

Triple Ratchet: A Bandwidth Efficient Hybrid-Secure Signal Protocol

Yevgeniy Dodis¹, Daniel Jost¹, Shuichi Katsumata^{2,3}, Thomas Prest², and Rolfe Schmidt⁴

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

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 now 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.

Table of Contents

1	Introduction	4
1.1	Triple Ratchet Design Overview	6
1.2	Lattice-based Katana RKEM Overview	8
2	Preliminary	10
3	Hybrid Secure Messaging	11
4	The Triple Ratchet	13
4.1	Construction	13
4.2	Correctness and Security	19
5	From Ratcheting Key Encapsulation Mechanism to CKA	20
5.1	Definition of Forward-Secure Ratcheting KEM	20
5.2	A Generic Construction of CKA from Ratcheting KEM	22
6	Katana: An Efficient Ratcheting KEM from Lattices	23
6.1	Construction of Katana	23
6.2	Security of Katana	24
6.3	Optimizing Katana with Bit-Dropping	26
6.4	Concrete Parameter Selection	26
7	Efficiency Analysis of Triple Ratchet	28
7.1	Effect of Our RKEM on Communication Costs	28
7.2	Effect of Chunk Encoding on Communication Costs	29
A	Omitted Preliminary	33
A.1	Lattices and Hardness Assumption	33
A.2	Rounding	33
A.3	CKA Security	34
A.4	Chunk Encoding	35
A.5	Cryptographic Primitives	35
B	More Details on Hybrid Messaging	36
C	Triple Ratchet: Security Proof	39
D	Details on the CKA from RKEM	44
D.1	Omitted Definition of RKEM	44
D.2	Protocol Description	45
D.3	Security	45
E	More Details on Our Lattice-based RKEM Katana	50
E.1	Correctness	50
E.2	FS-IND-CPA Security	51
E.3	Ratchet Simulatability	54
E.4	Details on Optimized RKEM	59
E.5	Details on Concrete Parameter Selection	60
F	Additional RKEM instantiation	61
F.1	Generic Construction	61
F.2	Non-Forward-Secure Lattice-based Construction	62
F.3	Diffie-Hellman Constructions	62

G Remark on Bad Randomness	64
----------------------------------	----

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 a famous *Double Ratchet* protocol [31] to provide important security properties called forward secrecy (FS) and post-compromise security (PCS). Signal (and the Double Ratchet protocol) has been widely deployed with heavily scrutinized open source implementations, and has been formally analyzed in [13,1,9,11,25,7,14] to show that it provides many desirable properties, including FS, PCS, but also mutual authentication and even certain form of deniability [36].

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 [34]. 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. [1] showed how to generalize Signal protocol to work with any key encapsulation mechanism (KEM). As a result, one could potentially replace the DH-based Signal with a post-quantum variant; for example, using recently standardized Kyber (i.e., ML-KEM) [33]. 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 Kyber 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 [26], an update to the X3DH [32] handshake component of the Signal Protocol, and formally verified that the updated protocol provides HNDL protection without removing any of the previous DH-based security guarantees [8]. Since this was only an update to the initial protocol handshake, 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 [4], — 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 [35] and machine verified by [29], but they do not fully capture what is done in the real world. Concretely, [35] 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 [22]. 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).⁵

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 one of the users is offline for an extended period of time,⁶ 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 [35].⁷

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 Signal. 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 [1] 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, [1,17,27] already described lattice-based protocols (either directly for Kyber, or equivalent variants over other rings) which seemingly achieve this goal. Unfortunately, the protocol of [17] achieves almost no saving (less than 2%, as noticed by the authors) as compared to using the generic Kyber, while the protocols of [1,27] 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 Signal or its variants.

OUR CONTRIBUTIONS. In this work, we provide a practical hybrid-secure Double Ratchet protocol called the *Triple Ratchet* protocol.⁸ 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.

⁵ This heuristics is related to “on-demand” ratcheting suggested by [10].

⁶ E.g., when using devices which are periodically turned off.

⁷ [29] explicitly models this optional sending behavior, but does not model the repetition of KEM public key and ciphertext messages required for immediate decryption [22].

⁸ This should not be confused with the protocol by [9] with the same name.

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 Sec. 7, and give an overview of some of the technical challenges we resolved in Sec. 1.1.

Second, we design a novel *Continuous Key Agreement* (CKA) protocol based on Kyber, which we call *Katana-CKA*, which could be used inside our Triple Ratchet protocol. Recall, CKA was a generic building block used by [1] to abstract out the design of the Double Ratchet Protocol. [1] 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 [1,27], 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 them with a novel proof relying on the recently introduced hint-MLWE assumption [24,18].

In more detail, we first generalize the KEM-based CKA from [1] 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 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*,⁹ which explains the name *Katana-CKA* for our new CKA. We expand on our technique in Sec. 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 [1] to allow the use of a third “post-quantum

⁹ Similar to Kyber, *Katana* is a certain type of an ancient (Japanese) sword.

ratchet”. 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 ratchets 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 move forward. We resolve it by having two root keys, carefully deriving two separate message keys (also using two separate¹⁰ 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 ratchet, 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 chunk* 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. 3 and 4 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 [1], 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.

¹⁰ 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.

With these important changes, our resulting protocol could be viewed as a natural, and still very modular, generalization of the Double Ratchet abstraction from [1]. 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.

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 CKA built from such RKEMs is exactly the generic construction of CKA based on KEMs by [1]. 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 K is then updated by mixing g^{ab} into 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, [9] 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 [1,17,27]. Notably, [1,27] 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 $\mathbf{s}_A \in R_q^k$ and Bob holds $\mathbf{u}_A = \mathbf{D} \cdot \mathbf{s}_A + \mathbf{e}_A \in R_q^k$, where \mathbf{s}_A and \mathbf{e}_A are short. Bob samples short vectors $\mathbf{s}_B, \mathbf{e}_B \in R_q^k$ and $\tilde{e}_B \in R_q$ from appropriate distributions, and a random $\text{seed} \xleftarrow{\$} \{0, 1\}^n \subset R_q$. It then sends $(\mathbf{u}_B, v_A) := (\mathbf{D}^\top \cdot \mathbf{s}_B + \mathbf{e}_B, \mathbf{u}_A^\top \cdot \mathbf{s}_B + \tilde{e}_B + \text{seed} \cdot \lfloor q/2 \rfloor)$ to Alice.¹¹ Alice first interprets (\mathbf{u}_B, v_A) as a ciphertext and decrypts seed by rounding $v_A - \mathbf{u}_B^\top \cdot \mathbf{s}_A$ to the nearest multiple of $\lfloor q/2 \rfloor$. The seed is then mixed into the shared key K to ratchet the state forward. Alice then interprets part of the ciphertext \mathbf{u}_B as Bob’s public key so that it can perform similar ratcheting. As \mathbf{u}_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 [1,27] contain the same flaw in the CKA security proof, rendering their scheme insecure for certain parameter regime. Recall that Signal’s Double Ratchet is proven to satisfy PCS 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 .

¹¹ Note that while [1] bases their construction on FrodoKEM, our explanation is based on a Kyber-like KEM as in [27]. These differences will have no importance to our argument.

(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 $\mathbf{s}_A \in R_q^k$ is compromised:

(L.1) $v_A := \mathbf{u}_A^\top \cdot \mathbf{s}_B + \tilde{e}_B + \text{seed} \cdot [q/2]$ can be simulated without Bob's secret key \mathbf{s}_B .

(L.2) $(\mathbf{u}'_A, \mathbf{u}_B, v'_B) = (\mathbf{D} \cdot \mathbf{s}'_A + \mathbf{e}'_A, \mathbf{D}^\top \cdot \mathbf{s}_B + \mathbf{e}_B, \mathbf{u}_B^\top \cdot \mathbf{s}'_A + \tilde{e}'_A + \text{seed}' \cdot [q/2])$ is indistinguishable from $(\mathbf{u}'_A, \mathbf{u}_B, v)$ 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_A is identically expressible as $v_A = (\mathbf{D} \cdot \mathbf{s}_A + \mathbf{e}_A)^\top \cdot \mathbf{s}_B + \tilde{e}_B + \text{seed} \cdot [q/2] = \mathbf{u}_B^\top \cdot \mathbf{s}_A - \mathbf{e}_B^\top \cdot \mathbf{s}_A + \mathbf{e}_A^\top \cdot \mathbf{s}_B + \tilde{e}_B + \text{seed} \cdot [q/2]$, where we plug in $\mathbf{u}_B = \mathbf{D}^\top \cdot \mathbf{s}_B + \mathbf{e}_B$. Denoting the underlined value as h , it is clear that h *cannot* be simulated only using Alice's secret \mathbf{s}_A (and \mathbf{e}_A). In fact, an adversary with \mathbf{s}_A can directly compute h to infer statistical knowledge of \mathbf{s}_B and \mathbf{e}_B . Even worse, since the adversary learns slight information about \mathbf{s}_B , we can inductively see that the adversary also learns no information even on the *updated* key \mathbf{s}'_A , creating a vicious cycle.

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}_B from a distribution super-polynomially larger than $-\mathbf{e}_B^\top \cdot \mathbf{s}_A + \mathbf{e}_A^\top \cdot \mathbf{s}_B$ (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 [24,18] 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 key to make the recent trick by Bienstock et al. [9] improving FS to work in the lattice-setting. The main idea of [9] was for Alice to run the same Signal's Double Ratchet protocol, but to store $\hat{a} := a' + \mathbf{H}(g^{a'b})$ as opposed to a' . The intuition is that even if \hat{a} is compromised, $g^{a'b}$ remains secure assuming \mathbf{H} is a random oracle (or ElGamal encryption is circular secure), hence offering better FS. In the lattice-setting however, the updated $\hat{\mathbf{s}}_A := \mathbf{s}'_A + \mathbf{H}(\text{seed}')$ must still remain *short* for decryption to work, and as such, leaking $\hat{\mathbf{s}}_A$ again statistically leaks information on \mathbf{s}'_A . For more detail of the proof, we refer to Sec. 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 ratcheting information is $\approx 40\%$ better than naively using Kyber as the KEM-based CKA [1].

2 Preliminary

Notations. When S is a finite set, we let $\mathcal{U}(S)$ denote the uniform distribution over S , and abbreviate $x \stackrel{\$}{\leftarrow} S$ for $x \stackrel{\$}{\leftarrow} \mathcal{U}(S)$. 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 $\forall i \in [N], x_i \stackrel{\$}{\leftarrow} D$. 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 $x \stackrel{\$}{\leftarrow} D\{\text{rand}\}$ 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.

Lattices and Hardness Assumption. We recall the standard notion of lattices and the definition of module learning with errors (MLWE) problem in App. A.1. In our work, we rely on a generalization of the recent *hint* MLWE problem by Kim et al. [24], stating that MLWE remains hard even if some leakage of the secret is provided. The following is based on [18].

Definition 2.1 (hint-MLWE). *Let k, ℓ, q be integers, χ and $\tilde{\chi}$ be probability distributions over \mathcal{R}_q^k and \mathcal{R}_q^ℓ , 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 hint-MLWE $_{q,k,\ell,\chi,\tilde{\chi},\mathcal{F}}$ problem is defined as:*

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

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

The hint-MLWE problem is known to be as hard as the MLWE problem with appropriately set parameters. We include the formal statement in App. A.1.

Chunk Encoding. We use a standard erasure code instantiated using Reed-Solomon erasure codes to implement our “chunking” strategy of post-quantum CKA messages. The details are given in App. A.4.

Cryptographic Primitives. We use an *authenticated encryption with associated data* (AEAD) scheme $\text{AEAD} = (\text{Enc}, \text{Dec})$ as well as a number of *key derivation functions* KDF. The latter is assumed to be variable input and output length. The details are given in App. A.5.

Continuous Key Agreement. We follow the abstraction of continuous key agreement (CKA) put forth by Alwen, Coretti, and Dodis [1]. 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 2.2. A continuous key agreement (CKA) protocol Π_{CKA} with initial key space \mathcal{I}_{CKA} , key space \mathcal{K} consists of PPT algorithms $(\text{CKA-Init-KeyGen}, (\text{CKA-Init-P}, \text{CKA-Send-P}, \text{CKA-Rec-P})_{P \in \{A, B\}})$ defined as follows:

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

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

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

$\text{CKA-Rec-A}(\mathsf{st}_A, \rho) \rightarrow (\mathsf{K}, \mathsf{st}_A)$: It takes as input a state st_A of party A and a message ρ , and outputs a key $\mathsf{K} \in \mathcal{K} \cup \{\perp\}$, and an updated state st_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 2.1 (Alternating Communication). Following Alwen, Coretti, and Dodis [1], we always assume parties A and B execute the sending and receiving algorithms in an alternating order. That is, $\text{CKA-Send-A} \rightarrow \text{CKA-Rec-B} \rightarrow \text{CKA-Send-B} \rightarrow \text{CKA-Rec-A} \rightarrow \dots$. 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 [1] with the latter phrased as a real-or-random experiment for a (fixed) challenge epoch \hat{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 \hat{t}^* . More concretely, a party must recover from a state compromise within Δ_{PCS} epochs and a state compromise must not endanger keys more than Δ_{FS} epochs from the past. See App. A.3 for a formal version of the game.

