construction above is* $(\Delta^{\mathsf{C}}_{\mathsf{FS}}, \Delta^{\mathsf{C}}_{\mathsf{PCS}}, \tau^{\mathsf{C}})$*-secure if* $\mathsf{CKA}^{\mathsf{C}}$ *is secure, and* $(\Delta^{\mathsf{Q}}_{\mathsf{FS}}, \Delta^{\mathsf{Q}}_{\mathsf{PCS}} + 1, \tau^{\mathsf{Q}})$ *secure if* $\mathsf{CKA}^{\mathsf{Q}}$ *is secure respectively. More concretely, let* $q$ *be an upper bound on the oracle invocations . Then we have*

$$\mathsf{Adv}^{\mathsf{SM}}_{\mathcal{A},\Delta^{\mathsf{Q}}_{\mathsf{PCS}}+1,\Delta^{\mathsf{Q}}_{\mathsf{FS}},\tau^{\mathsf{Q}}}(1^\lambda) \leqslant \mathsf{Adv}^{\mathsf{CKA\text{-}corr}^{\mathsf{C}}}_{\mathcal{A}_1}(1^\lambda) + \mathsf{Adv}^{\mathsf{CKA\text{-}corr}^{\mathsf{Q}}}_{\mathcal{A}_2}(1^\lambda) + 2q^2 \cdot \Big(\mathsf{Adv}^{\mathsf{CKA}^{\mathsf{Q}}}_{\mathcal{B},\Delta^{\mathsf{Q}}_{\mathsf{PCS}},\Delta^{\mathsf{Q}}_{\mathsf{FS}}}(1^\lambda) \\ + q \cdot \mathsf{Adv}^{\mathsf{KDF}_1}_{\mathcal{C}}(1^\lambda) + q \cdot \mathsf{Adv}^{\mathsf{KDF}_2}_{\mathcal{D}}(1^\lambda) + \mathsf{Adv}^{\mathsf{KDF}_3}_{\mathcal{E}}(1^\lambda) + \mathsf{Adv}^{\mathsf{AEAD}}_{\mathcal{F}}(1^\lambda)\Big)$$

*in case the* post-quantum *sub-protocol* $\mathsf{CKA}^{\mathsf{Q}}$ *is secure. Moreover, in case the* classical *sub-protocol* $\mathsf{CKA}^{\mathsf{C}}$ *is secure,* $\mathsf{Adv}^{\mathsf{SM}}_{\mathcal{A},\Delta^{\mathsf{C}}_{\mathsf{PCS}},\Delta^{\mathsf{C}}_{\mathsf{FS}},\tau^{\mathsf{C}}}(1^\lambda)$ *can be bounded by the same term, except with the* $\mathsf{CKA}$ *advantage replaced by* $\mathsf{Adv}^{\mathsf{CKA}^{\mathsf{C}}}_{\mathcal{B},\Delta^{\mathsf{C}}_{\mathsf{PCS}},\Delta^{\mathsf{C}}_{\mathsf{FS}}}$*, respectively.*

*In the above,* $\mathsf{Adv}^{\mathsf{CKA\text{-}corr}}_{\mathcal{A}'}$ *denotes the advantage of* $\mathcal{A}'$ *breaking the correctness*[11] *of the* $\mathsf{CKA}$ *and the remaining advantage terms formalizing the aforementioned security assumptions on the underlying primitives.*

*Remark* 4.3 (Instantiations). For the Triple Ratchet protocol, we propose to instantiate the two $\mathsf{CKA}$s using a generic $\mathsf{CKA}$ construction from $\mathsf{RKEM}$ presented in Section 5, with the classical one using a forward-secure Diffie-Hellman $\mathsf{RKEM}$ — modularizing the protocol proposed by Bienstock et al. [BFG⁺22a] — and the post-quantum one using our $\mathsf{Katana\text{-}RKEM}$. Therefore, both $\mathsf{CKA}$s will have $\Delta^{\mathsf{CKA}}_{\mathsf{FS}} = 0$ and $\Delta^{\mathsf{CKA}}_{\mathsf{PCS}} = 2$. Therefore, for the $\mathsf{TR}$ protocol, we obtain classical PCS within $\Delta^{\mathsf{TR}}_{\mathsf{PCS}} = 2$ epochs and post-quantum PCS within $\Delta^{\mathsf{TR}}_{\mathsf{PCS}} = 3$ (albeit slower) epochs. The additional epoch it takes for the post-quantum protocol is due to the protocol already having sampled the key material for the next epoch when still distributing it. In other words, a corruption may already compromise the secret key material of the next epoch.

As observed in [ACD19], the $\mathsf{SM}$ security game can be split into separate games for correctness, authenticity, and confidentiality, as stated by the following lemma.

**Lemma 4.4.** *In the following, let*

- $\mathsf{Game}^{\mathsf{SM\text{-}corr}}_{\mathcal{A}}$ *be a variant of* $\mathsf{Game}^{\mathsf{SM}}_{\mathcal{A},\Delta_{\mathsf{PCS}},\Delta_{\mathsf{FS}},\tau}$ *whose only winning condition is breaking the correctness in the* $\mathsf{Receive\text{-}A}$ *and* $\mathsf{Receive\text{-}B}$ *oracles (whose challenge oracles has been removed and where the adversary loses upon a successful injection).*
- $\mathsf{Game}^{\mathsf{SM\text{-}auth}}_{\mathcal{A},\Delta_{\mathsf{PCS}},\Delta_{\mathsf{FS}},\tau}$ *be a game whose only winning condition is breaking authenticity, i.e., triggering* $[\![\mathsf{M}' = \bot]\!] \vee [\![\mathsf{record} \in L_{\mathsf{comp}}]\!]$*, with the adversary losing when breaking correctness and the challenge oracle removed.*
- $\mathsf{Game}^{\mathsf{SM\text{-}conf}}_{\mathcal{A},\Delta_{\mathsf{PCS}},\Delta_{\mathsf{FS}},\tau}$ *be a variant where the adversary loses if they break correctness or cause a non-trivial injection, i.e., trigger* $[\![\mathsf{M}' = \bot]\!] \vee [\![\mathsf{record} \in L_{\mathsf{comp}}]\!]$.

*It holds that*

$$\mathsf{Adv}^{\mathsf{SM}}_{\mathcal{A},\Delta_{\mathsf{PCS}},\Delta_{\mathsf{FS}},\tau}(1^\lambda) \leqslant \mathsf{Adv}^{\mathsf{SM\text{-}corr}}_{\mathcal{A}}(1^\lambda) + \mathsf{Adv}^{\mathsf{SM\text{-}auth}}_{\mathcal{A},\Delta_{\mathsf{PCS}},\Delta_{\mathsf{FS}},\tau}(1^\lambda) + \mathsf{Adv}^{\mathsf{SM\text{-}conf}}_{\mathcal{A},\Delta_{\mathsf{PCS}},\Delta_{\mathsf{FS}},\tau}(1^\lambda).$$

### 4.2.1 Correctness

For correctness of the Triple Ratchet protocol, we require both CKAs to be correct. For simplicity, we assume the $\mathsf{AEAD}$ scheme to decrypt correctly with probability 1.

**Lemma 4.5.** *Assuming the* $\mathsf{AEAD}$ *to have perfect correctness, then the Triple Ratchet is correct as long as both the classical* $\mathsf{CKA}$ *and the post-quantum* $\mathsf{CKA}$ *are correct. More concretely,*

$$\mathsf{Adv}^{\mathsf{SM\text{-}corr}}_{\mathcal{A}}(1^\lambda) \leqslant \mathsf{Adv}^{\mathsf{CKA\text{-}corr}^{\mathsf{C}}}_{\mathcal{A}_1}(1^\lambda) + \mathsf{Adv}^{\mathsf{CKA\text{-}corr}^{\mathsf{Q}}}_{\mathcal{A}_2}(1^\lambda),$$

*where* $\mathsf{Adv}^{\mathsf{CKA\text{-}corr}}_{\mathcal{A}}(1^\lambda)$ *the notes the advantage of* $\mathcal{A}$ *to just trigger the correctness property in the* $\mathsf{CKA}$ *game.*

[11] Technically, of winning a variant of the $\mathsf{CKA}$ game where the challenge oracle has been removed such that breaking correctness is the only winning condition.

*Proof.* This follows mostly by inspection. For the classical $\mathsf{CKA}$, observe that it is easy to argue to both parties $\mathsf{A}$ and $\mathsf{B}$ absorb the same keys $\mathsf{K}_{\mathsf{CKA}}$ into their root key $\mathsf{K}_{\mathsf{root}}$. Therefore, for the same epoch $\mathsf{t}$ and period $\mathsf{iS}$, they produce the same $\mathsf{AEAD}$ key $\mathsf{K}^{\mathsf{C}}_{\mathsf{aead}}$. Analogously, for the post-quantum protocol, $\mathsf{Decode}$ is guaranteed to produce the correct $\mathsf{CKA}$ message $\rho$ and, therefore, correctness of the $\mathsf{CKA}$ scheme implies they produce the same sending and receiving keys $\mathsf{KS}$ and $\mathsf{KR}$ as well. As a result, for each message index $\mathsf{idx}$, both parties produce the same $\mathsf{AEAD}$ key $\mathsf{K}_{\mathsf{aead}} := \mathsf{KDF}_3(\mathsf{K}^{\mathsf{C}}_{\mathsf{aead}}, \mathsf{K}^{\mathsf{Q}}_{\mathsf{aead}})$ and, therefore, by correctness of the $\mathsf{AEAD}$, the recipient outputs the correct message. □

### 4.2.2 Confidentiality

We now proceed to bound the advantage on the confidentiality game. Privacy holds as long as either of the $\mathsf{CKA}$ protocols is secure — with the speed of FS and PCS depending on whether the classical or the post-quantum $\mathsf{CKA}$ is assumed to be secure. While the proofs of both properties are essentially analogous, in the following we mainly focus on the post-quantum security. First, we establish some technical lemmas that allow us to simplify the proof.

**Lemma 4.6.** *Let* $\mathsf{Game}^{\mathsf{SM\text{-}conf\text{-}ss}}$ *be a variant of* $\mathsf{Game}^{\mathsf{SM\text{-}conf}}$ *with the following two modifications:*

- *The attacker $\mathcal{A}$ only gets to make a single challenge.*
- *The attacker has to selectively input the value $\mathsf{t}_{\mathsf{L}}$ that the game will have at the time of the challenge* at the beginning *of the interaction. We call this input $\mathsf{t}^*_{\mathsf{L}}$.*

*For any PCS and FS parameters $\Delta_{\mathsf{PCS}}$ and $\Delta_{\mathsf{FS}}$, respectively, and any epoch function $\tau$, we then get*

$$\mathsf{Adv}^{\mathsf{SM\text{-}conf}}_{\mathcal{A},\Delta_{\mathsf{PCS}},\Delta_{\mathsf{FS}},\tau}(1^\lambda) \leqslant q^2 \cdot \mathsf{Adv}^{\mathsf{SM\text{-}conf\text{-}ss}}_{\mathcal{A}',\Delta_{\mathsf{PCS}},\Delta_{\mathsf{FS}},\tau}(1^\lambda).$$

*Proof.* The reduction to a single challenge follows using a standard hybrid argument, losing a factor in the number of challenge queries, which is at most $q$. Simply put, one can consider hybrids where the first $n$ challenges encrypt message $\mathsf{M}_1$ while all challenges thereafter encrypt message $\mathsf{M}_0$; the first hybrid clearly corresponds to the original game with $b = 0$ while the last hybrid corresponds to the original game with $b = 1$, while distinguishing two subsequent hybrids reduces to the one-challenge game with emulating the other challenges using the regular sending oracle. Selective security then follows by a reduction that simply guesses the input, losing another factor $q$. □

**Lemma 4.7.** *Assuming either the classical protocol $\mathsf{CKA}^{\mathsf{C}}$ or the post-quantum protocol $\mathsf{CKA}^{\mathsf{Q}}$ to be secure, then confidentiality holds for $\mathsf{TR}$ protocol. More concretely, let $q$ be an upper bound on the oracle invocations and, for $\mathsf{X} \in \{\mathsf{C}, \mathsf{Q}\}$, let $\Delta^{\mathsf{X}}_{\mathsf{PCS}}$ and $\Delta^{\mathsf{X}}_{\mathsf{FS}}$ denote the PCS and FS parameters for the classical and post-quantum CKAs, respectively, and let $\tau^{\mathsf{C}}$ and $\tau^{\mathsf{Q}}$ denote the respective epoch functions (as discussed in Remark 4.1). Then we have*

$$\begin{aligned}\mathsf{Adv}^{\mathsf{SM\text{-}conf\text{-}ss}}_{\mathcal{A}',\Delta^{\mathsf{C}}_{\mathsf{PCS}},\Delta^{\mathsf{C}}_{\mathsf{FS}},\tau^{\mathsf{C}}}(1^\lambda) \leqslant \mathsf{Adv}^{\mathsf{CKA}^{\mathsf{C}}}_{\mathcal{B},\Delta^{\mathsf{C}}_{\mathsf{PCS}},\Delta^{\mathsf{C}}_{\mathsf{FS}}}(1^\lambda) + q \cdot \mathsf{Adv}^{\mathsf{PRF\text{-}PRNG}}_{\mathcal{C}}(1^\lambda)& \\ + q \cdot \mathsf{Adv}^{\mathsf{PRG}}_{\mathcal{D}}(1^\lambda) + \mathsf{Adv}^{\mathsf{dPRF}}_{\mathcal{E}}(1^\lambda) + \mathsf{Adv}^{\mathsf{AEAD}}_{\mathcal{F}}(1^\lambda)&\end{aligned}$$

*in case the* classical *part $\mathsf{CKA}^{\mathsf{C}}$ is secure, and*

$$\begin{aligned}\mathsf{Adv}^{\mathsf{SM\text{-}conf\text{-}ss}}_{\mathcal{A}',\Delta^{\mathsf{Q}}_{\mathsf{PCS}}+1,\Delta^{\mathsf{Q}}_{\mathsf{FS}},\tau^{\mathsf{Q}}}(1^\lambda) \leqslant \mathsf{Adv}^{\mathsf{CKA}^{\mathsf{Q}}}_{\mathcal{B},\Delta^{\mathsf{Q}}_{\mathsf{PCS}},\Delta^{\mathsf{Q}}_{\mathsf{FS}}}(1^\lambda) + q \cdot \mathsf{Adv}^{\mathsf{PRF\text{-}PRNG}}_{\mathcal{C}}(1^\lambda)& \\ + q \cdot \mathsf{Adv}^{\mathsf{PRG}}_{\mathcal{D}}(1^\lambda) + \mathsf{Adv}^{\mathsf{dPRF}}_{\mathcal{E}}(1^\lambda) + \mathsf{Adv}^{\mathsf{AEAD}}_{\mathcal{F}}(1^\lambda)&\end{aligned}$$

*in case the* post-quantum *part $\mathsf{CKA}^{\mathsf{Q}}$ is secure.*

*Proof.* We show this using a sequence of hybrids. The overall approach closely follows the proof in [ACD19]. All the changes, unless specifically mentioned otherwise, are only performed to the $\mathsf{CKA}$ of $\mathsf{TR}$ that is assumed to be secure. The proofs for the two CKAs are mostly analogous, with small deviations mentioned when they arise.

$\mathsf{Hybrid}_1$: In the first hybrid, we modify $\mathsf{Game}^{\mathsf{SM\text{-}conf\text{-}ss}}_{\mathcal{A}',\Delta_{\mathsf{PCS}},\Delta_{\mathsf{FS}},\tau}$ as follows:

- We replace the key $\mathsf{K}_{\mathsf{CKA}}$ of epoch $\mathsf{t}^*_{\mathsf{L}} + \Delta_{\mathsf{PCS}}$ with a fresh independent one. That is, we replace it in both TR-Send-P, when output by CKA-Send-P, and in TR-Rec-P, when output by CKA-Rec-P, with the same freshly sampled key.
- If $\mathsf{t}^*_{\mathsf{L}} = -\infty$, i.e., if no corruption occurs before the challenge, then $\mathsf{Hybrid}_1$ behaves as the original game.

The latter case is trivially indistinguishable; we focus on the former case (with some corruption) in the following. Note that the sender of the key of $\mathsf{t}^*_{\mathsf{L}} + \Delta_{\mathsf{PCS}}$, i.e., the party P executing CKA-Send-P, did so between epoch $\mathsf{t}^*_{\mathsf{L}} + \Delta_{\mathsf{PCS}} - 1$ and $\mathsf{t}^*_{\mathsf{L}} + \Delta_{\mathsf{PCS}}$. By definition of $\mathsf{t}^*_{\mathsf{L}}$, we can therefore conclude that this must have been done with good randomness, as safe-chall(P) at this point was still false and, therefore, using bad randomness would have updated $\mathsf{t}_{\mathsf{L}}$. Using $\Delta_{\mathsf{PCS}} = \Delta^{\mathsf{CKA}}_{\mathsf{PCS}} + 1$, we can moreover observe that $\hat{\mathsf{t}}^* := \mathsf{t}^*_{\mathsf{L}} + \Delta_{\mathsf{PCS}} - 1$ is a valid challenge epoch for the CKA game. In other words, safe-corr(P) and safe-chall(P) ensure that the CKA state can only be leaked for strictly before $\hat{\mathsf{t}}^* - \Delta^{\mathsf{CKA}}_{\mathsf{PCS}}$ and after $\hat{\mathsf{t}}^* + \Delta_{\mathsf{FS}}$. Therefore, there exists a simple reduction to $\mathsf{Game}^{\mathsf{CKA}}_{\mathcal{A}',\hat{\mathsf{t}}^*}$ as follows:

- Whenever TR-Send-P invokes CKA-Send-P, the reduction uses the Send-P oracle of the CKA game instead to obtain $\rho$ for epoch other than $\mathsf{t}^*_{\mathsf{L}}$. Similarly, the reduction uses the Chall-P oracle to obtain $\rho$ for the challenge epoch $\mathsf{t}^*_{\mathsf{L}}$. The reduction then keeps track of the corresponding key $\mathsf{K}_{\mathsf{CKA}}$ and uses that one to mix into the root key $\mathsf{K}_{\mathsf{root}}$.
- Whenever TR-Rec-P invokes CKA-Rec-P on a decoded message $\rho$ that has been sent by the other party, as chunks, then the reduction invokes the Receive-P oracle of the CKA game to advance the party's CKA state. In a bit more detail, once at least $n_{\mathsf{chunk}}$ many SM messages have been honestly delivered (without any non-trivial injection), the reduction invokes the delivery oracle. Using correctness of the erasure code, we know that the game delivers the same $\rho$ that Decode would recover. It then mixes in the key $\mathsf{K}_{\mathsf{CKA}}$ that was output as part of sending $\rho$ (which by correctness is the same key the protocol obtains).
- Whenever the attacker $\mathcal{A}'$ corrupts a party P in the SM game, the reduction corrupts the corresponding party in the CKA game to obtain their CKA state. As argued above, whenever a corruption is valid in the SM game, it is also valid in the CKA game.
- For injections, recall that we disallowed so-called non-trivial injections, i.e., only allow injections for messages sufficiently in the past such that both parties have healed in the meantime. Note, however, that delivering old out-of-order messages causes the Triple Ratchet protocol to just look up the skipped key in StoredKeys — not affecting the CKA state. Therefore, the reduction running these parts internally can properly emulate any effect of such injections.

As a consequence, for the post-quantum CKA we obtain

$$\left|\Pr\left[\mathsf{Game}^{\mathsf{SM\text{-}conf\text{-}ss}}_{\mathcal{A}',\Delta_{\mathsf{PCS}}+1,\Delta_{\mathsf{FS}},\tau^{\mathsf{Q}}}(1^\lambda) = 1\right] - \Pr\left[\mathsf{Hybrid}_1(1^\lambda) = 1\right]\right| \leqslant \mathsf{Adv}^{\mathsf{CKA}^{\mathsf{Q}}}_{\mathcal{B},\Delta_{\mathsf{PCS}},\Delta_{\mathsf{FS}}}(1^\lambda),$$

and the analogous result for the classical CKA with the tighter $\Delta_{\mathsf{PCS}}$ bound.

$\mathsf{Hybrid}_2$: In the second hybrid, we modify $\mathsf{Hybrid}_1$ as follows:

- For all epochs starting from $\mathsf{t}^*_{\mathsf{L}} + \Delta_{\mathsf{PCS}}$ to the challenge epoch, we replace the output of $\mathsf{KDF}_1$, i.e., $\mathsf{K}_{\mathsf{root}}$, KS and $\mathsf{KR}_{+1}$, with freshly sampled independent keys. (In the case of the classically secure CKA, $\mathsf{KDF}_1$ just outputs two keys, which we replace by fresh ones.)

Observe that the first of those $\mathsf{KDF}_1(\mathsf{K}_{\mathsf{root}}, \mathsf{K}_{\mathsf{CKA}})$ invocations in $\mathsf{Hybrid}_1$ uses a fresh and independent $\mathsf{K}_{\mathsf{CKA}}$. Therefore, by PRF-PRNG security of $\mathsf{KDF}_1$, the outputs will be indistinguishable from freshly sampled outputs. Moreover, the game disallows any corruption of the involved keys. Therefore, using a sequence of hybrids we observe that for all the subsequent epochs, until the challenge epoch, the $\mathsf{K}_{\mathsf{root}}$

input now is a secure key and, thus, by PRF-PRNG security we can replace the subsequent outputs. (Note that for confidentiality, we only need the "PRNG" property of PRF-PRNG security. The "PRF" aspect of it will be vital for authenticity.) As a result, we can deduce

$$\left|\Pr\left[\mathsf{Hybrid}_2(1^\lambda) = 1\right] - \Pr\left[\mathsf{Hybrid}_1(1^\lambda) = 1\right]\right| \leqslant q \cdot \mathsf{Adv}_{\mathcal{C}}^{\mathsf{PRF\text{-}PRNG}}(1^\lambda).$$

$\mathsf{Hybrid}_3$**:** In the third hybrid, we modify $\mathsf{Hybrid}_2$ as follows:

- For all epochs starting from $\mathsf{t}_{\mathsf{L}}^* + \Delta_{\mathsf{PCS}}$ to the challenge epoch, we replace the output of $\mathsf{KDF}_2$, i.e. $\mathsf{KS}$, of the sending party with freshly sampled independent keys. For the receiving party, $\mathsf{KR}$ is replaced with the same key, i.e., the key used by the sender for the same epoch and period.
- In the challenge epoch, only invocations up to the actual challenge are replaced.

Since the initial keys $\mathsf{KS}$ (or $\mathsf{KR}$, respectively) have been fresh in $\mathsf{Hybrid}_2$, PRG security of $\mathsf{KDF}_2$ ensures that those outputs are indistinguishable. In particular, recall that for epochs between $\mathsf{t}_{\mathsf{L}}^* + \Delta_{\mathsf{PCS}}$ and (before) the challenge epoch the game does not allow corruptions. While corruptions may be allowed for the challenge epoch in case of $\Delta_{\mathsf{FS}} = 0$, they are in particular only allowed *after* the challenge. However, $\mathsf{KS}$ can be safely leaked after the challenge with the challenge $\mathsf{K}_{\mathsf{aead}}^{\mathsf{X}}$ still appearing independent and uniform at random. Therefore, we obtain

$$\left|\Pr\left[\mathsf{Hybrid}_3(1^\lambda) = 1\right] - \Pr\left[\mathsf{Hybrid}_2(1^\lambda) = 1\right]\right| \leqslant q \cdot \mathsf{Adv}_{\mathcal{D}}^{\mathsf{PRG}}(1^\lambda).$$

$\mathsf{Hybrid}_4$**:** Finally, we modify $\mathsf{Hybrid}_3$ as follows:

- For the challenge, we replace the output of $\mathsf{K}_{\mathsf{aead}} := \mathsf{KDF}_3(\mathsf{K}_{\mathsf{aead}}^{\mathsf{C}}, \mathsf{K}_{\mathsf{aead}}^{\mathsf{Q}})$ with a fresh independent key.

Note that in $\mathsf{Hybrid}_3$ either $\mathsf{K}_{\mathsf{aead}}^{\mathsf{C}}$ or $\mathsf{K}_{\mathsf{aead}}^{\mathsf{Q}}$ has been substituted with a fresh independent key. Therefore, dual-PRF security ensures that the output is indistinguishable from a uniform random key in either case.

$$\left|\Pr\left[\mathsf{Hybrid}_4(1^\lambda) = 1\right] - \Pr\left[\mathsf{Hybrid}_3(1^\lambda) = 1\right]\right| \leqslant \mathsf{Adv}_{\mathcal{E}}^{\mathsf{dPRF}}(1^\lambda).$$

Finally, we consider the probability of $\mathcal{A}'$ winning $\mathsf{Hybrid}_4$, i.e., of correctly guessing which of the messages was encrypted as part of the challenge. Since $\mathsf{Hybrid}_4$ uses a fresh uniform random key to encrypt the challenge using $\mathsf{AEAD}$, this probability trivially reduced to $\mathsf{AEAD}$ security:

$$\mathsf{Adv}_{\mathcal{A}'}^{\mathsf{Hybrid}_4}(1^\lambda) \leqslant \mathsf{Adv}_{\mathcal{F}}^{\mathsf{AEAD}}(1^\lambda).$$

The overall confidentiality statement then follows directly by adding the respective error terms, for both the classical and the post-quantum parts. □

### 4.2.3 Authenticity

Finally, we bound the advantage on the authenticity game. Analogous to confidentiality, authenticity holds as long as either of the $\mathsf{CKA}$ protocols is secure — with the speed of FS and PCS depending on whether the classical or the post-quantum $\mathsf{CKA}$ is assumed to be secure.

**Lemma 4.8.** *Let* $\mathsf{Game}^{\mathsf{SM\text{-}auth\text{-}ss}}$ *be a variant of* $\mathsf{Game}^{\mathsf{SM\text{-}auth}}$ *with the following two modifications:*

- *The attacker has to selectively input the epoch* $\mathsf{t}^*$ *they try to attack, as well as* $\mathsf{t}_{\mathsf{L}}^*$*, the value of the last epoch corrupted* $\mathsf{t}_{\mathsf{L}}$ *beforehand.*
- *Any non-trivial injection is forbidden unless in epoch* $\mathsf{t}^*$*.*
- *Corruptions are disallowed for a parties in epoch* $\mathsf{t}^*$*.*

*For any PCS and FS parameters $\Delta_{\mathsf{PCS}}$ and $\Delta_{\mathsf{FS}}$, respectively, and any epoch function $\tau$, we for every PPT adversary $\mathcal{A}$, there exists a PPT adversary $\mathcal{A}'$ such that*

$$\mathsf{Adv}^{\mathsf{SM\text{-}auth}}_{\mathcal{A},\Delta_{\mathsf{PCS}},\Delta_{\mathsf{FS}},\tau}(1^\lambda) \leqslant q^2 \cdot \mathsf{Adv}^{\mathsf{SM\text{-}auth\text{-}ss}}_{\mathcal{A}',\Delta_{\mathsf{PCS}},\Delta_{\mathsf{FS}},\tau}(1^\lambda).$$

*Proof.* $\mathcal{A}'$ works by internally running $\mathcal{A}$ and simulating the original game based on the restricted one. To this end, the reduction tries to guess the epoch $\mathsf{t}^*$ of *the first successful* (non-trivial) injection and the last corruption beforehand. (Note that any corruption in $\mathsf{t}^*$ would need to happen after the successful injection for the injection to be allowed; therefore, we can simply disregard such injections.) To this end, it chooses $\mathsf{t}^*$ uniformly at random in $\{1, \ldots, q\}$ and $\mathsf{t}^*_{\mathsf{L}}$ in $\{\infty, 1, \ldots, q - \Delta_{\mathsf{PCS}}\}$. It remains to briefly argue that if the guesses are correct, then the reduction can successfully simulate the original game *until* the successful injection. (Note that the game is considered won the moment a successful injection occurs. Therefore, the behavior of $\mathcal{A}'$ afterward is irrelevant.) This can be achieved trivially by simply rejecting all non-trivial injection attempt before $\mathsf{t}^*$, since for $\mathsf{TR}$ the state remains unchanged in case an injection is rejected. □

**Lemma 4.9.** *For $\mathsf{X} \in \{\mathsf{C}, \mathsf{Q}\}$, let $\Delta^{\mathsf{X}}_{\mathsf{PCS}}$ and $\Delta^{\mathsf{X}}_{\mathsf{FS}}$ denote the PCS and FS parameters for the classical and post-quantum CKAs, respectively, and let $\tau^{\mathsf{C}}$ and $\tau^{\mathsf{Q}}$ denote the respective epoch functions (as discussed in Remark 4.1). Then we have*

$$\mathsf{Adv}^{\mathsf{SM\text{-}auth\text{-}ss}}_{\mathcal{A}',\Delta^{\mathsf{C}}_{\mathsf{PCS}},\Delta^{\mathsf{C}}_{\mathsf{FS}},\tau}(1^\lambda) \leqslant \mathsf{Adv}^{\mathsf{CKA}^{\mathsf{C}}}_{\mathcal{B},\Delta^{\mathsf{C}}_{\mathsf{PCS}},\Delta^{\mathsf{C}}_{\mathsf{FS}}}(1^\lambda) + q \cdot \mathsf{Adv}^{\mathsf{PRF\text{-}PRNG}}_{\mathcal{C}}(1^\lambda) \\ + q \cdot \mathsf{Adv}^{\mathsf{PRG}}_{\mathcal{D}}(1^\lambda) + \mathsf{Adv}^{\mathsf{dPRF}}_{\mathcal{E}}(1^\lambda) + \mathsf{Adv}^{\mathsf{AEAD}}_{\mathcal{F}}(1^\lambda)$$

*in case the* classical *part* $\mathsf{CKA}^{\mathsf{C}}$ *is secure, and*

$$\mathsf{Adv}^{\mathsf{SM\text{-}auth\text{-}ss}}_{\mathcal{A}',\Delta^{\mathsf{Q}}_{\mathsf{PCS}}+1,\Delta^{\mathsf{Q}}_{\mathsf{FS}},\tau}(1^\lambda) \leqslant \mathsf{Adv}^{\mathsf{CKA}^{\mathsf{Q}}}_{\mathcal{B},\Delta^{\mathsf{Q}}_{\mathsf{PCS}},\Delta^{\mathsf{Q}}_{\mathsf{FS}}}(1^\lambda) + q \cdot \mathsf{Adv}^{\mathsf{PRF\text{-}PRNG}}_{\mathcal{C}}(1^\lambda) \\ + q \cdot \mathsf{Adv}^{\mathsf{PRG}}_{\mathcal{D}}(1^\lambda) + \mathsf{Adv}^{\mathsf{dPRF}}_{\mathcal{E}}(1^\lambda) + \mathsf{Adv}^{\mathsf{AEAD}}_{\mathcal{F}}(1^\lambda)$$

*in case the* post-quantum *part* $\mathsf{CKA}^{\mathsf{Q}}$ *is secure.*

*Proof.* First, we consider the injections for "old" epochs, i.e., where $\tau(\mathsf{idx}') < \mathsf{t}^*_{\mathsf{L}}$, but the party $\mathsf{P}$ is in epoch $\mathsf{t}^*$ when processing the injection. Note that at this point $\mathsf{P}$ already got the (correct) number of messages sent during $\tau(\mathsf{idx}')$ and has stored the individual $\mathsf{AEAD}$ keys in $\mathsf{StoredKeys}$. Therefore, $\mathsf{P}$ will not accept an injection for a period counter, according to the period function $\imath$, for which no message has been sent. Moreover, the $\mathsf{CKA}$ already moved on sufficiently such that those "trivial" injections no longer affect the protocol state. Therefore, we will ignore them in the following, for simplicity.

In the remainder, we bound the probability of a "non-trivial" attack using a sequence of hybrids. The sequence closely follows the one of the confidentiality proof — we mainly outline the differences.

$\mathsf{Hybrid}_1$**:** In the first hybrid, we modify $\mathsf{Game}^{\mathsf{SM\text{-}auth\text{-}ss}}_{\mathcal{A}',\Delta_{\mathsf{PCS}},\Delta_{\mathsf{FS}},\tau}$ as follows:

- We replace the key $\mathsf{K}_{\mathsf{CKA}}$ of epoch $\mathsf{t}^*_{\mathsf{L}} + \Delta_{\mathsf{PCS}}$ with a fresh independent one. That is, we replace it in both $\mathsf{TR\text{-}Send\text{-}P}$, when output by $\mathsf{CKA\text{-}Send\text{-}P}$, and in $\mathsf{TR\text{-}Rec\text{-}P}$, when output by $\mathsf{CKA\text{-}Rec\text{-}P}$, with the same freshly sampled key.
- If $\mathsf{t}^*_{\mathsf{L}} = -\infty$, i.e., if no corruption occurs the injection oracle becomes available during epoch $\mathsf{t}^*$, then $\mathsf{Hybrid}_1$ behaves as the original game.

Note that while processing the message that delivers this $\mathsf{K}_{\mathsf{CKA}}$ to the receiver, the receiver is still in the prior epoch. Therefore, injections are disallowed by $\mathsf{safe\text{-}inj}$ at this point. As a result, the argument becomes essentially the same as in the confidentiality case: the respective sender sampled the key using good randomness and no corruption exposing it is allowed. This yields a simple reduction to the $\mathsf{CKA}$

game in which either the correct key or an independently sampled one is produced. As a consequence, for the post-quantum $\mathsf{CKA}$ we obtain

$$\left|\Pr\left[\mathsf{Game}^{\mathsf{SM\text{-}auth\text{-}ss}}_{\mathcal{A}',\Delta_{\mathsf{PCS}}+1,\Delta_{\mathsf{FS}},\tau^{\mathsf{Q}}}(1^\lambda)=1\right]-\Pr\left[\mathsf{Hybrid}_1(1^\lambda)=1\right]\right|\leqslant \mathsf{Adv}^{\mathsf{CKA}^{\mathsf{Q}}}_{\mathcal{B},\Delta_{\mathsf{PCS}},\Delta_{\mathsf{FS}}}(1^\lambda),$$

and the analogous result for the classical $\mathsf{CKA}$ with the tighter $\Delta_{\mathsf{PCS}}$ bound.