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 [1], 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 Sec. 4, we will instantiate this primitive with a hybrid secure messaging protocol.

Syntax. To define the syntax of a secure messaging scheme, we mostly follow [1]. 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 [12].

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

- SM-Init-KeyGen(1^λ) \rightarrow l_K : It takes as input the security parameter 1^λ and outputs an initial key $l_K \in \mathcal{I}_K$.
- SM-Init-A(l_K) \rightarrow st_A : It takes as input an initial key $l_K \in \mathcal{I}_K$ and outputs an initial state st_A for party A.
- SM-Send-A(st_A, M) \rightarrow (ct, st'_A) : It takes as input a state st of party A and a message $M \in \mathcal{M}$, and outputs a ciphertext ct and an updated state st_A .
- SM-Rec-A(st_A, ct) \rightarrow (M, idx, st'_A) : It takes as input a state st of party A and a ciphertext ct , and outputs a message $M \in \mathcal{M}$, a message index $idx \in \mathcal{Idx}$, and an updated state 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 st_A to store A's current index $idx(st_A)$, and analogously for user B.¹²

Security. We formalize correctness and security as part of a combined security game. For space reasons, we defer to App. B for the formal game. 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 M_0 or M_1 is sent depending on the game's challenge bit b , and can try to break authenticity by injecting their own ciphertexts. In a bit more detail, 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.

¹² Alternatively, SM-Init-A, SM-Send-A, and SM-Rec-A could each output this index.

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 Δ_{PCS} epochs. Here, we measure epochs using an (efficiently computable) *epoch function* $\tau(\text{idx})$ 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 Δ_{PCS} epochs after a state compromise. Furthermore, *forward secrecy* (FS) dictates that messages sent as least Δ_{FS} epochs prior to a state compromise also remain secure. In other words, a state compromise may reveal messages of the last Δ_{FS} and the next Δ_{PCS} epochs.

Note that our game generalizes the one by Alwen et al. [1] to hybrid messaging and deviates in the following ways:

Message indices: Whereas the game in [1] 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 τ to formalize FS and PCS. We remark that for our concrete scheme the two choices of τ , 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 [1] considered adversarially chosen randomness, we consider honestly sampled but leaked randomness only (cf. App. G).

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 [1]. Since the post-quantum CKA messages are significantly larger than their classical counterparts, however, each post-quantum CKA message is split into n_{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. 1 to 4. To ease presentation, we use the following conventions: We depict the classical part in the left column with the post-quantum part in

Epochs	tR	The epoch under which received messages are encrypted.
	tS	The key epoch under which sent messages are encrypted. Invariant: $tS \in \{tR, tR + 1\}$
	$tCurr$	The epoch for which key material is being exchanged. Invariant: $tCurr \in \{tS, tS + 1\}$
Periods	iS	The number of messages sent in epoch tS .
	iS_{-1}, iS_{-2}	The number of messages sent in epochs $tS - 1$ and $tS - 2$.
	iR	The number of messages received in epoch tR .
Keys	K_{root}	The current root key of the asymmetric ratchet.
	KS	The current sending key for epoch tS .
	KS_{+1}	The sending key for epoch $tS + 1$ (if already known)
	KR	The current receiving key for epoch tR .
	KR_{+1}, KR_{+2}	The receiving keys for epoch $tR + 1$ and $tR + 2$ (if already known).
	$StoredKeys[t, i]$	Stored keys for processing out-of-order messages.
Chunks	cR	The number of chunks received for $tCurr$ (for receiving epochs).
	cS	The number of chunks sent for $tCurr$ (for sending epochs).
	$cAck$	The number of chunks acknowledged $tCurr$ (for receiving epochs). Invariant: $cAck \leq cS$.
	L	The set of chunk-period pairs received of the next CKA message.

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

the right column.¹³ 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 C (for classical) or Q (for post-quantum) — for example, $tCurr^C$ and $tCurr^Q$ denote the two independent epoch counters of the two CKAs. Finally, to reduce clutter we omit those superscripts whenever clear from the context which protocol part they refer to. In the following, we mainly describe the post-quantum part of the protocol, referring to [1] 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.

¹³ Note that the classical part technically can be seen as a simplification of the post-quantum protocol for $n_{chunk} = 1$ with certain optimizations applied.

TR-Init-KeyGen(1^λ)	
1 : $I_{CKA} \xleftarrow{\$} \text{CKA-Init-KeyGen}^C(1^\lambda)$	4 : $I_{CKA} \xleftarrow{\$} \text{CKA-Init-KeyGen}^Q(1^\lambda)$
2 : $(K_{\text{root}}, K_{CKA}) \xleftarrow{\$} \mathcal{K}_{PP}^C \times \mathcal{K}_{CKA}^C$	5 : $(K_{\text{root}}, K_{CKA}) \xleftarrow{\$} \mathcal{K}_{PP}^Q \times \mathcal{K}_{CKA}^Q$
3 : $I_K^C \leftarrow (I_{CKA}, K_{\text{root}}, K_{CKA})$	6 : $I_K^Q \leftarrow (I_{CKA}, K_{\text{root}}, K_{CKA})$
7 : return $I_K := (I_K^C, I_K^Q)$	
TR-Init-A(I_K)	
1 : parse $(I_K^C, I_K^Q) \leftarrow I_K$	
2 : parse $(I_{CKA}, K_{\text{root}}, K_{CKA}) \leftarrow I_K^C$	9 : parse $(I_{CKA}, K_{\text{root}}, K_{CKA}) \leftarrow I_K^Q$
3 : $(K_{\text{root}}, KR) \leftarrow \text{KDF}_1(K_{\text{root}}, K_{CKA})$	10 : $(K_{\text{root}}, KS, KR_{+1}) \leftarrow \text{KDF}_1(K_{\text{root}}, K_{CKA})$
4 : $KS \leftarrow \perp$	11 : $(KS_{+1}, KR, KR_{+2}) \leftarrow \perp$
5 : $(tCurr, iR, iS, iS_{-2}) \leftarrow 0$	12 : $(tCurr, tS, iR, iS, iS_{-1}, iS_{-2}) \leftarrow 0$
6 : $\overline{st}_A \xleftarrow{\$} \text{CKA-Init-A}(I_{CKA})$	13 : $tR \leftarrow -1$
7 : $\rho \leftarrow \perp$	14 : $(cS, cAck, cR) \leftarrow (0, 0, n_{\text{chunk}})$
8 : $\text{StoredKeys}^C[\cdot] := \perp$	15 : $\overline{st}_A \xleftarrow{\$} \text{CKA-Init-A}(I_{CKA})$
	16 : $\rho \leftarrow \perp$
	17 : $\text{StoredKeys}^Q[\cdot] := \perp$
TR-Init-B(I_K)	
1 : parse $(I_K^C, I_K^Q) \leftarrow I_K$	
2 : parse $(I_{CKA}, K_{\text{root}}, K_{CKA}) \leftarrow I_K^C$	9 : parse $(I_{CKA}, K_{\text{root}}, K_{CKA}) \leftarrow I_K^Q$
3 : $(K_{\text{root}}, KS) \leftarrow \text{KDF}_1(K_{\text{root}}, K_{CKA})$	10 : $(K_{\text{root}}, KR_{+1}, KS) \leftarrow \text{KDF}_1(K_{\text{root}}, K_{CKA})$
4 : $KR \leftarrow \perp$	11 : $(KS_{+1}, KR, KR_{+2}) \leftarrow \perp$
5 : $(tCurr, iR, iS, iS_{-2}) \leftarrow 0$	12 : $(tCurr, tS, iR, iS, iS_{-1}, iS_{-2}) \leftarrow 0$
6 : $\overline{st}_B \xleftarrow{\$} \text{CKA-Init-B}(I_{CKA})$	13 : $tR \leftarrow -1$
7 : $\rho \leftarrow \perp$	14 : $(cS, cAck, cR) \leftarrow (n_{\text{chunk}}, n_{\text{chunk}}, 0)$
8 : $\text{StoredKeys}^C[\cdot] := \perp$	15 : $\overline{st}_B \xleftarrow{\$} \text{CKA-Init-B}(I_{CKA})$
	16 : $\rho \leftarrow \perp$
	17 : $\text{StoredKeys}^Q[\cdot] := \perp$

Fig. 1: 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 C and Q, respectively. For ease of reading, we omit those superscripts whenever clear from the context.

$\text{skip}^X(t, iR')$ // for $X \in \{C, Q\}$	$\text{try-skipped}^X(t, i)$ // for $X \in \{C, Q\}$
1 : while $iR^X < iR'$	1 : $K_{\text{aead}}^X \leftarrow \text{StoredKeys}[t, i]$
2 : $iR^X += 1$	2 : $\text{StoredKeys}^X[t, i] \leftarrow \perp$
3 : $(KR^X, K_{\text{aead}}^X) \leftarrow \text{KDF}_2(KR^X)$	3 : return K_{aead}^X
4 : $\text{StoredKeys}^X[t, iR^X] \leftarrow K_{\text{aead}}$	

Fig. 2: Helper algorithms of the Triple Ratchet protocol.

TR-Send-A(M)	
<pre> 1 : if $\llbracket \text{tCurr is even} \rrbracket$ then 2 : tCurr += 1 3 : $(K_{\text{CKA}}, \rho, \overline{\text{st}}_A) \xleftarrow{\\$} \text{CKA-Send-A}(\overline{\text{st}}_A)$ 4 : $(K_{\text{root}}, \text{KS}) \leftarrow \text{KDF}_1(K_{\text{root}}, K_{\text{CKA}})$ 5 : $(iS, iS_{-2}) \leftarrow (0, iS)$ 6 : endif 7 : iS += 1 8 : $h^C := (\text{tCurr}, iS, \rho, iS_{-2})$ 9 : $(\text{KS}, K_{\text{aead}}^C) \leftarrow \text{KDF}_2(\text{KS})$ 30 : $K_{\text{aead}} \leftarrow \text{KDF}_3(K_{\text{aead}}^C, K_{\text{aead}}^Q)$ 31 : $h \leftarrow (h^C, h^Q)$ 32 : $e \xleftarrow{\\$} \text{AEAD.Enc}(K_{\text{aead}}, h, M)$ 33 : return ct := (h, e) </pre>	<pre> 10 : if $\llbracket \text{tS} = \text{tCurr} \rrbracket \wedge \llbracket \text{tCurr is even} \rrbracket$ then 11 : tCurr += 1 // start sending next key 12 : $(K_{\text{CKA}}, \rho, \overline{\text{st}}_A) \xleftarrow{\\$} \text{CKA-Send-A}(\overline{\text{st}}_A)$ 13 : $(K_{\text{root}}, \text{KS}_{+1}, \text{KR}_{+2}) \leftarrow \text{KDF}_1(K_{\text{root}}, K_{\text{CKA}})$ 14 : if $\llbracket \text{tR} = \text{tS} \rrbracket$ then 15 : $(\text{KR}_{+1}, \text{KR}_{+2}) \leftarrow (\text{KR}_{+2}, \perp)$ 16 : $(c_S, c_{\text{Ack}}) \leftarrow (0, 0)$ 17 : if $\llbracket \text{tCurr is odd} \rrbracket$ then // sending chunks 18 : $\rho_{\text{enc}} \leftarrow \text{Encode}(\rho, c_S)$ 19 : $c_S += 1$ 20 : if $\llbracket \text{tS} < \text{tCurr} \rrbracket \wedge \llbracket c_{\text{Ack}} + 1 \geq n_{\text{chunk}} \rrbracket$ then 21 : tS ← tCurr // start using next key 22 : $(iS, iS_{-1}, iS_{-2}) \leftarrow (0, iS, iS_{-1})$ 23 : $(\text{KS}, \text{KS}_{+1}) \leftarrow (\text{KS}_{+1}, \perp)$ 24 : iS += 1 25 : $h^Q := (\text{tS}, iS, \text{tCurr}, \rho_{\text{enc}}, \perp, iS_{-1}, iS_{-2})$ 26 : else // acknowledging chunks 27 : iS += 1 28 : $h^Q := (\text{tS}, iS, \text{tCurr}, \perp, c_R, iS_{-1}, iS_{-2})$ 29 : $(\text{KS}, K_{\text{aead}}^Q) \leftarrow \text{KDF}_2(\text{KS})$ </pre>

Fig. 3: 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 KDF_1 swapped with the output becoming $(K_{\text{root}}, \text{KR}, \text{KS})$ for consistency.

- In the former case (lines 18-25) A sends an additional CKA chunk ρ_{enc} to the receiver. It keeps track of the number of chunks sent for the current CKA message ρ using c_S . (Note that c_S is not necessarily equal to the sending period iS , as the sending epoch can change while sending ρ .)
- In the latter case (lines 27 and 28) A simply acknowledges the number of chunks received c_R of ρ sent by the other party. The other party B then uses this information (as stored in c_{Ack}) to deduce when the new CKA key becomes usable an TR-Rec-B and TR-Send-B (as discussed later).

TR-Rec-A(ct)

```

1 : parse (h, e) ← ct
2 : parse (hC, hQ) ← h
3 : parse (t, i, ρ, i-2) ← hC
4 : req [t ≤ tCurr + 1]
5 : if [t = tCurr + 1] then
6 :   skip(t - 2, i-2)
7 :   (tCurr, iR) ← (t, 0)
8 :   (KCKA, s̄tA) ←s CKA-Rec-A(s̄tA, ρ)
9 :   (Kroot, KR) ← KDF1(Kroot, KCKA)
10 : endif
11 : Kaead ← try-skipped(t, i)
12 : if Kaead = ⊥ then
13 :   skip(t, i - 1)
14 :   iR += 1
15 :   (KR, KaeadC) ← KDF2(KR)
16 :   parse (t, i, tCurr', ρenc, c'Ack, i-1, i-2) ← hQ
17 :   req [tCurr' ≤ tCurr + 1] ∧ [t ≤ tR + 2]
18 :   [tCurr' - 1 ≤ t ≤ tCurr']
19 :   if [t = tR + 2] then
20 :     skip(t - 2, i-2)
21 :     (KR, KR+1, KR+2) ← (KR+1, KR+2, ⊥)
22 :     if [t > tR] then
23 :       skip(t - 1, i-1)
24 :       (KR, KR+1, KR+2) ← (KR+1, KR+2, ⊥)
25 :       (tR, iR) ← (t, 0)
26 :       if [tCurr' = tCurr + 1] then
27 :         if [tS < tCurr] then
28 :           tS ← tCurr
29 :           (iS, iS-1, iS-2) ← (0, iS, iS-1)
30 :           (KS, KS+1) ← (KS+1, ⊥)
31 :           (tCurr, cR) ← (tCurr', 0)
32 :           if [tCurr' = tCurr] ∧ [tCurr is even] then
33 :             cR += 1
34 :             L ←+ (i, ρenc)
35 :             if [cR ≥ nchunk] ∧ [tCurr > tS] then
36 :               tS ← tCurr
37 :               (iS, iS-1, iS-2) ← (0, iS, iS-1)
38 :               ρ ← Decode(L)
39 :               (KCKA, s̄tA) ←s CKA-Rec-A(s̄tA, ρ)
40 :               if [tR = tS] then
41 :                 (Kroot, KS, KR) ← KDF1(Kroot, KCKA)
42 :               else
43 :                 (Kroot, KS, KR+1) ← KDF1(Kroot, KCKA)
44 :                 L ← ∅
45 :               elseif [tCurr' = tCurr] then
46 :                 cAck ← c'Ack
47 :                 Kaead ← try-skipped(t, i)
48 :                 if Kaead = ⊥ then
49 :                   skip(t, i - 1)
50 :                   iR += 1
51 :                   (KR, Kaead) ← KDF2(KR)
52 :                   Kaead ← KDF3(KaeadC, KaeadQ)
53 :                   M ← AEAD.Dec(Kaead, h, e)
54 :                   if M = ⊥ then error
55 :                   return (M, (tCurrC, iSC, tSQ, iSQ))

```

Fig. 4: 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 KDF₁ swapped with the output becoming (K_{root}, KR, KS) for consistency. skip and try-skipped are defined in Fig. 2.

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 are 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 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 K_{aead}^C and K_{aead}^Q 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 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 counter t_{Curr} , the epoch for which CKA messages are currently being exchanged, and t_S , 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 t_R , 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 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 K_{root} , a sending key, and a receiving key. See lines 12 and 13 of TR-Send-A.
- However, unless $n_{\text{chunk}} = 1$, those keys cannot be immediately used for sending. Instead, A at this point simply schedules the keys for further use. In case of the sending key, it will be the one used next, KS_{+1} . The party can use it once either B either starts sending chunks for the next key (lines 25-30 in TR-Rec-A) or A knows that B will know sufficiently many chunks with the next one received (lines 20-23 of TR-Send-A). The receiving key can either be the next or the one there after, depending on whether B has already used to current one.
- Similarly, during a “receiving epoch” the user reconstructs ρ once they received sufficiently many chunks. The party can then immediately start using the new sending key (lines 34-38 in TR-Rec-A) while the receiving key may have to be scheduled for later use unless the other party already uses it (lines 39-42 in TR-Rec-A). In either case, for the next sending operation A then can initiate the next sending epoch.

4.2 Correctness and Security

This section establishes SM security of the TR protocol from Sec. 4.1. Recall that the FS and PCS properties of SM security are defined with respect to an epoch function τ , 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 function for both cases.

Remark 4.1 (Epoch functions). For the protocol TR, we define $\tau^C := \text{tCurr}^C$ and $\tau^Q := \text{tS}^Q$. Observe that for the classical CKA the epoch function directly corresponds to epochs as introduced in [1] and increment on every change in direction. For the post-quantum protocol, once A enters an even epoch, it takes the following for A to advance to the next odd epoch:

1. A needs to send at least $n_{\text{chunk}} - 1$ many messages that need to be received by B (any subset of $n_{\text{chunk}} - 1$ many does, in case more are sent)
2. B sends a message that is received by A.
3. If B received at least n_{chunk} many messages before (2), then A immediately increments the epoch; otherwise A increments the epoch upon the next send action.

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

Theorem 4.1 (Security of TR). *For the TR protocol, let τ^C and τ^Q denote the respective epoch functions as discussed in Remark 4.1. Assume that*

- CKA^C is $(\Delta_{\text{FS}}^C, \Delta_{\text{PCS}}^C)$ -secure CKA scheme or CKA^Q is $(\Delta_{\text{FS}}^Q, \Delta_{\text{PCS}}^Q)$ -secure CKA scheme;
- CKA^C and CKA^Q are both correct;
- KDF_1 is a secure PRF-PRNG, KDF_2 is a secure PRG, and KDF_3 is a secure dual-PRF;
- AEAD is a secure authenticated encryption scheme with associated data.

Then, the TR construction above is $(\Delta_{\text{FS}}^C, \Delta_{\text{PCS}}^C, \tau^C)$ -secure if CKA^C is secure, and $(\Delta_{\text{FS}}^Q, \Delta_{\text{PCS}}^Q + 1, \tau^Q)$ secure if CKA^Q is secure respectively.

Remark 4.2 (Instantiations). For the Triple Ratchet protocol, we propose to instantiate the two CKA using a generic CKA construction from RKEM presented in Sec. 5, with the classical one using a forward-secure Diffie-Hellman RKEM—modularizing the protocol proposed by Bienstock et al. [9]—and the post-quantum one using our Katana-RKEM. Therefore, both CKA will have $\Delta_{\text{FS}}^{\text{CKA}} = 0$ and $\Delta_{\text{PCS}}^{\text{CKA}} = 2$. Therefore, for the TR protocol, we obtain classical PCS within $\Delta_{\text{PCS}}^{\text{TR}} = 2$ epochs and post-quantum PCS within $\Delta_{\text{PCS}}^{\text{TR}} = 3$ (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.

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 particular, the ciphertext can depend on the encapsulation key received in the previous round.

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

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

$\text{RKeyGen-P}(\text{par}, \text{mode}) \rightarrow (\text{ek}_P, \text{dk}_P)$: It takes as input the public parameter par and outputs encapsulation and decapsulation keys $(\text{ek}_P, \text{dk}_P) \in \mathcal{RK}_P$ if $\text{mode} = \perp$ and $(\text{ek}_P, \text{dk}_P) \in \widehat{\mathcal{RK}}_P$ if $\text{mode} = \text{updated}$. In case $\text{mode} = \perp$, we may simply ignore mode from the input when the context is clear.¹⁴

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

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

In the above, we define algorithms REnc-B and RDec-B analogously with roles of parties A and B swapped.

Remark 5.1 (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 $\widehat{\text{dk}}_P = \text{dk}_P$ and $\widehat{\text{ek}}_P = \text{ek}_P$ in algorithms REnc-P and RDec-P , respectively, for $P \in \{A, B\}$. While we can alternatively remove $\widehat{\text{dk}}_P$ and $\widehat{\text{ek}}_P$ 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.

¹⁴ Indeed, RKeyGen-P with $\text{mode} = \text{updated}$ is mainly used for security analysis and will otherwise only appear in the setup of our construction. As such, we will typically omit mode outside this section.

To aid readability, we define $\mathcal{D}_{\text{RKeyGen-P}}(\text{par})$ (resp. $\widehat{\mathcal{D}}_{\text{RKeyGen-P}}(\text{par})$) for $P \in \{A, B\}$ as the distribution of sampling $(ek_P, dk_P) \xleftarrow{\$} \text{RKeyGen-P}(\text{par}, \text{mode})$ with $\text{mode} = \perp$ (resp. $\text{mode} = \text{updated}$) for $\text{par} \in \text{RSetup}(1^\lambda)$. We use the shorthand $\mathcal{D}_{\text{RKeyGen-P}}$ and $\widehat{\mathcal{D}}_{\text{RKeyGen-P}}$ when par is randomly generated from $\text{RSetup}(1^\lambda)$. In the following correctness and security definitions, we assume par is sampled and fixed once and for all, and only use $\mathcal{D}_{\text{RKeyGen-P}}$ and $\widehat{\mathcal{D}}_{\text{RKeyGen-P}}$. While we omit par for readability, it is understood that the probability is taken over the randomness of generating 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 $\widehat{\mathcal{D}}_{\text{RKeyGen-P}}$. 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 5.2 (Correctness). *We say a ratcheting KEM Π_{RKEM} is correct if it satisfies two properties. The first property, correctness with updated keys, requires the following to hold:*

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

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

While we do not require it, we can define correctness with respect non-updated keys by replacing $(\widehat{ek}_B, \widehat{dk}_B) \xleftarrow{\$} \widehat{\mathcal{D}}_{\text{RKeyGen-B}}$ with $(ek_B, dk_B) \xleftarrow{\$} \mathcal{D}_{\text{RKeyGen-B}}$ above.

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 5.3 (FS-IND-CPA Security). *We say a ratcheting KEM Π_{RKEM} is forward-secure IND-CPA (FS-IND-CPA) secure if the advantages*

$$\text{Adv}_{\mathcal{A}}^{\text{FS-IND-CPA-A}}(1^\lambda) := \left| \Pr \left[\begin{array}{l} b \xleftarrow{\$} \{0, 1\}, K_1 \xleftarrow{\$} \mathcal{K}, \\ (ek_A, dk_A) \xleftarrow{\$} \mathcal{D}_{\text{RKeyGen-A}}, (\widehat{ek}_B, \widehat{dk}_B) \xleftarrow{\$} \widehat{\mathcal{D}}_{\text{RKeyGen-B}}, \\ (ct_B, K_0, \widehat{dk}_A) \xleftarrow{\$} \text{REnc-A}(\widehat{ek}_B, dk_A), \\ (\cdot, \widehat{ek}_A) \xleftarrow{\$} \text{RDec-B}(\widehat{dk}_B, ct_B, ek_A), \\ b' \xleftarrow{\$} \mathcal{A}(ek_A, \widehat{ek}_A, \widehat{ek}_B, ct_B, \widehat{dk}_A, K_b) \end{array} : b = b' \right] - \frac{1}{2} \right|$$

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

As a special case, we say a (non-forward-secure) RKEM (cf. Remark 5.1) is simply IND-CPA secure if $\widehat{\text{dk}}_A$ is not given to \mathcal{A} in the above game.

Lastly, we define *ratchet simulatability*. This comes with two properties: updated key and ciphertext simulatability. The former property stipulates that the updated key $(\widehat{\text{ek}}_P, \widehat{\text{dk}}_P)$ can be simulated only from the non-updated key $(\text{ek}_P, \text{dk}_P)$. Importantly, the updated key does not depend on the peers encapsulation key $\text{ek}_{\bar{P}}$ required to run REnc-P . This is used to break the dependence on the updated keys from the peers keys, allowing us to prove security of CKA based on induction. The latter property stipulates that the ciphertext ct_P generated using the peer \bar{P} 's decapsulation key can be simulated using instead user P 's (updated) decapsulation key. Put differently, ct_P along with the knowledge of \bar{P} 's decapsulation key does not leak any information of P 's decapsulation key. This is a key property to argue PCS for CKA as it is used to argue that once P generates a fresh pair of key, it will *heal* P despite \bar{P} being corrupt. The formal definition is given in App. D.1.