$\mathsf{Hybrid}_2$**:** In the second hybrid, we modify $\mathsf{Hybrid}_1$ as follows:

- For all epochs starting from $\mathsf{t}^*_{\mathsf{L}}+\Delta_{\mathsf{PCS}}$ to $\mathsf{t}^*$, we replace the output of $\mathsf{KDF}_1$, i.e., $\mathsf{K}_{\mathsf{root}}$, $\mathsf{KS}$ and $\mathsf{KR}_{+1}$, with freshly sampled independent keys. (In the case of the classically secure $\mathsf{CKA}$, $\mathsf{KDF}_1$ just outputs two keys, which we replace by fresh ones.)
- In epoch $\mathsf{t}^*$, if the receiver is still in epoch $\mathsf{t}^*-1$ then we replace the keys by independent ones *for any injected ciphertext*, using the ones consistent with the sender for the honest delivery.

Again, the argument is fairly similar to the one from confidentiality, as no corruptions are allowed for that period. Some care, however, has to be taken with respect to (non-trivial) injections. For epochs $\mathsf{t}^*_{\mathsf{L}}+\Delta_{\mathsf{PCS}}$ to $\mathsf{t}^*-1$ no injections are allowed by $\mathsf{Hybrid}_1$. Thus, we can therefore use a simple sequence of additional hybrids to replace those keys by fresh ones.

In contrast, injections are allowed for $\mathsf{t}^*$. In particular, the receiver of such an injection might be still at epoch $\mathsf{t}^*-1$ at this stage, processing the injection attempt. We know from the prior argument that the $\mathsf{K}_{\mathsf{root}}$ the receiver stores at this point is fresh. Here we crucially rely on the "PRF" property of PRF-PRNG security of $\mathsf{KDF}_1$ to argue that we can replace all the resulting keys with independent and uniformly distributed ones. As a result, we can deduce

$$\left|\Pr\left[\mathsf{Hybrid}_2(1^\lambda)=1\right]-\Pr\left[\mathsf{Hybrid}_1(1^\lambda)=1\right]\right|\leqslant q\cdot\mathsf{Adv}^{\mathsf{PRF\text{-}PRNG}}_{\mathcal{C}}(1^\lambda).$$

$\mathsf{Hybrid}_3$**:** In the third hybrid, we modify $\mathsf{Hybrid}_2$ as follows:

- In epoch $\mathsf{t}^*$, we replace the output of $\mathsf{KDF}_2$, i.e. $\mathsf{KS}$, of the sending party with freshly sampled independent keys. For the receiving party, $\mathsf{KR}$ is replaced with the same key, i.e., the key used by the sender for the same period.

Due to the absence of corruptions, this simply follows by PRG-security of $\mathsf{KDF}_2$. Therefore, we obtain

$$\left|\Pr\left[\mathsf{Hybrid}_3(1^\lambda)=1\right]-\Pr\left[\mathsf{Hybrid}_2(1^\lambda)=1\right]\right|\leqslant q\cdot\mathsf{Adv}^{\mathsf{PRG}}_{\mathcal{D}}(1^\lambda).$$

$\mathsf{Hybrid}_4$**:** Finally, we modify $\mathsf{Hybrid}_3$ as follows:

- For all injection attempts in epoch $\mathsf{t}^*$, we replace the output of $\mathsf{K}_{\mathsf{aead}}:=\mathsf{KDF}_3(\mathsf{K}^{\mathsf{C}}_{\mathsf{aead}},\mathsf{K}^{\mathsf{Q}}_{\mathsf{aead}})$ with a fresh independent key, subject to consistency. For example, assume we consider $\mathsf{CKA}^{\mathsf{Q}}$ to be secure, then for each unique value of $\mathsf{K}^{\mathsf{C}}_{\mathsf{aead}}$, we replace the output with a fresh uniform key.

Assume $\mathsf{CKA}^{\mathsf{Q}}$ is assumed to be secure. Then, in $\mathsf{Hybrid}_3$ we replaced $\mathsf{K}^{\mathsf{Q}}_{\mathsf{aead}}$ for each injection attempt with independent and fresh values. Even if the attacker can cause the receiving party to reuse $\mathsf{K}^{\mathsf{C}}_{\mathsf{aead}}$ across injections, dual-PRF security of $\mathsf{KDF}_3$ ensures the outputs to be independent and freshly sampled. The analogous argument holds if we assume $\mathsf{CKA}^{\mathsf{C}}$ to be secure. Therefore, dual-PRF security ensures that the output is indistinguishable from a uniform random key in either case.

$$\left|\Pr\left[\mathsf{Hybrid}_4(1^\lambda)=1\right]-\Pr\left[\mathsf{Hybrid}_3(1^\lambda)=1\right]\right|\leqslant\mathsf{Adv}^{\mathsf{dPRF}}_{\mathcal{E}}(1^\lambda).$$

Finally, we consider the probability of $\mathcal{A}'$ winning $\mathsf{Hybrid}_4$ by succeeding with one of the injection attempts.
There are three cases to consider:

- The attacker injects in the transition from epoch $\mathsf{t}^* - 1$ to $\mathsf{t}^*$ with a modified $\mathsf{CKA}$ message, i.e., such that the $\mathsf{K}_{\mathsf{CKA}}$ differs from what the sender uses.
- The attacker injects in the transition from epoch $\mathsf{t}^* - 1$ to $\mathsf{t}^*$, but the receiver obtains the $\mathsf{K}_{\mathsf{CKA}}$ the sender used.
- The attacker injects after the receiver already honestly transitioned to $\mathsf{t}^*$.

In the first case, the attacker essentially tries to inject to a fresh key $\mathsf{K}_{\mathsf{aead}}$ (see $\mathsf{Hybrid}_4$) for which they have no information about (in particular not even seen a ciphertext for). $\mathsf{AEAD}$ security rules out such an injection. In the second and third cases, the attacker has seen a valid ciphertext under that key, from the sender, but tries to inject a different one. Again, $\mathsf{AEAD}$ security prevents such an attack. Overall, this probability trivially reduced to $\mathsf{AEAD}$ security:

$$\mathsf{Adv}_{\mathcal{A}'}^{\mathsf{Hybrid}_4}(1^\lambda) \leqslant \mathsf{Adv}_{\mathcal{F}}^{\mathsf{AEAD}}(1^\lambda).$$

The overall confidentiality statement then follows directly by adding the respective error terms, for both the classical and the post-quantum parts. □

# 5 From Ratcheting Key Encapsulation Mechanism to CKA

## 5.1 Definition of Forward-Secure Ratcheting KEM

In this section, we define a *forward-secure ratcheting* KEM (RKEM), serving as the main building block to construct a CKA. RKEM is a two party protocol, with parties exchanging encapsulation keys and ciphertexts in a ping-pong manner. In contrast to a regular KEM, the ciphertext not only depends on the encapsulation key received in the previous round, but additionally on the fresh decapsulation key for the current round.

**Definition 5.1.** *A* forward-secure ratcheting key encapsulation mechanism (RKEM) $\Pi_{\mathsf{RKEM}}$ *with key space $\mathcal{K}$, ciphertext space $\mathcal{CT}$, and ratcheting key spaces $\mathcal{RK}_{\mathsf{P}}$ and $\widehat{\mathcal{RK}}_{\mathsf{P}}$ for parties $\mathsf{P} \in \{\mathsf{A}, \mathsf{B}\}$ consists of PPT algorithms $\big(\mathsf{RSetup}, (\mathsf{RKeyGen\text{-}P}, \mathsf{REnc\text{-}P}, \mathsf{RDec\text{-}P})_{\mathsf{P} \in \{\mathsf{A},\mathsf{B}\}}\big)$ defined as follows:*

$\mathsf{RSetup}(1^\lambda) \xrightarrow{\$} \mathsf{par}$*: It takes as input the security parameter $1^\lambda$ and outputs a public parameter* $\mathsf{par}$*. We assume all algorithms to take* $\mathsf{par}$ *as input and may omit it for simplicity.*

$\mathsf{RKeyGen\text{-}P}(\mathsf{par}, \mathsf{mode}) \xrightarrow{\$} (\mathsf{ek}_{\mathsf{P}}, \mathsf{dk}_{\mathsf{P}})$ *: It takes as input the public parameter* $\mathsf{par}$ *and outputs encapsulation and decapsulation keys $(\mathsf{ek}_{\mathsf{P}}, \mathsf{dk}_{\mathsf{P}}) \in \mathcal{RK}_{\mathsf{P}}$ if $\mathsf{mode} = \perp$ and $(\mathsf{ek}_{\mathsf{P}}, \mathsf{dk}_{\mathsf{P}}) \in \widehat{\mathcal{RK}}_{\mathsf{P}}$ if $\mathsf{mode} = \mathtt{updated}$. In case $\mathsf{mode} = \perp$, we may simply ignore $\mathsf{mode}$ from the input when the context is clear.*[12]

$\mathsf{REnc\text{-}A}(\mathsf{ek}_{\mathsf{B}}, \mathsf{dk}_{\mathsf{A}}) \xrightarrow{\$} (\mathsf{ct}_{\mathsf{B}}, \mathsf{K}, \widehat{\mathsf{dk}}_{\mathsf{A}})$ *: It takes as input an encapsulation key $\mathsf{ek}_{\mathsf{B}}$ for party $\mathsf{B}$ and a decapsulation key for party $\mathsf{A}$, and outputs a ciphertext $\mathsf{ct}_{\mathsf{B}}$, a shared key $\mathsf{K} \in \mathcal{K}$, and a possibly updated decapsulation key $\widehat{\mathsf{dk}}_{\mathsf{A}}$.*

$\mathsf{RDec\text{-}A}(\mathsf{dk}_{\mathsf{A}}, \mathsf{ct}_{\mathsf{A}}, \mathsf{ek}_{\mathsf{B}}) \xrightarrow{\$} (\mathsf{K}, \widehat{\mathsf{ek}}_{\mathsf{B}})$ *: It takes as input a decapsulation key $\mathsf{dk}_{\mathsf{A}}$ for party $\mathsf{A}$, a ciphertext $\mathsf{ct}_{\mathsf{A}}$, and an encapsulation key for party $\mathsf{B}$, and outputs a shared key $\mathsf{K} \in \mathcal{K}$ and a possibly updated encapsulation key $\widehat{\mathsf{ek}}_{\mathsf{B}}$.*

*In the above, we define algorithms* $\mathsf{REnc\text{-}B}$ *and* $\mathsf{RDec\text{-}B}$ *analogously with roles of parties* $\mathsf{A}$ *and* $\mathsf{B}$ *swapped.*

*Remark* 5.2 (Non-forward-secure RKEM). Our definition of a forward-secure RKEM can naturally handle a *non*-forward-secure scheme as well. In this work we define a non-forward-secure RKEM by restricting $\widehat{\mathsf{dk}}_{\mathsf{P}} = \mathsf{dk}_{\mathsf{P}}$ and $\widehat{\mathsf{ek}}_{\mathsf{P}} = \mathsf{ek}_{\mathsf{P}}$ in algorithms $\mathsf{REnc\text{-}P}$ and $\mathsf{RDec\text{-}P}$, respectively, for $\mathsf{P} \in \{\mathsf{A}, \mathsf{B}\}$. While we can alternatively remove $\widehat{\mathsf{dk}}_{\mathsf{P}}$ and $\widehat{\mathsf{ek}}_{\mathsf{P}}$ from the outputs, we chose the former approach to be consistent with our forward-secure formalization, allowing us to construct secure messaging protocol in a unified framework.

[12] Indeed, $\mathsf{RKeyGen\text{-}P}$ with $\mathsf{mode} = \mathtt{updated}$ is mainly used for security analysis and will otherwise only appear in the setup of our construction. As such, we will typically omit $\mathsf{mode}$ outside this section.

To aid readability, we define $\mathcal{D}_{\mathsf{RKeyGen\text{-}P}}(\mathsf{par})$ (resp. $\widehat{\mathcal{D}}_{\mathsf{RKeyGen\text{-}P}}(\mathsf{par})$) for $\mathsf{P} \in \{\mathsf{A}, \mathsf{B}\}$ as the distribution of sampling $(\mathsf{ek_P}, \mathsf{dk_P}) \xleftarrow{\$} \mathsf{RKeyGen\text{-}P}(\mathsf{par}, \mathsf{mode})$ with $\mathsf{mode} = \perp$ (resp. $\mathsf{mode} = \mathtt{updated}$) for $\mathsf{par} \in \mathsf{RSetup}(1^\lambda)$. We use the shorthand $\mathcal{D}_{\mathsf{RKeyGen\text{-}P}}$ and $\widehat{\mathcal{D}}_{\mathsf{RKeyGen\text{-}P}}$ when $\mathsf{par}$ is randomly generated from $\mathsf{RSetup}(1^\lambda)$. In the following correctness and security definitions, we assume $\mathsf{par}$ is sampled and fixed once and for all, and only use $\mathcal{D}_{\mathsf{RKeyGen\text{-}P}}$ and $\widehat{\mathcal{D}}_{\mathsf{RKeyGen\text{-}P}}$. While we omit $\mathsf{par}$ for readability, it is understood that the probability is taken over the randomness of generating $\mathsf{par}$.

We first define correctness. Correctness comes in two flavors. First, we require that a ciphertext generated using an *updated* encapsulation key can be decrypted correctly using an *updated* decapsulation key. Second, we require that the updated keys generated during the encapsulation and decapsulation algorithms have the same distribution as keys sampled directly using $\widehat{\mathcal{D}}_{\mathsf{RKeyGen\text{-}P}}$. This is a key property that allows us to effectively focus only on one round of interaction between the parties, as opposed to arguing correctness of a ping-pong interaction in its entirety.

**Definition 5.3 (Correctness).** *We say a ratcheting* KEM $\Pi_{\mathsf{RKEM}}$ *is* correct *if it satisfies two properties. The first property,* correctness with updated keys*, requires the following to hold:*

$$\Pr\left[\begin{array}{c} (\mathsf{ek_A}, \mathsf{dk_A}) \xleftarrow{\$} \mathcal{D}_{\mathsf{RKeyGen\text{-}A}}, (\widehat{\mathsf{ek}}_\mathsf{B}, \widehat{\mathsf{dk}}_\mathsf{B}) \xleftarrow{\$} \widehat{\mathcal{D}}_{\mathsf{RKeyGen\text{-}B}}, \\ (\mathsf{ct_B}, \mathsf{K}, \widehat{\mathsf{dk}}_\mathsf{A}) \xleftarrow{\$} \mathsf{REnc\text{-}A}(\widehat{\mathsf{ek}}_\mathsf{B}, \mathsf{dk_A}), \\ (\mathsf{K}', \widehat{\mathsf{ek}}_\mathsf{A}) \xleftarrow{\$} \mathsf{RDec\text{-}B}(\widehat{\mathsf{dk}}_\mathsf{B}, \mathsf{ct_B}, \mathsf{ek_A}) \end{array} : \mathsf{K} = \mathsf{K}' \right] = 1 - \mathsf{negl}(\lambda).$$

*We require the above to hold with the roles of parties* A *and* B *swapped. We denote the marginal distribution of* $(\widehat{\mathsf{ek}}_\mathsf{A}, \widehat{\mathsf{dk}}_\mathsf{A})$ *generated through the above process as* $\mathcal{D}'_{\mathsf{RKeyGen\text{-}A}}$*, and define* $\mathcal{D}'_{\mathsf{RKeyGen\text{-}B}}$ *similarly. The second property,* correctness of update key distribution*, then requires that* $\mathcal{D}'_{\mathsf{RKeyGen\text{-}P}}$ *is statistically close to* $\widehat{\mathcal{D}}_{\mathsf{RKeyGen\text{-}P}}$ *for* $\mathsf{P} \in \{\mathsf{A}, \mathsf{B}\}$.

We next define *forward-secure* IND-CPA security. This is captured through an extension of a natural IND-CPA security game where the adversary is provided with the updated decapsulation key along the challenge ciphertext.

**Definition 5.4 (FS-IND-CPA Security).** *We say a ratcheting* KEM $\Pi_{\mathsf{RKEM}}$ *is* forward-secure IND-CPA (FS-IND-CPA) *secure if the advantages*

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{FS\text{-}IND\text{-}CPA\text{-}A}}(1^\lambda) := \left| \Pr\left[\begin{array}{c} b \xleftarrow{\$} \{0,1\}, \mathsf{K}_1 \xleftarrow{\$} \mathcal{K}, \\ (\mathsf{ek_A}, \mathsf{dk_A}) \xleftarrow{\$} \mathcal{D}_{\mathsf{RKeyGen\text{-}A}}, (\widehat{\mathsf{ek}}_\mathsf{B}, \widehat{\mathsf{dk}}_\mathsf{B}) \xleftarrow{\$} \widehat{\mathcal{D}}_{\mathsf{RKeyGen\text{-}B}}, \\ (\mathsf{ct_B}, \mathsf{K}_0, \widehat{\mathsf{dk}}_\mathsf{A}) \xleftarrow{\$} \mathsf{REnc\text{-}A}(\widehat{\mathsf{ek}}_\mathsf{B}, \mathsf{dk_A}), \\ (\,\cdot\,, \widehat{\mathsf{ek}}_\mathsf{A}) \xleftarrow{\$} \mathsf{RDec\text{-}B}(\widehat{\mathsf{dk}}_\mathsf{B}, \mathsf{ct_B}, \mathsf{ek_A}), \\ b' \xleftarrow{\$} \mathcal{A}(\mathsf{ek_A}, \widehat{\mathsf{ek}}_\mathsf{A}, \widehat{\mathsf{ek}}_\mathsf{B}, \mathsf{ct_B}, \widehat{\mathsf{dk}}_\mathsf{A}, \mathsf{K}_b) \end{array} : b = b' \right] - \frac{1}{2} \right|$$

*and* $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{FS\text{-}IND\text{-}CPA\text{-}B}}$*, defined analogously with the roles of parties* A *and* B *swapped, are negligible. We denote* $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{FS\text{-}IND\text{-}CPA}} := \max_{\mathsf{P} \in \{\mathsf{A},\mathsf{B}\}} \left(\mathsf{Adv}_{\mathcal{A}}^{\mathsf{FS\text{-}IND\text{-}CPA\text{-}P}}(1^\lambda)\right)$.

*As a special case, we say a (non-forward-secure)* RKEM *(cf. Remark 5.2) is simply* IND-CPA *secure if* $\widehat{\mathsf{dk}}_\mathsf{A}$ *is not given to* $\mathcal{A}$ *in the above game.*

Lastly, we define *ratchet simulatability*. This comes with (roughly) two properties: updated key and ciphertext simulatability. The former property stipulates that the updated key $(\widehat{\mathsf{ek}}_\mathsf{P}, \widehat{\mathsf{dk}}_\mathsf{P})$ can be simulated from the non-updated key $(\mathsf{ek_P}, \mathsf{dk_P})$. Importantly, the updated key does not depend on the peer's encapsulation key $\mathsf{ek}_{\bar{\mathsf{P}}}$ required to run REnc-P. This is used to break the dependence on the updated keys from the peer's keys, allowing us to prove security of CKA based on induction. The latter property stipulates that the ciphertext $\mathsf{ct_P}$ generated using the peer $\bar{\mathsf{P}}$'s decapsulation key can be simulated using instead user P's

```
Distribution $\mathcal{D}^{\mathsf{KeyBaseSim}}_{\mathsf{A},0}$
    $(\widehat{\mathsf{ek}}_{\mathsf{A}}, \widehat{\mathsf{dk}}_{\mathsf{A}}) \xleftarrow{\$} \widehat{\mathcal{D}}_{\mathsf{RKeyGen\text{-}A}}$
    return $(\widehat{\mathsf{ek}}_{\mathsf{A}}, \widehat{\mathsf{dk}}_{\mathsf{A}})$
```

```
Distribution $\mathcal{D}^{\mathsf{KeyBaseSim}}_{\mathsf{A},1}$
    $(\mathsf{ek}_{\mathsf{A}}, \mathsf{dk}_{\mathsf{A}}) \xleftarrow{\$} \mathcal{D}_{\mathsf{RKeyGen\text{-}A}}$
    $(\widehat{\mathsf{ek}}_{\mathsf{A}}, \widehat{\mathsf{dk}}_{\mathsf{A}}, \_) \xleftarrow{\$} \mathsf{RSimKey\text{-}A}_1(\mathsf{ek}_{\mathsf{A}}, \mathsf{dk}_{\mathsf{A}})$
    return $(\widehat{\mathsf{ek}}_{\mathsf{A}}, \widehat{\mathsf{dk}}_{\mathsf{A}})$
```

Figure 10: Base Key simulatability.

```
Distribution $\mathcal{D}^{\mathsf{KeyUpdSim}}_{\mathsf{A},0}$
    $(\mathsf{ek}_{\mathsf{B}}, \mathsf{dk}_{\mathsf{B}}) \xleftarrow{\$} \mathcal{D}_{\mathsf{RKeyGen\text{-}B}}\{\mathsf{rand}_0\}$
    $(\widehat{\mathsf{ek}}_{\mathsf{B}}, \widehat{\mathsf{dk}}_{\mathsf{B}}, \mathsf{aux}_0) \xleftarrow{\$} \mathsf{RSimKey\text{-}B}_1(\mathsf{ek}_{\mathsf{B}}, \mathsf{dk}_{\mathsf{B}})$
    $(\mathsf{ek}_{\mathsf{A}}, \mathsf{dk}_{\mathsf{A}}) \xleftarrow{\$} \mathcal{D}_{\mathsf{RKeyGen\text{-}A}}\{\mathsf{rand}_1\}$
    $(\mathsf{ct}_{\mathsf{B}}, \mathsf{K}, \widehat{\mathsf{dk}}_{\mathsf{A}}) \leftarrow \mathsf{REnc\text{-}A}(\widehat{\mathsf{ek}}_{\mathsf{B}}, \mathsf{dk}_{\mathsf{A}}; \mathsf{rand}_2)$
    $(\mathsf{K}', \widehat{\mathsf{ek}}_{\mathsf{A}}) \xleftarrow{\$} \mathsf{RDec\text{-}B}(\widehat{\mathsf{dk}}_{\mathsf{B}}, \mathsf{ct}_{\mathsf{B}}, \mathsf{ek}_{\mathsf{A}})$
    return $\big((\widehat{\mathsf{ek}}_{\mathsf{B}}, \widehat{\mathsf{dk}}_{\mathsf{B}}), (\widehat{\mathsf{ek}}_{\mathsf{A}}, \widehat{\mathsf{dk}}_{\mathsf{A}}), \mathsf{ct}_{\mathsf{B}}, \mathsf{K}, \mathsf{K}',$
        $\mathsf{aux}_0, \mathsf{rand}_0, \mathsf{rand}_1, \mathsf{rand}_2\big)$
```

```
Distribution $\mathcal{D}^{\mathsf{KeyUpdSim}}_{\mathsf{A},1}$
    $(\mathsf{ek}_{\mathsf{B}}, \mathsf{dk}_{\mathsf{B}}) \xleftarrow{\$} \mathcal{D}_{\mathsf{RKeyGen\text{-}B}}\{\mathsf{rand}_0\}$
    $(\widehat{\mathsf{ek}}_{\mathsf{B}}, \widehat{\mathsf{dk}}_{\mathsf{B}}, \mathsf{aux}_0) \xleftarrow{\$} \mathsf{RSimKey\text{-}B}_1(\mathsf{ek}_{\mathsf{B}}, \mathsf{dk}_{\mathsf{B}})$
    $(\mathsf{ek}_{\mathsf{A}}, \mathsf{dk}_{\mathsf{A}}) \xleftarrow{\$} \mathcal{D}_{\mathsf{RKeyGen\text{-}A}}\{\mathsf{rand}_1\}$
    $(\widehat{\mathsf{ek}}_{\mathsf{A}}, \widehat{\mathsf{dk}}_{\mathsf{A}}, \mathsf{aux}_1) \xleftarrow{\$} \mathsf{RSimKey\text{-}A}_1(\mathsf{ek}_{\mathsf{A}}, \mathsf{dk}_{\mathsf{A}})$
    $(\mathsf{ct}_{\mathsf{B}}, \mathsf{K}, \mathsf{K}', \mathsf{rand}_2) \xleftarrow{\$} \mathsf{RSimKey\text{-}A}_2(\widehat{\mathsf{ek}}_{\mathsf{B}}, \widehat{\mathsf{dk}}_{\mathsf{B}}, \mathsf{aux}_1)$
    return $\big((\widehat{\mathsf{ek}}_{\mathsf{B}}, \widehat{\mathsf{dk}}_{\mathsf{B}}), (\widehat{\mathsf{ek}}_{\mathsf{A}}, \widehat{\mathsf{dk}}_{\mathsf{A}}), \mathsf{ct}_{\mathsf{B}}, \mathsf{K}, \mathsf{K}',$
        $\mathsf{aux}_0, \mathsf{rand}_0, \mathsf{rand}_1, \mathsf{rand}_2\big)$
```

Figure 11: Updated Key simulatability. The text highlighted in blue denotes the main differences between the two distributions. Recall $D\{\mathsf{rand}\}$ denotes the process of sampling from the distribution $D$ with randomness $\mathsf{rand}$. Above, we assume $\mathsf{rand}$ (except for those output by $\mathsf{RSimKey\text{-}A}_2$) to be distributed uniformly over their respective domain.

```
Distribution $\mathcal{D}^{\mathsf{CtxtSim}}_{\mathsf{B},0}$
    $(\mathsf{ek}_{\mathsf{A}}, \mathsf{dk}_{\mathsf{A}}) \xleftarrow{\$} \mathcal{D}_{\mathsf{RKeyGen\text{-}A}}\{\mathsf{rand}\}$
    $(\widehat{\mathsf{ek}}_{\mathsf{A}}, \widehat{\mathsf{dk}}_{\mathsf{A}}, \mathsf{aux}) \xleftarrow{\$} \mathsf{RSimKey\text{-}A}_1(\mathsf{ek}_{\mathsf{A}}, \mathsf{dk}_{\mathsf{A}})$
    $(\mathsf{ek}_{\mathsf{B}}, \mathsf{dk}_{\mathsf{B}}) \xleftarrow{\$} \mathcal{D}_{\mathsf{RKeyGen\text{-}B}}$
    $(\mathsf{ct}_{\mathsf{A}}, \mathsf{K}, \widehat{\mathsf{dk}}_{\mathsf{B}}) \xleftarrow{\$} \mathsf{REnc\text{-}B}(\widehat{\mathsf{ek}}_{\mathsf{A}}, \mathsf{dk}_{\mathsf{B}})$
    $(\mathsf{K}', \widehat{\mathsf{ek}}_{\mathsf{B}}) \xleftarrow{\$} \mathsf{RDec\text{-}A}(\widehat{\mathsf{dk}}_{\mathsf{A}}, \mathsf{ct}_{\mathsf{A}}, \mathsf{ek}_{\mathsf{B}})$
    return $\big(\mathsf{aux}, \mathsf{rand}, (\widehat{\mathsf{ek}}_{\mathsf{A}}, \widehat{\mathsf{dk}}_{\mathsf{A}}), \mathsf{ct}_{\mathsf{A}},$
        $(\mathsf{ek}_{\mathsf{B}}, \widehat{\mathsf{ek}}_{\mathsf{B}}), (\mathsf{K}, \mathsf{K}')\big)$
```

```
Distribution $\mathcal{D}^{\mathsf{CtxtSim}}_{\mathsf{B},1}$
    $(\mathsf{ek}_{\mathsf{A}}, \mathsf{dk}_{\mathsf{A}}) \xleftarrow{\$} \mathcal{D}_{\mathsf{RKeyGen\text{-}A}}\{\mathsf{rand}\}$
    $(\widehat{\mathsf{ek}}_{\mathsf{A}}, \widehat{\mathsf{dk}}_{\mathsf{A}}, \mathsf{aux}) \xleftarrow{\$} \mathsf{RSimKey\text{-}A}_1(\mathsf{ek}_{\mathsf{A}}, \mathsf{dk}_{\mathsf{A}})$
    $(\widehat{\mathsf{ek}}_{\mathsf{B}}, \widehat{\mathsf{dk}}_{\mathsf{B}}) \xleftarrow{\$} \widehat{\mathcal{D}}_{\mathsf{RKeyGen\text{-}B}}$
    $(\mathsf{ct}_{\mathsf{A}}, \mathsf{ek}_{\mathsf{B}}, \mathsf{K}, \mathsf{K}') \xleftarrow{\$} \mathsf{RSimCtxt\text{-}B}(\widehat{\mathsf{ek}}_{\mathsf{B}}, \widehat{\mathsf{ek}}_{\mathsf{A}}, \widehat{\mathsf{dk}}_{\mathsf{A}})$
    return $\big(\mathsf{aux}, \mathsf{rand}, (\widehat{\mathsf{ek}}_{\mathsf{A}}, \widehat{\mathsf{dk}}_{\mathsf{A}}), \mathsf{ct}_{\mathsf{A}},$
        $(\mathsf{ek}_{\mathsf{B}}, \widehat{\mathsf{ek}}_{\mathsf{B}}), (\mathsf{K}, \mathsf{K}')\big)$
```

Figure 12: Ciphertext simulatability. The text highlighted in blue denotes the main differences between the two distributions.

(updated) decapsulation key. Put differently, $\mathsf{ct}_{\mathsf{P}}$ along with the knowledge of $\bar{\mathsf{P}}$'s decapsulation key does not leak any information of $\mathsf{P}$'s decapsulation key. This is a key property to argue PCS for CKA as it is used to argue that once $\mathsf{P}$ generates a fresh pair of key, it will *heal* $\mathsf{P}$ despite $\bar{\mathsf{P}}$ being corrupt. Lastly, we have one more additional property named base-key simulatability. This is a minor property required to capture the *first* keys that are shared among the users in the CKA protocol.

**Definition 5.5 (Ratchet Simulatability).** *For $b \in \{0,1\}$, let $\mathcal{D}^{\mathsf{KeyBaseSim}}_{\mathsf{A},b}$ be the distributions as defined in*

*Fig. 10, $\mathcal{D}^{\mathsf{KeyUpdSim}}_{\mathsf{A},b}$ be the distributions as defined in Fig. 11, and $\mathcal{D}^{\mathsf{CtxtSim}}_{\mathsf{B},b}$ be the distributions as defined in Fig. 12. Moreover, let $\mathcal{D}^{\mathsf{KeyBaseSim}}_{\mathsf{B},b}$ and $\mathcal{D}^{\mathsf{KeyUpdSim}}_{\mathsf{B},b}$ and $\mathcal{D}^{\mathsf{CtxtSim}}_{\mathsf{B},b}$ be defined analogously with the roles of the two parties swapped in the respective experiments. We say a ratcheting* KEM $\Pi_{\mathsf{RKEM}}$ *is* ratchet simulatable *if there exists efficient simulators* $(\mathsf{RSimKey\text{-}P}_1, \mathsf{RSimKey\text{-}P}_2, \mathsf{RSimCtxt\text{-}P})_{\mathsf{P}\in\{\mathsf{A},\mathsf{B}\}}$ *such that the advantage against* base-key simulatability

$$\mathsf{Adv}^{\mathsf{KeyBaseSim\text{-}P}}_{\mathcal{A}}(1^\lambda) := \left|\Pr[b \xleftarrow{\$} \{0,1\}, x \xleftarrow{\$} \mathcal{D}^{\mathsf{KeyBaseSim}}_{\mathsf{P},b}, b' \xleftarrow{\$} \mathcal{A}(x) : b' = b] - \frac{1}{2}\right|,$$

*the advantage against* updated key simulatability

$$\mathsf{Adv}^{\mathsf{KeyUpdSim\text{-}P}}_{\mathcal{A}}(1^\lambda) := \left|\Pr[b \xleftarrow{\$} \{0,1\}, x \xleftarrow{\$} \mathcal{D}^{\mathsf{KeyUpdSim}}_{\mathsf{P},b}, b' \xleftarrow{\$} \mathcal{A}(x) : b' = b] - \frac{1}{2},\right|$$

*and the advantage against* ciphertext simulatability

$$\mathsf{Adv}^{\mathsf{CtxtSim\text{-}P}}_{\mathcal{A}}(1^\lambda) := \left|\Pr[b \xleftarrow{\$} \{0,1\}, x \xleftarrow{\$} \mathcal{D}^{\mathsf{CtxtSim}}_{\mathsf{P},b}, b' \xleftarrow{\$} \mathcal{A}(x) : b' = b] - \frac{1}{2}\right|$$

*for both* $\mathsf{P} \in \{\mathsf{A},\mathsf{B}\}$ *are negligible. We denote* $\mathsf{Adv}^{\mathsf{KeyBaseSim}}_{\mathcal{A}} := \max_{\mathsf{P}\in\{\mathsf{A},\mathsf{B}\}}\left(\mathsf{Adv}^{\mathsf{KeyBaseSim\text{-}P}}_{\mathcal{A}}(1^\lambda)\right)$, $\mathsf{Adv}^{\mathsf{KeyUpdSim}}_{\mathcal{A}} := \max_{\mathsf{P}\in\{\mathsf{A},\mathsf{B}\}}\left(\mathsf{Adv}^{\mathsf{KeyUpdSim\text{-}P}}_{\mathcal{A}}(1^\lambda)\right)$ *and* $\mathsf{Adv}^{\mathsf{CtxtSim}}_{\mathcal{A}} := \max_{\mathsf{P}\in\{\mathsf{A},\mathsf{B}\}}\left(\mathsf{Adv}^{\mathsf{CtxtSim\text{-}P}}_{\mathcal{A}}(1^\lambda)\right)$.