Instantiations. In this work, we consider five instantiations of RKEM. A generic instantiation based of any KEM is presented in App. F. 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 Sec. 6. The Diffie-Hellman based instantiations modularize the Double Ratchet and the forward-secure variant thereof by Bienstock et al. [9] — for completeness they are presented in App. F.

5.2 A Generic Construction of CKA from Ratcheting KEM

We now present a simple construction of CKA based on RKEM. In the protocol, for each send operation, a party P first samples 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. 5 while a formal description is presented in App. D.2.

A proof of the following theorem is presented in App. D.3.

Theorem 5.1 (Key indistinguishability). *For any correct and forward-secure RKEM, the protocol from Fig. 15 is a correct and secure CKA protocol with $\Delta_{\text{FS}} = 0$ and $\Delta_{\text{PCS}} = 2$. Moreover, if the RKEM is non-forward secure, then the protocol is a secure CKA with $\Delta_{\text{FS}} = 1$ and $\Delta_{\text{PCS}} = 2$.*

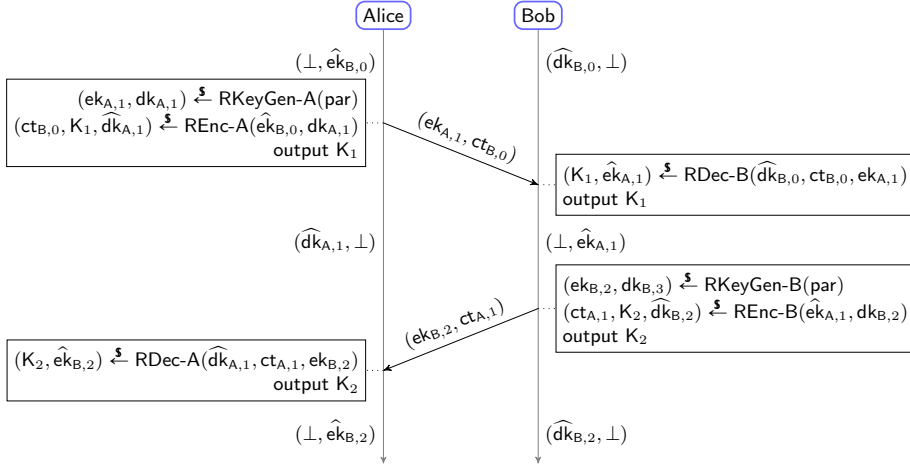


Fig. 5: The first two messages of the RKEM based CKA. Computation for CKA-Send-P and CKA-Rec-P are shown in boxes, while the state kept in between operations is shown next to the party.

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 $(\mathbf{u}, \text{seed}) \in R_q^k \times \{0, 1\}^\lambda$, outputs a tuple $(K, \mathbf{s}, \mathbf{e})$ 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 $\text{Encode} : \{0, 1\}^\lambda \rightarrow \mathcal{R}_q$ be a function that maps $\text{seed} \in \{0, 1\}^\lambda \subset \mathcal{R}_q$ to $\lfloor q/2 \rfloor \cdot \text{seed}$, where seed is viewed as a degree $\lambda - 1$ polynomial in \mathcal{R}_q with binary coefficients. Let $\text{Decode} : \mathcal{R}_q \rightarrow \{0, 1\}^\lambda$ be a function that maps each coefficient $w \in \mathcal{R}_q$ to 0 (resp. 1) if it is close to 0 (resp. $\lfloor q/2 \rfloor$) in absolute value.

Katana is based on the (IND-CPA secure) KEM by Lyubashevsky et al. [30] and Lindner and Peikert [28], underlying the NIST standard Kyber [33]. The construction is given in Fig. 6. For simplicity, we first provide the simplified variant where we do not perform bit-dropping. The optimized variant is given in Sec. 6.3. As we provide the correctness statement for our optimized scheme in Sec. 6.3, we defer the correctness of the non-optimized variant to App. E.1.

RKeyGen-P(par, mode)	REnc-P(\widehat{ek}_P, dk_P)	RDec-P($\widehat{dk}_P, ct_P, ek_{\bar{P}}$)
1 : if $\llbracket \text{mode} = \perp \rrbracket$	1 : $(\mathbf{u}_P, \mathbf{s}_P) := dk_P$	1 : $m := ct_P - ek_{\bar{P}}^\top \cdot \widehat{dk}_P$
2 : $(\mathbf{s}_P, \mathbf{e}_P) \stackrel{\$}{\leftarrow} \chi \times \chi$	2 : $seed \stackrel{\$}{\leftarrow} \{0, 1\}^\lambda$	2 : $seed := \text{Decode}(m)$
3 : else // mode = updated	3 : $m \leftarrow \text{Encode}(seed)$ // $m \in R_q$	3 : $(\mathbf{K}, \mathbf{s}, \mathbf{e}) := \text{H}(ek_{\bar{P}}, seed)$
4 : $(\mathbf{s}_P, \mathbf{e}_P) \stackrel{\$}{\leftarrow} \widehat{\chi} \times \widehat{\chi}$	4 : $(\mathbf{K}, \mathbf{s}, \mathbf{e}) := \text{H}(\mathbf{u}_P, seed)$	// Update $ek_{\bar{P}}$
5 : if $\llbracket P = A \rrbracket$ then	5 : $\tilde{e}_P \stackrel{\$}{\leftarrow} \tilde{\chi}$	4 : if $\llbracket P = A \rrbracket$ then
6 : $\mathbf{u}_A := \mathbf{D} \cdot \mathbf{s}_A + \mathbf{e}_A \in R_q^k$	6 : $v_{\bar{P}} := \widehat{ek}_{\bar{P}}^\top \cdot \mathbf{s}_P + \tilde{e}_P + m \in R_q$	5 : $\widehat{ek}_B := ek_B + \mathbf{D}^\top \cdot \mathbf{s} + \mathbf{e}$
7 : else // P = B	7 : $ct_{\bar{P}} := v_{\bar{P}}$	6 : else // P = B
8 : $\mathbf{u}_B := \mathbf{D}^\top \cdot \mathbf{s}_B + \mathbf{e}_B \in R_q^k$	// Update and erase dk_P	7 : $\widehat{ek}_A := ek_A + \mathbf{D} \cdot \mathbf{s} + \mathbf{e}$
9 : if $\llbracket \text{mode} = \perp \rrbracket$	8 : $\widehat{dk}_P := \mathbf{s}_P + \mathbf{s} \in R_q^k$	8 : return $(\mathbf{K}, \widehat{ek}_B)$
10 : $(ek_P, dk_P) := (\mathbf{u}_P, (\mathbf{u}_P, \mathbf{s}_P))$	9 : return $(ct_{\bar{P}}, \mathbf{K}, \widehat{dk}_P)$	RSetup(1^λ)
11 : else // mode = updated		1 : $par := \mathbf{D} \stackrel{\$}{\leftarrow} R_q^{k \times k}$
12 : $(ek_P, dk_P) := (\mathbf{u}_P, \mathbf{s}_P)$		2 : return par
13 : return (ek_P, dk_P)		

Fig. 6: Katana without the bit-dropping optimization. Above, $(P, \bar{P}) = (A, B)$ or (B, A) .

6.2 Security of Katana

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

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

Theorem 6.1 (FS-IND-CPA security). *Our RKEM Katana is FS-IND-CPA secure assuming the hardness of the $\text{MLWE}_{q,k,\chi}$ and the $\text{hint-MLWE}_{q,k,2k,\chi,\chi,\mathcal{F}_{\text{cpa}}}$ assumptions with $\mathcal{F}_{\text{cpa}} := \mathcal{U}(\{\mathbf{I}_{2k \times 2k}\})$.*

Proof Overview. The main technical argument of the proof is to prove that the encryption algorithm $\text{REnc-P}(\widehat{ek}_{\bar{P}}, dk_P)$ run by user P can be simulated *without* knowledge of P’s decapsulation key $dk_P = \mathbf{s}_P$. Focusing on $P = A$,

Notations	Explanation
R_q	Polynomial ring $R_q = \mathbb{Z}[X]/(q, X^n + 1)$ with $n \geq \lambda$
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 $\text{H} : R_q^k \times \{0, 1\}^\lambda \rightarrow \{0, 1\}^\lambda \times R_q^k$ 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 .

this would be a straightforward proof if not for forward-secrecy as $(\mathbf{ek}_A, \mathbf{ct}_B) = (\mathbf{D} \cdot \mathbf{s}_A + \mathbf{e}_A, \widehat{\mathbf{ek}}_B^\top \cdot \mathbf{s}_A + \tilde{e}_A + m)$ consists of a standard Kyber-like ciphertext. However, this is non-trivialized by the fact that we must simulate this all while being able to simulate the *updated* decapsulation key $\widehat{\mathbf{dk}}_A = \mathbf{s}_P + \mathbf{s}$ to the adversary, as required by forward-security, where recall \mathbf{s} is generated as $H(\mathbf{ek}_{\bar{P}}, \text{seed})$ with $m = \text{Encode}(\text{seed})$.

In our proof, we first modify the way the reduction simulates the random oracle H so that \mathbf{s} is no longer output by it. This modification allows the reduction to only require knowledge of $\widehat{\mathbf{dk}}_A = \mathbf{s}_A + \mathbf{s}$ as opposed to explicitly knowing \mathbf{s} . However, since we haven't established that m does not leak from the ciphertext \mathbf{ct}_B yet, we cannot immediately argue that this modification cannot be detected by the adversary. We will use deferred analysis (cf. [19]) to formally argue this. After this step is completed, we can rely on the *hint-MLWE* problem to argue that $(\mathbf{ek}_A, \mathbf{ct}_B)$ looks random while viewing the updated key $\widehat{\mathbf{dk}}_A$ as the *hint*. The full proof is given in App. E.2. \square

Ratchet Simulatability. The following theorem establishes the ratchet simulatability of Katana. Below, we rely on the *hint-MLWE* $_{q,k,1,\chi,\tilde{\chi},\mathcal{F}_{\text{sim}}}$ problem, where \mathcal{F}_{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 6.2 (Ratchet simulatability). *Our RKEM Katana is ratchet simulatable assuming the hardness of the $\text{MLWE}_{q,k,\tilde{\chi}}$ and $\text{hint-MLWE}_{q,k,1,\chi,\tilde{\chi},\mathcal{F}_{\text{sim}}}$ assumptions.*

Proof Overview. Ratchet simulatability consists of proving key and ciphertext simulatability. The former can be proven in a straightforward manner by observing that the information required to perform key update does not depend on the peer's (updated) encapsulation key. The latter is the more technical proof. Similarly to FS-IND-CPA security, the main technical argument of the proof is to prove that the encryption algorithm $\text{REnc-P}(\widehat{\mathbf{ek}}_{\bar{P}}, \mathbf{dk}_P)$ run by user P can be simulated. However, there are two critical differences: (i) the goal is *not* to hide the message m , and more importantly, (ii) the simulation must be performed while the adversary holds user \bar{P} 's decapsulation keys $(\mathbf{dk}_{\bar{P}}, \widehat{\mathbf{dk}}_{\bar{P}})$ — in particular, arguing $\mathbf{ct}_{\bar{P}}$ is random based on *hint-MLWE* does not work as the adversary can use the decapsulation keys to check this.

In our proof, we first aim to *statistically* simulate $\mathbf{ct}_{\bar{P}}$ using $\widehat{\mathbf{dk}}_{\bar{P}}$, without knowledge of \mathbf{dk}_P . Focusing on the case $P = A$, the ciphertext $\mathbf{ct}_A = \widehat{\mathbf{ek}}_A^\top \cdot \mathbf{s}_B + \tilde{e}_B + m$ can be equally expressed as $\widehat{\mathbf{s}}_A^\top \cdot \mathbf{ek}_B - \widehat{\mathbf{s}}_A^\top \cdot \mathbf{e}_B + \widehat{\mathbf{e}}_A^\top \cdot \mathbf{s}_B + \tilde{e}_B$, where we denote the underlined values as h . In particular, \mathbf{ct}_A can be simulated using $\widehat{\mathbf{dk}}_A = \widehat{\mathbf{s}}_A$ and h . However, notice the information of $\mathbf{dk}_B = (\mathbf{u}_B, \mathbf{s}_B)$ is required to compute h , and thus, \mathbf{ct}_A cannot be made independent from \mathbf{dk}_B .¹⁵ We thus only aim to *computationally* simulate \mathbf{ct}_A , where we rely on the *hint-MLWE* problem again,

¹⁵ While ratchet simulatability is a new notion in our work, the same argument should be required by [1,27], and this is the part where they contained a flaw in the proof.

but with the difference that the hint is now given in the form h . The actual proof consists of carefully setting up the game hybrids so that we can invoke this core argument. The full proof is given in App. E.3. \square

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 [33]. Namely, we drop the d lower bits of the ciphertext $\text{ct}_P = \widehat{\text{ek}}_P^\top \cdot \mathbf{s}_P + \tilde{e}_P + m \in R_q$ and map it to an element over $\text{mod } q_d$. These are standard techniques and the details are provided in App. E.4.