**Instantiations.** In this work, we consider five instantiations of RKEM. A generic instantiation based of any KEM is presented in Appendix A. Second, we have an optimized forward-secure and an optimized non-forward secure instantiation based on lattices and based on Diffie-Hellman, each. The lattice based constructions, called Katana-RKEM, are presented in Section 6. The Diffie-Hellman based instantiations modularize the Double Ratchet and the forward-secure variant thereof by Bienstock et al. [BFG+22a] — for completeness they are presented in Appendix A.

## 5.2 A Generic Construction of CKA from Ratcheting KEM

We now present a simple construction of CKA based on RKEM. In the CKA protocol, for each send operation, a party P first samples a fresh key pair using RKeyGen-P. P then encapsulates a symmetric key to the other party under the latest public key from the other party; the freshly sampled secret key is updated as part of this process. The resulting ciphertext along the freshly sampled public key is then sent to the other party while the updated secret key is stored. The receiving party analogously simply uses their secret key to decapsulate the received ciphertext and public key, and stores the updated public key while erasing their own secret key. The protocol assumes a public-secret key pair of B to be distributed as setup such that A can initiate the first send operation. A schematic overview of the protocol is depicted in Fig. 13 while a formal description is presented in Fig. 14.

## 5.3 Security

Lastly, we provide the security proof for the generic CKA construction based on RKEM.

**Theorem 5.6.** *For any correct and forward-secure* RKEM, *the protocol from Fig. 14 is a correct and secure* CKA *protocol with* $\Delta_{\mathsf{FS}} = 0$ *and* $\Delta_{\mathsf{PCS}} = 2$. *Moreover, if the* RKEM *is non-forward secure, then the protocol is a secure* CKA *with* $\Delta_{\mathsf{FS}} = 1$ *and* $\Delta_{\mathsf{PCS}} = 2$.

*More specifically, let $q$ denote an upper bound on the number of epochs $\mathcal{A}$ creates and let $\epsilon^{\mathsf{RKEM}}_{\mathsf{corr}}$ denote the correctness error of the* RKEM. *Then we have*

$$\mathsf{Adv}^{\mathsf{CKA}}_{\mathcal{A},\Delta_{\mathsf{FS}},\Delta_{\mathsf{PCS}}}(1^\lambda) \leqslant q \cdot \epsilon^{\mathsf{RKEM}}_{\mathsf{corr}} + \mathsf{Adv}^{\mathsf{KeyBaseSim\text{-}B}}_{\mathcal{B}}(1^\lambda) + (q-1) \cdot \mathsf{Adv}^{\mathsf{KeyUpdSim}}_{\mathcal{C}}(1^\lambda)$$

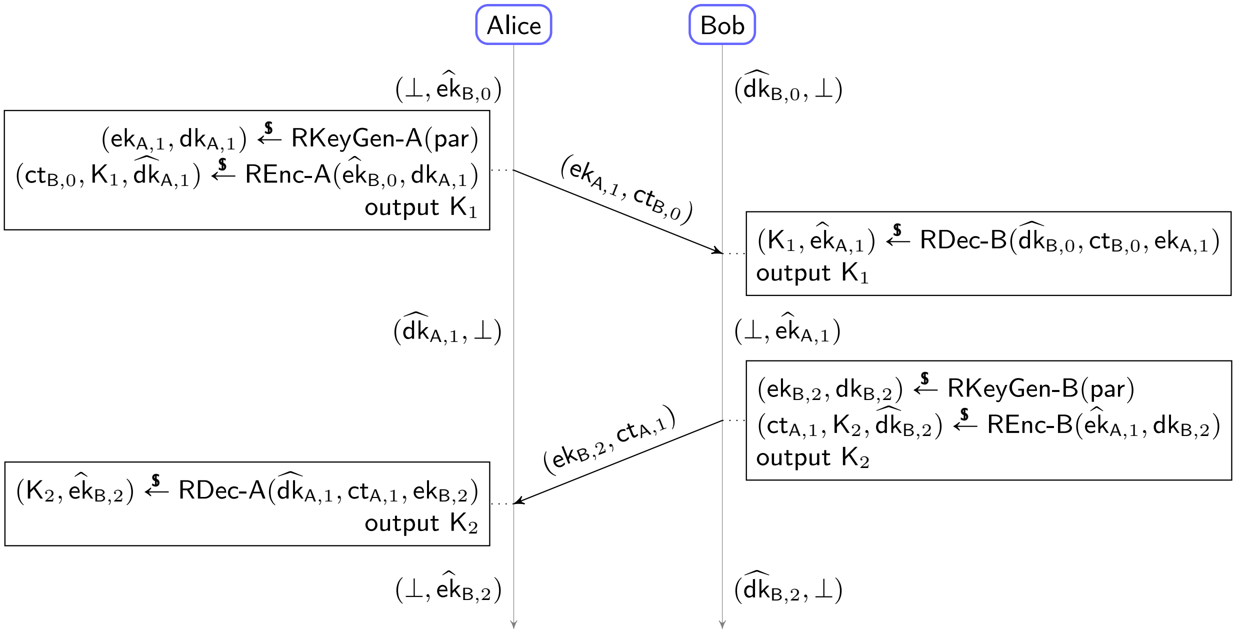
Alice
Bob
$(\perp, \widehat{\mathsf{ek}}_{\mathsf{B},0})$
$(\widehat{\mathsf{dk}}_{\mathsf{B},0}, \perp)$
$(\mathsf{ek}_{\mathsf{A},1}, \mathsf{dk}_{\mathsf{A},1}) \xleftarrow{\$} \mathsf{RKeyGen\text{-}A}(\mathsf{par})$
$(\mathsf{ct}_{\mathsf{B},0}, \mathsf{K}_1, \widehat{\mathsf{dk}}_{\mathsf{A},1}) \xleftarrow{\$} \mathsf{REnc\text{-}A}(\widehat{\mathsf{ek}}_{\mathsf{B},0}, \mathsf{dk}_{\mathsf{A},1})$
output $\mathsf{K}_1$
$(ek_{A,1}, ct_{B,0})$
$(\mathsf{K}_1, \widehat{\mathsf{ek}}_{\mathsf{A},1}) \xleftarrow{\$} \mathsf{RDec\text{-}B}(\widehat{\mathsf{dk}}_{\mathsf{B},0}, \mathsf{ct}_{\mathsf{B},0}, \mathsf{ek}_{\mathsf{A},1})$
output $\mathsf{K}_1$
$(\widehat{\mathsf{dk}}_{\mathsf{A},1}, \perp)$
$(\perp, \widehat{\mathsf{ek}}_{\mathsf{A},1})$
$(\mathsf{ek}_{\mathsf{B},2}, \mathsf{dk}_{\mathsf{B},2}) \xleftarrow{\$} \mathsf{RKeyGen\text{-}B}(\mathsf{par})$
$(\mathsf{ct}_{\mathsf{A},1}, \mathsf{K}_2, \widehat{\mathsf{dk}}_{\mathsf{B},2}) \xleftarrow{\$} \mathsf{REnc\text{-}B}(\widehat{\mathsf{ek}}_{\mathsf{A},1}, \mathsf{dk}_{\mathsf{B},2})$
output $\mathsf{K}_2$
$(\mathsf{ek}_{\mathsf{B},2}, \mathsf{ct}_{\mathsf{A},1})$
$(\mathsf{K}_2, \widehat{\mathsf{ek}}_{\mathsf{B},2}) \xleftarrow{\$} \mathsf{RDec\text{-}A}(\widehat{\mathsf{dk}}_{\mathsf{A},1}, \mathsf{ct}_{\mathsf{A},1}, \mathsf{ek}_{\mathsf{B},2})$
output $\mathsf{K}_2$
$(\perp, \widehat{\mathsf{ek}}_{\mathsf{B},2})$
$(\widehat{\mathsf{dk}}_{\mathsf{B},2}, \perp)$

Figure 13: The first two messages of the RKEM based CKA. Computation for CKA-Send-P and CKA-Rec-P are shown in boxes, while the state kept in between operations is shown next to the party.

$$+ \ \mathsf{Adv}_{\mathcal{D}}^{\mathsf{CtxtSim}}(1^\lambda) + \mathsf{Adv}_{\mathcal{E}}^{\mathsf{FS\text{-}IND\text{-}CPA}}(1^\lambda),$$

*with* $\Delta_{\mathsf{PCS}} = 2$ *and* $\Delta_{\mathsf{FS}} = 0$ *if the* RKEM *is forward secure. If the* RKEM *is non-forward secure, we obtain the same bound except with* $\mathsf{Adv}_{\mathcal{E}}^{\mathsf{IND\text{-}CPA}}(1^\lambda)$ *and for* $\Delta_{\mathsf{FS}} = 1$.

*Proof.* Consider the CKA game $\mathsf{Game}_{\mathcal{A},\mathsf{t}^*}^{\mathsf{CKA}}$ as depicted in Fig. 15 with the protocol from Fig. 14 inlined and some minor syntactic changes. In particular, we keep the protocol state expanded as part of the game's state rather than parsing and reassembling $\mathsf{st}_\mathsf{P}$ for each operation. Analogously, we keep $\mathsf{I}_\mathsf{K}$ and CKA messages $\rho_\mathsf{t}$ in their expanded form, which especially implies that CKA-Init-P which just parses $\mathsf{I}_\mathsf{K}$ becomes vacuous. Furthermore, we observe that the epoch counters maintained by the game and the ones maintained by the protocol match, and therefore unify them into a single counter $\mathsf{t}_\mathsf{P}$ per party. Finally, we remove some redundant checks on the epoch counters in CKA-Rec-P that always hold when messages are honestly delivered by an adversary that respects alternating communication.

**Correctness:** We first argue correctness of the scheme; namely that line 3 of Receive-P (cf. Fig. 4) never applies. To this end, observe that by correctness of RKEM, $\mathsf{K}_1' = \mathsf{K}_1$ and the keypair $(\widehat{\mathsf{ek}}_1, \widehat{\mathsf{dk}}_1)$ is indistinguishable from a fresh one, when only considering the keys themselves. Since the protocol for epoch $\mathsf{t}$ only uses the keys $(\widehat{\mathsf{ek}}_{\mathsf{t}-1}, \widehat{\mathsf{dk}}_{\mathsf{t}-1})$ (and no side information thereof) we can therefore inductively invoke correctness and argue that by correctness $\mathsf{K}_\mathsf{t}' = \mathsf{K}_\mathsf{t}$ and that the key pair $(\widehat{\mathsf{ek}}_\mathsf{t}, \widehat{\mathsf{dk}}_\mathsf{t})$ is indistinguishable from a fresh keypair (when ignoring side information).

In the following, we therefore consider a modification of $\mathsf{Game}_{\mathcal{A},\mathsf{t}^*,b}^{\mathsf{CKA}}$ where the correctness condition has been removed and bound the respective advantage of $\mathcal{A}$.

**$\mathsf{Hybrid}_{\mathcal{A},\mathsf{t}^*,b,0}^{\mathsf{CKA}}$ to $\mathsf{Hybrid}_{\mathcal{A},\mathsf{t}^*,b,\mathsf{t}^*-2}^{\mathsf{CKA}}$:** We define a sequence of hybrids $\mathsf{Hybrid}_{\mathcal{A},\mathsf{t}^*,b,i}^{\mathsf{CKA}}$ for $0 \leqslant i \leqslant \mathsf{t}^* - 2$ and $b \in \{0, 1\}$. The hybrids are based on Fig. 15 with modifications described below — they are depicted in Fig. 16.

```
$\mathsf{CKA\text{-}Init\text{-}KeyGen}(1^\lambda)$
    $\mathsf{par} \xleftarrow{\$} \mathsf{RSetup}(1^\lambda)$
    $(\widehat{\mathsf{ek}}_\mathsf{B}, \widehat{\mathsf{dk}}_\mathsf{B}) \xleftarrow{\$} \mathsf{RKeyGen\text{-}B}(\mathsf{par}, \texttt{updated})$
    $\textbf{return } \mathsf{I_K} := (\widehat{\mathsf{ek}}_\mathsf{B}, \widehat{\mathsf{dk}}_\mathsf{B}, \mathsf{par})$

$\mathsf{CKA\text{-}Init\text{-}A}(\mathsf{I_K})$
    $\textbf{parse } (\widehat{\mathsf{ek}}_\mathsf{B}, \widehat{\mathsf{dk}}_\mathsf{B}, \mathsf{par}) \leftarrow \mathsf{I_K}$
    $\mathsf{t_A} := 0$
    $\mathsf{st_A} := (\mathsf{t_A}, \bot, \widehat{\mathsf{ek}}_\mathsf{B}, \mathsf{par})$
    $\textbf{return } \mathsf{st_A}$

$\mathsf{CKA\text{-}Init\text{-}B}(\mathsf{I_K})$
    $\textbf{parse } (\widehat{\mathsf{ek}}_\mathsf{B}, \widehat{\mathsf{dk}}_\mathsf{B}, \mathsf{par}) \leftarrow \mathsf{I_K}$
    $\mathsf{t_B} := 0$
    $\mathsf{st_B} := (\mathsf{t_B}, \widehat{\mathsf{dk}}_\mathsf{B}, \bot, \mathsf{par})$
    $\textbf{return } \mathsf{st_B}$
```

```
$\mathsf{CKA\text{-}Send\text{-}A}(\mathsf{st_A})$
    $\textbf{parse } (\mathsf{t_A}, \_, \widehat{\mathsf{ek}}_\mathsf{B}, \mathsf{par}) \leftarrow \mathsf{st_A}$
    $\textbf{req } [\![\mathsf{t_A} \text{ is even}]\!]$
    $\mathsf{t_A} \mathrel{+}= 1$
    $(\mathsf{ek_A}, \mathsf{dk_A}) \xleftarrow{\$} \mathsf{RKeyGen\text{-}A}(\mathsf{par})$
    $(\mathsf{ct_B}, \mathsf{K}, \widehat{\mathsf{dk}}_\mathsf{A}) \xleftarrow{\$} \mathsf{REnc\text{-}A}(\widehat{\mathsf{ek}}_\mathsf{B}, \mathsf{dk_A})$
    $\rho := (\mathsf{t_A}, \mathsf{ek_A}, \mathsf{ct_B})$    // Send $(\mathsf{ek_A}, \mathsf{ct_B})$
    $\mathsf{st_A} := (\mathsf{t_A}, \widehat{\mathsf{dk}}_\mathsf{A}, \bot, \mathsf{par})$
    $\textbf{return } (\mathsf{K}, \rho, \mathsf{st_A})$

$\mathsf{CKA\text{-}Rec\text{-}B}(\mathsf{st_B}, \rho)$
    $\textbf{parse } (\mathsf{t_B}, \widehat{\mathsf{dk}}_\mathsf{B}, \_, \mathsf{par}) \leftarrow \mathsf{st_B}$
    $\textbf{req } [\![\mathsf{t_B} \text{ is even}]\!]$
    $\textbf{parse } (\mathsf{t_A}, \mathsf{ek_A}, \mathsf{ct_B}) \leftarrow \rho$
    $\textbf{req } [\![\mathsf{t_A} = \mathsf{t_B} + 1]\!]$
    $\mathsf{t_B} \mathrel{+}= 1$
    $(\mathsf{K}, \widehat{\mathsf{ek}}_\mathsf{A}) \xleftarrow{\$} \mathsf{RDec\text{-}B}(\widehat{\mathsf{dk}}_\mathsf{B}, \mathsf{ct_B}, \mathsf{ek_A})$
    $\mathsf{st_B} := (\mathsf{t_B}, \bot, \widehat{\mathsf{ek}}_\mathsf{A}, \mathsf{par})$
    $\textbf{return } (\mathsf{K}, \mathsf{st_B})$
```

Figure 14: A generic construction of a CKA from ratcheting KEM. Algorithms CKA-Send-B and CKA-Rec-A are defined analogously with the roles of parties A and B swapped, and the algorithms checking for the epoch number to be odd instead of even.

**Initial setup:** Instead of directly sampling $(\widehat{\mathsf{ek}}_0, \widehat{\mathsf{dk}}_0)$ using the RKeyGen-B algorithm, all hybrids first sample $(\mathsf{ek}_0, \mathsf{dk}_0)$ instead, and then use $\mathsf{RSimKey\text{-}B}_1$ to derive $(\widehat{\mathsf{ek}}_0, \widehat{\mathsf{dk}}_0)$. It is easy to see that this is indistinguishable by base-key simulatability, and more concretely there exists a simple reduction $\mathcal{B}_0$ such that

$$\left|\Pr\left[\mathsf{Game}^{\mathsf{CKA}}_{\mathcal{A},\mathsf{t}^*,b}(1^\lambda) = 1\right] - \Pr\left[\mathsf{Hybrid}^{\mathsf{CKA}}_{\mathcal{A},\mathsf{t}^*,b,0}(1^\lambda) = 1\right]\right| \leqslant \mathsf{Adv}^{\mathsf{KeyBaseSim\text{-}B}}_{\mathcal{B}}(1^\lambda).$$

**Sending and receiving:** For epochs $1 \leqslant \mathsf{t_P} \leqslant i$, we moreover change CKA-Send-P to use $\mathsf{RSimKey\text{-}P}_1$ and $\mathsf{RSimKey\text{-}P}_2$ to generate the key pair $(\widehat{\mathsf{ek}}_{\mathsf{t_P}}, \widehat{\mathsf{dk}}_{\mathsf{t_P}})$ as well as the ciphertext $\mathsf{ct}_{\mathsf{t_P}-1}$ and key $\mathsf{K}_{\mathsf{t_P}}$ for epoch $\mathsf{t_P}$. Note that the updated decyrption key $\widehat{\mathsf{dk}}_{\mathsf{t_P}}$ is already generated by the simulator and, thus, we skip RDec-P in CKA-Rec-P for those epochs. (Observe that defining the key earlier does not otherwise change the game's behavior, as it is only leaked as part of a corruption once the respective message has been leaked.)

Note that this behavior exactly corresponds to $\mathcal{D}^{\mathsf{KeyUpdSim}}_{\mathsf{P},1}$ while the regular protocol behavior exactly corresponds to $\mathcal{D}^{\mathsf{KeyUpdSim}}_{\mathsf{P},0}$. Therefore, there exists a simple reduction to updated-key simulatability, i.e.,

$$\left|\Pr\left[\mathsf{Game}^{\mathsf{CKA}}_{\mathcal{A},\mathsf{t}^*,b}(1^\lambda) = 1\right] - \Pr\left[\mathsf{Hybrid}^{\mathsf{CKA}}_{\mathcal{A},\mathsf{t}^*,b,0}(1^\lambda) = 1\right]\right| \leqslant \mathsf{Adv}^{\mathsf{KeyUpdSim\text{-}P}}_{\mathcal{C}}(1^\lambda),$$

where $\mathsf{P} = \mathsf{A}$ for odd $\mathsf{t_P}$ and $\mathsf{P} = \mathsf{B}$ for even $\mathsf{t_P}$.

$\mathsf{Hybrid}^{\mathsf{CKA}}_{\mathcal{A},\mathsf{t}^*,b,\mathsf{t}^*-1}$: Next, consider a hybrid depicted in Fig. 17 that changes how the keys for epoch $\mathsf{t}^* - 1$ are sampled. More concretely, it samples the key pair $(\widehat{\mathsf{ek}}_{\mathsf{t_P}}, \widehat{\mathsf{dk}}_{\mathsf{t_P}})$ freshly and then uses the simulator RSimCtxt-P

```
$\mathsf{Game}^{\mathsf{CKA}}_{\mathcal{A},\mathsf{t}^*}(1^\lambda)$
  $b \xleftarrow{\$} \{0,1\}$
  CKA-Init-KeyGen
    $\mathsf{par} \leftarrow \mathsf{RSetup}(1^\lambda)$
    $(\widehat{\mathsf{ek}}_0, \widehat{\mathsf{dk}}_0) \xleftarrow{\$} \mathsf{RKeyGen\text{-}B}(\mathsf{par}, \mathtt{updated})$
  for $\mathsf{P} \in \{\mathsf{A}, \mathsf{B}\}$
    $\mathsf{t_P} := 0$   // CKA-Init-P does nothing
  $b' \xleftarrow{\$} \mathcal{A}(\widehat{\mathsf{t}}^*)^{\mathsf{Send\text{-}P}(),\mathsf{Receive\text{-}P}(),\mathsf{Chall\text{-}P}(),\mathsf{Corr\text{-}P}()}$
  return $[\![b = b']\!]$
```

```
$\mathsf{Send\text{-}P}(\mathsf{rleak})$
  $\mathsf{t_P} \leftarrow \mathsf{t_P} + 1$
  $\mathsf{rand} := (\mathsf{rand}_1, \mathsf{rand}_2) \xleftarrow{\$} \mathcal{R}$
  CKA-Send-P
    $(\mathsf{ek}_{\mathsf{t_P}}, \mathsf{dk}_{\mathsf{t_P}}) \leftarrow \mathsf{RKeyGen\text{-}P}(\mathsf{par}; \mathsf{rand}_1)$
    $(\mathsf{ct}_{\mathsf{t_P}-1}, \mathsf{K}, \widehat{\mathsf{dk}}_{\mathsf{t_P}}) \leftarrow \mathsf{REnc\text{-}P}(\widehat{\mathsf{ek}}_{\mathsf{t_P}-1}, \mathsf{dk}_{\mathsf{t_P}}; \mathsf{rand}_2)$
    $\rho := (\mathsf{t_P}, \mathsf{ek}_{\mathsf{t_P}}, \mathsf{ct}_{\mathsf{t_P}-1})$
  if $[\![\mathsf{rleak}]\!]$ then   // Leak randomness
    // Allow leaking randomness $\Delta_{\mathsf{PCS}}$-epoch before $\mathsf{t}^*$
    req $[\![\mathsf{t_A}, \mathsf{t_B} \leqslant \mathsf{t}^* - \Delta_{\mathsf{PCS}}]\!]$
  else   // Secure randomness (for challenge epoch)
    $\mathsf{rand} \leftarrow \bot$
  $\mathsf{K}_{\mathsf{t_P}} \leftarrow \mathsf{K}$
  return $(\mathsf{K}, \rho, \mathsf{rand})$
```

```
$\mathsf{Chall\text{-}P}()$
  $\mathsf{t_P} \leftarrow \mathsf{t_P} + 1$
  req $[\![\mathsf{t_P} = \widehat{\mathsf{t}}^*]\!]$   // Challenge epoch $\mathsf{t}^*$
  CKA-Send-P
    $(\mathsf{ek}_{\mathsf{t_P}}, \mathsf{dk}_{\mathsf{t_P}}) \xleftarrow{\$} \mathsf{RKeyGen\text{-}P}(\mathsf{par})$
    $(\mathsf{ct}_{\mathsf{t_P}-1}, \mathsf{K}, \widehat{\mathsf{dk}}_{\mathsf{t_P}}) \xleftarrow{\$} \mathsf{REnc\text{-}P}(\widehat{\mathsf{ek}}_{\mathsf{t_P}-1}, \mathsf{dk}_{\mathsf{t_P}})$
    $\rho := (\mathsf{t_P}, \mathsf{ek}_{\mathsf{t_P}}, \mathsf{ct}_{\mathsf{t_P}-1})$
  $\mathsf{K}_{\mathsf{t_P}} \leftarrow \mathsf{K}$
  if $[\![b = 1]\!]$ then
    $\mathsf{K} \xleftarrow{\$} \mathcal{K}$   // Replace with random key
  return $(\mathsf{K}, \rho)$
```

```
$\mathsf{Receive\text{-}P}()$
  $\mathsf{t_P} \leftarrow \mathsf{t_P} + 1$
  CKA-Rec-P
    $(\mathsf{K}, \widehat{\mathsf{ek}}_{\mathsf{t_P}}) \xleftarrow{\$} \mathsf{RDec\text{-}P}(\widehat{\mathsf{dk}}_{\mathsf{t_P}-1}, \mathsf{ct}_{\mathsf{t_P}-1}, \mathsf{ek}_{\mathsf{t_P}})$
  assert $[\![\mathsf{K} = \mathsf{K}_{\mathsf{t_P}}]\!]$   // Correctness
```

```
$\mathsf{Corr\text{-}P}()$
  // Allow corrupting $\Delta_{\mathsf{PCS}}$-epoch before $\widehat{\mathsf{t}}^*$
  req $[\![\widehat{\mathsf{t}}_{\mathsf{A}}, \widehat{\mathsf{t}}_{\mathsf{B}} \leqslant \widehat{\mathsf{t}}^* - \Delta_{\mathsf{PCS}}]\!]$
  // Allow corrupting $\Delta_{\mathsf{PCS}}$-epoch after $\widehat{\mathsf{t}}^*$
  req $[\![\widehat{\mathsf{t}}_{\mathsf{P}} \geqslant \widehat{\mathsf{t}}^* + \Delta_{\mathsf{FS}}]\!]$
  Protocol state
    if P is sender in $\mathsf{t_P}$ then
      $\mathsf{st_P} := (\mathsf{t_P}, \widehat{\mathsf{dk}}_{\mathsf{t_P}}, \bot, \mathsf{par})$
    else
      $\mathsf{st_P} := (\mathsf{t_P}, \bot, \widehat{\mathsf{ek}}_{\mathsf{t_P}}, \mathsf{par})$
  return $\mathsf{st_P}$
```

Figure 15: The CKA security game with our specific RKEM based protocol inlined for clarity. Some trivial simplifications have been applied, such as unifying the epoch counters shared between the game and the protocol, and storing the individual components of CKA messages and states to avoid repeated parsing.

```
$\mathsf{Hybrid}^{\mathsf{CKA}}_{\mathcal{A},\mathsf{t}^*,b,i}(1^\lambda)$
    CKA-Init-KeyGen
        $\mathsf{par} \xleftarrow{\$} \mathsf{RSetup}(1^\lambda)$
        $(\mathsf{ek}_0, \mathsf{dk}_0) \xleftarrow{\$} \mathcal{D}_{\mathsf{RKeyGen\text{-}B}}(\mathsf{par})$
        $(\widehat{\mathsf{ek}}_0, \widehat{\mathsf{dk}}_0, \cdot) \xleftarrow{\$} \mathsf{RSim\text{-}KeyB}_1(\mathsf{ek}_0, \mathsf{dk}_0)$
    for $\mathsf{P} \in \{\mathsf{A}, \mathsf{B}\}$
        $\hat{\mathsf{t}}_\mathsf{P} := 0$    // CKA-Init-P does nothing
    $b' \xleftarrow{\$} \mathcal{A}(\hat{\mathsf{t}}^*)^{\mathsf{Send\text{-}P}(),\mathsf{Receive\text{-}P}(),\mathsf{Chall\text{-}P}(),\mathsf{Corr\text{-}P}()}$
    return b'
```

```
$\mathsf{Send\text{-}P}(\mathsf{rleak})$
    $\mathsf{t}_\mathsf{P} \leftarrow \mathsf{t}_\mathsf{P} + 1$
    $\mathsf{rand} := (\mathsf{rand}_1, \mathsf{rand}_2) \xleftarrow{\$} \mathcal{R}$
    CKA-Send-P
        $(\mathsf{ek}_{\mathsf{t}_\mathsf{P}}, \mathsf{dk}_{\mathsf{t}_\mathsf{P}}) \xleftarrow{\$} \mathcal{D}_{\mathsf{RKeyGen\text{-}P}}(\mathsf{par}; \mathsf{rand}_1)$
        if $[\![\mathsf{t}_\mathsf{P} \leqslant i]\!]$ then
            $(\widehat{\mathsf{ek}}_{\mathsf{t}_\mathsf{P}}, \widehat{\mathsf{dk}}_{\mathsf{t}_\mathsf{P}}, \mathsf{aux}) \xleftarrow{\$} \mathsf{RSim\text{-}KeyP}_1(\mathsf{ek}_{\mathsf{t}_\mathsf{P}}, \mathsf{dk}_{\mathsf{t}_\mathsf{P}})$
            $(\mathsf{ct}_{\mathsf{t}_\mathsf{P}-1}, \mathsf{K}, \mathsf{K}'_{\mathsf{t}_\mathsf{P}}, \mathsf{rand}_2)$
                $\xleftarrow{\$} \mathsf{RSim\text{-}KeyP}_2(\widehat{\mathsf{ek}}_{\mathsf{t}_\mathsf{P}-1}, \widehat{\mathsf{dk}}_{\mathsf{t}_\mathsf{P}-1}, \mathsf{aux})$
        else
            $(\mathsf{ct}_{\mathsf{t}_\mathsf{P}-1}, \mathsf{K}, \widehat{\mathsf{dk}}_{\mathsf{t}_\mathsf{P}}) \leftarrow \mathsf{REnc\text{-}P}(\widehat{\mathsf{ek}}_{\mathsf{t}_\mathsf{P}-1}, \mathsf{dk}_{\mathsf{t}_\mathsf{P}}; \mathsf{rand}_2)$
        $\rho := (\mathsf{t}_\mathsf{P}, \mathsf{ek}_{\mathsf{t}_\mathsf{P}}, \mathsf{ct}_{\mathsf{t}_\mathsf{P}-1})$
    if $[\![\mathsf{rleak}]\!]$ then    // Leak randomness
        // Allow leaking randomness $\Delta_{\mathsf{PCS}}$-epoch *before* $\mathsf{t}^*$
        req $[\![\mathsf{t}_\mathsf{A}, \mathsf{t}_\mathsf{B} \leqslant \mathsf{t}^* - \Delta_{\mathsf{PCS}}]\!]$
    else    // Secure randomness (for challenge epoch)
        $\mathsf{rand} \leftarrow \perp$
    $\mathsf{K}_{\mathsf{t}_\mathsf{P}} \leftarrow \mathsf{K}$
    return $(\mathsf{K}, \rho, \mathsf{rand})$
```

```
$\mathsf{Chall\text{-}P}()$
    $\mathsf{t}_\mathsf{P} \leftarrow \mathsf{t}_\mathsf{P} + 1$
    req $[\![\mathsf{t}_\mathsf{P} = \hat{\mathsf{t}}^*]\!]$    // Challenge epoch $\mathsf{t}^*$
    CKA-Send-P
        $(\mathsf{ek}_{\mathsf{t}_\mathsf{P}}, \mathsf{dk}_{\mathsf{t}_\mathsf{P}}) \xleftarrow{\$} \mathsf{RKeyGen\text{-}P}(\mathsf{par})$
        $(\mathsf{ct}_{\mathsf{t}_\mathsf{P}-1}, \mathsf{K}, \widehat{\mathsf{dk}}_{\mathsf{t}_\mathsf{P}}) \xleftarrow{\$} \mathsf{REnc\text{-}P}(\widehat{\mathsf{ek}}_{\mathsf{t}_\mathsf{P}-1}, \mathsf{dk}_{\mathsf{t}_\mathsf{P}})$
        $\rho := (\mathsf{t}_\mathsf{P}, \mathsf{ek}_{\mathsf{t}_\mathsf{P}}, \mathsf{ct}_{\mathsf{t}_\mathsf{P}-1})$
    $\mathsf{K}_{\mathsf{t}_\mathsf{P}} \leftarrow \mathsf{K}$
    if $[\![b = 1]\!]$ then
        $\mathsf{K} \xleftarrow{\$} \mathcal{K}$    // Replace with random key
    return $(\mathsf{K}, \rho)$
```

```
$\mathsf{Receive\text{-}P}()$
    $\mathsf{t}_\mathsf{P} \leftarrow \mathsf{t}_\mathsf{P} + 1$
    CKA-Rec-P
        if $[\![\mathsf{t}_\mathsf{P} > i]\!]$ then
            $(\mathsf{K}, \widehat{\mathsf{ek}}_{\mathsf{t}_\mathsf{P}}) \xleftarrow{\$} \mathsf{RDec\text{-}P}(\widehat{\mathsf{dk}}_{\mathsf{t}_\mathsf{P}-1}, \mathsf{ct}_{\mathsf{t}_\mathsf{P}-1}, \mathsf{ek}_{\mathsf{t}_\mathsf{P}})$
```

```
$\mathsf{Corr\text{-}P}()$
    // Allow corrupting $\Delta_{\mathsf{PCS}}$-epoch *before* $\hat{\mathsf{t}}^*$
    req $[\![\hat{\mathsf{t}}_\mathsf{A}, \hat{\mathsf{t}}_\mathsf{B} \leqslant \hat{\mathsf{t}}^* - \Delta_{\mathsf{PCS}}]\!]$
    // Allow corrupting $\Delta_{\mathsf{PCS}}$-epoch *after* $\hat{\mathsf{t}}^*$
    req $[\![\hat{\mathsf{t}}_\mathsf{P} \geqslant \hat{\mathsf{t}}^* + \Delta_{\mathsf{FS}}]\!]$
    Protocol state
        if P is sender in $\mathsf{t}_\mathsf{P}$ then
            $\mathsf{st}_\mathsf{P} := (\mathsf{t}_\mathsf{P}, \widehat{\mathsf{dk}}_{\mathsf{t}_\mathsf{P}}, \perp, \mathsf{par})$
        else
            $\mathsf{st}_\mathsf{P} := (\mathsf{t}_\mathsf{P}, \perp, \widehat{\mathsf{ek}}_{\mathsf{t}_\mathsf{P}}, \mathsf{par})$
    return $\mathsf{st}_\mathsf{P}$
```