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 $\delta \stackrel{\$}{\leftarrow} \chi_{\text{round}}$ induced by bit-dropping for tighter concrete parameters. The proof is included in App. E.4.

Lemma 6.1 (Correctness with bit-dropping). *Our optimized RKEM Katana is correct assuming*

$$\Pr [\|\widehat{\mathbf{s}}^\top \cdot \mathbf{e} - \widehat{\mathbf{e}}^\top \cdot \mathbf{s} + \tilde{e} + \delta\|_\infty \leq q/4] = 1 - \text{negl}(\lambda), \quad (1)$$

where the probability is taken over the randomness to sample $(\mathbf{s}, \mathbf{e}) \stackrel{\$}{\leftarrow} \chi \times \chi$, $(\widehat{\mathbf{s}}, \widehat{\mathbf{e}}) \stackrel{\$}{\leftarrow} \widehat{\chi} \times \widehat{\chi}$, $\tilde{e} \stackrel{\$}{\leftarrow} \tilde{\chi}$, and $\delta \stackrel{\$}{\leftarrow} \chi_{\text{round}}$. Here, χ_{round} is some distribution over R_q such that $\Pr[\delta \stackrel{\$}{\leftarrow} \chi_{\text{round}} : \|\delta\|_\infty \leq \lfloor \frac{q}{2^{d+1}} \rfloor] = 1$.

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 5) must satisfy, where note that some requirements are subsumed by others. The first requirement stems from correctness (cf. Lemma 6.1), the second and third stem from FS-IND-CPA security (cf. Theorem 6.1), and the second and fourth stem from ratchet simulatability (cf. Theorem 6.2).

- (R1) The correctness error is below $2^{-\lambda}$ (see Lemma 6.1, Eq. (1)).
- (R2) The $\text{MLWE}_{q,k,\chi}$ and $\text{MLWE}_{q,k,\tilde{\chi}}$ problems are hard.
- (R3) The $\text{hint-MLWE}_{q,k,2k,\chi,\chi,\mathcal{F}_{\text{cpa}}}$ problem is hard, where $\mathcal{F}_{\text{cpa}} := \mathcal{U}(\{\mathbf{I}_{2k \times 2k}\})$, i.e., a distribution always outputting the identity matrix $\mathbf{I}_{2k \times 2k} \in R_q^{2k \times 2k}$.

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

Below, we focus on the conditions required by FS-IND-CPA security. The conditions Items (R2) and (R4) required by ratchet simulatability is obtained in a similar fashion. The condition Item (R1) on correctness is similar to Kyber, and in particular, we re-employ their analysis, [33, see Eqs. (7) and (8)] in order to characterize the distribution in Eq. (1). More details are found in App. E.5

FS-IND-CPA Security. 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 χ follows a Gaussian distribution of parameter σ , then the hint-MLWE reduction (cf. Theorem A.1) tells us that hint-MLWE $_{q,k,2k,\chi,\chi,\mathcal{F}_{\text{cpa}}}$ is at least as hard as MLWE $_{q,k,2k,\chi'}$, where χ' is the discrete Gaussian of parameter σ_0 : $\frac{1}{\sigma_0^2} = 2 \left(\frac{1}{\sigma^2} + \frac{s_1(\mathbf{I}_{2k})^2}{\sigma^2} \right) = \frac{4}{\sigma^2}$. In our case, we rely on two heuristics:

Heuristic 1: Replace Gaussians. While Theorem A.1 holds when the secret and noise are sampled from Gaussian distributions, we assume that it also the case with non-Gaussian distributions of equivalent variance σ^2 . In our case, we will sample $\tilde{\mathbf{e}}_p \stackrel{\$}{\leftarrow} \tilde{\chi}$ as a sum of uniforms and $\mathbf{s} \stackrel{\$}{\leftarrow} \chi$ from a binomial distribution (see below **Summary** for discussion); both types of distributions become ‘‘Gaussian-like’’ for some parameter regimes. It has also been argued in Raccoon [16,15] that in Rényi divergence-based arguments, sum of uniforms behave similarly to discrete Gaussians of identical variance σ^2 . For the present analysis, we conjecture that this is also the case in hint-MLWE.

Heuristic 2: Remove factor 2. We remove the factor 2 in the above equation. This is motivated by the fact that in [24], 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, the above equation simplifies to $\sigma_0 = \sigma/\sqrt{2}$, where σ is the standard variation of χ , and χ is not necessarily discrete Gaussian. We may then estimate the hardness of MLWE $_{q,k,2k,\chi'}$ using the lattice estimator¹⁶.

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

- $\chi = \text{CBD}_\eta$ is a centered binomial distribution, that is $\text{CBD}_\eta = [\eta] \cdot (\mathcal{B} - \mathcal{B})$, where $\mathcal{B} = \mathcal{U}(\{0,1\})$ is the Bernoulli distribution of parameter 1/2. This distribution is used in Kyber [33].

¹⁶ <https://github.com/malb/lattice-estimator>

Target λ	CoreSVP hardness	n	k	q	q_a	χ	$\tilde{\chi}$	d	$ \text{ek} $	$ \text{ct} $	$ \text{ek} + \text{ct} $
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 3: 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.

- $\tilde{\chi} = \text{SU}(u, T)$ is the sum of T uniformly random variates over $\{-2^{u-1}, \dots, 2^{u-1} - 1\}$, that is $\text{SU}(u, T) = [T] \cdot \mathcal{U}(\{-2^{u-1}, \dots, 2^{u-1} - 1\})$. This distribution is used in *Raccoon* [16].

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 3, 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. App. F). 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 4, 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 3).

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 [22].

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 $\Delta_{\text{PCS}}^{\text{PQ3}} = 2$ where $\Delta_{\text{PCS}}^{\text{TR}} = 3$. 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 4 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 Δ_{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.

	PQ3	TR with Kyber-768	TR with 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 4: 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.

References

1. J. Alwen, S. Coretti, and Y. Dodis. The double ratchet: Security notions, proofs, and modularization for the Signal protocol. *EUROCRYPT 2019, Part I*, pp. 129–158, Darmstadt, Germany, 2019.
2. J. Alwen, D. Hartmann, E. Kiltz, and M. Mularczyk. Server-aided continuous group key agreement. *ACM CCS 2022*, pp. 69–82, Los Angeles, CA, USA, 2022.
3. J. Alwen, D. Jost, and M. Mularczyk. On the insider security of MLS. *CRYPTO 2022, Part II*, pp. 34–68, Santa Barbara, CA, USA, 2022.
4. Apple Security Engineering and Architecture (SEAR). iMessage with PQ3: The new state of the art in quantum-secure messaging at scale.
5. F. Balli, P. Rösler, and S. Vaudenay. Determining the core primitive for optimally secure ratcheting. *ASIACRYPT 2020, Part III*, pp. 621–650, Daejeon, South Korea, 2020.
6. M. Bellare, A. C. Singh, J. Jaeger, M. Nyayapati, and I. Stepanovs. Ratcheted encryption and key exchange: The security of messaging. *CRYPTO 2017, Part III*, pp. 619–650, Santa Barbara, CA, USA, 2017.
7. K. Bhargavan, A. Bichhawat, Q. H. Do, P. Hosseini, R. Küsters, G. Schmitz, and T. Würtele. DY*: A modular symbolic verification framework for executable cryptographic protocol code. In *2021 IEEE European Symposium on Security and Privacy*, pp. 523–542, Vienna, Austria, 2021.
8. K. Bhargavan, C. Jacomme, F. Kiefer, and R. Schmidt. Formal verification of the PQXDH post-quantum key agreement protocol for end-to-end secure messaging. *USENIX Security 2024*, Philadelphia, PA, USA, 2024.
9. A. Bienstock, J. Fairoze, S. Garg, P. Mukherjee, and S. Raghuraman. A more complete analysis of the Signal double ratchet algorithm. *CRYPTO 2022, Part I*, pp. 784–813, Santa Barbara, CA, USA, 2022.
10. A. Caforio, F. B. Durak, and S. Vaudenay. Beyond security and efficiency: On-demand ratcheting with security awareness. *PKC 2021, Part II*, pp. 649–677, Virtual Event, 2021.
11. R. Canetti, P. Jain, M. Swanberg, and M. Varia. Universally composable end-to-end secure messaging. *CRYPTO 2022, Part II*, pp. 3–33, Santa Barbara, CA, USA, 2022.

12. S. Chen and M. Fischlin. Integrating causality in messaging channels. *EUROCRYPT 2024, Part III*, pp. 251–282, Zurich, Switzerland, 2024.
13. K. Cohn-Gordon, C. Cremers, B. Dowling, L. Garratt, and D. Stebila. A formal security analysis of the Signal messaging protocol. *Journal of Cryptology*, 33(4):1914–1983.
14. D. Collins, D. Riepel, and S. A. O. Tran. On the tight security of the double ratchet. *ACM CCS 2024*, pp. 4747–4761, Salt Lake City, UT, USA, 2024.
15. R. d Pino, S. Katsumata, T. Prest, and M. Rossi. Raccoon: A masking-friendly signature proven in the probing model. *CRYPTO 2024, Part I*, pp. 409–444, Santa Barbara, CA, USA, 2024.
16. R. d Pino, T. Espitau, S. Katsumata, M. Maller, F. Mouhartem, T. Prest, M. Rossi, and M.-J. 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>.
17. N. Drucker and S. Gueron. Continuous key agreement with reduced bandwidth. In *International Symposium on Cyber Security Cryptography and Machine Learning*, pp. 33–46. Springer.
18. M. F. Esgin, T. Espitau, G. Niot, T. Prest, A. Sakzad, and R. Steinfeld. Plover: Masking-friendly hash-and-sign lattice signatures. *EUROCRYPT 2024, Part VII*, pp. 316–345, Zurich, Switzerland, 2024.
19. R. Gennaro and V. Shoup. A note on an encryption scheme of kurosawa and desmedt. *Cryptology ePrint Archive*, Report 2004/194.
20. K. Hashimoto, S. Katsumata, E. Postlethwaite, T. Prest, and B. Westerbaan. A concrete treatment of efficient continuous group key agreement via multi-recipient PKEs. *ACM CCS 2021*, pp. 1441–1462, Virtual Event, Republic of Korea, 2021.
21. K. Hashimoto, S. Katsumata, and T. Prest. How to hide MetaData in MLS-like secure group messaging: Simple, modular, and post-quantum. *ACM CCS 2022*, pp. 1399–1412, Los Angeles, CA, USA, 2022.
22. F. Jacobs. Designing imessage pq3: Quantum-secure messaging at scale. Invited talk at the Real World Crypto Symposium 2025, 2024.
23. D. Jost, U. Maurer, and M. Mularczyk. Efficient ratcheting: Almost-optimal guarantees for secure messaging. *EUROCRYPT 2019, Part I*, pp. 159–188, Darmstadt, Germany, 2019.
24. D. Kim, D. Lee, J. Seo, and Y. Song. Toward practical lattice-based proof of knowledge from hint-MLWE. *CRYPTO 2023, Part V*, pp. 549–580, Santa Barbara, CA, USA, 2023.
25. N. Kobeissi, K. Bhargavan, and B. 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*, pp. 435–450.
26. E. Kret and R. Schmidt. The pqxdh key agreement protocol.
27. J. Lee, J. Kwon, and J. S. Shin. Efficient continuous key agreement with reduced bandwidth from a decomposable kem. *IEEE Access*, 11:33224–33235.
28. R. Lindner and C. Peikert. Better key sizes (and attacks) for LWE-based encryption. *CT-RSA 2011*, pp. 319–339, San Francisco, CA, USA, 2011.
29. F. Linker, R. Sasse, and D. Basin. A formal analysis of apple’s iMessage PQ3 protocol. *Cryptology ePrint Archive*, Paper 2024/1395.
30. V. Lyubashevsky, C. Peikert, and O. Regev. On ideal lattices and learning with errors over rings. *EUROCRYPT 2010*, pp. 1–23, French Riviera, 2010.
31. M. Marlinspike and T. Perrin. The double ratchet algorithm.

32. M. Marlinspike and T. Perrin. The x3dh key agreement protocol.
33. P. Schwabe, R. Avanzi, J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, G. Seiler, D. Stehlé, and J. 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>.
34. P. W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *35th FOCS*, pp. 124–134, Santa Fe, NM, USA, 1994.
35. D. Stebila. Security analysis of the iMessage PQ3 protocol. Cryptology ePrint Archive, Report 2024/357.
36. N. Vatandas, R. Gennaro, B. Ithurburn, and H. Krawczyk. On the cryptographic deniability of the Signal protocol. *ACNS 20International Conference on Applied Cryptography and Network Security, Part II*, pp. 188–209, Rome, Italy, 2020.

SUPPLEMENTARY MATERIALS

A Omitted Preliminary

A.1 Lattices and Hardness Assumption

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 $\mathbf{M}^t \cdot \mathbf{M}$. 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} .

Hardness Assumption. In this work, we also rely on the standard module learning with errors (MLWE) problem, formally defined as follows.

Definition A.1 (MLWE). *Let k, q be integers and χ be a probability distribution over \mathcal{R}_q . The advantage of an adversary \mathcal{A} against the Module Learning with Errors $\text{MLWE}_{q,k,\chi}$ problem is defined as:*

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

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 $\text{MLWE}_{q,k,\chi}$ assumption states that any efficient adversary \mathcal{A} has negligible advantage.

The following result establishes the hardness of the hint-MLWE problem based on the MLWE problem. This is a simple adaptation of the original proof [24], formally appearing in [18]. Below, we denote \mathcal{D}_σ as a discrete Gaussian distribution with standard deviation σ .

Theorem A.1 (Hardness of hint-MLWE). *For any integers k, ℓ, q, n , let \mathcal{F} be a probability distribution over $\mathcal{R}_q^{\ell \times 2k}$, χ and $\tilde{\chi}$ be discrete Gaussian distributions \mathcal{D}_{σ_1} and \mathcal{D}_{σ_2} , respectively, and \mathcal{B}, σ a positive real such that*

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

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

A.2 Rounding

In our work, we use the rounding definition used by Kyber [33]. 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 $-\frac{q}{2} < x' \leq \frac{q}{2}$ (resp. $-\frac{q-1}{2} < x' \leq \frac{q-1}{2}$) 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 $0 \leq x' < q$ such that $x' = x \bmod q$. We simply write $x \bmod q$ when the representation is not important. Also, for an element in $x \in \mathbb{Q}$, $\lfloor x \rfloor$ 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 $d < \lceil \log_2(q) \rceil$:

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

$$\begin{aligned} \text{Decompress}_q : \mathbb{Z}_{2^d} &\longrightarrow \mathbb{Z}_q \\ y &\longmapsto \left\lfloor \frac{q}{2^d} \cdot y \right\rfloor. \end{aligned} \tag{3}$$

For these functions, we have the following:

Lemma A.1. 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

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

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

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

A.3 CKA Security

In this section, we recall CKA security as introduced in [1]. Note that we use separate parameters Δ_{FS} and Δ_{PCS} for FS and PCS, respectively, whereas [1] hardcoded $\Delta_{\text{PCS}} = 2$.

Definition A.2 (Key Indistinguishability). Let Δ_{FS} and Δ_{PCS} be positive integers, dictating how fast forward secrecy and post-compromise security come into effect. For a CKA protocol Π_{CKA} , the advantage of an adversary \mathcal{A} against key indistinguishability is defined as

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

where $\text{Game}_{\mathcal{A}, \Delta_{\text{FS}}, \Delta_{\text{PCS}}, \hat{t}^*}^{\text{CKA}}(1^\lambda)$ for any challenge epoch $\hat{t}^* \in \mathbb{N}$ is described in Fig. 7.

We say Π_{CKA} is $(\Delta_{\text{FS}}, \Delta_{\text{PCS}})$ -key indistinguishable if for any efficient \mathcal{A} that respects alternating communications (cf. Remark 2.1), we have $\text{Adv}_{\mathcal{A}, \Delta_{\text{FS}}, \Delta_{\text{PCS}}}^{\text{CKA}}(1^\lambda) = \text{negl}(\lambda)$. 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 .

<p>Game_{$\mathcal{A}, \Delta_{\text{FS}}, \Delta_{\text{PCS}}, \hat{t}^*$}^{CKA}($1^\lambda$)</p> <hr/> <p>1 : $b \xleftarrow{\\$} \{0, 1\}$</p> <p>2 : $\text{l}_{\text{CKA}} \xleftarrow{\\$} \text{CKA-Init-KeyGen}(1^\lambda)$ // Initial key</p> <p>3 : for $P \in \{A, B\}$</p> <p>4 : $\text{st}_P \xleftarrow{\\$} \text{CKA-Init-P}(\text{l}_{\text{CKA}})$</p> <p>5 : $\hat{t}_P \leftarrow 0$</p> <p>6 : $b' \xleftarrow{\\$} \mathcal{A}(\hat{t}^*)^{\text{Send-P}(), \text{Receive-P}(), \text{Chall-P}(), \text{Corr-P}()}$</p> <p>7 : return $\llbracket b = b' \rrbracket$</p> <hr/> <p>Send-P(rleak)</p> <hr/> <p>1 : $\hat{t}_P \leftarrow \hat{t}_P + 1$</p> <p>2 : if $\llbracket \text{rleak} \rrbracket$ // Leak randomness // Allow leaking rand. Δ_{PCS}-epoch before \hat{t}^*</p> <p>3 : req $\llbracket \hat{t}_A, \hat{t}_B \leq \hat{t}^* - \Delta_{\text{PCS}} \rrbracket$</p> <p>4 : $\text{rand} \xleftarrow{\\$} \mathcal{R}$</p> <p>5 : $(K_{\hat{t}_P}, \rho_{\hat{t}_P}, \text{st}_P) \leftarrow \text{CKA-Send-P}(\text{st}_P; \text{rand})$</p> <p>6 : else // Secure randomness</p> <p>7 : $\text{rand} \leftarrow \perp$</p> <p>8 : $(K_{\hat{t}_P}, \rho_{\hat{t}_P}, \text{st}_P) \xleftarrow{\\$} \text{CKA-Send-P}(\text{st}_P)$</p> <p>9 : return $(K_{\hat{t}_P}, \rho_{\hat{t}_P}, \text{rand})$</p>	<p>Chall-P()</p> <hr/> <p>1 : $\hat{t}_P \leftarrow \hat{t}_P + 1$</p> <p>2 : req $\llbracket \hat{t}_P = \hat{t}^* \rrbracket$ // Challenge epoch \hat{t}^*</p> <p>3 : $(K_{\hat{t}_P}, \rho_{\hat{t}_P}, \text{st}_P) \xleftarrow{\\$} \text{CKA-Send-P}(\text{st}_P)$</p> <p>4 : if $\llbracket b = 0 \rrbracket$ then</p> <p>5 : $K \leftarrow K_{\hat{t}_P}$</p> <p>6 : else</p> <p>7 : $K \xleftarrow{\\$} \mathcal{K}$ // Replace with random key</p> <p>8 : return $(K, \rho_{\hat{t}_P})$</p> <hr/> <p>Receive-P()</p> <hr/> <p>1 : $\hat{t}_P \leftarrow \hat{t}_P + 1$</p> <p>2 : $(K, \text{st}_P) \xleftarrow{\\$} \text{CKA-Rec-P}(\text{st}_P, \rho_{\hat{t}_P})$</p> <p>3 : assert $\llbracket K = K_{\hat{t}_P} \rrbracket$ // Correctness</p> <hr/> <p>Corr-P()</p> <hr/> <p> // Allow corrupting Δ_{PCS}-epoch before \hat{t}^*</p> <p>1 : req $\llbracket \hat{t}_A, \hat{t}_B \leq \hat{t}^* - \Delta_{\text{PCS}} \rrbracket$</p> <p> // Allow corrupting Δ_{PCS}-epoch after \hat{t}^*</p> <p>2 : req $\llbracket \hat{t}_P \geq \hat{t}^* + \Delta_{\text{FS}} \rrbracket$</p> <p>3 : return st_P</p>
---	--

Fig. 7: 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 .

Remark A.1 (Bad randomness). We deviate from [1] (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 App. G.

A.4 Chunk Encoding

Definition A.3. An erasure code for a set of symbols Σ , a block length N , and a message size n_{chunk} consists of PPT algorithms $\text{Encode}, \text{Decode}$ defined as follows:

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

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

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

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_{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_{chunk} symbols.

A.5 Cryptographic Primitives

Definition A.4 (Authenticated Encryption). An authenticated encryption with associated data (AEAD) scheme is a pair of algorithms $\text{AEAD} := (\text{Enc}, \text{Dec})$ with the following syntax:

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

$\text{Dec}(K, h, c) \rightarrow M'$: It takes a key K , authenticated data h , and a ciphertext c , and outputs a plaintext M' .

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

We say that an AEAD scheme is correct, if for all keys K and all pairs (h, M) ,

$$\text{Dec}(K, h, \text{Enc}(K, h, m)) = m.$$

We require AEAD to be one-time IND-CCA secure, formalized by the game in Fig. 8, and define the following advantage

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

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

$\text{Game}_{\mathcal{A}}^{\text{AEAD}}(1^\lambda)$	$\text{encrypt}(h, M)$	$\text{decrypt}(h, e)$
1 : $b \xleftarrow{\$} \{0, 1\}$	1 : if $\llbracket b = 0 \rrbracket$ then	1 : if $\llbracket e = e^* \rrbracket \vee \llbracket b = 1 \rrbracket$ then
2 : $K \xleftarrow{\$} \{0, 1\}^\lambda$	2 : $e^* \leftarrow \text{Enc}(K, h, M)$	2 : return \perp
3 : $e^* \leftarrow \perp$	3 : else	3 : return $\text{Dec}(K, h, e)$
4 : $b' \xleftarrow{\$} \mathcal{A}(1^\lambda)^{\text{encrypt}(), \text{decrypt}()}$	4 : $e^* \xleftarrow{\$} \mathcal{C}$	
5 : return $\llbracket b = b' \rrbracket$	5 : return e^*	

Fig. 8: The one-time IND-CCA game of an AEAD scheme (Enc, Dec) with ciphertext space \mathcal{C} , where encrypt is a one-time oracle.

Key Derivation Functions (KDF). We use several KDF in our work. Syntax wise, 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 KDF, in slight abuse of notation we overload the same function to output a variable number of arguments. In practice, each KDF would be instantiated by a hash-based construction such as HKDF. In the following we outline the different security assumptions we need.

Definition A.5 (Pseudorandom generator (PRG)). A KDF with one input argument is said to behave like a PRG, if

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

is negligible for every PPT \mathcal{A} .

Definition A.6 ((dual)-PRF). A KDF with two input arguments is said to behave like a pseudo-random function (PRF), if

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

is negligible for every PPT \mathcal{A} , for the game from Fig. 9. 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 σ and I are swapped, i.e., where initially I is sampled and chall , eval take σ as input, and \mathcal{F} is indexed by σ .

$\text{Game}_{\mathcal{A}}^{\text{PRF}}(1^\lambda)$	$\text{chall}(I)$	$\text{eval}(I)$
1 : $b \xleftarrow{\$} \{0, 1\}$	1 : $(\sigma', R) \leftarrow \text{KDF}(\sigma, I)$	1 : if $\llbracket \mathcal{F}[I] = \perp \rrbracket$ then
2 : $\sigma \xleftarrow{\$} \{0, 1\}^\lambda$	2 : if $\llbracket b = 1 \rrbracket$ then	2 : $\mathcal{F}[I] \xleftarrow{\$} \mathcal{R}$
3 : $\mathcal{F}[\cdot] \leftarrow \perp$	3 : $R \leftarrow \text{eval}(I)$	3 : return $\mathcal{F}[I]$
4 : $b' \xleftarrow{\$} \mathcal{A}(1^\lambda)^{\text{chall}}$	4 : return (σ', R)	
5 : return $\llbracket b = b' \rrbracket$		

Fig. 9: PRF security of a two-input KDF. If KDF expands to more than two return values, then the first one is σ and the remaining outputs should be considered as R over an appropriate composite space \mathcal{R} .

Definition A.7 (PRF-PRNG). A KDF with two input arguments is said to have PRF-PRNG security, if

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

is negligible for every PPT \mathcal{A} , for the game from Fig. 10.

B More Details on Hybrid Messaging

In this section, we present the formal security and correctness definition of a (hybrid) secure messaging scheme. The game is shown in Fig. 11. The game is a generalization of the game from Alwen et al. and we, therefore, refer to [1] for an indepth discussion thereof.

On a high-level, 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 to predicate **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 [1], 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.