Figure 16: A sequence of hybrid games for $0 \leqslant i \leqslant \mathsf{t}^* - 2$. Changes with respect to Fig. 15 are highlighted.

```
$\mathsf{Hybrid}^{\mathsf{CKA}}_{\mathcal{A},\mathsf{t}^*,b,\mathsf{t}^*-1}(1^\lambda)$
    CKA-Init-KeyGen
        $\mathsf{par} \xleftarrow{\$} \mathsf{RSetup}(1^\lambda)$
        $(\mathsf{ek}_0, \mathsf{dk}_0) \xleftarrow{\$} \mathcal{D}_{\mathsf{RKeyGen\text{-}B}}(\mathsf{par})$
        $(\widehat{\mathsf{ek}}_0, \widehat{\mathsf{dk}}_0, \cdot\,) \xleftarrow{\$} \mathsf{RSim\text{-}KeyB}_1(\mathsf{ek}_0, \mathsf{dk}_0)$
    for $\mathsf{P} \in \{\mathsf{A}, \mathsf{B}\}$
        $\hat{\mathsf{t}}_{\mathsf{P}} := 0$    // CKA-Init-P does nothing
    $b' \xleftarrow{\$} \mathcal{A}(\hat{\mathsf{t}}^*)^{\mathsf{Send\text{-}P}(),\mathsf{Receive\text{-}P}(),\mathsf{Chall\text{-}P}(),\mathsf{Corr\text{-}P}()}$
    return $[\![b = b']\!]$
```

```
Send-P(rleak)
    $\mathsf{t}_{\mathsf{P}} \leftarrow \mathsf{t}_{\mathsf{P}} + 1$
    $\mathsf{rand} := (\mathsf{rand}_1, \mathsf{rand}_2) \xleftarrow{\$} \mathcal{R}$
    CKA-Send-P
        if $[\![\mathsf{t}_{\mathsf{P}} \leqslant \mathsf{t}^* - 2]\!]$ then
            $(\mathsf{ek}_{\mathsf{t_P}}, \mathsf{dk}_{\mathsf{t_P}}) \xleftarrow{\$} \mathcal{D}_{\mathsf{RKeyGen\text{-}P}}(\mathsf{par}; \mathsf{rand}_1)$
            $(\widehat{\mathsf{ek}}_{\mathsf{t_P}}, \widehat{\mathsf{dk}}_{\mathsf{t_P}}, \mathsf{aux}) \xleftarrow{\$} \mathsf{RSimKey\text{-}P}_1(\mathsf{ek}_{\mathsf{t_P}}, \mathsf{dk}_{\mathsf{t_P}})$
            $(\mathsf{ct}_{\mathsf{t_P}-1}, \mathsf{K}, \mathsf{K}'_{\mathsf{t_P}}, \mathsf{rand}_2)$
                $\xleftarrow{\$} \mathsf{RSimKey\text{-}P}_2(\widehat{\mathsf{ek}}_{\mathsf{t_P}-1}, \widehat{\mathsf{dk}}_{\mathsf{t_P}-1}, \mathsf{aux})$
        elseif $[\![\mathsf{t}_{\mathsf{P}} = \mathsf{t}^* - 1]\!]$ then
            $(\widehat{\mathsf{ek}}_{\mathsf{t_P}}, \widehat{\mathsf{dk}}_{\mathsf{t_P}}) \xleftarrow{\$} \widehat{\mathcal{D}}_{\mathsf{RKeyGen\text{-}P}}(\mathsf{par})$
            $(\mathsf{ct}_{\mathsf{t_P}-1}, \mathsf{ek}_{\mathsf{t_P}}, \mathsf{K}, \mathsf{K}'_{\mathsf{t_P}}) \xleftarrow{\$} \mathsf{RSimCtxt\text{-}P}(\widehat{\mathsf{ek}}_{\mathsf{t_P}}, \widehat{\mathsf{ek}}_{\mathsf{t_P}-1}, \widehat{\mathsf{dk}}_{\mathsf{t_P}-1})$
        else
            $(\mathsf{ek}_{\mathsf{t_P}}, \mathsf{dk}_{\mathsf{t_P}}) \xleftarrow{\$} \mathcal{D}_{\mathsf{RKeyGen\text{-}P}}(\ ; \mathsf{rand}_1)$
            $(\mathsf{ct}_{\mathsf{t_P}-1}, \mathsf{K}, \widehat{\mathsf{dk}}_{\mathsf{t_P}}) \leftarrow \mathsf{REnc\text{-}P}(\widehat{\mathsf{ek}}_{\mathsf{t_P}-1}, \mathsf{dk}_{\mathsf{t_P}}; \mathsf{rand}_2)$
        $\rho := (\mathsf{t_P}, \mathsf{ek}_{\mathsf{t_P}}, \mathsf{ct}_{\mathsf{t_P}-1})$
    if $[\![\mathsf{rleak}]\!]$ then    // Leak randomness
        // Allow leaking randomness $\Delta_{\mathsf{PCS}}$-epoch before $\mathsf{t}^*$
        req $[\![\mathsf{t_A}, \mathsf{t_B} \leqslant \mathsf{t}^* - \Delta_{\mathsf{PCS}}]\!]$
    else    // Secure randomness (for challenge epoch)
        $\mathsf{rand} \leftarrow \bot$
    $\mathsf{K}_{\mathsf{t_P}} \leftarrow \mathsf{K}$
    return $(\mathsf{K}, \rho, \mathsf{rand})$
```

```
Chall-P()
    $\mathsf{t_P} \leftarrow \mathsf{t_P} + 1$
    req $[\![\mathsf{t_P} = \hat{\mathsf{t}}^*]\!]$    // Challenge epoch $\mathsf{t}^*$
    CKA-Send-P
        $(\mathsf{ek}_{\mathsf{t_P}}, \mathsf{dk}_{\mathsf{t_P}}) \xleftarrow{\$} \mathsf{RKeyGen\text{-}P}(\mathsf{par})$
        $(\mathsf{ct}_{\mathsf{t_P}-1}, \mathsf{K}, \widehat{\mathsf{dk}}_{\mathsf{t_P}}) \xleftarrow{\$} \mathsf{REnc\text{-}P}(\widehat{\mathsf{ek}}_{\mathsf{t_P}-1}, \mathsf{dk}_{\mathsf{t_P}})$
        $\rho := (\mathsf{t_P}, \mathsf{ek}_{\mathsf{t_P}}, \mathsf{ct}_{\mathsf{t_P}-1})$
    $\mathsf{K}_{\mathsf{t_P}} \leftarrow \mathsf{K}$
    if $[\![b = 1]\!]$ then
        $\mathsf{K} \xleftarrow{\$} \mathcal{K}$    // Replace with random key
    return $(\mathsf{K}, \rho)$
```

```
Receive-P()
    $\mathsf{t_P} \leftarrow \mathsf{t_P} + 1$
    CKA-Rec-P
        if $[\![\mathsf{t_P} > \mathsf{t}^* - 1]\!]$ then
            $(\mathsf{K}, \widehat{\mathsf{ek}}_{\mathsf{t_P}}) \xleftarrow{\$} \mathsf{RDec\text{-}P}(\widehat{\mathsf{dk}}_{\mathsf{t_P}-1}, \mathsf{ct}_{\mathsf{t_P}-1}, \mathsf{ek}_{\mathsf{t_P}})$
```

```
Corr-P()
    // Allow corrupting $\Delta_{\mathsf{PCS}}$-epoch before $\hat{\mathsf{t}}^*$
    req $[\![\hat{\mathsf{t}}_{\mathsf{A}}, \hat{\mathsf{t}}_{\mathsf{B}} \leqslant \hat{\mathsf{t}}^* - \Delta_{\mathsf{PCS}}]\!]$
    // Allow corrupting $\Delta_{\mathsf{PCS}}$-epoch after $\hat{\mathsf{t}}^*$
    req $[\![\hat{\mathsf{t}}_{\mathsf{P}} \geqslant \hat{\mathsf{t}}^* + \Delta_{\mathsf{FS}}]\!]$
    Protocol state
        if P is sender in $\mathsf{t_P}$ then
            $\mathsf{st_P} := (\mathsf{t_P}, \widehat{\mathsf{dk}}_{\mathsf{t_P}}, \bot, \mathsf{par})$
        else
            $\mathsf{st_P} := (\mathsf{t_P}, \bot, \widehat{\mathsf{ek}}_{\mathsf{t_P}}, \mathsf{par})$
    return $\mathsf{st_P}$
```

Figure 17: An additional hybrid game. Changes with respect to $\mathsf{Hybrid}^{\mathsf{CKA}}_{\mathcal{A},\mathsf{t}^*,b,\mathsf{t}^*-2}$ are highlighted.

to simulate $\mathsf{ek}_{\mathsf{t_P}}$, the key $\mathsf{K}_{\mathsf{t_P}}$ and the ciphertext $\mathsf{ct}_{\mathsf{t_P}-1}$. (The private key $\mathsf{dk}_{\mathsf{t_P}}$ is not needed by the hybrid.) In addition, we also emit the decryption for epoch $\mathsf{t}^* - 1$ as $\widehat{\mathsf{ek}}_{\mathsf{t_P}}$ has already been produced. Observe that this matches the sampling strategy of $\mathcal{D}^{\mathsf{CtxtSim}}_{\mathsf{P},1}$, while the old strategy of $\mathsf{Hybrid}^{\mathsf{CKA}}_{\mathcal{A},\mathsf{t}^*,b,\mathsf{t}^*-2}$ matches $\mathcal{D}^{\mathsf{CtxtSim}}_{\mathsf{P},0}$.

Therefore, we obtain

$$\left|\Pr\left[\mathsf{Hybrid}^{\mathsf{CKA}}_{\mathcal{A},\mathsf{t}*,b,\mathsf{t}*-2}(1^\lambda) = 1\right] - \Pr\left[\mathsf{Hybrid}^{\mathsf{CKA}}_{\mathcal{A},\mathsf{t}*,b,\mathsf{t}*-1}(1^\lambda) = 1\right]\right| \leqslant \mathsf{Adv}^{\mathsf{CtxtSim\text{-}P}}_{\mathcal{D}}(1^\lambda),$$

for an appropriate reduction $\mathcal{D}$.

**Embedding the challenge:** In $\mathsf{Hybrid}^{\mathsf{CKA}}_{\mathcal{A},\mathsf{t}*,b,\mathsf{t}*-1}$, we now switch from $b = 0$ to $b = 1$ based on FS-IND-CPA security of the RKEM. Observe the following:

- $(\widehat{\mathsf{ek}}_{\mathsf{t}*-1}, \widehat{\mathsf{dk}}_{\mathsf{t}*-1})$ is a fresh key pair drawn from the same distribution the key generation algorithm produces. Moreover, with $\Delta_{\mathsf{PCS}} = 2$, the adversary is not allowed to leak the key pair's randomness.
- The only place $\widehat{\mathsf{dk}}_{\mathsf{t}*-1}$ is used in the game is in CKA-Rec-P to update $\mathsf{ek}_{\mathsf{t}*}$ to $\widehat{\mathsf{ek}}_{\mathsf{t}*}$.
- The only place $\widehat{\mathsf{ek}}_{\mathsf{t}*-1}$ is used is in Chall-P where the challenge is encrypted under this key, and a real-or-random key is returned based on the bit $b$.

This directly corresponds to FS-IND-CPA security of the RKEM. Thus, we obtain

$$\left|\Pr\left[\mathsf{Hybrid}^{\mathsf{CKA}}_{\mathcal{A},\mathsf{t}*,b=0,\mathsf{t}*-1}(1^\lambda) = 1\right] - \Pr\left[\mathsf{Hybrid}^{\mathsf{CKA}}_{\mathcal{A},\mathsf{t}*,b=1,\mathsf{t}*-1}(1^\lambda) = 1\right]\right| \leqslant \mathsf{Adv}^{\mathsf{FS\text{-}IND\text{-}CPA}}_{\mathcal{E}}(1^\lambda).$$

Note that if the RKEM is non-forward secure, then $\widehat{\mathsf{dk}}_{\mathsf{t}*} = \mathsf{dk}_{\mathsf{t}*}$ cannot be leaked as $\Delta_{\mathsf{FS}} = 1$. Moreover, the reduction does not need to consider the use $\widehat{\mathsf{dk}}_{\mathsf{t}*}$ to update the next public key. Therefore, there is no need for the reduction to know $\mathsf{dk}_{\mathsf{t}*}$ to simulate the further protocol execution, and the reduction to $\mathsf{Adv}^{\mathsf{IND\text{-}CPA}}$ works analogously.

**Putting it all together:** By fixing the bit $b$ in the CKA game and taking $b'$ as its output — technically the version without the correctness condition — we can rewrite the advantage as

$$\mathsf{Adv}^{\mathsf{CKA}}_{\mathcal{A}}(1^\lambda) = \left|\Pr\left[\mathsf{Game}^{\mathsf{CKA}}_{\mathcal{A},\mathsf{t}*,b=0}(1^\lambda) = 1\right] - \Pr\left[\mathsf{Game}^{\mathsf{CKA}}_{\mathcal{A},\mathsf{t}*,b=1}(1^\lambda) = 1\right]\right|$$

Using the sequence of hybrids

$$\begin{aligned}\mathsf{Game}^{\mathsf{CKA}}_{\mathcal{A},\mathsf{t}*,b=0} &\to \mathsf{Hybrid}^{\mathsf{CKA}}_{\mathcal{A},\mathsf{t}*,b=0,0} \to \ldots \to \mathsf{Hybrid}^{\mathsf{CKA}}_{\mathcal{A},\mathsf{t}*,b=0,\mathsf{t}*-1} \\ &\to \mathsf{Hybrid}^{\mathsf{CKA}}_{\mathcal{A},\mathsf{t}*,b=1,\mathsf{t}*-1} \to \ldots \to \mathsf{Hybrid}^{\mathsf{CKA}}_{\mathcal{A},\mathsf{t}*,b=1,0} \to \mathsf{Game}^{\mathsf{CKA}}_{\mathcal{A},\mathsf{t}*,b=1}\end{aligned}$$

then yields the desired bound. □

# 6 Katana: An Efficient Ratcheting KEM from Lattices

In this section, we construct a ratcheting KEM (RKEM) from lattices which we call Katana. As with typical practice-oriented lattice-based constructions, we first analyze our construction based on asymptotic bounds and later set concrete parameters based on cryptanalysis.

## 6.1 Construction of Katana

The notations used in this section is summarized in Table 2. H is a function which on input $(\mathbf{u}, \mathsf{seed}) \in R_q^k \times \{0,1\}^\lambda$, outputs a tuple $(\mathsf{K}, \mathbf{s}, \mathbf{e})$ distributed over $\{0,1\}^\lambda \times \chi \times \chi$. This function is modeled as a random oracle in the security proof. In practice, H can output randomness used to sample from the target distributions. Moreover, let $\mathsf{Encode} : \{0,1\}^\lambda \to \mathcal{R}_q$ be a function that maps $\mathsf{seed} \in \{0,1\}^\lambda \subset \mathcal{R}_q$ to $\lfloor q/2 \rceil \cdot \mathsf{seed}$, where seed is viewed as a degree $\lambda - 1$ polynomial in $\mathcal{R}_q$ with binary coefficients. Let $\mathsf{Decode} : \mathcal{R}_q \to \{0,1\}^\lambda$ be a function that maps each coefficient $w \in \mathcal{R}_q$ to 0 (resp. 1) if it is close to 0 (resp. $\lfloor q/2 \rceil$) in absolute value.

Katana is based on the (IND-CPA secure) KEM by Lyubashevsky et al. [LPR10] and Lindner and Peikert [LP11], underlying the ML-KEM NIST standard (i.e., Kyber) [SAB+22]. The construction is given in Fig. 18. For simplicity, we first provide the simplified variant where we do not perform bit-dropping. The optimized variant is given in Section 6.3.

```
$\mathsf{RKeyGen\text{-}P}(\mathsf{par}, \mathsf{mode})$
if $[\![\mathsf{mode} = \perp]\!]$
    $(\mathbf{s}_\mathsf{P}, \mathbf{e}_\mathsf{P}) \xleftarrow{\$} \chi \times \chi$
else    // $\mathsf{mode} = \mathsf{updated}$
    $(\mathbf{s}_\mathsf{P}, \mathbf{e}_\mathsf{P}) \xleftarrow{\$} \widehat{\chi} \times \widehat{\chi}$
if $[\![\mathsf{P} = \mathsf{A}]\!]$ then
    $\mathbf{u}_\mathsf{A} := \mathbf{D} \cdot \mathbf{s}_\mathsf{A} + \mathbf{e}_\mathsf{A} \in R_q^k$
else    // $\mathsf{P} = \mathsf{B}$
    $\mathbf{u}_\mathsf{B} := \mathbf{D}^\top \cdot \mathbf{s}_\mathsf{B} + \mathbf{e}_\mathsf{B} \in R_q^k$
if $[\![\mathsf{mode} = \perp]\!]$
    $(\mathsf{ek}_\mathsf{P}, \mathsf{dk}_\mathsf{P}) := (\mathbf{u}_\mathsf{P}, (\mathbf{u}_\mathsf{P}, \mathbf{s}_\mathsf{P}))$
else    // $\mathsf{mode} = \mathsf{updated}$
    $(\mathsf{ek}_\mathsf{P}, \mathsf{dk}_\mathsf{P}) := (\mathbf{u}_\mathsf{P}, \mathbf{s}_\mathsf{P})$
return $(\mathsf{ek}_\mathsf{P}, \mathsf{dk}_\mathsf{P})$
```

```
$\mathsf{REnc\text{-}P}(\widehat{\mathsf{ek}}_{\bar{\mathsf{P}}}, \mathsf{dk}_\mathsf{P})$
$(\mathbf{u}_\mathsf{P}, \mathbf{s}_\mathsf{P}) := \mathsf{dk}_\mathsf{P}$
$\mathsf{seed} \xleftarrow{\$} \{0,1\}^\lambda$
$m \leftarrow \mathsf{Encode}(\mathsf{seed})$    // $m \in R_q$
$(\mathsf{K}, \mathbf{s}, \mathbf{e}) := \mathsf{H}(\mathbf{u}_\mathsf{P}, \mathsf{seed})$
$\tilde{e}_\mathsf{P} \xleftarrow{\$} \tilde{\chi}$
$v_{\bar{\mathsf{P}}} := \widehat{\mathsf{ek}}_{\bar{\mathsf{P}}}^\top \cdot \mathbf{s}_\mathsf{P} + \tilde{e}_\mathsf{P} + m \in R_q$
$\mathsf{ct}_{\bar{\mathsf{P}}} := v_{\bar{\mathsf{P}}}$
    // Update and erase $\mathsf{dk}_\mathsf{P}$
$\widehat{\mathsf{dk}}_\mathsf{P} := \mathbf{s}_\mathsf{P} + \mathbf{s} \in R_q^k$
return $(\mathsf{ct}_{\bar{\mathsf{P}}}, \mathsf{K}, \widehat{\mathsf{dk}}_\mathsf{P})$
```

```
$\mathsf{RDec\text{-}P}(\widehat{\mathsf{dk}}_\mathsf{P}, \mathsf{ct}_\mathsf{P}, \mathsf{ek}_{\bar{\mathsf{P}}})$
$m := \mathsf{ct}_\mathsf{P} - \mathsf{ek}_{\bar{\mathsf{P}}}^\top \cdot \widehat{\mathsf{dk}}_\mathsf{P}$
$\mathsf{seed} := \mathsf{Decode}(m)$
$(\mathsf{K}, \mathbf{s}, \mathbf{e}) := \mathsf{H}(\mathsf{ek}_{\bar{\mathsf{P}}}, \mathsf{seed})$
    // Update $\mathsf{ek}_{\bar{\mathsf{P}}}$
if $[\![\mathsf{P} = \mathsf{A}]\!]$ then
    $\widehat{\mathsf{ek}}_\mathsf{B} := \mathsf{ek}_\mathsf{B} + \mathbf{D}^\top \cdot \mathbf{s} + \mathbf{e}$
else    // $\mathsf{P} = \mathsf{B}$
    $\widehat{\mathsf{ek}}_\mathsf{A} := \mathsf{ek}_\mathsf{A} + \mathbf{D} \cdot \mathbf{s} + \mathbf{e}$
return $(\mathsf{K}, \widehat{\mathsf{ek}}_{\bar{\mathsf{P}}})$

$\mathsf{RSetup}(1^\lambda)$
$\mathsf{par} := \mathbf{D} \xleftarrow{\$} R_q^{k \times k}$
return $\mathsf{par}$
```

Figure 18: Katana without the bit-dropping optimization. Above, $(\mathsf{P}, \bar{\mathsf{P}}) = (\mathsf{A}, \mathsf{B})$ or $(\mathsf{B}, \mathsf{A})$.

**Correctness.** Correctness can be shown through a standard check on the size of the decapsulation noise. While it is easy to show that the assumption required for the correctness holds for specific distributions of $\chi, \widehat{\chi}$, and $\tilde{\chi}$ (e.g., discrete Gaussian distributions), we leave it general to allow any distribution. See Sections 6.1 and 6.4 for more detail.

**Lemma 6.1 (Correctness).** *Our* RKEM Katana *is correct assuming*

$$\Pr\left[\|\widehat{\mathbf{s}}^\top \cdot \mathbf{e} - \widehat{\mathbf{e}}^\top \cdot \mathbf{s} + \tilde{e}\|_\infty \leqslant q/4\right] = 1 - \mathsf{negl}(\lambda),$$

*where the probability is taken over the randomness to sample* $(\mathbf{s}, \mathbf{e}) \xleftarrow{\$} \chi \times \chi, (\widehat{\mathbf{s}}, \widehat{\mathbf{e}}) \xleftarrow{\$} \widehat{\chi} \times \widehat{\chi}$*, and* $\tilde{e} \xleftarrow{\$} \tilde{\chi}$*.*

*Proof.* Recalling that $\widehat{\chi}$ is defined as $[2] \cdot \chi$ (i.e., convolution of two independent copies of $\chi$), correctness of update key distribution is immediate. Let us show correctness with updated keys. Due to symmetry, we only focus on the case where user A runs RDec-A. Namely, we have the following

$$\begin{aligned}\mathsf{ct}_\mathsf{A} - \mathsf{ek}_\mathsf{B}^\top \cdot \widehat{\mathsf{dk}}_\mathsf{A} &= \widehat{\mathsf{ek}}_\mathsf{A}^\top \cdot \mathbf{s}_\mathsf{B} + \tilde{e}_\mathsf{B} + m - \mathsf{ek}_\mathsf{B}^\top \cdot \widehat{\mathsf{dk}}_\mathsf{A} \\ &= (\mathbf{D}\widehat{\mathbf{s}}_\mathsf{A} + \widehat{\mathbf{e}}_\mathsf{A})^\top \cdot \mathbf{s}_\mathsf{B} + \tilde{e}_\mathsf{B} + m - (\mathbf{D}^\top \mathbf{s}_\mathsf{B} + \mathbf{e}_\mathsf{B})^\top \cdot \widehat{\mathbf{s}}_\mathsf{A}\end{aligned}$$

| Notations | Explanation |
|---|---|
| $R_q$ | Polynomial ring $R_q = \mathbb{Z}[X]/(q, X^n + 1)$ with $n \geqslant \lambda$ |
| $k$ | Dimension of public matrix $\mathbf{D} \in R_q^{k \times k}$ |
| $\chi, \tilde{\chi}$ | Distributions for secrets and noises in ek and ct |
| $\widehat{\chi}$ | Distribution for "updated" secrets: $\widehat{\chi} := [2] \cdot \chi$ |
| H | A function $\mathsf{H} : R_q^k \times \{0,1\}^\lambda \to \{0,1\}^\lambda \times R_q^k$ modeled as a RO. |
| Encode, Decode | Encoding and decoding elements in $\{0,1\}^\lambda$ to $R_q$ |

Table 2: Overview of the notations. See the accompanying text for more details. Recall $[N] \cdot D$ is the convolution of $N$ independent copies of $D$.

$$= m + \underbrace{\widehat{\mathbf{e}}_{\mathsf{A}}^{\top} \cdot \mathbf{s}_{\mathsf{B}} - \mathbf{e}_{\mathsf{B}}^{\top} \cdot \widehat{\mathbf{s}}_{\mathsf{A}} + \tilde{e}_{\mathsf{B}}}_{=:z},$$

where $(\widehat{\mathbf{s}}_{\mathsf{A}}, \widehat{\mathbf{e}}_{\mathsf{A}}) \xleftarrow{\$} \widehat{\chi} \times \widehat{\chi}$ and $(\mathbf{s}_{\mathsf{B}}, \mathbf{e}_{\mathsf{B}}, \tilde{e}_{\mathsf{B}}) \xleftarrow{\$} \chi \times \chi \times \tilde{\chi}$. If each coefficient of $z \in R_q$ is smaller than $q/4$ (i.e., $\|z\|_{\infty} \leqslant q/4$), Decode will correctly decode to $m$ as desired. □

## 6.2 Security of Katana

Below, we prove that Katana is FS-IND-CPA secure and ratchet simulatable.

### 6.2.1 FS-IND-CPA Security.

The following theorem establishes the FS-IND-CPA security of Katana.

**Theorem 6.2 (FS-IND-CPA security).** *Our* RKEM Katana *is* FS-IND-CPA *secure assuming the hardness of the* MLWE *and the* hint-MLWE *assumptions.*

*Formally, for any adversary $\mathcal{A}$ against the* FS-IND-CPA *security making at most $Q$ queries to the random oracle* H*, there exists adversary $\mathcal{B}_{\mathsf{MLWE}}$ against the* $\mathsf{MLWE}_{q,k,\chi}$ *problem and adversaries $\mathcal{B}_{\mathsf{hint\text{-}MLWE},1}$ and $\mathcal{B}_{\mathsf{hint\text{-}MLWE},2}$ against the* $\mathsf{hint\text{-}MLWE}_{q,k,2k,\chi,\chi,\mathcal{F}_{\mathsf{cpa}}}$ *problem with $\mathcal{F}_{\mathsf{cpa}} := \mathcal{U}(\{\mathbf{I}_{2k \times 2k}\})$ such that*

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{FS\text{-}IND\text{-}CPA\text{-}A}}(1^{\lambda}) \leqslant \mathsf{Adv}_{\mathcal{B}_{\mathsf{MLWE}}}^{\mathsf{MLWE}}(1^{\lambda}) + \mathsf{Adv}_{\mathcal{B}_{\mathsf{hint\text{-}MLWE},1}}^{\mathsf{hint\text{-}MLWE}}(1^{\lambda}) \\ + \ 2 \cdot \mathsf{Adv}_{\mathcal{B}_{\mathsf{hint\text{-}MLWE},2}}^{\mathsf{hint\text{-}MLWE}}(1^{\lambda}) + \epsilon_{\mathsf{corr}} + \frac{Q}{2^{\lambda-1}},$$

*where $\epsilon_{\mathsf{corr}}$ is the probability that correctness with updated keys fails (cf. Definition 5.3).*

*Proof.* Due to the symmetry of users A and B, we only focus on bounding the advantage $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{FS\text{-}IND\text{-}CPA\text{-}A}}(1^{\lambda})$ (cf. Definition 5.4). The theorem is proven in a sequence of hybrid games given in Figs. 19 and 20. The first $\mathsf{Game}_0$ is the real FS-IND-CPA security game, where $\mathsf{Game}_6$ is a game in which even an unbounded adversary has negligible advantage. Our proof consists of bounding the advantage of an adversary $\mathcal{A}$ of the adjacent games. Below, $\epsilon_i$ denotes the advantage of $\mathcal{A}$ in $\mathsf{Game}_i$ and $Q$ denotes the number of random oracle queries performed by $\mathcal{A}$.

$\mathsf{Game}_0$: This is the real FS-IND-CPA security game. For reference, in Fig. 19, we provide the full details of the game.

$\mathsf{Game}_1$: In this game, the challenger reuses $\mathsf{K}, \mathbf{s}, \mathbf{e}$ from algorithm REnc-A as opposed to generating them through executing RDec-B. This follows from the same argument made to prove correctness: $m$ used during REnc-A and $m'$ generated during RDec-B are the same with all but a negligible probability. Hence, we have

$$|\epsilon_0 - \epsilon_1| \leqslant \epsilon_{\mathsf{corr}},$$

where $\epsilon_{\mathsf{corr}}$ is the probability that correctness with updated keys fails (cf. Definition 5.3).

$\mathsf{Game}_2$: In this game, the challenger samples a random $\widehat{\mathbf{u}}_{\mathsf{B}}$ from $\mathcal{R}_q^k$ as opposed to generating them as an MLWE instance. It is straight forward to see that the $\mathsf{Game}_2$ is indistinguishable from $\mathsf{Game}_1$ under the MLWE assumption. Formally, we can construct an adversary $\mathcal{B}_{\mathsf{MLWE}}$ against the $\mathsf{MLWE}_{q,k,\chi}$ problem such that

$$|\epsilon_1 - \epsilon_2| \leqslant \mathsf{Adv}_{\mathcal{B}_{\mathsf{MLWE}}}^{\mathsf{MLWE}}(1^{\lambda}).$$