$\text{Game}_{\mathcal{A}}^{\text{PRF-PRNG}}(1^\lambda)$	$\text{process}(I)$	$\text{chall-prng}(I)$
1 : $b \xleftarrow{\$} \{0, 1\}$ 2 : $\sigma \xleftarrow{\$} \{0, 1\}^\lambda$ 3 : $\text{corr}, \text{prng}, \text{prf} \leftarrow \text{false}$ 4 : $\mathcal{F}[\cdot] \leftarrow \perp$ 5 : $b' \xleftarrow{\$} \mathcal{A}(1^\lambda)^{\text{process}, \text{chall-prf}, \text{chall-prng}, \text{corr}}$ 6 : return $\llbracket b = b' \rrbracket$	1 : $I \leftarrow \text{sample-if-nec}(I)$ 2 : $(\sigma, R) \leftarrow \text{KDF}(\sigma, I)$ 3 : return R	1 : $I \leftarrow \text{sample-if-nec}(I)$ 2 : req $\llbracket \neg \text{corr} \rrbracket \wedge \llbracket \neg \text{prf} \rrbracket$ 3 : $\text{prng} \leftarrow \text{true}$ 4 : $(\sigma, R) \leftarrow \text{KDF}(\sigma, I)$ 5 : if $\llbracket b = 1 \rrbracket$ then 6 : $R \xleftarrow{\$} \mathcal{R}$ 7 : return R
corr() 1 : req $\llbracket \neg \text{prf} \rrbracket$ 2 : $\text{corr} \leftarrow \text{true}$ 3 : return σ	chall-prf (I) 1 : req $\llbracket \neg \text{corr} \rrbracket \wedge \llbracket \neg \text{prng} \rrbracket$ 2 : $\text{prf} \leftarrow \text{true}$ 3 : $(\sigma', R) \leftarrow \text{KDF}(\sigma, I)$ 4 : if $\llbracket b = 1 \rrbracket$ then 5 : $R \leftarrow \text{eval}(I)$ 6 : return (σ', R)	
	sample-if-nec (I) 1 : if $\llbracket I = \perp \rrbracket$ then 2 : $I \xleftarrow{\$} \mathcal{I}$ 3 : $\text{corr} \leftarrow \text{false}$ 4 : return I	eval (I) 1 : if $\llbracket \mathcal{F}[I] = \perp \rrbracket$ then 2 : $\mathcal{F}[I] \xleftarrow{\$} \mathcal{R}$ 3 : return $\mathcal{F}[I]$

Fig. 10: PRF-PRNG security of a two-input KDF. If KDF expands to more than two return values, then the first one is σ and the remaining tuple of outputs should be considered R over an appropriate composite space \mathcal{R} .

In the security game, correctness thus enforces that the receipt outputs the correct message index (in addition to the correct message) as part of the respective oracle. The game moreover uses an *epoch function* $\tau(\text{id}_x)$ 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 is then defined in the number of epochs Δ_{FS} and Δ_{PCS} , respectively, describing the corruption window. The game uses the helper algorithm `epoch-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 `corr-mgmt` moreover keeps track of corrupted epochs — updating the last corrupted epoch \mathfrak{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 an *period function* $\iota(\text{id}_x)$ 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 `safe-inj`.) If all messages have been delivered, then we expect that no injections can be performed outside such a windows 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_{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_{comp} only one injection happens with the same epoch-period pair. (The overall message index, however, may differ.)

Definition B.1. For a SM protocol Π_{SM} with message index space \mathcal{Idx} , let $\tau: \mathcal{Idx} \rightarrow \mathbb{N}$ be an epoch function that dictates how fast forward secrecy and post-compromise security come into effect, measured as positive integers Δ_{FS} and Δ_{PCS} , respectively. The advantage of an adversary \mathcal{A} is defined as

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

where the game is described in Fig. 11. We say Π_{SM} is $(\Delta_{\text{FS}}, \Delta_{\text{PCS}}, \tau)$ -secure if for any efficient \mathcal{A} we have $\text{Adv}_{\mathcal{A}, \Delta_{\text{PCS}}, \Delta_{\text{FS}}, \tau}^{\text{SM}}(1^\lambda) = \text{negl}(\lambda)$.

<p>Game$_{A, \Delta_{PCS}, \Delta_{FS}, \tau}^{SM}(1^\lambda)$</p> <hr/> <pre> 1 : $b \xleftarrow{\\$} \{0, 1\}$ 2 : $k \xleftarrow{\\$} \text{SM-Init-KeyGen}(1^\lambda)$ // Sample initial key 3 : for $P \in \{A, B\}$ 4 : $st_P \xleftarrow{\\$} \text{SM-Init-P}(k)$ 5 : $(t_{\text{Chall-P}}, \text{idx}_P) \leftarrow (0, -\infty)$ 6 : $t_L \leftarrow -\infty$ 7 : $L_{\text{trans}}, L_{\text{chall}}, L_{\text{comp}} \leftarrow \emptyset$ 8 : $b' \xleftarrow{\\$} \mathcal{A}(1^\lambda)^{\text{Send-P}(), \text{Receive-P}(), \text{Chall-P}(), \text{Corr-P}()}$ 9 : return $\llbracket b = b' \rrbracket$ </pre>	<p>Send-A(M, rleak)</p> <hr/> <pre> 1 : $\text{rand} \xleftarrow{\\$} \mathcal{R}$ 2 : $(ct, st_A) \xleftarrow{\\$} \text{SM-Send-A}(st_A, M; \text{rand})$ 3 : $\text{epoch-mgmt}(A, \text{send}, \text{rleak})$ 4 : $\text{record} := (A, M, \text{idx}_A, ct)$ 5 : $L_{\text{trans}} \xleftarrow{+} \text{record}$ 6 : if $\llbracket \neg \text{safe-chall}(A) \rrbracket$ then 7 : $L_{\text{comp}} \xleftarrow{+} \text{record}$ 8 : if $\llbracket \neg \text{rleak} \rrbracket$ then $\text{rand} \leftarrow \perp$ 9 : return $(ct, \text{idx}_A, \text{rand})$ </pre>	
<p>Chall-A(M₀, M₁, rleak)</p> <hr/> <pre> 1 : $\text{rand} \xleftarrow{\\$} \mathcal{R}$ 2 : req $\llbracket M_0 = M_1 \rrbracket$ 3 : $(ct, st_A) \xleftarrow{\\$} \text{SM-Send-A}(st_A, M_b; \text{rand})$ 4 : $\text{epoch-mgmt}(A, \text{chall}, \text{rleak})$ 5 : req $\llbracket \text{safe-chall}(A) \rrbracket$ 6 : $\text{record} := (A, M_b, \text{idx}_A, ct)$ 7 : $L_{\text{trans}}, L_{\text{chall}}, L_{\text{comp}} \xleftarrow{+} \text{record}$ 8 : if $\llbracket \neg \text{rleak} \rrbracket$ then $\text{rand} \leftarrow \perp$ 9 : return $(ct, \text{idx}_A, \text{rand})$ </pre>	<p>Receive-A(ct)</p> <hr/> <pre> 1 : req $\llbracket (B, _, _) \in L_{\text{trans}} \rrbracket$ 2 : $(M', \text{idx}', st_A) \xleftarrow{\\$} \text{SM-Rec-A}(st_A, ct)$ 3 : $\text{epoch-mgmt}(A, \text{receive}, \text{false})$ 4 : $\text{record} := (B, M', \text{idx}', ct)$ // Correctness guarantee 5 : assert $\llbracket \text{record} \in L_{\text{trans}} \rrbracket$ 6 : if $\llbracket \text{record} \in L_{\text{chall}} \rrbracket$ then 7 : $M' \leftarrow \perp$ 8 : $L_{\text{trans}}, L_{\text{chall}}, L_{\text{comp}} \xleftarrow{-} \text{record}$ 9 : return (M', idx') </pre>	
<p>Inject-A(ct)</p> <hr/> <pre> 1 : req $\llbracket (B, _, _) \notin L_{\text{trans}} \rrbracket \wedge \llbracket \text{safe-inj}() \rrbracket$ 2 : $(M', \text{idx}', st_A) \xleftarrow{\\$} \text{SM-Rec-A}(st_A, ct)$ 3 : $\text{epoch-mgmt}(A, \text{receive}, \text{false})$ // Authenticity guarantee 4 : if $\llbracket M' \neq \perp \rrbracket$ then 5 : assert $\exists \text{idx}'' : \llbracket \text{equiv}(\text{idx}', \text{idx}'') \rrbracket$ $\wedge \llbracket (B, _, \text{idx}'', _) \in L_{\text{comp}} \rrbracket$ 6 : foreach $\text{idx}'' : \text{equiv}(\text{idx}', \text{idx}'')$ 7 : if $\llbracket (B, _, \text{idx}'', _) \in L_{\text{trans}} \rrbracket$ then 8 : $L_{\text{trans}}, L_{\text{chall}}, L_{\text{comp}} \xleftarrow{-} (B, _, \text{idx}'', _)$ 9 : return (M', idx') </pre>	<p>Corr-A()</p> <hr/> <pre> 1 : req $\llbracket (B, _, _) \notin L_{\text{chall}} \rrbracket \wedge \llbracket \text{safe-corr}(A) \rrbracket$ 2 : foreach $(B, M', \text{idx}', ct') \in L_{\text{trans}}$ 3 : $L_{\text{comp}} \xleftarrow{+} (B, M', \text{idx}', ct')$ 4 : $t_L \leftarrow \max(t_A, t_B)$ 5 : return st_A </pre>	
<p>epoch-mgmt(P, act, rleak)</p> <hr/> <pre> 1 : $\text{idx} \leftarrow \text{idx}(st_P)$ 2 : $(t, t_P) \leftarrow (\tau(\text{idx}), \tau(\text{idx}_P))$ 3 : $(i, i_P) \leftarrow (\iota(\text{idx}), \iota(\text{idx}_P))$ 4 : if $\llbracket \text{act} \in \{\text{send}, \text{chall}\} \rrbracket$ then 5 : assert $\llbracket \text{idx} > \text{idx}_P \rrbracket$ 6 : else assert $\llbracket \text{idx} \geq \text{idx}_P \rrbracket$ 7 : assert $\llbracket t \geq t_P \rrbracket$ 8 : if $\llbracket t > t_P \rrbracket$ then 9 : assert $\llbracket i = 1 \rrbracket$ 10 : elseif $\llbracket \text{act} \in \{\text{send}, \text{chall}\} \rrbracket$ then 11 : assert $\llbracket i = i_P + 1 \rrbracket$ 12 : else assert $\llbracket i = i_P \rrbracket$ 13 : $\text{corr-mgmt}(P, \text{act}, \text{rleak}, t, t_P)$ 14 : $\text{idx}_P \leftarrow \text{idx}$ </pre>	<p>corr-mgmt(P, act, rleak, t, t_P)</p> <hr/> <pre> 1 : if $\llbracket \text{act} = \text{chall} \rrbracket$ then $t_{\text{Chall-P}} \leftarrow t$ 2 : if $\llbracket \text{rleak} \rrbracket$ then $\text{bad-rand}_P \leftarrow \text{true}$ 3 : if $\llbracket t > t_P \rrbracket \wedge \llbracket \text{bad-rand}_P \rrbracket$ then 4 : while $\llbracket t_P < t \rrbracket$ 5 : $t_P += 1$ 6 : if $\llbracket \text{sending-ep}(P, t) \rrbracket$ $\wedge \llbracket \neg \text{safe-chall}(P) \rrbracket$ then 7 : $t_L \leftarrow \max(t_L, t)$ 8 : $\text{bad-rand}_P \leftarrow \text{false}$ </pre> <p>equiv(idx₁, idx₂)</p> <hr/> <pre> 1 : return $\llbracket \tau(\text{idx}_1) = \tau(\text{idx}_2) \rrbracket$ $\wedge \llbracket i(\text{idx}_1) = i(\text{idx}_2) \rrbracket$ </pre>	<p>sending-ep(P, t)</p> <hr/> <pre> 1 : return $\llbracket P = A \text{ and } t \text{ is odd} \rrbracket$ $\vee \llbracket P = B \text{ and } t \text{ is even} \rrbracket$ </pre> <p>safe-chall(P) // Δ_{PCS} after last corruption</p> <hr/> <pre> 1 : return $\llbracket t_P \geq t_L + \Delta_{PCS} \rrbracket$ </pre> <p>safe-inj() // Once both parties healed</p> <hr/> <pre> 1 : return $\llbracket \min(t_A, t_B) \geq t_L + \Delta_{PCS} \rrbracket$ </pre> <p>safe-corr(P) // Δ_{FS} epochs after last challenge</p> <hr/> <pre> 1 : return $\llbracket t_P \geq t_{\text{Chall-P}} + \Delta_{FS} \rrbracket$ </pre>

Fig. 11: The SM security game parametrized in the epoch function τ , the number of epochs Δ_{PCS} for PCS, and the number of epochs Δ_{FS} for FS. The period function ι is used for bookkeeping purposes and not security relevant. The oracles for B are defined analogously.