$\mathsf{Game}_3$: In this game, the challenger first samples $(\mathsf{K}_0, \mathbf{s}, \mathbf{e}) \xleftarrow{\$} \{0,1\}^{\lambda} \times \chi \times \chi$ and later programs the random oracle H on input $(\mathsf{ek}_{\mathsf{A}}, \mathsf{seed})$. In case the input is already queried (i.e., $Q_{\mathsf{H}}[\mathsf{ek}_{\mathsf{A}}, \mathsf{seed}] \neq \perp$), then the

```
$\mathsf{Game}_0$    // Original FS-IND-CPA security game
 1 :  $Q_\mathsf{H}[\cdot] := \bot$    // Prepare empty RO
 2 :  $b \xleftarrow{\$} \{0,1\}$
 3 :  $\mathsf{K}_1 \xleftarrow{\$} \{0,1\}^\lambda$
      // Sample from $\widehat{\mathcal{D}}_{\mathsf{RKeyGen\text{-}B}}$
 4 :  $(\hat{\mathbf{s}}_\mathsf{B}, \hat{\mathbf{e}}_\mathsf{B}) \xleftarrow{\$} \hat{\chi} \times \hat{\chi}$
 5 :  $\hat{\mathbf{u}}_\mathsf{B} := \mathbf{D}^\top \cdot \hat{\mathbf{s}}_\mathsf{B} + \hat{\mathbf{e}}_\mathsf{B} \in R_q^k$
 6 :  $(\widehat{\mathsf{ek}}_\mathsf{B}, \widehat{\mathsf{dk}}_\mathsf{B}) := (\hat{\mathbf{u}}_\mathsf{B}, \hat{\mathbf{s}}_\mathsf{B})$
      // Sample from $\mathcal{D}_{\mathsf{RKeyGen\text{-}A}}$
 7 :  $(\mathbf{s}_\mathsf{A}, \mathbf{e}_\mathsf{A}) \xleftarrow{\$} \chi \times \chi$
 8 :  $\mathbf{u}_\mathsf{A} := \mathbf{D} \cdot \mathbf{s}_\mathsf{A} + \mathbf{e}_\mathsf{A} \in R_q^k$
 9 :  $(\mathsf{ek}_\mathsf{A}, \mathsf{dk}_\mathsf{A}) := (\mathbf{u}_\mathsf{A}, (\mathbf{u}_\mathsf{A}, \mathbf{s}_\mathsf{A}))$
      // Run $\mathsf{REnc\text{-}A}(\widehat{\mathsf{ek}}_\mathsf{B}, \mathsf{dk}_\mathsf{A})$
10 :  $\mathsf{seed} \xleftarrow{\$} \{0,1\}^\lambda$
11 :  $m \leftarrow \mathsf{Encode}(\mathsf{seed})$    // $m \in R_q$
12 :  $(\mathsf{K}_0, \mathbf{s}, \mathbf{e}) := \mathsf{H}(\mathbf{u}_\mathsf{A}, \mathsf{seed})$
13 :  $\tilde{e}_\mathsf{A} \xleftarrow{\$} \tilde{\chi}$
14 :  $v_\mathsf{B} := \widehat{\mathsf{ek}}_\mathsf{B}^\top \cdot \mathbf{s}_\mathsf{A} + \tilde{e}_\mathsf{A} + m \in R_q$
15 :  $\mathsf{ct}_\mathsf{B} := v_\mathsf{B}$
16 :  $\widehat{\mathsf{dk}}_\mathsf{A} := \mathbf{s}_\mathsf{A} + \mathbf{s} \in R_q^k$    // Update and erase $\mathsf{dk}_\mathsf{A}$
      // Run $\mathsf{RDec\text{-}B}(\widehat{\mathsf{dk}}_\mathsf{B}, \mathsf{ct}_\mathsf{B}, \mathsf{ek}_\mathsf{A})$
17 :  $m' := \mathsf{ct}_\mathsf{B} - \mathsf{ek}_\mathsf{A}^\top \cdot \widehat{\mathsf{dk}}_\mathsf{B}$
18 :  $\mathsf{seed}' := \mathsf{Decode}(m')$
19 :  $(\mathsf{K}', \mathbf{s}', \mathbf{e}') := \mathsf{H}(\mathsf{ek}_\mathsf{A}', \mathsf{seed}')$
20 :  $\widehat{\mathsf{ek}}_\mathsf{A} := \mathsf{ek}_\mathsf{A} + \mathbf{D} \cdot \mathbf{s}' + \mathbf{e}'$
      // Run adversary $\mathcal{A}$
21 :  $b' \xleftarrow{\$} \mathcal{A}(\mathsf{ek}_\mathsf{A}, \widehat{\mathsf{ek}}_\mathsf{A}, \widehat{\mathsf{ek}}_\mathsf{B}, \mathsf{ct}_\mathsf{B}, \widehat{\mathsf{dk}}_\mathsf{A}, \mathsf{K}_b)$
22 :  return $[\![b = b']\!]$
```

```
$\mathsf{Game}_1$
      // Same up till $\mathsf{Game}_0$, line 16
16 :  $\widehat{\mathsf{ek}}_\mathsf{A} := \mathsf{ek}_\mathsf{A} + \mathbf{D} \cdot \mathbf{s} + \mathbf{e}$    // Reuse $\mathsf{K}, \mathbf{s}, \mathbf{e}$ from $\mathsf{REnc\text{-}A}$
17 :  $b' \xleftarrow{\$} \mathcal{A}(\mathsf{ek}_\mathsf{A}, \widehat{\mathsf{ek}}_\mathsf{A}, \widehat{\mathsf{ek}}_\mathsf{B}, \mathsf{ct}_\mathsf{B}, \widehat{\mathsf{dk}}_\mathsf{A}, \mathsf{K}_b)$
18 :  return $[\![b = b']\!]$
```

```
$\mathsf{Game}_2$
 1 :  $Q_\mathsf{H}[\cdot] := \bot$
 2 :  $b \xleftarrow{\$} \{0,1\}$
 3 :  $\mathsf{K}_1 \xleftarrow{\$} \{0,1\}^\lambda$
 4 :  $\hat{\mathbf{u}}_\mathsf{B} \xleftarrow{\$} R_q^k$
 5 :  $\widehat{\mathsf{ek}}_\mathsf{B} := \hat{\mathbf{u}}_\mathsf{B}$    // Remove $\widehat{\mathsf{dk}}_\mathsf{B}$
      // Sample from $\mathcal{D}_{\mathsf{RKeyGen\text{-}A}}$
      // Same from $\mathsf{Game}_1$, line 7
```

```
$\mathsf{Game}_3$
 1 :  $Q_\mathsf{H}[\cdot] := \bot$
 2 :  $b \xleftarrow{\$} \{0,1\}$
 3 :  $(\mathsf{K}_0, \mathbf{s}, \mathbf{e}) \xleftarrow{\$} \{0,1\}^\lambda \times \chi \times \chi$    // Sample w/o RO
 4 :  $\mathsf{K}_1 \xleftarrow{\$} \{0,1\}^\lambda$
 5 :  $\widehat{\mathsf{ek}}_\mathsf{B} \xleftarrow{\$} \mathcal{R}_q^k$
 6 :  $(\mathbf{s}_\mathsf{A}, \mathbf{e}_\mathsf{A}) \xleftarrow{\$} \chi \times \chi$
 7 :  $\mathbf{u}_\mathsf{A} := \mathbf{D} \cdot \mathbf{s}_\mathsf{A} + \mathbf{e}_\mathsf{A} \in R_q^k$
 8 :  $(\mathsf{ek}_\mathsf{A}, \mathsf{dk}_\mathsf{A}) := (\mathbf{u}_\mathsf{A}, (\mathbf{u}_\mathsf{A}, \mathbf{s}_\mathsf{A}))$
 9 :  $\mathsf{seed} \xleftarrow{\$} \{0,1\}^\lambda$
10 :  $m \leftarrow \mathsf{Encode}(\mathsf{seed})$
11 :  if $[\![Q_\mathsf{H}[\mathbf{u}_\mathsf{A}, \mathsf{seed}] \neq \bot]\!]$ then
12 :    return 1    // Declare $\mathcal{A}$ wins
13 :  $Q_\mathsf{H}[\mathbf{u}_\mathsf{A}, \mathsf{seed}] := (\mathsf{K}_0, \mathbf{s}, \mathbf{e})$    // Program RO
14 :  $\tilde{e}_\mathsf{A} \xleftarrow{\$} \tilde{\chi}$
15 :  $v_\mathsf{B} := \widehat{\mathsf{ek}}_\mathsf{B}^\top \cdot \mathbf{s}_\mathsf{A} + \tilde{e}_\mathsf{A} + m \in R_q$
16 :  $\mathsf{ct}_\mathsf{B} := v_\mathsf{B}$
17 :  $\widehat{\mathsf{dk}}_\mathsf{A} := \mathbf{s}_\mathsf{A} + \mathbf{s} \in R_q^k$
18 :  $\widehat{\mathsf{ek}}_\mathsf{A} := \mathsf{ek}_\mathsf{A} + \mathbf{D} \cdot \mathbf{s} + \mathbf{e}$
19 :  $b' \xleftarrow{\$} \mathcal{A}(\mathsf{ek}_\mathsf{A}, \widehat{\mathsf{ek}}_\mathsf{A}, \widehat{\mathsf{ek}}_\mathsf{B}, \mathsf{ct}_\mathsf{B}, \widehat{\mathsf{dk}}_\mathsf{A}, \mathsf{K}_b)$
20 :  return $[\![b = b']\!]$
```

```
$\mathsf{H}(\mathbf{u}_\mathsf{A}, \mathsf{seed})$    // Used by $\mathsf{Game}_0$ to $\mathsf{Game}_3$
 1 :  if $[\![Q_\mathsf{H}[\mathbf{u}, \mathsf{seed}] = \bot]\!]$ then
 2 :    $(\mathsf{K}_0, \mathbf{s}, \mathbf{e}) \xleftarrow{\$} \{0,1\}^\lambda \times \chi \times \chi$
 3 :    $Q_\mathsf{H}[\mathbf{u}, \mathsf{seed}] \leftarrow (\mathsf{K}_0, \mathbf{s}, \mathbf{e})$
 4 :  return $Q_\mathsf{H}[\mathbf{u}, \mathsf{seed}]$
```

Figure 19: Hybrid games $\mathsf{Game}_0$ to $\mathsf{Game}_3$ used for the proof of FS-IND-CPA. The text highlighted in blue denotes the main difference between the previous hybrid.

challenger declares the adversary $\mathcal{A}$ wins and outputs 1 as the output of the game. This game is identical to $\mathsf{Game}_2$ as long as $Q_{\mathsf{H}}[\mathsf{ek}_{\mathsf{A}}, \mathsf{seed}] \neq \bot$. Since $\mathsf{seed}$ is sampled uniformly random over $\{0,1\}^\lambda$, the probability of this occurring is $Q/2^\lambda$.

Hence, we have

$$|\epsilon_2 - \epsilon_3| \leqslant \frac{Q}{2^\lambda}.$$

$\mathsf{Game}_4$: In this game, the challenger no longer programs the random oracle. Instead, it aborts the game and declares the adversary wins when the random oracle is queried on $(\mathbf{u}_{\mathsf{A}}^*, \mathsf{seed}^*)$. We denote this event by $\mathsf{E}_4$. Clearly, as long as event $\mathsf{E}_4$ does not occur, $\mathsf{Game}_3$ and $\mathsf{Game}_4$ proceed identically. Hence, we have

$$|\epsilon_3 - \epsilon_4| \leqslant \Pr[\mathsf{E}_4].$$

As we cannot bound $\Pr[\mathsf{E}_4]$ yet, we postpone bounding it to later.

$\mathsf{Game}_5$: In this game, the challenger computes user $\mathsf{A}$'s updated key $\widehat{\mathsf{ek}}_{\mathsf{A}}$ directly without using $\mathsf{ek}_{\mathsf{A}}$. Since this is only a conceptual change, we have

$$\epsilon_4 = \epsilon_5.$$

Moreover, denoting $\mathsf{E}_5$ the event that the adversary triggers the abort condition in $\mathsf{Game}_5$, we also have

$$\Pr[\mathsf{E}_4] = \Pr[\mathsf{E}_5].$$

$\mathsf{Game}_6$: In the final game, the challenger samples a random $\mathbf{u}_{\mathsf{A}}$ from $\mathcal{R}_q^k$ and sets user $\mathsf{A}$'s key as $\mathsf{ek}_{\mathsf{A}} := \mathbf{u}_{\mathsf{A}}$. Moreover, it samples random $v'_{\mathsf{B}}$ from $\mathcal{R}_q$ and sets the ciphertext as $\mathsf{ct}_{\mathsf{B}} := v'_{\mathsf{B}} + m$. Recall in the previous game, $\mathbf{u}_{\mathsf{A}}$ and $v_{\mathsf{B}}$ were set as $\mathsf{MLWE}$ instances. Since $\mathbf{s}_{\mathsf{A}}$ and $\mathbf{e}_{\mathsf{A}}$ are partially leaked to the adversary $\mathcal{A}$ via $\widehat{\mathsf{dk}}_{\mathsf{A}} = \mathbf{s}_{\mathsf{A}} + \mathbf{s}$ and $\widehat{\mathsf{ek}}_{\mathsf{A}}$, we cannot rely on the standard $\mathsf{MLWE}$ assumption to argue indistinguishability of the two games. However, noticing that $\mathbf{s}, \mathbf{e} \xleftarrow{\$} \chi$ are information theoretically hidden to $\mathcal{A}$ conditioning on the game not aborting, we can rely instead on the *hint* $\mathsf{MLWE}$ assumption. Let $\mathsf{E}_6$ denote the event that $\mathcal{A}$ triggers an abort in $\mathsf{Game}_6$ and let $\mathsf{Win}_i$ denote the event that $\mathcal{A}$ wins in $\mathsf{Game}_i$. Then, we can construct an adversary $\mathcal{B}_{\mathsf{hint\text{-}MLWE},1}$ against the $\mathsf{hint\text{-}MLWE}_{q,k,2k,\chi,\chi,\mathcal{F}_{\mathsf{cpa}}}$ problem with $\mathcal{F} := \mathcal{U}(\{\mathbf{I}_{2k\times 2k}\})$ such that

$$|\Pr[\mathsf{Win}_5 \wedge \neg\mathsf{E}_5] - \Pr[\mathsf{Win}_6 \wedge \neg\mathsf{E}_6]| \leqslant \mathsf{Adv}^{\mathsf{hint\text{-}MLWE}}_{\mathcal{B}_{\mathsf{hint\text{-}MLWE},1}}(1^\lambda).$$

Indeed, the reduction is straightforward as $\mathcal{B}_{\mathsf{hint\text{-}MLWE},1}$ receives as hints $\widehat{\mathsf{dk}}_{\mathsf{A}}$ and $\widehat{\mathbf{e}}_{\mathsf{A}}$ and can efficiently simulate $\mathsf{Game}_5$ or $\mathsf{Game}_6$ to $\mathcal{A}$ depending on whether it receives a random or valid $\mathsf{MLWE}$ instance. Here, recall $\mathcal{F}$ outputs $\mathbf{I}_{2k\times 2k}$ (i.e., the identity matrix in $R_q^{2k\times 2k}$) with probability 1. Moreover, observe we have

$$\begin{aligned}
&|\epsilon_5 - \epsilon_6| \\
&= \big|\big(\Pr[\mathsf{E}_5]\cdot\Pr[\mathsf{Win}_5|\mathsf{E}_5] + \Pr[\mathsf{Win}_5 \wedge \neg\mathsf{E}_5]\big) - \big(\Pr[\mathsf{E}_6]\cdot\Pr[\mathsf{Win}_6|\mathsf{E}_6] + \Pr[\mathsf{Win}_6 \wedge \neg\mathsf{E}_6]\big)\big| \\
&\leqslant \Pr[\mathsf{E}_5] + \Pr[\mathsf{E}_6] + |\Pr[\mathsf{Win}_5 \wedge \neg\mathsf{E}_5] - \Pr[\mathsf{Win}_6 \wedge \neg\mathsf{E}_6]| \\
&\leqslant \Pr[\mathsf{E}_5] + \Pr[\mathsf{E}_6] + \mathsf{Adv}^{\mathsf{hint\text{-}MLWE}}_{\mathcal{B}_{\mathsf{hint\text{-}MLWE},1}}(1^\lambda),
\end{aligned}$$

where we use the fact that $\Pr[\mathsf{Win}_i|\mathsf{E}_i] = 1$ for $i \in \{5,6\}$. In addition, it can be checked that the differences between $\Pr[\mathsf{E}_5]$ and $\Pr[\mathsf{E}_6]$ are negligible assuming the hardness of the $\mathsf{hint\text{-}MLWE}$ problem. Formally, we can construct an adversary $\mathcal{B}_{\mathsf{hint\text{-}MLWE},2}$ against the $\mathsf{hint\text{-}MLWE}_{q,k,2k,\chi,\chi,\mathcal{F}_{\mathsf{cpa}}}$ problem such that

$$|\Pr[\mathsf{E}_5] - \Pr[\mathsf{E}_6]| \leqslant \mathsf{Adv}^{\mathsf{hint\text{-}MLWE}}_{\mathcal{B}_{\mathsf{hint\text{-}MLWE},2}}(1^\lambda).$$

$\mathsf{Game}_4$

```
1 :  $Q_\mathsf{H}[\cdot] := \bot$
2 :  $(\mathbf{u}_\mathsf{A}^*, \mathsf{seed}^*) := (\bot, \bot)$
3 :  $b \xleftarrow{\$} \{0,1\}$
4 :  $(\mathsf{K}_0, \mathbf{s}, \mathbf{e}) \xleftarrow{\$} \{0,1\}^\lambda \times \chi \times \chi$   // Sample w/o RO
5 :  $\mathsf{K}_1 \xleftarrow{\$} \{0,1\}^\lambda$
6 :  $\widehat{\mathsf{ek}}_\mathsf{B} \xleftarrow{\$} \mathcal{R}_q^k$
7 :  $(\mathbf{s}_\mathsf{A}, \mathbf{e}_\mathsf{A}) \xleftarrow{\$} \chi \times \chi$
8 :  $\mathbf{u}_\mathsf{A} := \mathbf{D} \cdot \mathbf{s}_\mathsf{A} + \mathbf{e}_\mathsf{A} \in R_q^k$
9 :  $(\mathsf{ek}_\mathsf{A}, \mathsf{dk}_\mathsf{A}) := (\mathbf{u}_\mathsf{A}, (\mathbf{u}_\mathsf{A}, \mathbf{s}_\mathsf{A}))$
10 : $\mathsf{seed} \xleftarrow{\$} \{0,1\}^\lambda$
11 : $m \leftarrow \mathsf{Encode}(\mathsf{seed})$
12 : if $[\![Q_\mathsf{H}[\mathbf{u}_\mathsf{A}, \mathsf{seed}] \neq \bot]\!]$ then
13 :   return 1   // Declare $\mathcal{A}$ wins
14 : $(\mathbf{u}_\mathsf{A}^*, \mathsf{seed}^*) \leftarrow (\mathbf{u}_\mathsf{A}, \mathsf{seed})$
15 : $\tilde{e}_\mathsf{A} \xleftarrow{\$} \tilde{\chi}$
16 : $v_\mathsf{B} := \widehat{\mathsf{ek}}_\mathsf{B}^\top \cdot \mathbf{s}_\mathsf{A} + \tilde{e}_\mathsf{A} + m \in R_q$
17 : $\mathsf{ct}_\mathsf{B} := v_\mathsf{B}$
18 : $\widehat{\mathsf{dk}}_\mathsf{A} := \mathbf{s}_\mathsf{A} + \mathbf{s} \in R_q^k$
19 : $\widehat{\mathsf{ek}}_\mathsf{A} := \mathsf{ek}_\mathsf{A} + \mathbf{D} \cdot \mathbf{s} + \mathbf{e}$
20 : $b' \xleftarrow{\$} \mathcal{A}(\mathsf{ek}_\mathsf{A}, \widehat{\mathsf{ek}}_\mathsf{A}, \widehat{\mathsf{ek}}_\mathsf{B}, \mathsf{ct}_\mathsf{B}, \widehat{\mathsf{dk}}_\mathsf{A}, \mathsf{K}_b)$
21 : return $[\![b = b']\!]$
```

$\mathsf{H}(\mathbf{u}, \mathsf{seed})$ // Used by $\mathsf{Game}_4$ to $\mathsf{Game}_6$

```
1 : if $[\![(\mathbf{u}_\mathsf{A}^*, \mathsf{seed}^*) \neq (\bot, \bot)]\!]$ then
2 :   if $[\![(\mathbf{u}, \mathsf{seed}) = (\mathbf{u}_\mathsf{A}^*, \mathsf{seed}^*)]\!]$ then
3 :     abort
4 : if $[\![Q_\mathsf{H}[\mathbf{u}, \mathsf{seed}] = \bot]\!]$ then
5 :   $(\mathsf{K}_0, \mathbf{s}, \mathbf{e}) \xleftarrow{\$} \{0,1\}^\lambda \times \chi \times \chi$
6 :   $Q_\mathsf{H}[\mathbf{u}, \mathsf{seed}] \leftarrow (\mathsf{K}_0, \mathbf{s}, \mathbf{e})$
7 : return $Q_\mathsf{H}[\mathbf{u}, \mathsf{seed}]$
```

$\mathsf{Game}_5$

```
     // Same up till $\mathsf{Game}_4$, line 17
18 : $\widehat{\mathsf{dk}}_\mathsf{A} := \mathbf{s}_\mathsf{A} + \mathbf{s} \in R_q^k$
19 : $\hat{\mathbf{e}}_\mathsf{A} := \mathbf{e}_\mathsf{A} + \mathbf{e} \in R_q^k$
20 : $\widehat{\mathsf{ek}}_\mathsf{A} := \mathbf{D} \cdot \widehat{\mathsf{dk}}_\mathsf{A} + \hat{\mathbf{e}}_\mathsf{A}$
21 : $b' \xleftarrow{\$} \mathcal{A}(\mathsf{ek}_\mathsf{A}, \widehat{\mathsf{ek}}_\mathsf{A}, \widehat{\mathsf{ek}}_\mathsf{B}, \mathsf{ct}_\mathsf{B}, \widehat{\mathsf{dk}}_\mathsf{A}, \mathsf{K}_b)$
22 : return $[\![b = b']\!]$
```

$\mathsf{Game}_6$

```
1 :  $Q_\mathsf{H}[\cdot] := \bot$
2 :  $(\mathbf{u}_\mathsf{A}^*, \mathsf{seed}^*) := (\bot, \bot)$
3 :  $b \xleftarrow{\$} \{0,1\}$
4 :  $(\mathsf{K}_0, \mathbf{s}, \mathbf{e}) \xleftarrow{\$} \{0,1\}^\lambda \times \chi \times \chi$   // Sample w/o RO
5 :  $\mathsf{K}_1 \xleftarrow{\$} \{0,1\}^\lambda$
6 :  $\widehat{\mathsf{ek}}_\mathsf{B} \xleftarrow{\$} \mathcal{R}_q^k$
7 :  $(\mathbf{s}_\mathsf{A}, \mathbf{e}_\mathsf{A}) \xleftarrow{\$} \chi \times \chi$
8 :  $\mathbf{u}_\mathsf{A} \xleftarrow{\$} R_q^k$
9 :  $\mathsf{ek}_\mathsf{A} := \mathbf{u}_\mathsf{A}$   // Remove $\mathsf{dk}_\mathsf{A}$
10 : $\mathsf{seed} \xleftarrow{\$} \{0,1\}^\lambda$
11 : $m \leftarrow \mathsf{Encode}(\mathsf{seed})$
12 : if $[\![Q_\mathsf{H}[\mathbf{u}_\mathsf{A}, \mathsf{seed}] \neq \bot]\!]$ then
13 :   return 1   // Declare $\mathcal{A}$ wins
14 : else
15 :   $(\mathbf{u}_\mathsf{A}^*, \mathsf{seed}^*) \leftarrow (\mathbf{u}_\mathsf{A}, \mathsf{seed})$
16 : $v'_\mathsf{B} \xleftarrow{\$} R_q$
17 : $v_\mathsf{B} := v'_\mathsf{B} + m$
18 : $\mathsf{ct}_\mathsf{B} := v_\mathsf{B}$
19 : $\widehat{\mathsf{dk}}_\mathsf{A} := \mathbf{s}_\mathsf{A} + \mathbf{s} \in R_q^k$
20 : $\hat{\mathbf{e}}_\mathsf{A} := \mathbf{e}_\mathsf{A} + \mathbf{e} \in R_q^k$
21 : $\widehat{\mathsf{ek}}_\mathsf{A} := \mathbf{D} \cdot \widehat{\mathsf{dk}}_\mathsf{A} + \hat{\mathbf{e}}_\mathsf{A}$
22 : $b' \xleftarrow{\$} \mathcal{A}(\mathsf{ek}_\mathsf{A}, \widehat{\mathsf{ek}}_\mathsf{A}, \widehat{\mathsf{ek}}_\mathsf{B}, \mathsf{ct}_\mathsf{B}, \widehat{\mathsf{dk}}_\mathsf{A}, \mathsf{K}_b)$
23 : return $[\![b = b']\!]$
```

Figure 20: Hybrid games $\mathsf{Game}_4$ to $\mathsf{Game}_6$ used for the proof of FS-IND-CPA. The text highlighted in blue denotes the main difference between the previous hybrid. **abort** indicates the game terminates and returns 1.

Lastly, observe that in the final game, $m$ is information theoretically hidden from $\mathcal{A}$ as $v'_\mathsf{B}$ is sampled uniformly at random. Put differently, seed is hidden from $\mathcal{A}$, and thus, we have $\Pr[\mathsf{E}_6] \leqslant Q/2^\lambda$.

```
$\mathsf{RSimKey\text{-}P}_1(\mathsf{ek_P}, \mathsf{dk_P})$
 1 :  $(\mathbf{u}_\mathsf{P}, \mathbf{s}_\mathsf{P}) := \mathsf{dk_P}$
 2 :  $\mathsf{seed} \xleftarrow{\$} \mathcal{U}(\{0,1\}^\lambda)\{\mathsf{rand_{seed}}\}$
 3 :  $(\mathsf{K}, \mathbf{s}, \mathbf{e}) := \mathsf{H}(\mathbf{u}_\mathsf{P}, \mathsf{seed})$
 4 :  if $[\![\mathsf{P} = \mathsf{A}]\!]$ then
 5 :    $\widehat{\mathsf{ek}}_\mathsf{A} := \mathbf{u}_\mathsf{A} + \mathbf{D} \cdot \mathbf{s} + \mathbf{e}$
 6 :  else    // $\mathsf{P} = \mathsf{B}$
 7 :    $\widehat{\mathsf{ek}}_\mathsf{B} := \mathbf{u}_\mathsf{B} + \mathbf{D}^\top \cdot \mathbf{s} + \mathbf{e}$
 8 :  $\widehat{\mathsf{dk}}_\mathsf{P} := \mathbf{s}_\mathsf{P} + \mathbf{s} \in R_q^k$
 9 :  $\mathsf{aux}_1 := (\mathsf{rand_{seed}}, \mathsf{dk_P})$
10 :  return $(\widehat{\mathsf{ek}}_\mathsf{P}, \widehat{\mathsf{dk}}_\mathsf{P}, \mathsf{aux}_1)$
```

```
$\mathsf{RSimKey\text{-}P}_2(\widehat{\mathsf{ek}}_{\bar{\mathsf{P}}}, \widehat{\mathsf{dk}}_{\bar{\mathsf{P}}}, \mathsf{aux}_1)$
 1 :  $(\mathsf{rand_{seed}}, \mathsf{dk_P}) := \mathsf{aux}_1$
 2 :  $(\mathbf{u}_\mathsf{P}, \mathbf{s}_\mathsf{P}) := \mathsf{dk_P}$
 3 :  $\mathsf{seed} \xleftarrow{\$} \mathcal{U}(\{0,1\}^\lambda)\{\mathsf{rand_{seed}}\}$
 4 :  $(\mathsf{K}, \mathbf{s}, \mathbf{e}) := \mathsf{H}(\mathbf{u}_\mathsf{P}, \mathsf{seed})$
 5 :  $m \leftarrow \mathsf{Encode}(\mathsf{seed})$    // $m \in R_q$
 6 :  $\tilde{e}_\mathsf{P} \xleftarrow{\$} \tilde{\chi}\{\mathsf{rand}_{\tilde{e}}\}$
 7 :  $v_{\bar{\mathsf{P}}} := \widehat{\mathsf{ek}}_{\bar{\mathsf{P}}}^\top \cdot \mathbf{s}_\mathsf{P} + \tilde{e}_\mathsf{P} + m \in R_q$
 8 :  $\mathsf{ct}_{\bar{\mathsf{P}}} := v_{\bar{\mathsf{P}}}$
 9 :  $\mathsf{rand}_2 := (\mathsf{rand_{seed}}, \mathsf{rand}_{\tilde{e}})$
10 :  return $(\mathsf{ct}_{\bar{\mathsf{P}}}, \mathsf{K}, \mathsf{K}, \mathsf{rand}_2)$
```

```
$\mathsf{RSimCtxt\text{-}P}(\widehat{\mathsf{ek}}_\mathsf{P}, \widehat{\mathsf{ek}}_{\bar{\mathsf{P}}}, \widehat{\mathsf{dk}}_{\bar{\mathsf{P}}})$
 1 :  $(\mathsf{K}, \mathbf{s}, \mathbf{e}) \xleftarrow{\$} \{0,1\}^\lambda \times \chi \times \chi$
 2 :  if $[\![\mathsf{P} = \mathsf{A}]\!]$ then
 3 :    $\mathsf{ek_A} := \widehat{\mathsf{ek}}_\mathsf{A} - \mathbf{D} \cdot \mathbf{s} - \mathbf{e}$    // ek before update
 4 :    $\hat{\mathbf{e}}_\mathsf{B} := \widehat{\mathsf{ek}}_\mathsf{B} - \mathbf{D}^\top \cdot \widehat{\mathsf{dk}}_\mathsf{B}$    // MLWE noise in $\widehat{\mathsf{ek}}$
 5 :  else    // $\mathsf{P} = \mathsf{B}$
 6 :    $\mathsf{ek_B} := \widehat{\mathsf{ek}}_\mathsf{B} - \mathbf{D}^\top \cdot \mathbf{s} - \mathbf{e}$
 7 :    $\hat{\mathbf{e}}_\mathsf{A} := \widehat{\mathsf{ek}}_\mathsf{A} - \mathbf{D} \cdot \widehat{\mathsf{dk}}_\mathsf{A}$
 8 :  $\mathsf{seed} \xleftarrow{\$} \{0,1\}^\lambda$
 9 :  $\mathsf{H}(\mathsf{ek_P}, \mathsf{seed}) := (\mathsf{K}, \mathbf{s}, \mathbf{e})$    // Program RO
10 :  $m \leftarrow \mathsf{Encode}(\mathsf{seed})$    // $m \in R_q$
11 :  $(\mathbf{s}_\mathsf{P}, \mathbf{e}_\mathsf{P}, \tilde{e}_\mathsf{P}) \xleftarrow{\$} \hat{\chi} \times \hat{\chi} \times \tilde{\chi}$
12 :  $v_{\bar{\mathsf{P}}} := \hat{\mathbf{s}}_{\bar{\mathsf{P}}}^\top \cdot \mathsf{ek_P} - \hat{\mathbf{s}}_{\bar{\mathsf{P}}}^\top \cdot \mathbf{e}_\mathsf{P} + \hat{\mathbf{e}}_{\bar{\mathsf{P}}}^\top \cdot \mathbf{s}_\mathsf{P} + \tilde{e}_\mathsf{P} + m$
13 :  $\mathsf{ct}_{\bar{\mathsf{P}}} := v_{\bar{\mathsf{P}}}$    // Simulate $v_{\bar{\mathsf{P}}}$ w/o user P secret
14 :  return $(\mathsf{ct}_{\bar{\mathsf{P}}}, \mathsf{ek_P}, \mathsf{K}, \mathsf{K})$
```

Figure 21: Simulators for base key, updated key, and ciphertext simulatability with $(\mathsf{P}, \bar{\mathsf{P}}) = (\mathsf{A}, \mathsf{B})$ or $(\mathsf{B}, \mathsf{A})$. Recall $\mathcal{U}(S)\{\mathsf{rand}\}$ denotes the process of sampling uniformly from the set $S$ using randomness $\mathsf{rand}$. Moreover, in line 9 of RSimCtxt-P, we assume the simulator outputs $\perp$ in case the random oracle is already programed.

Combining everything together, we have

$$|\epsilon_5 - \epsilon_6| \leqslant \mathsf{Adv}^{\mathsf{hint\text{-}MLWE}}_{\mathcal{B}_{\mathsf{hint\text{-}MLWE},1}}(1^\lambda) + \mathsf{Adv}^{\mathsf{hint\text{-}MLWE}}_{\mathcal{B}_{\mathsf{hint\text{-}MLWE},2}}(1^\lambda) + \frac{Q}{2^\lambda}.$$

The bound in the theorem statement follows by collecting all the bounds. This completes the proof. □

### 6.2.2 Ratchet Simulatability.

The following theorem establishes the ratchet simulatability of Katana. Below, we rely on the $\mathsf{hint\text{-}MLWE}_{q,k,1,\chi,\tilde{\chi},\mathcal{F}_{\mathsf{sim}}}$ problem, where $\mathcal{F}_{\mathsf{sim}}$ is a distribution over $R_q^{1\times 2k}$ that outputs $[-\hat{\mathbf{s}}^\top | \hat{\mathbf{e}}^\top]$ with $\hat{\mathbf{s}}, \hat{\mathbf{e}} \xleftarrow{\$} \hat{\chi}$.

**Theorem 6.3 (Ratchet simulatability).** *Our* RKEM Katana *is ratchet simulatable assuming the hardness of the* MLWE *and* hint-MLWE *assumptions.*

*Formally, for any adversary $\mathcal{A}$ against ratchet simulatability making at most $Q$ queries to the random oracle* H*, there exists adversary $\mathcal{B}_{\mathsf{MLWE}}$ against the $\mathsf{MLWE}_{q,k,\hat{\chi}}$ problem and adversary $\mathcal{B}_{\mathsf{hint\text{-}MLWE}}$ against the $\mathsf{hint\text{-}MLWE}_{q,k,1,\chi,\tilde{\chi},\mathcal{F}_{\mathsf{sim}}}$ problem such that*

$$\mathsf{Adv}^{\mathsf{KeyUpdSim\text{-}P}}_{\mathcal{A}}(1^\lambda) \leqslant \epsilon_{\mathsf{corr}}$$

*and*

$$\mathsf{Adv}^{\mathsf{CtxtSim\text{-}P}}_{\mathcal{A}}(1^\lambda) \leqslant \mathsf{Adv}^{\mathsf{MLWE}}_{\mathcal{B}_{\mathsf{MLWE}}}(1^\lambda) + \mathsf{Adv}^{\mathsf{hint\text{-}MLWE}}_{\mathcal{B}_{\mathsf{hint\text{-}MLWE}}}(1^\lambda) + \epsilon_{\mathsf{corr}} + \frac{Q}{2^\lambda},$$

*where $\epsilon_{\mathsf{corr}}$ is the probability that correctness with updated keys fails (cf. Definition 5.3).*

*Proof.* We first provide the simulators $(\mathsf{RSimKey\text{-}P}_1, \mathsf{RSimKey\text{-}P}_2, \mathsf{RSimCtxt\text{-}P})_{\mathsf{P}\in\{\mathsf{A},\mathsf{B}\}}$ used to prove ratchet simulatability in Fig. 21. As base key simulatability trivially holds from construction, below we show update key simulatability and ciphertext simulatability.

```
Distribution $\mathcal{D}_{\mathsf{A},0} := \mathcal{D}^{\mathsf{KeyUpdSim}}_{\mathsf{A},0}$
1 :  $(\mathsf{ek}_\mathsf{B}, \mathsf{dk}_\mathsf{B}) \xleftarrow{\$} \mathcal{D}_{\mathsf{RKeyGen\text{-}B}}\{\mathsf{rand}_0\}$
2 :  $(\widehat{\mathsf{ek}}_\mathsf{B}, \widehat{\mathsf{dk}}_\mathsf{B}, \mathsf{aux}_0) \xleftarrow{\$} \mathsf{RSimKey\text{-}B}_1(\mathsf{ek}_\mathsf{B}, \mathsf{dk}_\mathsf{B})$
3 :  $(\mathsf{ek}_\mathsf{A}, \mathsf{dk}_\mathsf{A}) \xleftarrow{\$} \mathcal{D}_{\mathsf{RKeyGen\text{-}A}}\{\mathsf{rand}_1\}$
     // $\mathsf{REnc\text{-}A}$ in full detail
4 :  $(\mathsf{rand}_\mathsf{seed}, \mathsf{rand}_{\tilde{e}}) := \mathsf{rand}_2$
5 :  $(\mathbf{u}_\mathsf{A}, \mathbf{s}_\mathsf{A}) := \mathsf{dk}_\mathsf{A}$
6 :  $\mathsf{seed} \xleftarrow{\$} \mathcal{U}(\{0,1\}^\lambda)\{\mathsf{rand}_\mathsf{seed}\}$
7 :  $m \leftarrow \mathsf{Encode}(\mathsf{seed})$   // $m \in R_q$
8 :  $(\mathsf{K}, \mathbf{s}, \mathbf{e}) := \mathsf{H}(\mathbf{u}_\mathsf{A}, \mathsf{seed})$
9 :  $\tilde{e}_\mathsf{A} \xleftarrow{\$} \tilde{\chi}\{\mathsf{rand}_{\tilde{e}}\}$
10 : $v_\mathsf{B} := \widehat{\mathsf{ek}}_\mathsf{B}^\top \cdot \mathbf{s}_\mathsf{A} + \tilde{e}_\mathsf{A} + m \in R_q$
11 : $\mathsf{ct}_\mathsf{B} := v_\mathsf{B}$
12 : $\widehat{\mathsf{dk}}_\mathsf{A} := \mathbf{s}_\mathsf{A} + \mathbf{s} \in R_q^k$
     // $\mathsf{RDec\text{-}B}$ in full detail
13 : $m' := \mathsf{ct}_\mathsf{P} - \mathsf{ek}_{\bar{\mathsf{P}}}^\top \cdot \widehat{\mathsf{dk}}_\mathsf{P}$
14 : $\mathsf{seed}' := \mathsf{Decode}(m')$
15 : $(\mathsf{K}', \mathsf{s}', \mathbf{e}') := \mathsf{H}(\mathsf{ek}_{\bar{\mathsf{P}}}, \mathsf{seed}')$
16 : $\widehat{\mathsf{ek}}_\mathsf{A} := \mathsf{ek}_\mathsf{A} + \mathbf{D} \cdot \mathbf{s}' + \mathbf{e}'$
17 : return $\big((\widehat{\mathsf{ek}}_\mathsf{B}, \widehat{\mathsf{dk}}_\mathsf{B}), (\widehat{\mathsf{ek}}_\mathsf{A}, \widehat{\mathsf{dk}}_\mathsf{A}), \mathsf{ct}_\mathsf{B}, \mathsf{K}, \mathsf{K}',$
            $\mathsf{aux}_0, \mathsf{rand}_0, \mathsf{rand}_1, \mathsf{rand}_2\big)$
```

```
Distribution $\mathcal{D}_{\mathsf{A},1} := \mathcal{D}^{\mathsf{KeyUpdSim}}_{\mathsf{A},1}$
     // Same up till $\mathcal{D}^{\mathsf{KeyUpdSim}}_0$, line 12
13 : $\widehat{\mathsf{ek}}_\mathsf{A} := \mathsf{ek}_\mathsf{A} + \mathbf{D} \cdot \mathbf{s} + \mathbf{e}$ // Reuse $(\mathsf{K}, \mathbf{s}, \mathbf{e})$ from $\mathsf{REnc\text{-}A}$
14 : return $\big((\widehat{\mathsf{ek}}_\mathsf{B}, \widehat{\mathsf{dk}}_\mathsf{B}), (\widehat{\mathsf{ek}}_\mathsf{A}, \widehat{\mathsf{dk}}_\mathsf{A}), \mathsf{ct}_\mathsf{B}, \mathsf{K}, \mathsf{K},$
            $\mathsf{aux}_0, \mathsf{rand}_0, \mathsf{rand}_1, \mathsf{rand}_2\big)$
```

```
Distribution $\mathcal{D}_{\mathsf{A},2}$
     // Same up till $\mathcal{D}^{\mathsf{KeyUpdSim}}_0$, line 3
     // Run $\mathsf{RSimKey\text{-}A}_1(\mathsf{ek}_\mathsf{A}, \mathsf{dk}_\mathsf{A})$
4 :  $(\mathbf{u}_\mathsf{A}, \mathbf{s}_\mathsf{A}) := \mathsf{dk}_\mathsf{A}$   // $\mathsf{ek}_\mathsf{A} = \mathbf{u}_\mathsf{A}$
5 :  $\mathsf{seed} \xleftarrow{\$} \mathcal{U}(\{0,1\}^\lambda)\{\mathsf{rand}_\mathsf{seed}\}$   // Sample only $\mathsf{rand}_\mathsf{seed}$
6 :  $(\mathsf{K}, \mathbf{s}, \mathbf{e}) := \mathsf{H}(\mathbf{u}_\mathsf{A}, \mathsf{seed})$
7 :  $\widehat{\mathsf{ek}}_\mathsf{A} := \mathsf{ek}_\mathsf{A} + \mathbf{D} \cdot \mathbf{s} + \mathbf{e}$   // Compute at the beginning
8 :  $\widehat{\mathsf{dk}}_\mathsf{A} := \mathbf{s}_\mathsf{A} + \mathbf{s} \in R_q^k$
9 :  $\mathsf{aux}_1 := (\mathsf{rand}_\mathsf{seed}, \mathsf{dk}_\mathsf{A})$
     // Run $\mathsf{RSimKey\text{-}A}_2(\widehat{\mathsf{ek}}_\mathsf{B}, \widehat{\mathsf{dk}}_\mathsf{B}, \mathsf{aux}_1)$, ignoring the initial steps
10 : $m \leftarrow \mathsf{Encode}(\mathsf{seed})$   // $m \in R_q$
11 : $\tilde{e}_\mathsf{A} \xleftarrow{\$} \tilde{\chi}\{\mathsf{rand}_{\tilde{e}}\}$
12 : $v_\mathsf{B} := \widehat{\mathsf{ek}}_\mathsf{B}^\top \cdot \mathbf{s}_\mathsf{A} + \tilde{e}_\mathsf{A} + m \in R_q$
13 : $\mathsf{ct}_\mathsf{B} := v_\mathsf{B}$
14 : $\mathsf{rand}_2 := (\mathsf{rand}_\mathsf{seed}, \mathsf{rand}_{\tilde{e}})$   // Set $\mathsf{rand}_2$ at the end
15 : return $\big((\widehat{\mathsf{ek}}_\mathsf{B}, \widehat{\mathsf{dk}}_\mathsf{B}), (\widehat{\mathsf{ek}}_\mathsf{A}, \widehat{\mathsf{dk}}_\mathsf{A}), \mathsf{ct}_\mathsf{B}, \mathsf{K}, \mathsf{K},$
            $\mathsf{aux}_0, \mathsf{rand}_0, \mathsf{rand}_1, \mathsf{rand}_2\big)$
```

Figure 22: Hybrid distributions used for the proof of updated key simulatability. The text highlighted in blue denotes the main difference between the previous hybrid.

**Updated key simulatability.** Due to the symmetry of users A and B, we only focus on bounding the distinguishing advantage of distributions $\mathcal{D}^{\mathsf{KeyUpdSim}}_{\mathsf{A},0}$ and $\mathcal{D}^{\mathsf{KeyUpdSim}}_{\mathsf{A},1}$ (cf. Definition 5.5). This is proven in a sequence of hybrid distributions given in Fig. 22. Let us assume an adversary $\mathcal{A}$ is given a sample from the distribution $\mathcal{D}_{\mathsf{A},i}$ for $i \in \{0,1,2\}$ and outputs a bit. Let $\epsilon_i$ denote the probability that $\mathcal{A}$ outputs 1 given a sample from $\mathcal{D}_{\mathsf{A},i}$. The goal is then to bound the difference between $\epsilon_0$ and $\epsilon_2$.

Distribution $\mathcal{D}_{\mathsf{A},0}$: This is the distribution $\mathcal{D}^{\mathsf{KeyUpdSim}}_{\mathsf{A},0}$. For reference, in Fig. 22, we provide the full details of the distribution.

Distribution $\mathcal{D}_{\mathsf{A},1}$: The only difference between the prior distribution is that we reuse $\mathsf{K}, \mathbf{s}, \mathbf{e}$ from algorithm $\mathsf{REnc\text{-}A}$ as opposed to generating them through executing $\mathsf{RDec\text{-}B}$. This follows from the same argument made to prove correctness: $m$ used during $\mathsf{REnc\text{-}A}$ and $m'$ generated during $\mathsf{RDec\text{-}B}$ are the same with all but a negligible probability. Hence, we have

$$|\epsilon_0 - \epsilon_1| \leqslant \epsilon_\mathsf{corr},$$

where $\epsilon_{\mathsf{corr}}$ is the probability that correctness with updated keys fails (cf. Definition 5.3).

Distribution $\mathcal{D}_{\mathsf{A},2}$: The only difference between the prior distribution is that we push the generation of $\widehat{\mathsf{ek}}_{\mathsf{A}}$ and $\widehat{\mathsf{dk}}_{\mathsf{A}}$ once we sample $(\mathsf{K}, \mathbf{s}, \mathbf{e})$. This can be done because $\widehat{\mathsf{ek}}_{\mathsf{A}}$ no longer depends on user $\mathsf{B}$'s information due to the modification we made in $\mathcal{D}_{\mathsf{A},1}$. Since this modification is conceptual, we have

$$\epsilon_1 = \epsilon_2.$$

Lastly, observe that $\mathcal{D}_{\mathsf{A},2}$ implicitly defines the desired $\mathsf{RSimKey\text{-}A}_1$ and $\mathsf{RSimKey\text{-}A}_2$. Hence, $\mathcal{D}_{\mathsf{A},2}$ has the same distribution as $\mathcal{D}^{\mathsf{KeyUpdSim}}_{\mathsf{A},1}$. This completes the proof of updated key simulatability.

**Ciphertext simulatability.** Similarly to above, we only focus on the distinguishing advantage of the distributions $\mathcal{D}^{\mathsf{CtxtSim}}_{\mathsf{B},0}$ and $\mathcal{D}^{\mathsf{CtxtSim}}_{\mathsf{B},1}$. This is proven in a sequence of hybrid distributions given in Figs. 23 and 24. Again, let $\epsilon_i$ denote the probability that $\mathcal{A}$ outputs 1 given a sample from $\mathcal{D}_{\mathsf{B},i}$. The goal is then to bound the difference between $\epsilon_0$ and $\epsilon_6$.

Distribution $\mathcal{D}_{\mathsf{B},0}$: This is the distribution $\mathcal{D}^{\mathsf{CtxtSim}}_{\mathsf{B},0}$. For reference, in Fig. 23, we provide the full details of the distribution.

Distribution $\mathcal{D}_{\mathsf{B},1}$: The only difference between the prior distribution is that we reuse $\mathsf{K}, \mathbf{s}, \mathbf{e}$ from algorithm $\mathsf{REnc\text{-}B}$ as opposed to generating them through executing $\mathsf{RDec\text{-}A}$. This follows from the same argument made to prove correctness: $m$ used during $\mathsf{REnc\text{-}B}$ and $m'$ generated during $\mathsf{RDec\text{-}A}$ are the same with all but a negligible probability. Hence, we have

$$|\epsilon_0 - \epsilon_1| \leqslant \epsilon_{\mathsf{corr}},$$

where $\epsilon_{\mathsf{corr}}$ is the probability that correctness with updated keys fails (cf. Definition 5.3).

Distribution $\mathcal{D}_{\mathsf{B},2}$: The main difference between the prior distribution is that we compute the ciphertext $\mathsf{ct}_{\mathsf{A}} = v_{\mathsf{A}}$ in a different way. Below, we show that these two ways of computing $v_{\mathsf{A}}$ are identical, where the first equality is how $v_{\mathsf{A}}$ is computed in the prior distribution:

$$\begin{aligned} v_{\mathsf{A}} - m &= \widehat{\mathsf{ek}}_{\mathsf{A}}^{\top} \cdot \mathbf{s}_{\mathsf{B}} + \tilde{e}_{\mathsf{B}} \\ &= (\mathbf{D} \cdot \widehat{\mathbf{s}}_{\mathsf{A}} + \widehat{\mathbf{e}}_{\mathsf{A}})^{\top} \cdot \mathbf{s}_{\mathsf{B}} + \tilde{e}_{\mathsf{B}} \\ &= \widehat{\mathbf{s}}_{\mathsf{A}}^{\top} \cdot (\mathbf{D}^{\top} \cdot \mathbf{s}_{\mathsf{B}} + \mathbf{e}_{\mathsf{B}}) - \widehat{\mathbf{s}}_{\mathsf{A}}^{\top} \cdot \mathbf{e}_{\mathsf{B}} + \widehat{\mathbf{e}}_{\mathsf{A}}^{\top} \cdot \mathbf{s}_{\mathsf{B}} + \tilde{e}_{\mathsf{B}} \\ &= \widehat{\mathbf{s}}_{\mathsf{A}}^{\top} \cdot \mathsf{ek}_{\mathsf{B}} - \widehat{\mathbf{s}}_{\mathsf{A}}^{\top} \cdot \mathbf{e}_{\mathsf{B}} + \widehat{\mathbf{e}}_{\mathsf{A}}^{\top} \cdot \mathbf{s}_{\mathsf{B}} + \tilde{e}_{\mathsf{B}} \end{aligned}$$

The last equation is exactly how $v_{\mathsf{A}}$ is computed in $\mathcal{D}_{\mathsf{B},2}$. Hence, we have

$$\epsilon_1 = \epsilon_2.$$

Distribution $\mathcal{D}_{\mathsf{B},3}$: The difference between the prior distribution is that we sample $(\mathsf{K}_0, \mathbf{s}, \mathbf{e}) \xleftarrow{\$} \{0,1\}^{\lambda} \times \chi \times \chi$ and later program the random oracle $\mathsf{H}$ on input $(\mathsf{ek}_{\mathsf{A}}, \mathsf{seed})$. In case the input is already queried (i.e., $Q_{\mathsf{H}}[\mathsf{ek}_{\mathsf{A}}, \mathsf{seed}] \neq \perp$), the distribution outputs a special symbol $\top$. This distribution is identical to the previous one as long as $Q_{\mathsf{H}}[\mathsf{ek}_{\mathsf{A}}, \mathsf{seed}] \neq \perp$. Since $\mathsf{seed}$ is sampled uniformly random over $\{0,1\}^{\lambda}$, the probability of this occurring is $Q/2^{\lambda}$.

Hence, we have

$$|\epsilon_2 - \epsilon_3| \leqslant \frac{Q}{2^{\lambda}}.$$

```
Distribution $\mathcal{D}_{\mathsf{B},0} := \mathcal{D}^{\mathsf{CtxtSim}}_{\mathsf{B},0}$
 1 : $Q_{\mathsf{H}}[\cdot] := \bot$   // Prepare empty RO
 2 : $(\mathsf{ek}_{\mathsf{A}}, \mathsf{dk}_{\mathsf{A}}) \xleftarrow{\$} \mathcal{D}_{\mathsf{RKeyGen\text{-}A}}\{\mathsf{rand}\}$
 3 : $(\widehat{\mathsf{ek}}_{\mathsf{A}}, \widehat{\mathsf{dk}}_{\mathsf{A}}, \mathsf{aux}) \xleftarrow{\$} \mathsf{RSimKey\text{-}A}_1(\mathsf{ek}_{\mathsf{A}}, \mathsf{dk}_{\mathsf{A}})$
      // Real user B procedure in full detail
      // $\mathcal{D}_{\mathsf{RKeyGen\text{-}B}}$ in full detail
 4 : $(\mathbf{s}_{\mathsf{B}}, \mathbf{e}_{\mathsf{B}}) \xleftarrow{\$} \chi \times \chi$
 5 : $\mathbf{u}_{\mathsf{B}} := \mathbf{D}^{\top} \cdot \mathbf{s}_{\mathsf{B}} + \mathbf{e}_{\mathsf{B}} \in R_q^k$
 6 : $(\mathsf{ek}_{\mathsf{B}}, \mathsf{dk}_{\mathsf{B}}) := (\mathbf{u}_{\mathsf{B}}, (\mathbf{u}_{\mathsf{B}}, \mathbf{s}_{\mathsf{B}}))$
      // $\mathsf{REnc\text{-}B}(\widehat{\mathsf{ek}}_{\mathsf{A}}, \mathsf{dk}_{\mathsf{B}})$ in full detail
 7 : $\mathsf{seed} \xleftarrow{\$} \{0,1\}^{\lambda}$
 8 : $m \leftarrow \mathsf{Encode}(\mathsf{seed})$   // $m \in R_q$
 9 : $(\mathsf{K}, \mathbf{s}, \mathbf{e}) := \mathsf{H}(\mathsf{ek}_{\mathsf{B}}, \mathsf{seed})$
10 : $\tilde{e}_{\mathsf{B}} \xleftarrow{\$} \tilde{\chi}$
11 : $v_{\mathsf{A}} := \widehat{\mathsf{ek}}_{\mathsf{A}}^{\top} \cdot \mathbf{s}_{\mathsf{B}} + \tilde{e}_{\mathsf{B}} + m \in R_q$
12 : $\mathsf{ct}_{\mathsf{A}} := v_{\mathsf{A}}$
13 : $\widehat{\mathsf{dk}}_{\mathsf{B}} := \mathbf{s}_{\mathsf{B}} + \mathbf{s} \in R_q^k$   // Update and erase $\mathsf{dk}_{\mathsf{B}}$
      // $\mathsf{RDec\text{-}A}(\widehat{\mathsf{dk}}_{\mathsf{A}}, \mathsf{ct}_{\mathsf{A}}, \mathsf{ek}_{\mathsf{B}})$ in full detail
14 : $m' := \mathsf{ct}_{\mathsf{A}} - \mathsf{ek}_{\mathsf{B}}^{\top} \cdot \widehat{\mathsf{dk}}_{\mathsf{A}}$
15 : $\mathsf{seed}' := \mathsf{Decode}(m')$
16 : $(\mathsf{K}', \mathbf{s}', \mathbf{e}') := \mathsf{H}(\mathsf{ek}_{\mathsf{B}}, \mathsf{seed}')$
17 : $\widehat{\mathsf{ek}}_{\mathsf{B}} := \mathsf{ek}_{\mathsf{B}} + \mathbf{D}^{\top} \cdot \mathbf{s}' + \mathbf{e}'$
18 : return $(\mathsf{aux}, \mathsf{rand}, (\widehat{\mathsf{ek}}_{\mathsf{A}}, \widehat{\mathsf{dk}}_{\mathsf{A}}),$
             $\mathsf{ct}_{\mathsf{A}}, (\mathsf{ek}_{\mathsf{B}}, \widehat{\mathsf{ek}}_{\mathsf{B}}), (\mathsf{K}, \mathsf{K}'))$
```

```
Distribution $\mathcal{D}_{\mathsf{B},1}$
      // Same up till $\mathcal{D}_{\mathsf{B},0}$, line 13
13 : $\widehat{\mathsf{ek}}_{\mathsf{B}} := \mathsf{ek}_{\mathsf{B}} + \mathbf{D}^{\top} \cdot \mathbf{s} + \mathbf{e}$   // Reuse $(\mathsf{K}, \mathbf{s}, \mathbf{e})$ from REnc-B
14 : return $(\mathsf{aux}, \mathsf{rand}, (\widehat{\mathsf{ek}}_{\mathsf{A}}, \widehat{\mathsf{dk}}_{\mathsf{A}}),$
             $\mathsf{ct}_{\mathsf{A}}, (\mathsf{ek}_{\mathsf{B}}, \widehat{\mathsf{ek}}_{\mathsf{B}}), (\mathsf{K}, \mathsf{K}))$
```

```
$\mathsf{H}(\mathbf{u}, \mathsf{seed})$   // Used by all distributions
 1 : if $[\![Q_{\mathsf{H}}[\mathbf{u}, \mathsf{seed}] = \bot]\!]$ then
 2 :   $(\mathsf{K}_0, \mathbf{s}, \mathbf{e}) \xleftarrow{\$} \{0,1\}^{\lambda} \times \chi \times \chi$
 3 :   $Q_{\mathsf{H}}[\mathbf{u}, \mathsf{seed}] \leftarrow (\mathsf{K}_0, \mathbf{s}, \mathbf{e})$
 4 : return $Q_{\mathsf{H}}[\mathbf{u}, \mathsf{seed}]$
```

```
Distribution $\mathcal{D}_{\mathsf{B},2}$
      // Same up till $\mathcal{D}_{\mathsf{B},0}$, line 3
 4 : $\widehat{\mathbf{s}}_{\mathsf{A}} := \widehat{\mathsf{dk}}_{\mathsf{A}}$
 5 : $\widehat{\mathbf{e}}_{\mathsf{A}} := \widehat{\mathsf{ek}}_{\mathsf{A}} - \mathbf{D} \cdot \widehat{\mathbf{s}}_{\mathsf{A}}$   // MLWE noise in $\widehat{\mathsf{ek}}_{\mathsf{A}}$
 6 : $(\mathbf{s}_{\mathsf{B}}, \mathbf{e}_{\mathsf{B}}, \tilde{e}_{\mathsf{B}}) \xleftarrow{\$} \chi \times \chi \times \tilde{\chi}$
 7 : $\mathbf{u}_{\mathsf{B}} := \mathbf{D}^{\top} \cdot \mathbf{s}_{\mathsf{B}} + \mathbf{e}_{\mathsf{B}} \in R_q^k$
 8 : $\mathsf{ek}_{\mathsf{B}} := \mathbf{u}_{\mathsf{B}}$   // Remove $\mathsf{dk}_{\mathsf{B}}$
 9 : $\mathsf{seed} \xleftarrow{\$} \{0,1\}^{\lambda}$
10 : $(\mathsf{K}, \mathbf{s}, \mathbf{e}) := \mathsf{H}(\mathsf{ek}_{\mathsf{B}}, \mathsf{seed})$
11 : $\widehat{\mathsf{ek}}_{\mathsf{B}} := \mathsf{ek}_{\mathsf{B}} + \mathbf{D}^{\top} \cdot \mathbf{s} + \mathbf{e}$   // Update immediately
12 : $m \leftarrow \mathsf{Encode}(\mathsf{seed})$// Generate $\mathsf{ct}_{\mathsf{A}}$ w/o $\mathsf{dk}_{\mathsf{B}}$
13 : $v_{\mathsf{A}} := \widehat{\mathbf{s}}_{\mathsf{A}}^{\top} \cdot \mathsf{ek}_{\mathsf{B}} - \widehat{\mathbf{s}}_{\mathsf{A}}^{\top} \cdot \mathbf{e}_{\mathsf{B}} + \widehat{\mathbf{e}}_{\mathsf{A}}^{\top} \cdot \mathbf{s}_{\mathsf{B}} + \tilde{e}_{\mathsf{B}} + m$
14 : $\mathsf{ct}_{\mathsf{A}} := v_{\mathsf{A}}$
15 : return $(\mathsf{aux}, \mathsf{rand}, (\widehat{\mathsf{ek}}_{\mathsf{A}}, \widehat{\mathsf{dk}}_{\mathsf{A}}),$
             $\mathsf{ct}_{\mathsf{A}}, (\mathsf{ek}_{\mathsf{B}}, \widehat{\mathsf{ek}}_{\mathsf{B}}), (\mathsf{K}, \mathsf{K}))$
```

```
Distribution $\mathcal{D}_{\mathsf{B},3}$
      // Same up till $\mathcal{D}_{\mathsf{B},0}$, line 3
 4 : $(\mathsf{K}, \mathbf{s}, \mathbf{e}) \xleftarrow{\$} \{0,1\}^{\lambda} \times \chi \times \chi$   // Sample w/o RO
 5 : $\widehat{\mathbf{s}}_{\mathsf{A}} := \widehat{\mathsf{dk}}_{\mathsf{A}}$
 6 : $\widehat{\mathbf{e}}_{\mathsf{A}} := \widehat{\mathsf{ek}}_{\mathsf{A}} - \mathbf{D} \cdot \widehat{\mathbf{s}}_{\mathsf{A}}$
 7 : $(\mathbf{s}_{\mathsf{B}}, \mathbf{e}_{\mathsf{B}}, \tilde{e}_{\mathsf{B}}) \xleftarrow{\$} \chi \times \chi \times \tilde{\chi}$
 8 : $\mathbf{u}_{\mathsf{B}} := \mathbf{D}^{\top} \cdot \mathbf{s}_{\mathsf{B}} + \mathbf{e}_{\mathsf{B}} \in R_q^k$
 9 : $\mathsf{ek}_{\mathsf{B}} := \mathbf{u}_{\mathsf{B}}$
10 : $\widehat{\mathsf{ek}}_{\mathsf{B}} := \mathsf{ek}_{\mathsf{B}} + \mathbf{D}^{\top} \cdot \mathbf{s} + \mathbf{e}$
11 : $\mathsf{seed} \xleftarrow{\$} \{0,1\}^{\lambda}$
12 : if $[\![Q_{\mathsf{H}}[\mathsf{ek}_{\mathsf{B}}, \mathsf{seed}] \neq \bot]\!]$ then
13 :   return $\top$   // Output special symbol $\top$
14 : $Q_{\mathsf{H}}[\mathsf{ek}_{\mathsf{B}}, \mathsf{seed}] := (\mathsf{K}_0, \mathbf{s}, \mathbf{e})$   // Program RO
15 : $m \leftarrow \mathsf{Encode}(\mathsf{seed})$
16 : $v_{\mathsf{A}} := \widehat{\mathbf{s}}_{\mathsf{A}}^{\top} \cdot \mathsf{ek}_{\mathsf{B}} - \widehat{\mathbf{s}}_{\mathsf{A}}^{\top} \cdot \mathbf{e}_{\mathsf{B}} + \widehat{\mathbf{e}}_{\mathsf{A}}^{\top} \cdot \mathbf{s}_{\mathsf{B}} + \tilde{e}_{\mathsf{B}} + m$
17 : $\mathsf{ct}_{\mathsf{A}} := v_{\mathsf{A}}$
18 : return $(\mathsf{aux}, \mathsf{rand}, (\widehat{\mathsf{ek}}_{\mathsf{A}}, \widehat{\mathsf{dk}}_{\mathsf{A}}),$
             $\mathsf{ct}_{\mathsf{A}}, (\mathsf{ek}_{\mathsf{B}}, \widehat{\mathsf{ek}}_{\mathsf{B}}), (\mathsf{K}, \mathsf{K}))$
```

Figure 23: Hybrid distributions $\mathcal{D}_{\mathsf{B},0}$ to $\mathcal{D}_{\mathsf{B},3}$ used for the proof of ciphertext simulatability. The text highlighted in blue denotes the main difference between the previous hybrid.

Distribution $\mathcal{D}_{\mathsf{B},4}$

```
     // Same up till $\mathcal{D}_{\mathsf{B},0}$, line 3
4 :  $(\mathsf{K}, \mathbf{s}, \mathbf{e}) \xleftarrow{\$} \{0,1\}^{\lambda} \times \chi \times \chi$
5 :  $\widehat{\mathbf{s}}_{\mathsf{A}} := \widehat{\mathsf{dk}}_{\mathsf{A}}$
6 :  $\widehat{\mathbf{e}}_{\mathsf{A}} := \widehat{\mathsf{ek}}_{\mathsf{A}} - \mathbf{D} \cdot \widehat{\mathbf{s}}_{\mathsf{A}}$
7 :  $(\mathbf{s}_{\mathsf{B}}, \mathbf{e}_{\mathsf{B}}, \tilde{e}_{\mathsf{B}}) \xleftarrow{\$} \chi \times \chi \times \tilde{\chi}$
8 :  $\mathbf{u}_{\mathsf{B}} \xleftarrow{\$} R_q^k$
9 :  $\mathsf{ek}_{\mathsf{B}} := \mathbf{u}_{\mathsf{B}}$
10 : $\widehat{\mathsf{ek}}_{\mathsf{B}} := \mathsf{ek}_{\mathsf{B}} + \mathbf{D}^{\top} \cdot \mathbf{s} + \mathbf{e}$
11 : $\mathsf{seed} \xleftarrow{\$} \{0,1\}^{\lambda}$
12 : if $[\![Q_{\mathsf{H}}[\mathsf{ek}_{\mathsf{B}}, \mathsf{seed}] \neq \bot]\!]$ then
13 :   return $\top$
14 : $Q_{\mathsf{H}}[\mathsf{ek}_{\mathsf{B}}, \mathsf{seed}] := (\mathsf{K}_0, \mathbf{s}, \mathbf{e})$
15 : $m \leftarrow \mathsf{Encode}(\mathsf{seed})$
16 : $v_{\mathsf{A}} := \widehat{\mathbf{s}}_{\mathsf{A}}^{\top} \cdot \mathsf{ek}_{\mathsf{B}} - \widehat{\mathbf{s}}_{\mathsf{A}}^{\top} \cdot \mathbf{e}_{\mathsf{B}} + \widehat{\mathbf{e}}_{\mathsf{A}}^{\top} \cdot \mathbf{s}_{\mathsf{B}} + \tilde{e}_{\mathsf{B}} + m$
17 : $\mathsf{ct}_{\mathsf{A}} := v_{\mathsf{A}}$
18 : return $(\mathsf{aux}, \mathsf{rand}, (\widehat{\mathsf{ek}}_{\mathsf{A}}, \widehat{\mathsf{dk}}_{\mathsf{A}}),$
               $\mathsf{ct}_{\mathsf{A}}, (\mathsf{ek}_{\mathsf{B}}, \widehat{\mathsf{ek}}_{\mathsf{B}}), (\mathsf{K}, \mathsf{K}))$
```

Distribution $\mathcal{D}_{\mathsf{B},5}$

```
     // Same up till $\mathcal{D}_{\mathsf{B},0}$, line 3
4 :  $(\mathsf{K}, \mathbf{s}, \mathbf{e}) \xleftarrow{\$} \{0,1\}^{\lambda} \times \chi \times \chi$
5 :  $\widehat{\mathbf{s}}_{\mathsf{A}} := \widehat{\mathsf{dk}}_{\mathsf{A}}$
6 :  $\widehat{\mathbf{e}}_{\mathsf{A}} := \widehat{\mathsf{ek}}_{\mathsf{A}} - \mathbf{D} \cdot \widehat{\mathbf{s}}_{\mathsf{A}}$
7 :  $\widehat{\mathbf{u}}_{\mathsf{B}} \xleftarrow{\$} R_q^k$
8 :  $\widehat{\mathsf{ek}}_{\mathsf{B}} := \widehat{\mathbf{u}}_{\mathsf{B}}$
9 :  $\mathsf{ek}_{\mathsf{B}} := \widehat{\mathsf{ek}}_{\mathsf{B}} - \mathbf{D}^{\top} \cdot \mathbf{s} - \mathbf{e}$
10 : $\mathsf{seed} \xleftarrow{\$} \{0,1\}^{\lambda}$
11 : if $[\![Q_{\mathsf{H}}[\mathsf{ek}_{\mathsf{B}}, \mathsf{seed}] \neq \bot]\!]$ then
12 :   return $\top$
13 : $Q_{\mathsf{H}}[\mathsf{ek}_{\mathsf{B}}, \mathsf{seed}] := (\mathsf{K}_0, \mathbf{s}, \mathbf{e})$
14 : $m \leftarrow \mathsf{Encode}(\mathsf{seed})$
15 : $(\mathbf{s}_{\mathsf{B}}, \mathbf{e}_{\mathsf{B}}, \tilde{e}_{\mathsf{B}}) \xleftarrow{\$} \chi \times \chi \times \tilde{\chi}$
16 : $v_{\mathsf{A}} := \widehat{\mathbf{s}}_{\mathsf{A}}^{\top} \cdot \mathsf{ek}_{\mathsf{B}} - \widehat{\mathbf{s}}_{\mathsf{A}}^{\top} \cdot \mathbf{e}_{\mathsf{B}} + \widehat{\mathbf{e}}_{\mathsf{A}}^{\top} \cdot \mathbf{s}_{\mathsf{B}} + \tilde{e}_{\mathsf{B}} + m$
17 : $\mathsf{ct}_{\mathsf{A}} := v_{\mathsf{A}}$
18 : return $(\mathsf{aux}, \mathsf{rand}, (\widehat{\mathsf{ek}}_{\mathsf{A}}, \widehat{\mathsf{dk}}_{\mathsf{A}}),$
               $\mathsf{ct}_{\mathsf{A}}, (\mathsf{ek}_{\mathsf{B}}, \widehat{\mathsf{ek}}_{\mathsf{B}}), (\mathsf{K}, \mathsf{K}))$
```

Distribution $\mathcal{D}_{\mathsf{B},6} := \mathcal{D}_{\mathsf{B},1}^{\mathsf{CtxtSim}}$

```
     // Same up till $\mathcal{D}_{\mathsf{B},0}$, line 3
     // Sample from $\widehat{\mathcal{D}}_{\mathsf{RKeyGen\text{-}B}}$
4 :  $(\widehat{\mathbf{s}}_{\mathsf{B}}, \widehat{\mathbf{e}}_{\mathsf{B}}) \xleftarrow{\$} \widehat{\chi} \times \widehat{\chi}$
5 :  $\widehat{\mathbf{u}}_{\mathsf{B}} := \mathbf{D}^{\top} \cdot \widehat{\mathbf{s}}_{\mathsf{B}} + \widehat{\mathbf{e}}_{\mathsf{B}} \in R_q^k$
6 :  $\widehat{\mathsf{ek}}_{\mathsf{B}} := \widehat{\mathbf{u}}_{\mathsf{B}}$    // Implicitly $\widehat{\mathsf{dk}}_{\mathsf{B}} := \widehat{\mathbf{s}}_{\mathsf{B}}$
     // Run $\mathsf{RSimCtxt\text{-}B}(\widehat{\mathsf{ek}}_{\mathsf{B}}, \widehat{\mathsf{ek}}_{\mathsf{A}}, \widehat{\mathsf{dk}}_{\mathsf{A}})$
7 :  $(\mathsf{K}, \mathbf{s}, \mathbf{e}) \xleftarrow{\$} \{0,1\}^{\lambda} \times \chi \times \chi$
8 :  $\widehat{\mathbf{s}}_{\mathsf{A}} := \widehat{\mathsf{dk}}_{\mathsf{A}}$
9 :  $\widehat{\mathbf{e}}_{\mathsf{A}} := \widehat{\mathsf{ek}}_{\mathsf{A}} - \mathbf{D} \cdot \widehat{\mathbf{s}}_{\mathsf{A}}$
10 : $\mathsf{ek}_{\mathsf{B}} := \widehat{\mathsf{ek}}_{\mathsf{B}} - \mathbf{D}^{\top} \cdot \mathbf{s} - \mathbf{e}$
11 : $\mathsf{seed} \xleftarrow{\$} \{0,1\}^{\lambda}$
12 : if $[\![Q_{\mathsf{H}}[\mathsf{ek}_{\mathsf{B}}, \mathsf{seed}] \neq \bot]\!]$ then
13 :   return $\top$
14 : $Q_{\mathsf{H}}[\mathsf{ek}_{\mathsf{B}}, \mathsf{seed}] := (\mathsf{K}_0, \mathbf{s}, \mathbf{e})$
15 : $m \leftarrow \mathsf{Encode}(\mathsf{seed})$
16 : $(\mathbf{s}_{\mathsf{B}}, \mathbf{e}_{\mathsf{B}}, \tilde{e}_{\mathsf{B}}) \xleftarrow{\$} \chi \times \chi \times \tilde{\chi}$
17 : $v_{\mathsf{A}} := \widehat{\mathbf{s}}_{\mathsf{A}}^{\top} \cdot \mathsf{ek}_{\mathsf{B}} - \widehat{\mathbf{s}}_{\mathsf{A}}^{\top} \cdot \mathbf{e}_{\mathsf{B}} + \widehat{\mathbf{e}}_{\mathsf{A}}^{\top} \cdot \mathbf{s}_{\mathsf{B}} + \tilde{e}_{\mathsf{B}} + m$
18 : $\mathsf{ct}_{\mathsf{A}} := v_{\mathsf{A}}$
19 : return $(\mathsf{aux}, \mathsf{rand}, (\widehat{\mathsf{ek}}_{\mathsf{A}}, \widehat{\mathsf{dk}}_{\mathsf{A}}),$
               $\mathsf{ct}_{\mathsf{A}}, (\mathsf{ek}_{\mathsf{B}}, \widehat{\mathsf{ek}}_{\mathsf{B}}), (\mathsf{K}, \mathsf{K}))$
```

Figure 24: Hybrid distributions $\mathcal{D}_{\mathsf{B},4}$ to $\mathcal{D}_{\mathsf{B},6}$ used for the proof of ciphertext simulatability. The text highlighted in blue denotes the main difference between the previous hybrid.

Distribution $\mathcal{D}_{\mathsf{B},4}$: The difference between the prior distribution is that we sample a random $\mathbf{u}_{\mathsf{B}}$ from $\mathcal{R}_q^k$ and sets user $\mathsf{B}$'s key as $\mathsf{ek}_{\mathsf{B}} := \mathbf{u}_{\mathsf{B}}$. Recall in the previous game, $\mathbf{u}_{\mathsf{B}}$ was set as $\mathsf{MLWE}$ instances. Since $\mathbf{s}_{\mathsf{B}}$ and $\mathbf{e}_{\mathsf{B}}$ are partially leaked via $\mathsf{ct}_{\mathsf{A}} = v_{\mathsf{A}}$ and $\widehat{\mathsf{dk}}_{\mathsf{A}} = \widehat{\mathbf{s}}_{\mathsf{A}}$, we cannot rely on the standard $\mathsf{MLWE}$ assumption to argue indistinguishability of the two distributions.[13] However, noticing that $\tilde{e} \xleftarrow{\$} \tilde{\chi}$ is information theoretically hidden to $\mathcal{A}$ conditioning on the game not aborting, we can rely instead on the *hint* $\mathsf{MLWE}$ assumption. Concretely, we can construct an adversary $\mathcal{B}_{\mathsf{hint\text{-}MLWE}}$ against the $\mathsf{hint\text{-}MLWE}_{q,k,1,\chi,\tilde{\chi},\mathcal{F}_{\mathsf{sim}}}$ problem such that

$$|\epsilon_3 - \epsilon_4| \leqslant \mathsf{Adv}^{\mathsf{hint\text{-}MLWE}}_{\mathcal{B}_{\mathsf{hint\text{-}MLWE}}}(1^\lambda),$$

where recall a sample from $\mathcal{F}_{\mathsf{sim}}$ has the form $[-\widehat{\mathbf{s}}^\top | \widehat{\mathbf{e}}^\top]$ with $\widehat{\mathbf{s}}, \widehat{\mathbf{e}} \xleftarrow{\$} \widehat{\chi}$. In more detail, $\mathcal{B}_{\mathsf{hint\text{-}MLWE}}$ obtains $\left(\mathbf{D}^\top, \mathbf{u}_{\mathsf{B}}, [-\widehat{\mathbf{s}}^\top | \widehat{\mathbf{e}}^\top], h = [-\widehat{\mathbf{s}}^\top | \widehat{\mathbf{e}}^\top] \begin{bmatrix} \widehat{\mathbf{e}}_{\mathsf{B}} \\ \widehat{\mathbf{s}}_{\mathsf{B}} \end{bmatrix} + \tilde{e}_{\mathsf{B}}\right)$ as input, where $\mathbf{u}_{\mathsf{B}}$ is either random or of the form $\mathbf{D}^\top \cdot \mathbf{s}_{\mathsf{B}} + \mathbf{e}_{\mathsf{B}}$. Here $h$ is the *hint*. It then simulates $(\widehat{\mathsf{ek}}_{\mathsf{A}}, \widehat{\mathsf{dk}}_{\mathsf{A}})$ by setting $(\widehat{\mathbf{s}}_{\mathsf{A}}, \widehat{\mathbf{e}}_{\mathsf{A}}) := (\widehat{\mathbf{s}}, \widehat{\mathbf{e}})$ and computing the ciphertext $\mathsf{ct}_{\mathsf{A}} = v_{\mathsf{A}} = \widehat{\mathbf{s}}^\top \cdot \mathbf{u}_{\mathsf{B}} + h + m$.

Distribution $\mathcal{D}_{\mathsf{B},5}$: The only difference from the prior distribution is that we sample user $\mathsf{B}$'s updated key $\widehat{\mathsf{ek}}_{\mathsf{B}} = \widehat{\mathbf{u}}_{\mathsf{B}}$ uniformly sample and then set $\mathsf{ek}_{\mathsf{B}}$. In the previous distribution, this was performed in the opposite direction. Since this produces an identical distribution, we have

$$\epsilon_4 = \epsilon_5.$$

Distribution $\mathcal{D}_{\mathsf{B},6}$: The only difference from the prior distribution is that we sample user $\mathsf{B}$'s updated key $\widehat{\mathsf{ek}}_{\mathsf{B}} = \widehat{\mathbf{u}}_{\mathsf{B}}$ as a valid $\mathsf{MLWE}$ instance. It is straight forward to see that the two distributions is indistinguishable under the $\mathsf{MLWE}$ assumption. Formally, we can construct an adversary $\mathcal{B}_{\mathsf{MLWE}}$ against the $\mathsf{MLWE}_{q,k,\widehat{\chi}}$ problem such that

$$|\epsilon_5 - \epsilon_6| \leqslant \mathsf{Adv}^{\mathsf{MLWE}}_{\mathcal{B}_{\mathsf{MLWE}}}(1^\lambda).$$

Lastly, it can be checked that the final distribution $\mathcal{D}_{\mathsf{B},6}$ is identical to $\mathcal{D}^{\mathsf{CtxtSim}}_{\mathsf{B},1}$ as the generation of the updated key $(\widehat{\mathsf{ek}}_{\mathsf{B}}, \widehat{\mathsf{dk}}_{\mathsf{B}})$ can be pushed before generating the non-updated key $(\mathsf{ek}_{\mathsf{B}}, \mathsf{dk}_{\mathsf{B}})$. Moreover, line 7 onward is identical to $\mathsf{RSimCtxt\text{-}B}$ as desired. The bound in the theorem statement follows by collecting all the bounds. This completes the proof. □

We note that our definition of RKEM is in the standard model, while our construction is in the random oracle model (ROM). We thus make an implicit assumption that the correctness and security definitions of RKEM are adapted in the standard way to allow adversaries to make RO queries. As common practice, we then assume the RKEM instantiated with a concrete hash function retains the same security, and view it as an RKEM in the standard model when using it as a building block to generically construct a CKA.

## 6.3 Optimizing Katana with Bit-Dropping

We can minimize the size of the ciphertext by performing bit-dropping, similarly to Kyber [SAB+22]. The additional notations we use and our optimized construction is provide in Table 3 and Fig. 25, respectively.

It can be checked that this optimization only affects the correctness of Katana. Specifically, the proof of FS-IND-CPA security and ratchet simulatability remains unchanged, except for the hybrids we rely on the correctness of the scheme. We therefore only provide the proof of correctness below. Below, similarly to Kyber, we consider an average case bound on the error $\delta \xleftarrow{\$} \chi_{\mathsf{round}}$ induced by bit-dropping for tighter concrete parameters.

[13] It is worth noting that at a high level, this is the argument where [ACD19] made a mistake by arguing indistinguishability solely on MLWE.

| Notations | Explanation |
|---|---|
| $d$ | Amount of bit dropping performed on ciphertext such that $d < \lceil \log_2(q) \rceil$ |
| $q_d$ | Rounded modulus satisfying $q_d := 2^d$ |
| $\mathsf{Compress}_q$<br>$\mathsf{Decompress}_q$ | Rounding operations from Kyber [SAB+22] |

Table 3: Parameters and notations regarding bit-dropping optimization. See Section 2.2.2 for the definitions of $\mathsf{Compress}_q$ and $\mathsf{Decompress}_q$.

```
REnc-P$(\widehat{\mathsf{ek}}_{\bar{\mathsf{P}}}, \mathsf{dk}_\mathsf{P})$
$(\mathbf{u}_\mathsf{P}, \mathbf{s}_\mathsf{P}) := \mathsf{dk}_\mathsf{P}$
$\mathsf{seed} \xleftarrow{\$} \{0,1\}^\lambda$
$m \leftarrow \mathsf{Encode}(\mathsf{seed})$   // $m \in R_q$
$(\mathsf{K}, \mathbf{s}, \mathbf{e}) := \mathsf{H}(\mathbf{u}_\mathsf{P}, \mathsf{seed})$
$\tilde{e}_\mathsf{P} \xleftarrow{\$} \tilde{\chi}$
$v_{\bar{\mathsf{P}}} := \widehat{\mathsf{ek}}_{\bar{\mathsf{P}}}^\top \cdot \mathbf{s}_\mathsf{P} + \tilde{e}_\mathsf{P} + m \in R_q$
$\mathsf{ct}_{\bar{\mathsf{P}}} := \mathsf{Compress}_q(v_{\bar{\mathsf{P}}}, d) \in R_{q_d}$
$\widehat{\mathsf{dk}}_\mathsf{P} := \mathbf{s}_\mathsf{P} + \mathbf{s} \in R_q^k$   // Update and erase $\mathsf{dk}_\mathsf{P}$
return $(\mathsf{ct}_{\bar{\mathsf{P}}}, \mathsf{K}, \widehat{\mathsf{dk}}_\mathsf{P})$
```

```
RDec-P$(\widehat{\mathsf{dk}}_\mathsf{P}, \mathsf{ct}_\mathsf{P}, \mathsf{ek}_{\bar{\mathsf{P}}})$
$\mathsf{ct}'_\mathsf{P} := \mathsf{Decompress}_q(\mathsf{ct}_\mathsf{P}, d) \in R_q$
$m := \mathsf{ct}'_\mathsf{P} - \mathsf{ek}_{\bar{\mathsf{P}}}^\top \cdot \widehat{\mathsf{dk}}_\mathsf{P} \in R_q$
$\mathsf{seed} := \mathsf{Decode}(m)$
$(\mathsf{K}, \mathbf{s}, \mathbf{e}) := \mathsf{H}(\mathsf{ek}_{\bar{\mathsf{P}}}, \mathsf{seed})$
// Update $\mathsf{ek}_{\bar{\mathsf{P}}}$
if $[\![\mathsf{P} = \mathsf{A}]\!]$ then
    $\widehat{\mathsf{ek}}_\mathsf{B} := \mathsf{ek}_\mathsf{B} + \mathbf{D}^\top \cdot \mathbf{s} + \mathbf{e}$
else   // $\mathsf{P} = \mathsf{B}$
    $\widehat{\mathsf{ek}}_\mathsf{A} := \mathsf{ek}_\mathsf{A} + \mathbf{D} \cdot \mathbf{s} + \mathbf{e}$
return $(\mathsf{K}, \widehat{\mathsf{ek}}_\mathsf{B})$
```

Figure 25: Our RKEM $\Pi_{\mathsf{RKEM}}$ with the bit-dropping optimization. The differences are highlighted in blue. RSetup and RKeyGen-P are defined identically to those in Fig. 18.

**Lemma 6.4 (Correctness with bit-dropping).** *Our optimized* RKEM Katana *is correct assuming*

$$\Pr\left[\|\hat{\mathbf{s}}^\top \cdot \mathbf{e} - \hat{\mathbf{e}}^\top \cdot \mathbf{s} + \tilde{e} + \delta\|_\infty \leqslant q/4\right] = 1 - \mathsf{negl}(\lambda), \tag{3}$$

*where the probability is taken over the randomness to sample* $(\mathbf{s}, \mathbf{e}) \xleftarrow{\$} \chi \times \chi$, $(\hat{\mathbf{s}}, \hat{\mathbf{e}}) \xleftarrow{\$} \hat{\chi} \times \hat{\chi}$, $\tilde{e} \xleftarrow{\$} \tilde{\chi}$, *and* $\delta \xleftarrow{\$} \chi_{\mathsf{round}}$. *Here,* $\chi_{\mathsf{round}}$ *is some distribution over* $R_q$ *such that* $\Pr[\delta \xleftarrow{\$} \chi_{\mathsf{round}} : \|\delta\|_\infty \leqslant \lfloor \frac{q}{2^{d+1}} \rceil] = 1$.

*Proof.* Since the encapsulation key is unchanged, correctness of update key distribution follows from Lemma 6.1. Let us show correctness with updated keys. Again, due to symmetry, we only focus on the case where user A runs RDec-A. First, observe that we have

$$\begin{aligned} v_\mathsf{A} &= \widehat{\mathsf{ek}}_\mathsf{A}^\top \cdot \mathbf{s}_\mathsf{B} + \tilde{e}_\mathsf{B} + m \\ &= (\mathbf{D} \cdot \hat{\mathbf{s}}_\mathsf{A} + \hat{\mathbf{e}}_\mathsf{A})^\top \cdot \mathbf{s}_\mathsf{B} + \tilde{e}_\mathsf{B} + m \\ &= (\mathbf{D}^\top \cdot \hat{\mathbf{s}}_\mathsf{B} + \mathbf{e}_\mathsf{B})^\top \cdot \mathbf{s}_\mathsf{A} + \hat{\mathbf{e}}_\mathsf{A}^\top \cdot \mathbf{s}_\mathsf{B} - \mathbf{e}_\mathsf{B}^\top \cdot \hat{\mathbf{s}}_\mathsf{A} + \tilde{e}_\mathsf{B} + m \\ &= \mathsf{ek}_\mathsf{B}^\top \cdot \widehat{\mathsf{dk}}_\mathsf{A} + \underbrace{\hat{\mathbf{e}}_\mathsf{A}^\top \cdot \mathbf{s}_\mathsf{B} - \mathbf{e}_\mathsf{B}^\top \cdot \hat{\mathbf{s}}_\mathsf{A} + \tilde{e}_\mathsf{B}}_{=:z} + m \end{aligned}$$

Plugging this into the decryption equation, we have

$$\begin{aligned} \mathsf{Decompress}_q(\mathsf{ct}_\mathsf{A}, d) - \mathsf{ek}_\mathsf{B}^\top \cdot \widehat{\mathsf{dk}}_\mathsf{A} &= \mathsf{Decompress}_q(\mathsf{Compress}_q(v_\mathsf{A}), d) - v_\mathsf{A} + z + m \\ &= m + z + \delta, \end{aligned}$$

where $\delta \in R_q$ such that $\|\delta\|_\infty \leqslant \lfloor \frac{q}{2^{d+1}} \rceil$ from Lemma 2.4 . If each coefficient of $z + \delta \in R_q$ is smaller than $q/4$ (i.e., $\|z\|_\infty \leqslant q/4$), Decode will correctly decode to $m$ as desired. □

Lastly, note that similarly to Kyber [SAB+22], we do not perform bit-dropping on the encapsulation key. Since this seems to require a non-trivial analysis, unlike the simpler bit-dropping on the ciphertext, we leave this optimization for future work.

## 6.4 Concrete Parameter Selection

We provide a concrete instantiation of Katana. For reference, we recall all the requirements our parameters (see Tables 2 and 3) must satisfy, where note that some requirements are subsumed by others. The first requirement stems from correctness (cf. Lemma 6.4), the second and third stem from FS-IND-CPA security (cf. Theorem 6.2), and the second and forth stem from ratchet simulatability (cf. Theorem 6.3).

(R1) The correctness error is below $2^{-\lambda}$ (see Lemma 6.4, Eq. (3)).

(R2) The $\mathsf{MLWE}_{q,k,\chi}$ and $\mathsf{MLWE}_{q,k,\tilde{\chi}}$ problems are hard.

(R3) The $\mathsf{hint}\text{-}\mathsf{MLWE}_{q,k,2k,\chi,\chi,\mathcal{F}_{\mathsf{cpa}}}$ problem is hard, where $\mathcal{F}_{\mathsf{cpa}} := \mathcal{U}(\{\mathbf{I}_{2k\times 2k}\})$, i.e., a distribution always outputting the identity matrix $\mathbf{I}_{2k\times 2k} \in R_q^{2k\times 2k}$.

(R4) The $\mathsf{hint}\text{-}\mathsf{MLWE}_{q,k,1,\chi,\tilde{\chi},\mathcal{F}_{\mathsf{sim}}}$ problem is hard, where $\mathcal{F}_{\mathsf{sim}}$ is a distribution over $R_q^{1\times 2k}$ that outputs $[-\widehat{\mathbf{s}}^\top|\widehat{\mathbf{e}}^\top]$ with $\widehat{\mathbf{s}}, \widehat{\mathbf{e}} \xleftarrow{\$} \widehat{\chi}$.

We study each condition in a separate subsection.

### 6.4.1 Correctness

We first study Item (R1). Our analysis of correctness is similar to the one of Kyber, and we use similar techniques. By symmetry, we can see that all integer coefficients of $\widehat{\mathbf{s}}^\top \cdot \mathbf{e} - \widehat{\mathbf{e}}^\top \cdot \mathbf{s} + \tilde{e} + \delta$ follow the same distribution, although they are not independent. We start by studying an arbitrary coefficient of $\widehat{\mathbf{s}}^\top \cdot \mathbf{e} - \widehat{\mathbf{e}}^\top \cdot \mathbf{s} + \tilde{e} + \delta$, let us note it $y_i$. If we completely ignore rounding, it is clear that $y_i$ is distributed as:

$$y_i \sim [2\,k\,n] \cdot \widehat{\chi}_0 \cdot \chi_0 + \tilde{\chi}_0, \tag{4}$$

where $\widehat{\chi}_0$ (resp. $\chi_0$, resp. $\tilde{\chi}_0$) is the distribution of each integer coefficient of $\widehat{\chi}$ (resp. $\chi$, resp. $\tilde{\chi}$). Now, let us note $\chi_{\mathsf{round}}$ the distribution entailed by rounding $v$ and $\chi_{0,\mathsf{round}}$ as the distribution of each integer coefficient. Since, we use exactly the same rounding method as in Kyber, we may re-employ their analysis, [SAB+22, see Eqs. (7) and (8)] in order to characterize the resulting distributions. Eq. (4) may then be adapted as follows:

$$y_i \sim [2\,k\,n] \cdot \widehat{\chi}_0 \cdot \chi_0 + \tilde{\chi}_0 + \chi_{0,\mathsf{round}}. \tag{5}$$

We compute explicitly the distribution in Eq. (5) using a Sage script. To keep the computation tractable, we continuously apply tailcutting (over a set of weight $\leqslant 2^{-\lambda}/n$). Finally, we use the union bound to ensure that $\|\widehat{\mathbf{s}}^\top \cdot \mathbf{e} - \widehat{\mathbf{e}}^\top \cdot \mathbf{s} + \tilde{e} + \delta\|_\infty \leqslant q/4$ with overwhelming probability.

### 6.4.2 FS-IND-CPA Security

Next, we study Items (R2) and (R3), which underlie FS-IND-CPA security. We note that Item (R3) strictly subsumes Item (R2), therefore we may study Item (R3) alone. If $\chi$ follows a Gaussian distribution of parameter $\sigma$, then the hint-MLWE reduction (cf. Theorem 2.3) tells us that $\mathsf{hint}\text{-}\mathsf{MLWE}_{q,k,2k,\chi,\chi,\mathcal{F}_{\mathsf{cpa}}}$ is at least as hard as $\mathsf{MLWE}_{q,k,2k,\chi'}$, where $\chi'$ is the discrete Gaussian of parameter $\sigma_0$:

$$\frac{1}{\sigma_0^2} = 2\left(\frac{1}{\sigma^2} + \frac{s_1(\mathbf{I}_{2k})^2}{\sigma^2}\right) = \frac{4}{\sigma^2} \tag{6}$$

In our case, we rely on two heuristics:

*Heuristic 1: Replace Gaussians.* While Theorem 2.3 holds when the secret and noise are sampled from Gaussian distributions, we assume that this is also the case with non-Gaussian distributions of equivalent variance $\sigma^2$. In our case, we sample $\tilde{e}_{\mathsf{P}} \xleftarrow{\$} \tilde{\chi}$ as a sum of uniforms and $\mathbf{s} \xleftarrow{\$} \chi$ from a binomial distribution (see Section 6.4.4 for discussion); both types of distributions become "Gaussian-like" for some parameter regimes. It has also been argued in Raccoon [dPEK$^+$23, dPKPR24] that in Rényi divergence-based arguments, sum of uniforms behave similarly to discrete Gaussians of identical variance $\sigma^2$. For the present analysis, we conjecture that this is also the case in $\mathsf{hint\text{-}MLWE}$.

*Heuristic 2: Remove factor* 2. We remove the factor 2 in Eq. (6). This is motivated by the fact that in [KLSS23], this factor seems to appear in order to simplify a smoothing parameter argument for discrete Gaussians. In particular, we can see that (i) if there is no hint then it is clear that the factor 2 is superfluous, and (ii) in our case, since the underlying distributions are not discrete Gaussians, this factor 2 serves no apparent purpose.

Under Heuristics 1 and 2, Eq. (6) simplifies to $\sigma_0 = \sigma/\sqrt{2}$, where $\sigma$ is the standard variation of $\chi$, and $\chi$ is not necessarily discrete Gaussian. We may then estimate the hardness of $\mathsf{MLWE}_{q,k,2k,\chi'}$ using the lattice estimator[14].

### 6.4.3 Ratchet Simulatability

Finally, we study Items (R2) and (R4), which underlie ratchet simulatability. Item (R4) strictly subsumes Item (R2) and is therefore studied alone. Using the same reasoning as above, under Heuristics 1 and 2, we may say that $\mathsf{hint\text{-}MLWE}_{q,k,1,\chi,\tilde{\chi},\mathcal{F}_{\mathsf{sim}}}$ is at least as hard as $\mathsf{MLWE}_{q,k,2k,\chi'}$, where $\chi'$ is the discrete Gaussian of parameter $\sigma_0$:[15]

$$\frac{1}{\sigma_0^2} = \frac{1}{\sigma^2} + \frac{s_1(\mathbf{M})^2}{\tilde{\sigma}^2} \tag{7}$$

where $\mathbf{M} = \left[-\hat{\mathbf{s}}^\top \mid \hat{\mathbf{e}}^\top\right]$, $\sigma$ (resp. $\tilde{\sigma}$, resp. $\hat{\sigma}$) is the standard variation of $\chi$ (resp. $\tilde{\chi}$, resp. $\hat{\chi}$), and $\chi$ (resp. $\tilde{\chi}$, resp. $\hat{\chi}$) is not necessarily Gaussian. In addition to Heuristics 1 and 2, we rely on a third heuristic:

*Heuristic 3: Approximate singular norm.* Instead of computing the worst-case bound $s_1(\mathbf{M}) = \max \frac{\|\mathbf{M}\cdot\mathbf{z}\|}{\|\mathbf{z}\|}$, we estimate the *average-case* value $\mathbb{E}\left[\frac{\|\mathbf{M}\cdot\mathbf{z}\|}{\|\mathbf{z}\|}\right]$. We observe that $\mathbb{E}\left[\|\mathbf{M}\cdot\mathbf{z}\|^2\right] = 2\,k\,n^2\,\sigma^2\,\hat{\sigma}^2$ and $\mathbb{E}\left[\|\mathbf{z}\|^2\right] = 2\,k\,n\,\sigma^2$. Therefore we heuristically estimate:

$$\mathbb{E}\left[\frac{\|\mathbf{M}\cdot\mathbf{z}\|^2}{\|\mathbf{z}\|^2}\right] \approx \frac{\mathbb{E}\left[\|\mathbf{M}\cdot\mathbf{z}\|^2\right]}{\mathbb{E}\left[\|\mathbf{z}\|^2\right]} = n\,\hat{\sigma}^2 \tag{8}$$

The reason why this is heuristic is because the expected value is only multiplicative for *independent* random variables. Since $\mathbf{M}\cdot\mathbf{z}$ and $\mathbf{z}$ are high-dimensional vectors, their norms are tightly concentrated around their expected values. Therefore we replace $s_1(\mathbf{M})^2$ in Eq. (7) by $n\,\hat{\sigma}^2$. Since $\hat{\sigma} = 2\sigma$, Eq. (7) becomes:

$$\frac{1}{\sigma_0^2} = \frac{1}{\sigma^2} + \frac{4\,n\,\sigma^2}{\tilde{\sigma}^2}. \tag{9}$$

Eq. (9) is minimized when $\tilde{\sigma} = 2\sqrt{n}\,\sigma^2$, in which case $\sigma_0 = \sigma/\sqrt{2}$. Interestingly, this is exactly the same value of $\sigma_0$ as the one obtained in the study of $\mathsf{FS\text{-}IND\text{-}CPA}$ security. Again, we use the lattice estimator to estimate the hardness of the underlying $\mathsf{MLWE}$ assumption.

[14] https://github.com/malb/lattice-estimator

[15] Recall that $s_1(\mathbf{M}^\top \cdot \mathbf{M}) = s_1(\mathbf{M})^2$.

| Target<br>$\lambda$ | CoreSVP<br>hardness | $n$ | $k$ | $q$ | $q_d$ | $\chi$ | $\tilde{\chi}$ | $d$ | $\lvert\mathsf{ek}\rvert$ | $\lvert\mathsf{ct}\rvert$ | $\lvert\mathsf{ek}\rvert + \lvert\mathsf{ct}\rvert$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 128 | 100 | 256 | 2 | 7681 | 8 | CBD(4) | SU(7, 4) | 3 | 832 | 48 | 880 |
| 192 | 158 | - | 3 | 10753 | - | - | - | - | 1344 | 72 | 1416 |
| 256 | 215 | - | 4 | 15361 | - | - | - | - | 1792 | 96 | 1888 |

Table 4: Parameter sets for Katana. The sizes of ek and ct are in bytes. The symbol "-" in a cell indicates that it has the same value as the cell directly above in the table.

### 6.4.4 Summary

We now specify the parameter sets. We introduce the error distributions that we use to instantiate our scheme:

- $\chi = \mathrm{CBD}_\eta$ is a centered binomial distribution, that is $\mathrm{CBD}_\eta = [\eta] \cdot (\mathcal{B} - \mathcal{B})$, where $\mathcal{B} = \mathcal{U}(\{0, 1\})$ is the Bernoulli distribution of parameter $1/2$. This distribution is used in Kyber [SAB+22].
- $\tilde{\chi} = \mathrm{SU}(u, T)$ is the sum of $T$ uniformly random variates over $\{-2^{u-1}, \ldots, 2^{u-1} - 1\}$, that is $\mathrm{SU}(u, T) = [T] \cdot \mathcal{U}(\{-2^{u-1}, \ldots, 2^{u-1} - 1\})$. This distribution is used in Raccoon [dPEK+23].

These distributions were chosen because they are easy to implement in a constant-time manner, unlike Gaussian distributions. An additional silver lining of sums of uniforms is that they provide slightly better correctness bounds than Gaussians (of identical variance), due to their tails decreasing faster. Finally, we propose parameters sets in Table 4, which target 128, 192 and 256 bits of security. We recall that the CoreSVP hardness is a crude measure of the bit-security of a lattice problem and that it ignores several polynomial factors. These factors typically represent about 30 bits of security. We choose primes $q$ that are NTT-friendly.

# 7 Efficiency Analysis of Triple Ratchet

We now examine the effects of our two main improvements — erasure coding and a better RKEM — on the efficiency of attaining *post-quantum* PCS (recall efficient classical PCS is inherited from using Signal's Double Ratchet protocol). For our RKEM improvement, the gain is clear as it reduces the combined encapsulation key and ciphertext size by approximately 37% when compared to a standard KEM (Kyber) at a comparable security level. For our coding improvement, PQ3 is a natural benchmark. It turns out that the gain (or loss) depends on the communication pattern, and to emphasize this point, in addition to comparing PQ3 to TR with Katana we also compare it with TR using a trivial RKEM based on Kyber-768. We focus on communication cost but note that higher efficiency can yield higher security: a protocol that is more efficient in communication cost can yield shorter epochs and faster PCS healing for a fixed communication overhead budget.

## 7.1 Effect of Our RKEM on Communication Costs

We can use Kyber to construct a trivial RKEM (cf. Appendix A). Compared to such RKEM, our optimized RKEM Katana has significantly smaller combined encapsulation key and ciphertext size at the same security level, and this leads directly to a smaller amount of data that must be transferred between parties in order to obtain PCS when building a post-quantum CKA. This can be seen by comparing the last two columns in Table 5, where the reduction in per message overhead comes entirely from the fact that Kyber-768 requires the transfer of 2272B per epoch where Katana only requires the transfer of 1416B (see also Table 4).

|  | PQ3 | TR with<br>Kyber-768 | TR with<br>Katana ($\lambda = 192$) |
|---|---|---|---|
| $p = 0$ | 6 488 | 9 000 | 6 500 |
| $p = 0.5$ | 11 176 | 9 540 | 6 890 |
| $p = 0.9$ | 48 680 | 13 860 | 10 010 |

Table 5: Expected communication cost in bytes to attain PCS for PQ3 and TR. See text for the parameter $p$. PQ3 is assumed to send a Kyber-768 encapsulation key and ciphertext every 50 messages. TR with Kyber-768 (resp. Katana) uses a post-quantum CKA based on Kyber-768 (resp. Katana with $\lambda = 192$). This includes base message cost of 36B for PQ3 and 46B for TR to account for the overhead of sending counters and DH keys but excludes the 64B signature used by PQ3 for fair comparison.

## 7.2 Effect of Chunk Encoding on Communication Costs

The benefits of our use of erasure codes is more nuanced and depends on messaging behavior. To understand this, recall that PQ3 attains post-quantum PCS by repeatedly sending Kyber encapsulation key and ciphertext messages until receiving an acknowledgement [Jac].

In a perfectly balanced conversation where every send is followed by a receive, this repeated sending imposes no cost and PQ3 actually has a structural advantage over TR because $\Delta^{PQ3}_{\mathsf{PCS}} = 2$ where $\Delta^{\mathsf{TR}}_{\mathsf{PCS}} = 3$. Real conversations are unbalanced and Signal's use of encrypted typing indicators - small, frequent messages that do not elicit a response - amplify this imbalance. Using PQ3 in this setting would lead to a large number of repeated KEM messages. The cost is significant and this can negatively impact a user's experience when it happens. Another Signal feature, linked devices, exacerbates the costs of repeated messages even further. Signal users often leave linked laptops and desktops off for hours or days, and each logical conversation with a user maintains separate protocol sessions with each of that user's linked devices. When someone leaves their laptop off overnight - or loses it - it can impose a significant cost on everyone messaging them. The resulting costs and user experience are unacceptable for the Signal team.

We illustrate these costs in Table 5 where we report the expected number of bytes transferred to attain PCS assuming a simple model of unbalanced communication where every sender has a probability $p$ of sending another message before receiving all incoming messages, independent of previous events. We compare PQ3 and two instantiations of TR. One is the TR where we use Signal's Double Ratchet protocol as the classical CKA with curve25519 and the post-quantum CKA based on the trivial RKEM with Kyber-768. The other TR, which is our main protocol, replaces the post-quantum CKA with one based on Katana at $\lambda = 192$. Chunk sizes are chosen so that all protocols attain PCS in 50 messages under ideal conditions. In row one we use $p = 0$ to capture perfectly balanced communication The advantage of PQ3 over the trivial RKEM, due to its smaller $\Delta_{\mathsf{PCS}}$, is clear, as is the advantage of Katana due to the smaller message size. In row two we use $p = 0.5$ to conservatively approximate the sending behavior of two online parties using typing indicators and read receipts, and we see that at this point both instantiations of TR have an advantage over PQ3. Finally, in row 3, we use $p = 0.9$ to approximate the behavior of a device that is offline for hours at a time, where PQ3 is more than 4 times as expensive as TR with Katana.

**Acknowledgement.** The third author was partially supported by JST, CREST Grant Number JPMJCR22M1, Japan. We also thank the EUROCRYPT'25 reviewers for their thorough feedbacks.

# References

[ACD19] Joël Alwen, Sandro Coretti, and Yevgeniy Dodis. The double ratchet: Security notions, proofs, and modularization for the Signal protocol. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part I*, volume 11476 of *LNCS*, pages 129–158, Darmstadt, Germany, May 19–23, 2019. Springer, Cham, Switzerland.

[AHKM22] Joël Alwen, Dominik Hartmann, Eike Kiltz, and Marta Mularczyk. Server-aided continuous group key agreement. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022*, pages 69–82, Los Angeles, CA, USA, November 7–11, 2022. ACM Press.

[AJM22] Joël Alwen, Daniel Jost, and Marta Mularczyk. On the insider security of MLS. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part II*, volume 13508 of *LNCS*, pages 34–68, Santa Barbara, CA, USA, August 15–18, 2022. Springer, Cham, Switzerland.

[App24] Apple Security Engineering and Architecture (SEAR). iMessage with PQ3: The new state of the art in quantum-secure messaging at scale, 2 2024. Available at https://security.apple.com/blog/imessage-pq3/.

[BBD$^+$21] Karthikeyan Bhargavan, Abhishek Bichhawat, Quoc Huy Do, Pedram Hosseyni, Ralf Küsters, Guido Schmitz, and Tim Würtele. DY*: A modular symbolic verification framework for executable cryptographic protocol code. In *2021 IEEE European Symposium on Security and Privacy*, pages 523–542, Vienna, Austria, September 6–10, 2021. IEEE Computer Society Press.

[BFG$^+$20] Jacqueline Brendel, Marc Fischlin, Felix Günther, Christian Janson, and Douglas Stebila. Towards post-quantum security for Signal's X3DH handshake. In Orr Dunkelman, Michael J. Jacobson, Jr., and Colin O'Flynn, editors, *SAC 2020*, volume 12804 of *LNCS*, pages 404–430, Halifax, NS, Canada (Virtual Event), October 21-23, 2020. Springer, Cham, Switzerland.

[BFG$^+$22a] Alexander Bienstock, Jaiden Fairoze, Sanjam Garg, Pratyay Mukherjee, and Srinivasan Raghuraman. A more complete analysis of the Signal double ratchet algorithm. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part I*, volume 13507 of *LNCS*, pages 784–813, Santa Barbara, CA, USA, August 15–18, 2022. Springer, Cham, Switzerland.

[BFG$^+$22b] Jacqueline Brendel, Rune Fiedler, Felix Günther, Christian Janson, and Douglas Stebila. Post-quantum asynchronous deniable key exchange and the Signal handshake. In Goichiro Hanaoka, Junji Shikata, and Yohei Watanabe, editors, *PKC 2022, Part II*, volume 13178 of *LNCS*, pages 3–34, Virtual Event, March 8–11, 2022. Springer, Cham, Switzerland.

[BJKS24] 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 Davide Balzarotti and Wenyuan Xu, editors, *USENIX Security 2024*, Philadelphia, PA, USA, August 14–16, 2024. USENIX Association.

[BRV20] Fatih Balli, Paul Rösler, and Serge Vaudenay. Determining the core primitive for optimally secure ratcheting. In Shiho Moriai and Huaxiong Wang, editors, *ASIACRYPT 2020, Part III*, volume 12493 of *LNCS*, pages 621–650, Daejeon, South Korea, December 7–11, 2020. Springer, Cham, Switzerland.

[BSJ$^+$17] Mihir Bellare, Asha Camper Singh, Joseph Jaeger, Maya Nyayapati, and Igors Stepanovs. Ratcheted encryption and key exchange: The security of messaging. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part III*, volume 10403 of *LNCS*, pages 619–650, Santa Barbara, CA, USA, August 20–24, 2017. Springer, Cham, Switzerland.

[CCD$^+$20] Katriel Cohn-Gordon, Cas Cremers, Benjamin Dowling, Luke Garratt, and Douglas Stebila. A formal security analysis of the Signal messaging protocol. *Journal of Cryptology*, 33(4):1914–1983, October 2020.

[CDV21] Andrea Caforio, F. Betül Durak, and Serge Vaudenay. Beyond security and efficiency: On-demand ratcheting with security awareness. In Juan Garay, editor, *PKC 2021, Part II*, volume 12711 of *LNCS*, pages 649–677, Virtual Event, May 10–13, 2021. Springer, Cham, Switzerland.

[CF24] Shan Chen and Marc Fischlin. Integrating causality in messaging channels. In Marc Joye and Gregor Leander, editors, *EUROCRYPT 2024, Part III*, volume 14653 of *LNCS*, pages 251–282, Zurich, Switzerland, May 26–30, 2024. Springer, Cham, Switzerland.

[CHN$^+$24] Daniel Collins, Loïs Huguenin-Dumittan, Ngoc Khanh Nguyen, Nicolas Rolin, and Serge Vaudenay. K-waay: Fast and deniable post-quantum X3DH without ring signatures. In Davide Balzarotti and Wenyuan Xu, editors, *USENIX Security 2024*, Philadelphia, PA, USA, August 14–16, 2024. USENIX Association.

[CJSV22] Ran Canetti, Palak Jain, Marika Swanberg, and Mayank Varia. Universally composable end-to-end secure messaging. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part II*, volume 13508 of *LNCS*, pages 3–33, Santa Barbara, CA, USA, August 15–18, 2022. Springer, Cham, Switzerland.

[CRT24] Daniel Collins, Doreen Riepel, and Si An Oliver Tran. On the tight security of the double ratchet. In Bo Luo, Xiaojing Liao, Jun Xu, Engin Kirda, and David Lie, editors, *ACM CCS 2024*, pages 4747–4761, Salt Lake City, UT, USA, October 14–18, 2024. ACM Press.

[DG19] Nir Drucker and Shay Gueron. Continuous key agreement with reduced bandwidth. In *International Symposium on Cyber Security Cryptography and Machine Learning*, pages 33–46. Springer, 2019.

[DG22] Samuel Dobson and Steven D. Galbraith. Post-quantum signal key agreement from SIDH. In Jung Hee Cheon and Thomas Johansson, editors, *Post-Quantum Cryptography - 13th International Workshop, PQCrypto 2022*, pages 422–450, Virtual Event, September 28–30, 2022. Springer, Cham, Switzerland.

[dPEK$^+$23] Rafaël del Pino, Thomas Espitau, Shuichi Katsumata, Mary Maller, Fabrice Mouhartem, Thomas Prest, Mélissa Rossi, and Markku-Juhani Saarinen. Raccoon. Technical report, National Institute of Standards and Technology, 2023. Available at https://csrc.nist.gov/Projects/pqc-dig-sig/round-1-additional-signatures.

[dPKPR24] Rafaël del Pino, Shuichi Katsumata, Thomas Prest, and Mélissa Rossi. Raccoon: A masking-friendly signature proven in the probing model. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part I*, volume 14920 of *LNCS*, pages 409–444, Santa Barbara, CA, USA, August 18–22, 2024. Springer, Cham, Switzerland.

[EEN$^+$24] Muhammed F. Esgin, Thomas Espitau, Guilhem Niot, Thomas Prest, Amin Sakzad, and Ron Steinfeld. Plover: Masking-friendly hash-and-sign lattice signatures. In Marc Joye and Gregor Leander, editors, *EUROCRYPT 2024, Part VII*, volume 14657 of *LNCS*, pages 316–345, Zurich, Switzerland, May 26–30, 2024. Springer, Cham, Switzerland.

[FG] Rune Fiedler and Felix Günther. Security analysis of signal's pqxdh handshake. To Appear in *PKC 2025*.

[HKKP21] 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 Juan Garay, editor, *PKC 2021, Part II*, volume 12711 of *LNCS*, pages 410–440, Virtual Event, May 10–13, 2021. Springer, Cham, Switzerland.

[HKKP22] 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. *Journal of Cryptology*, 35(3):17, July 2022.

[HKP$^{+}$21] Keitaro Hashimoto, Shuichi Katsumata, Eamonn Postlethwaite, Thomas Prest, and Bas Westerbaan. A concrete treatment of efficient continuous group key agreement via multi-recipient PKEs. In Giovanni Vigna and Elaine Shi, editors, *ACM CCS 2021*, pages 1441–1462, Virtual Event, Republic of Korea, November 15–19, 2021. ACM Press.

[HKP22] Keitaro Hashimoto, Shuichi Katsumata, and Thomas Prest. How to hide MetaData in MLS-like secure group messaging: Simple, modular, and post-quantum. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022*, pages 1399–1412, Los Angeles, CA, USA, November 7–11, 2022. ACM Press.

[HKW] Keitaro Hashimoto, Shuichi Katsumata, and Thom Wiggers. Bundled authenticated key exchange: A concrete treatment of (post-quantum) signal's handshake protocol. To Appear in *USENIX 2025*.

[Jac] Frederic Jacobs. Designing iMessage PQ3: Quantum-secure messaging at scale. Invited talk at the Real World Crypto Symposium 2024.

[JMM19] Daniel Jost, Ueli Maurer, and Marta Mularczyk. Efficient ratcheting: Almost-optimal guarantees for secure messaging. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part I*, volume 11476 of *LNCS*, pages 159–188, Darmstadt, Germany, May 19–23, 2019. Springer, Cham, Switzerland.

[KBB17] Nadim Kobeissi, Karthikeyan Bhargavan, and Bruno Blanchet. Automated verification for secure messaging protocols and their implementations: A symbolic and computational approach. In *2017 IEEE European Symposium on Security and Privacy, EuroS&P 2017, Paris, France, April 26-28, 2017*, pages 435–450. IEEE, 2017.

[KLSS23] Duhyeong Kim, Dongwon Lee, Jinyeong Seo, and Yongsoo Song. Toward practical lattice-based proof of knowledge from hint-MLWE. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part V*, volume 14085 of *LNCS*, pages 549–580, Santa Barbara, CA, USA, August 20–24, 2023. Springer, Cham, Switzerland.

[KS23] Ehren Kret and Rolfe Schmidt. The pqxdh key agreement protocol, 2023. Available at https://signal.org/docs/specifications/pqxdh/.

[LKS23] Joohee Lee, Jihoon Kwon, and Ji Sun Shin. Efficient continuous key agreement with reduced bandwidth from a decomposable kem. *IEEE Access*, 11:33224–33235, 2023.

[LP11] Richard Lindner and Chris Peikert. Better key sizes (and attacks) for LWE-based encryption. In Aggelos Kiayias, editor, *CT-RSA 2011*, volume 6558 of *LNCS*, pages 319–339, San Francisco, CA, USA, February 14–18, 2011. Springer Berlin Heidelberg, Germany.

[LPR10] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In Henri Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 1–23, French Riviera, May 30 – June 3, 2010. Springer Berlin Heidelberg, Germany.

[LSB24] Felix Linker, Ralf Sasse, and David Basin. A formal analysis of apple's iMessage PQ3 protocol. Cryptology ePrint Archive, Paper 2024/1395, 2024.

[MP16a] Moxie Marlinspike and Trevor Perrin. The double ratchet algorithm, 2016. Available at https://signal.org/docs/specifications/doubleratchet/.

[MP16b] Moxie Marlinspike and Trevor Perrin. The x3dh key agreement protocol, 2016. Available at https://signal.org/docs/specifications/x3dh/.

[SAB$^+$22] Peter Schwabe, Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Gregor Seiler, Damien Stehlé, and Jintai Ding. CRYSTALS-KYBER. Technical report, National Institute of Standards and Technology, 2022. available at https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022.

[Sho94] Peter W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *35th FOCS*, pages 124–134, Santa Fe, NM, USA, November 20–22, 1994. IEEE Computer Society Press.

[Ste24] Douglas Stebila. Security analysis of the iMessage PQ3 protocol. Cryptology ePrint Archive, Report 2024/357, 2024.

[VGIK20] Nihal Vatandas, Rosario Gennaro, Bertrand Ithurburn, and Hugo Krawczyk. On the cryptographic deniability of the Signal protocol. In Mauro Conti, Jianying Zhou, Emiliano Casalicchio, and Angelo Spognardi, editors, *ACNS 20International Conference on Applied Cryptography and Network Security, Part II*, volume 12147 of *LNCS*, pages 188–209, Rome, Italy, October 19–22, 2020. Springer, Cham, Switzerland.

```
$\mathsf{RKeyGen\text{-}P}(\mathsf{par}, \mathsf{mode})$
    $(\mathsf{ek_P}, \mathsf{dk_P}) \leftarrow \mathsf{KeyGen}(1^\lambda)$
    $\mathbf{return}\ (\mathsf{ek_P}, \mathsf{dk_P})$

$\mathsf{REnc\text{-}P}(\widehat{\mathsf{ek}}_{\bar{\mathsf{P}}}, \mathsf{dk_P})$
    $(\mathsf{ct}, \mathsf{K}) \xleftarrow{\$} \mathsf{Enc}(\widehat{\mathsf{ek}}_{\bar{\mathsf{P}}})$
    $(\widehat{\mathsf{ek}}_{\mathsf{P}}, \widehat{\mathsf{dk}}_{\mathsf{P}}) \leftarrow \mathsf{KeyGen}(1^\lambda)$
    $\mathsf{ct}_{\bar{\mathsf{P}}} := (\widehat{\mathsf{ek}}_{\mathsf{P}}, \mathsf{ct})$
    $\mathbf{return}\ (\mathsf{ct}_{\bar{\mathsf{P}}}, \mathsf{K}, \widehat{\mathsf{dk}}_{\mathsf{P}})$

$\mathsf{RDec\text{-}P}(\widehat{\mathsf{dk}}_{\mathsf{P}}, \mathsf{ct_P}, \mathsf{ek}_{\bar{\mathsf{P}}})$
    $\mathbf{parse}\ (\widehat{\mathsf{ek}}_{\bar{\mathsf{P}}}, \mathsf{ct}) \leftarrow \mathsf{ct}_{\bar{\mathsf{P}}}$
    $\mathsf{K} \leftarrow \mathsf{Dec}(\widehat{\mathsf{dk}}_{\mathsf{P}}, \mathsf{ct})$
    $\mathbf{return}\ (\mathsf{K}, \widehat{\mathsf{ek}}_{\bar{\mathsf{P}}})$
```

Figure 26: A generic forward-secure RKEM based on a KEM = (KeyGen, Enc, Dec). When using the RKEM to instantiate our generic CKA construction (cf. Section 5.2) this exactly yields the generic CKA construction analyzed by Alwen et al. [ACD19].

```
$\mathsf{RSimKey\text{-}P}_1(\mathsf{ek_P}, \mathsf{dk_P})$
    $(\widehat{\mathsf{dk}}_{\mathsf{P}}, \widehat{\mathsf{ek}}_{\mathsf{P}}) \leftarrow \mathsf{KeyGen}(1^\lambda; \mathsf{rand})$
    $\mathsf{aux}_1 := (\widehat{\mathsf{ek}}_{\mathsf{P}}, \mathsf{rand})$
    $\mathbf{return}\ (\widehat{\mathsf{dk}}_{\mathsf{P}}, \widehat{\mathsf{ek}}_{\mathsf{P}}, \mathsf{aux}_2)$

$\mathsf{RSimKey\text{-}P}_2(\widehat{\mathsf{ek}}_{\bar{\mathsf{P}}}, \widehat{\mathsf{dk}}_{\bar{\mathsf{P}}}, \mathsf{aux}_1)$
    $\mathbf{parse}\ (\widehat{\mathsf{ek}}_{\mathsf{P}}, \mathsf{rand}) \leftarrow \mathsf{aux}_1$
    $(\mathsf{ct}, \mathsf{K}) \leftarrow \mathsf{Enc}(\widehat{\mathsf{ek}}_{\bar{\mathsf{P}}}; \mathsf{rand}')$
    $\mathsf{K}' \leftarrow \mathsf{Dec}(\widehat{\mathsf{dk}}_{\bar{\mathsf{P}}}, \mathsf{ct})$
    $\mathsf{ct}_{\bar{\mathsf{P}}} := (\widehat{\mathsf{ek}}_{\mathsf{P}}, \mathsf{ct})$
    $\mathsf{rand}_2 := (\mathsf{aux}_1, \mathsf{rand}')$
    $\mathbf{return}\ (\mathsf{ct}_{\bar{\mathsf{P}}}, \mathsf{K}, \mathsf{K}, \mathsf{rand}_2)$

$\mathsf{RSimCtxt\text{-}P}(\widehat{\mathsf{ek}}_{\mathsf{P}}, \widehat{\mathsf{ek}}_{\bar{\mathsf{P}}}, \widehat{\mathsf{dk}}_{\bar{\mathsf{P}}})$
    $(\mathsf{ct}, \mathsf{K}) \xleftarrow{\$} \mathsf{Enc}(\widehat{\mathsf{ek}}_{\bar{\mathsf{P}}})$
    $\mathsf{K}' \leftarrow \mathsf{Dec}(\widehat{\mathsf{dk}}_{\bar{\mathsf{P}}}, \mathsf{ct})$
    $\mathsf{ct}_{\bar{\mathsf{P}}} := (\widehat{\mathsf{ek}}_{\mathsf{P}}, \mathsf{ct})$
    $\mathsf{ek_P} := \widehat{\mathsf{ek}}_{\mathsf{P}}$
    $\mathbf{return}\ (\mathsf{ct}_{\bar{\mathsf{P}}}, \mathsf{ek_P}, \mathsf{K}, \mathsf{K}')$
```

Figure 27: Simulators for key and ciphertext simulatability for the generic RKEM.

# A Additional RKEM instantiation

## A.1 Generic Construction

While the Ratchet KEM notion is geared towards abstracting the efficient forward-secure constructions that reuse (parts of) the KEM ciphertext to be the next round's public key, the abstraction can also be naively instantiated from any KEM with just using fresh keys for each round — at the cost of doubling the communication cost. The protocol is shown in Fig. 26.

**Theorem A.1.** *The construction from Fig. 26 is correct and* FS-IND-CPA *secure if the underlying* KEM *is correct and* IND-CPA *secure.*

*Proof.* This immediately follows using trivial reductions. In particular, observe that since $(\widehat{\mathsf{ek}}_{\mathsf{P}}, \widehat{\mathsf{dk}}_{\mathsf{P}})$ are just fresh keys unrelated to this round's KEM encapsulation, leaking $\widehat{\mathsf{dk}}_{\mathsf{P}}$ does not affect FS-IND-CPA security. □

Ratchet simulatability is furthermore trivial due to the complete independence of rounds. For completeness, we depict the simulators in Fig. 27. The proof of the following theorem follows by inspection.

**Theorem A.2.** *The construction from Fig. 26 is perfectly ratchet simulatable using the simulators from Fig. 27.*

## A.2 Non-Forward-Secure Lattice-based Construction

For completeness, we mention that our construction can be trivially "downgraded" to a lattice-based *non-forward-secure* RKEM (cf. Remark 5.2). The difference is that we modify the REnc-P algorithm so that $\mathsf{H}(\mathbf{u}_{\mathsf{P}}, \mathsf{seed})$ outputs only $\mathsf{K}$ rather than $(\mathsf{K}, \mathbf{s}, \mathbf{e})$, and skips the updating of $\mathsf{dk_P}$. Moreover, the RDec-P algorithm is modified similarly where $\mathsf{ek}_{\bar{\mathsf{P}}}$ is no longer updated.

```
$\mathsf{RKeyGen\text{-}P}(\mathsf{par}, \mathsf{mode})$
    $x \xleftarrow{\$} \mathbb{Z}_q$
    $(\mathsf{ek_P}, \mathsf{dk_P}) \leftarrow (g^x, x)$
    return $(\mathsf{ek_P}, \mathsf{dk_P})$

$\mathsf{REnc\text{-}P}(\widehat{\mathsf{ek}}_{\bar{\mathsf{P}}} := Y, \mathsf{dk_P} := x)$
    $\mathsf{ct}_{\bar{\mathsf{P}}} := ()$    // empty ciphertext
    $\mathsf{K} := Y^x$
    $\widehat{\mathsf{dk}}_{\mathsf{P}} := x \cdot \mathsf{H}(\mathsf{K})$
    return $(\mathsf{ct}_{\bar{\mathsf{P}}}, \mathsf{K}, \widehat{\mathsf{dk}}_{\mathsf{P}})$

$\mathsf{RDec\text{-}P}(\widehat{\mathsf{dk}}_{\mathsf{P}} := x, \mathsf{ct_P}, \mathsf{ek}_{\bar{\mathsf{P}}} := Y)$
    $\mathsf{K} := Y^x$
    $\widehat{\mathsf{ek}}_{\bar{\mathsf{P}}} := Y^{\mathsf{H}(\mathsf{K})}$
    return $(\mathsf{K}, \widehat{\mathsf{ek}}_{\bar{\mathsf{P}}})$
```

Figure 28: Diffie-Hellman based RKEM. When using the base protocol to instantiate our generic CKA construction (cf. Section 5.2) this exactly yields the Double Ratchet CKA as analyzed by Alwen et al. [ACD19]. When making the RKEM forward secure, this yields a forward-secure KEM, as analyzed by Bienstock et al. [BFG$^+$22a]. Note that for the forward secure variant, key generation is the same irrespective of the mode.

It is easy to check that correctness (cf. Lemma 6.1) holds, where the updated key distribution $\hat{\chi}$ is replaced by the non-updated one $\chi$. FS-IND-CPA security remains intact as well, with the main difference being that we rely on standard MLWE instead of hint-MLWE to bound the advantage of $\mathsf{Game}_5$ and $\mathsf{Game}_6$ in the proof of Theorem 6.2. The difference stems from the fact that the reduction no longer requires to simulate the updated decapsulation key $\widehat{\mathsf{dk}}_{\mathsf{P}}$. Lastly, ratcheting simulatability remains intact as well. It is worth highlighting that we still need hint-MLWE to simulate the ciphertext in distribution $\mathcal{D}_{\mathsf{B},2}$ in the proof of Theorem 6.3 as this argument does not stem from forward security.

## A.3 Diffie-Hellman Constructions

In this section, we show how both the Diffie-Hellman based Double Ratchet protocol and the forward-secure modification thereof introduced by Bienstock et al. [BFG$^+$22a] can be viewed as an instantiation of RKEM.[16]

**Protocol.** In the following, let $G = \langle g \rangle$ be a cyclic group of prime order $|G| = q$. Recall that in the (original) Double Ratchet protocol a party reuses a group element $g^{x_i}$ as (1) the KEM ciphertext and (2) the public key for the next round. In other words, assume Alice currently knows Bob's public key $Y_{i-1}$ and wants to initiate the next epoch. Then Alice sends $X_i = g^{x_i}$ and absorbs $\mathsf{K}_i = (Y_{i-1})^{x_i}$ as the CKA into the key chain. For the next message, Bob then sends $Y_{i+1} = g^{y_{i+1}}$ and outputs $\mathsf{K}_{i+1} = X_i^{y_{i+1}}$ — therefore reuses $X_i$. In the forward-secure modification, Bob instead computes $\widehat{X_i} := X_i^{\mathsf{H}(\mathsf{K}_i)}$, upon receiving $X_i$, and then encapsulates to that public key instead. Upon sending $X_i$, Alice updates her secret key $\widehat{x_i} := x_i \cdot \mathsf{H}(\mathsf{K}_i)$ analogously. Simply put, the idea is that leaking $\widehat{x_i}$ no longer exposed $\mathsf{K}_i$, which can be proven in the ROM. This neatly fits the RKEM abstraction as used by our generic CKA construction in Section 5.2. For completeness, we present both the original and the forward-secure instantiations in Fig. 28.

**Correctness and security.** We briefly argue correctness and security of the schemes. Note that the schemes are symmetrical between A and B. Therefore, in the following, we solely focus on A without loss of generality. Correctness of the scheme follows by correctness of the Diffie-Hellman key exchange and holds unconditionally.

**Theorem A.3.** *Both the non-forward secure and the forward secure* RKEM *constructions from Fig. 28 are perfectly correct.*

*Proof.* Let $(\mathsf{ek_A}, \mathsf{dk_A}) := (X, x)$ and $(\widehat{\mathsf{ek}}_{\mathsf{B}}, \widehat{\mathsf{dk}}_{\mathsf{B}}) := (Y, y)$, where $X = g^x$ and $Y = g^y$, respectively. The key $\mathsf{K}$ output by $\mathsf{REnc\text{-}A}(\widehat{\mathsf{ek}}_{\mathsf{B}}, \mathsf{dk_A})$ is $\mathsf{K} := Y^x = g^{xy}$, while $\mathsf{RDec\text{-}B}(\widehat{\mathsf{dk}}_{\mathsf{B}}, (), \mathsf{ek_A})$ outputs $\mathsf{K}' := X^y = g^{xy}$ as well. □

[16] Bienstock et al. [BFG$^+$22a] dubbed their forward secure protocol the "Triple Ratchet". To avoid confusion with our hybrid SM protocol, we omit this term.

Security of the basic scheme trivially reduces to the decisional Diffe-Hellman (DDH) assumption, while security of the forward secure variant additionally relies on the random oracle model.

**Theorem A.4.** *The non-forward secure* $\mathsf{RKEM}$ *construction from Fig. 28 is* $\mathsf{IND\text{-}CPA}$ *secure under the DDH assumption. When modelling* $\mathsf{H}$ *as a random oracle, the forward secure variant is* $\mathsf{FS\text{-}IND\text{-}CPA}$ *secure under the DDH assumption.*

*Proof.* Let $(\mathsf{ek_A}, \mathsf{dk_A}) := (X, x)$ and $(\widehat{\mathsf{ek}}_\mathsf{B}, \widehat{\mathsf{dk}}_\mathsf{B}) := (Y, y)$, where $X = g^x$ and $Y = g^y$, be sampled uniformly at random over the key space. In the non-forward secure scheme, $\mathcal{A}$ is given $\mathsf{ek_A} = X$, $\widehat{\mathsf{ek}}_\mathsf{B} = Y$ and either the real key $\mathsf{K}_0 = g^{xy}$ or an uniform random and independent key $\mathsf{K}_1 \in G$. This directly corresponds to a DDH instance. In the forward secure protocol, the adversary is given the following:

- $\mathsf{ek_A} = X$
- $\widehat{\mathsf{ek}}_\mathsf{A} = X^{\mathsf{H}(\mathsf{K}_0)}$
- $\widehat{\mathsf{ek}}_\mathsf{B} = Y$
- $\widehat{\mathsf{dk}}_\mathsf{A} = x \cdot \mathsf{H}(\mathsf{K}_0)$
- $\mathsf{K}_b$, i.e., either $\mathsf{K}_0$ or $\mathsf{K}_1$ depending on the bit $b$.

Note that $\widehat{\mathsf{ek}}_\mathsf{A}$ is redundant given $\widehat{\mathsf{dk}}_\mathsf{A}$. Furthermore, note that the reduction to the DDH instance can program the ROM for consistency as $\mathsf{H}(\mathsf{K}_b) := x^{-1} \cdot \widehat{\mathsf{dk}}_\mathsf{A}$, which does look like a uniform random element in $\mathbb{Z}_q$ to $\mathcal{A}$. Moreover, the key $\mathsf{K}_b$ is unpredictable to $\mathcal{A}$ upfront, i.e., before the key pair $(X, x)$ is sampled (and $\mathsf{REnc\text{-}A}$ is executed) meaning the programming will succeed with overwhelming probability $1 - \frac{1}{q}$. □

**Ratchet Simulatability.** In the following, we argue ratchet simulatability of the protocols. Note that ratchet simulatability for the non-forward secure protocol is almost trivial: Key simulatability can simply execute the protocol, and for ciphertext simulatability the only caveat is that the simulator knows the secret key of party $\bar{\mathsf{P}}$ instead of the one of $\mathsf{P}$. Due to the symmetry of Diffie-Hellman, this nevertheless allows computing the correct key. The forward-secure variant is slightly more elaborate and involves $\mathsf{RSimKey\text{-}P_1}$ actually choosing a fresh key pair and then $\mathsf{RSimKey\text{-}P_2}$ programming the ROM to make this appear consist.

**Theorem A.5.** *The forward-secure* $\mathsf{RKEM}$ *from Fig. 28 is ratchet simulatable with respect to the simulators from Fig. 29. The non-forward secure variant is ratchet simulatable with respect to the simulators form Fig. 30.*

*Proof.* For the non-forward secure variant, the proof follows by inspection. Let us now consider the forward secure protocol. Here, the proofs of the two properties mostly follow by inspection. In particular, observe that the programming of the ROM is consistent, in particular (1) it programs $\mathsf{H}(\mathsf{K})$ to a value that has the correct uniform distribution as in the real-world experiments, and (2) programs it at positions that $\mathcal{A}$ cannot guess beforehand, meaning the programming is still valid with overwhelming probability when attempted. □

# B Remark on Bad Randomness

The use of bad randomness can significantly affect a protocol's security. The secure messaging literature can roughly be divided into three camps with respect to the type of randomness corruption considered.

- **No randomness corruptions.** Some papers, such as [AHKM22, HKP22], do not consider randomness corruptions at all. More recent work often justifies this as a deliberate choice to reduce definitional complexity.
- **Uniform but leaked randomness.** A significant body of work, e.g. [CCD+20, BSJ+17, JMM19], considers "good" (i.e., uniformly sampled) randomness that can be revealed to the adversary.

```
$\mathsf{RSimKey\text{-}P}_1(\mathsf{ek}_\mathsf{P} := X, \mathsf{dk}_\mathsf{P} := x)$
    $(\widehat{x}, \widehat{X}) \xleftarrow{\$} \mathsf{RKeyGen\text{-}P}()$
    $\mathsf{aux}_1 := (x, \widehat{x})$
    return $(\widehat{x}, \widehat{X}, \mathsf{aux}_1)$

$\mathsf{RSimKey\text{-}P}_2(\widehat{\mathsf{ek}}_{\bar{\mathsf{P}}} := \widehat{Y}, \widehat{\mathsf{dk}}_{\bar{\mathsf{P}}} := \widehat{y}, \mathsf{aux}_1)$
    parse $(x, \widehat{x}) \leftarrow \mathsf{aux}_1$
    $\mathsf{K} := (\widehat{Y})^x$
    $\mathsf{H}(\mathsf{K}) := x^{-1} \cdot \widehat{x}$    // Program RO
    $\mathsf{ct}_{\bar{\mathsf{P}}}, \mathsf{rand}_2 := ()$
    return $(\mathsf{ct}_{\bar{\mathsf{P}}}, \mathsf{K}, \mathsf{K}, \mathsf{rand}_2)$

$\mathsf{RSimCtxt\text{-}P}(\widehat{\mathsf{ek}}_\mathsf{P} := \widehat{X}, \widehat{\mathsf{ek}}_{\bar{\mathsf{P}}} := \widehat{Y}, \widehat{\mathsf{dk}}_{\bar{\mathsf{P}}} := \widehat{y})$
    $z \xleftarrow{\$} \mathbb{Z}_q$
    $X := \widehat{X}^{-z}$
    $\mathsf{K} := X^{\widehat{y}}$
    $\mathsf{ct}_{\bar{\mathsf{P}}} := ()$
    return $(\mathsf{ct}_{\bar{\mathsf{P}}}, X, \mathsf{K}, \mathsf{K})$
```

Figure 29: Simulators for key and ciphertext simulatability for the forward-secure Diffie-Hellman based RKEM.

```
$\mathsf{RSimKey\text{-}P}_1(\mathsf{ek}_\mathsf{P} := X, \mathsf{dk}_\mathsf{P} := x)$
    $\mathsf{aux}_1 := x$
    return $(x, X, \mathsf{aux}_1)$

$\mathsf{RSimKey\text{-}P}_2(\widehat{\mathsf{ek}}_{\bar{\mathsf{P}}} := Y, \widehat{\mathsf{dk}}_{\bar{\mathsf{P}}} := y, \mathsf{aux}_1 := x)$
    $\mathsf{K} = Y^x$
    $\mathsf{ct}_{\bar{\mathsf{P}}} := (); \mathsf{rand}_2 := ()$
    return $(\mathsf{ct}_{\bar{\mathsf{P}}}, \mathsf{K}, \mathsf{K}, \mathsf{rand}_2)$

$\mathsf{RSimCtxt\text{-}P}(\widehat{\mathsf{ek}}_\mathsf{P} := X, \widehat{\mathsf{ek}}_{\bar{\mathsf{P}}} := Y, \widehat{\mathsf{dk}}_{\bar{\mathsf{P}}} := y)$
    $\mathsf{K} = X^y$
    $\mathsf{ct}_{\bar{\mathsf{P}}} := ()$
    return $(\mathsf{ct}_{\bar{\mathsf{P}}}, X, \mathsf{K}, \mathsf{K})$
```

Figure 30: Simulators for key and ciphertext simulatability for the non-forward secure Diffie-Hellman based RKEM.

- **Adversarial randomness.** Another line of work, e.g. [ACD19, HKP+21, BFG+22a, AJM22] assumes that the adversary gets to fully control the randomness.

As argued by Balli et al. [BRV20], the additional power of each corruption model does reflect to certain real-world attacks. For instance, there are certain real-world attacks that randomness leakage does not capture, but that is captured by adversarially chosen randomness. While thus appealing, we argue that the third model is problematic when considering post-quantum security: For realistic choices of parameters, most lattice-based KEMs are not perfectly correct, implying that an adversary choosing the randomness can induce arbitrary decryption failures. For "ratcheting" protocols that continuously exchange fresh key material, such correctness issues moreover can translate into security issues, if it allows the adversary to tamper with the decryption of a freshly exchanged public key.

Several countermeasures are conceivable:

- One may choose parameters in a regime where perfect correctness is guaranteed for lattice-based scheme. While this approach was taken in the initial (theoretical) post-quantum instantiation of the Double Ratchet in [ACD19], this ultimately is highly undesirable for practical applications where the size of post-quantum keys is one of the main obstacles towards adoption.
- One may harden the randomness as part of the cryptographic protocol. For instance in [HKP+21] the authors generate the randomness via an output of a hash function. In the ROM, one can then prove, using a union bound argument, that the probability of the adversary finding bad randomness triggering a decryption failure is negligible (cf. [HKP+21, Section 1.3]).

In this work we eschew the issue of adversarially chosen randomness and choose the model of honest-but-leaked randomness instead for several reasons. First, the model already captures many of the attacks from bad randomness. In particular, this captures the exposure of all *intermediate* values of a computation during a corruption. (In contrast, if an attacker only gets to see the state between operations and the operations can use fresh randomness, intermediate values may remain hidden.) Second, while a real-world attacker may realistically have *some* control over the randomness source, arbitrarily setting the randomness (but not allowing to tamper with other protocol state) seems to be an extremely strong assumption not met by the real-world attacks pointed out by [BRV20]. Finally, randomness hardening should preferably be performed at the operating system level and not the at the level of an individual cryptographic protocol.