C Triple Ratchet: Security Proof

In this section, we prove the SM security of the TR protocol, as stated in the following theorem.

Theorem C.1 (Theorem 4.1 restated). *For the TR protocol, let τ^C and τ^Q denote the epoch functions as discussed in Remark 4.1, and let $v^C := iS^C$ and $v^Q := iS^Q$ denote the period functions. Assume that*

- CKA^C is $(\Delta_{FS}^C, \Delta_{PCS}^C)$ -secure CKA scheme or CKA^Q is $(\Delta_{FS}^Q, \Delta_{PCS}^Q)$ -secure CKA scheme;
- CKA^C and CKA^Q are both correct;
- KDF_1 is a secure PRF-PRNG, KDF_2 is a secure PRG, and KDF_3 is a secure dual-PRF;
- AEAD is a secure authenticated encryption scheme with associated data.

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

$$\begin{aligned} \text{Adv}_{\mathcal{A}, \Delta_{PCS}^Q + 1, \Delta_{FS}^Q, \tau^Q}^{\text{SM}}(1^\lambda) &\leq \text{Adv}_{\mathcal{A}_1}^{\text{CKA-corr}^C}(1^\lambda) + \text{Adv}_{\mathcal{A}_2}^{\text{CKA-corr}^Q}(1^\lambda) \\ &\quad + 2q^2 \left(\text{Adv}_{\mathcal{B}, \Delta_{PCS}^Q, \Delta_{FS}^Q}^{\text{CKA}^Q}(1^\lambda) + q \cdot \text{Adv}_C^{\text{PRF-PRNG}}(1^\lambda) + q \cdot \text{Adv}_D^{\text{PRG}}(1^\lambda) \right. \\ &\quad \left. + \text{Adv}_E^{\text{dPRF}}(1^\lambda) + \text{Adv}_F^{\text{AEAD}}(1^\lambda) \right) \end{aligned}$$

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

In the above, $\text{Adv}_{\mathcal{A}'}^{\text{CKA-corr}}$ denotes the advantage of \mathcal{A}' breaking the correctness¹⁷ of the CKA and the remaining advantage terms formalizing the aforementioned security assumptions on the underlying primitives.

As observed in [1], the SM security game can be split into separate games for correctness, authenticity, and confidentiality, as stated by the following lemma.

Lemma C.1. *In the following, let*

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

It holds that

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

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

Lemma C.2. *Assuming the AEAD to have perfect correctness, then the Triple Ratchet is correct as long as both the classical CKA and the post-quantum CKA are correct. More concretely,*

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

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

¹⁷ Technically, of winning an variant of the 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 CKA, observe that it is easy to argue to both parties A and B absorb the same keys K_{CKA} into their root key K_{root} . Therefore, for the same epoch t and period iS , they produce the same AEAD key $K_{\text{aead}}^{\text{C}}$. Analogously, for the post-quantum protocol, Decode is guaranteed to produce the correct CKA message ρ and, therefore, correctness of the CKA scheme implies they produce the same sending and receiving keys KS and KR as well. As a result, for each message index idx , both parties produce the same AEAD key $K_{\text{aead}} := \text{KDF}_3(K_{\text{aead}}^{\text{C}}, K_{\text{aead}}^{\text{Q}})$ and, therefore, by correctness of the AEAD the recipient outputs the correct message. \square

Confidentiality. We now proceed to bound the advantage on the confidentiality game. Privacy holds as long as either of the CKA protocols is secure — with the speed of FS and PCS depending on whether the classical or the post-quantum 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 C.3. *Let $\text{Game}^{\text{SM-conf-ss}}$ be a variant of $\text{Game}^{\text{SM-conf}}$ with the following two modifications:*

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

For any PCS and FS parameters Δ_{PCS} and Δ_{FS} , respectively, and any epoch function τ , we then get

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

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 M_1 while all challenges thereafter encrypt message 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 . \square

Lemma C.4. *Assuming either the classical protocol CKA^{C} or the post-quantum protocol CKA^{Q} to be secure, then confidentiality holds for TR protocol. More concretely, let q be an upper bound on the oracle invocations and, for $X \in \{\text{C}, \text{Q}\}$, let Δ_{PCS}^X and Δ_{FS}^X denote the PCS and FS parameters for the classical and post-quantum CKAs, respectively, and let τ^{C} and τ^{Q} denote the respective epoch functions (as discussed in Remark 4.1). Then we have*

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

in case the classical part CKA^{C} is secure, and

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

in case the post-quantum part CKA^{Q} is secure.

Proof. We show this using a sequence of hybrids. The overall approach closely follows the proof in [1]. All of the changes, unless specifically mentioned otherwise, are only performed to the CKA of TR that is assumed to be secure. The proofs for the two CKAs are mostly analogous, with small deviations mentioned when they arise.

Hybrid₁: In the first hybrid, we modify $\text{Game}_{\mathcal{A}', \Delta_{\text{PCS}}, \Delta_{\text{FS}}, \tau}^{\text{SM-conf-ss}}$ as follows:

- We replace the key K_{CKA} of epoch $t_{\text{L}}^* + \Delta_{\text{PCS}}$ 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 $t_{\text{L}}^* = -\infty$, i.e., if no corruption occurs before the challenge, then Hybrid₁ 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 $t_L^* + \Delta_{PCS}$, i.e., the party P executing CKA-Send-P, did so between epoch $t_L^* + \Delta_{PCS} - 1$ and $t_L^* + \Delta_{PCS}$. By definition of t_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 t_L . Using $\Delta_{PCS} = \Delta_{PCS}^{CKA} + 1$, we can moreover observe that $\hat{t}^* := t_L^* + \Delta_{PCS} - 1$ 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 $\hat{t}^* - \Delta_{PCS}^{CKA}$ and after $\hat{t}^* + \Delta_{FS}$. Therefore, there exists a simple reduction to $\text{Game}_{\mathcal{A}', \hat{t}^*}^{CKA}$ as follows:

- Whenever TR-Send-P invokes CKA-Send-P, the reduction uses the Send-P oracle of the CKA game instead to obtain ρ for epoch other than t_L^* . Similarly, the reduction uses the Chall-P oracle to obtain ρ for the challenge epoch t_L^* . The reduction then keeps track of the corresponding key K_{CKA} and uses that one to mix into the root key K_{root} .
- Whenever TR-Rec-P invokes CKA-Rec-P on a decoded message ρ 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_{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 ρ that Decode would recover. It then mixes in the key K_{CKA} that was output as part of sending ρ (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

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

and the analogous result for the classical CKA with the tighter Δ_{PCS} bound.

Hybrid₂: In the second hybrid, we modify Hybrid_1 as follows:

- For all epochs starting from $t_L^* + \Delta_{PCS}$ to the challenge epoch, we replace the output of KDF_1 , i.e., K_{root} , KS and KR_{+1} , with freshly sampled independent keys. (In the case of the classically secure CKA, KDF_1 just outputs two keys, which we replace by fresh ones.)

Observe that the first of those $\text{KDF}_1(K_{root}, K_{CKA})$ invocations in Hybrid_1 uses a fresh and independent K_{CKA} . Therefore, by PRF-PRNG security of 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 K_{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

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

Hybrid₃: In the third hybrid, we modify Hybrid_2 as follows:

- For all epochs starting from $t_L^* + \Delta_{PCS}$ to the challenge epoch, we replace the output of KDF_2 , i.e. KS , of the sending party with freshly sampled independent keys. For the receiving party, 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 KS (or KR , respectively) have been fresh in Hybrid_2 , PRG security of KDF_2 ensures that those outputs are indistinguishable. In particular, recall that for epochs between $t_L^* + \Delta_{PCS}$ and (before) the challenge epoch the game does not allow corruptions. While corruptions may be allowed for the challenge epoch in case of $\Delta_{FS} = 0$, they are in particular only allowed *after* the challenge. However, KS can be safely leaked after the challenge with the challenge K_{aead}^X still appearing independent and uniform at random. Therefore, we obtain

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

Hybrid₄: Finally, we modify Hybrid₃ as follows:

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

Note that in Hybrid₃ either $K_{\text{aead}}^{\text{C}}$ or $K_{\text{aead}}^{\text{Q}}$ 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.

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

Finally, we consider the probability of \mathcal{A}' winning Hybrid₄, i.e., of correctly guessing which of the messages was encrypted as part of the challenge. Since Hybrid₄ uses a fresh uniform random key to encrypt the challenge using AEAD, this probability trivially reduced to AEAD security:

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

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

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

Lemma C.5. *Let Game^{SM-auth-ss} be a variant of Game^{SM-auth} with the following two modifications:*

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

For any PCS and FS parameters Δ_{PCS} and Δ_{FS} , respectively, and any epoch function τ , we for every PPT adversary \mathcal{A} , there exists a PPT adversary \mathcal{A}' such that

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

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 t^* of the first successful (non-trivial) injection and the last corruption beforehand. (Note that any corruption in 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 t^* uniformly at random in $\{1, \dots, q\}$ and t_{\perp}^* in $\{\infty, 1, \dots, q - \Delta_{\text{PCS}}\}$. 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}' afterwards is irrelevant.) This can be achieved trivially by simply rejecting all non-trivial injection attempts before t^* , since for TR the state remains unchanged in case an injection is rejected. \square

Lemma C.6. *For $X \in \{\text{C}, \text{Q}\}$, let Δ_{PCS}^X and Δ_{FS}^X denote the PCS and FS parameters for the classical and post-quantum CKAs, respectively, and let τ^{C} and τ^{Q} denote the respective epoch functions (as discussed in Remark 4.1). Then we have*

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

in case the classical part CKA^{C} is secure, and

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

in case the post-quantum part CKA^{Q} is secure.

Proof. First, we consider the injections for “old” epochs, i.e., where $\tau(\text{id}x') < t_L^*$, but the party P is in epoch t^* when processing the injection. Note that at this point P already got the (correct) number of messages sent during $\tau(\text{id}x')$ and has stored the individual AEAD keys in `StoredKeys`. Therefore, P will not accept an injection for a period counter, according to the period function ι , for which no message has been sent. Moreover, the 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.

Hybrid₁: In the first hybrid, we modify $\text{Game}_{A', \Delta_{\text{PCS}}, \Delta_{\text{FS}}, \tau}^{\text{SM-auth-ss}}$ as follows:

- We replace the key K_{CKA} of epoch $t_L^* + \Delta_{\text{PCS}}$ 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 $t_L^* = -\infty$, i.e., if no corruption occurs the injection oracle becomes available during epoch t^* , then Hybrid₁ behaves as the original game.

Note that while processing the message that delivers this K_{CKA} to the receiver, the receiver is still in the prior epoch. Therefore, injections are disallowed by `safe-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 CKA game in which either the correct key or an independently sampled one is produced. As a consequence, for the post-quantum CKA we obtain

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

and the analogous result for the classical CKA with the tighter Δ_{PCS} bound.

Hybrid₂: In the second hybrid, we modify Hybrid₁ as follows:

- For all epochs starting from $t_L^* + \Delta_{\text{PCS}}$ to t^* , we replace the output of KDF_1 , i.e., K_{root} , KS and KR_{+1} , with freshly sampled independent keys. (In the case of the classically secure CKA, KDF_1 just outputs two keys, which we replace by fresh ones.)
- In epoch t^* , if the receiver is still in epoch $t^* - 1$ 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 $t_L^* + \Delta_{\text{PCS}}$ to $t^* - 1$ no injections are allowed by Hybrid₁. Thus, we can therefore use a simple sequence of additional hybrids to replace those keys by fresh ones.

In contrast, injections are allowed for t^* . In particular, the receiver of such an injection might be still at epoch $t^* - 1$ at this stage, processing the injection attempt. We know from the prior argument that the K_{root} the receiver stores at this point is fresh. Here we crucially rely on the “PRF” property of PRF-PRNG security of KDF_1 to argue that we can replace all the resulting keys with independent and uniformly distributed ones. As a result, we can deduce

$$\left| \Pr \left[\text{Hybrid}_2(1^\lambda) = 1 \right] - \Pr \left[\text{Hybrid}_1(1^\lambda) = 1 \right] \right| \leq q \cdot \text{Adv}_C^{\text{PRF-PRNG}}(1^\lambda).$$

Hybrid₃: In the third hybrid, we modify Hybrid₂ as follows:

- In epoch t^* , we replace the output of KDF_2 , i.e. KS , of the sending party with freshly sampled independent keys. For the receiving party, 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 KDF_2 . Therefore, we obtain

$$\left| \Pr \left[\text{Hybrid}_3(1^\lambda) = 1 \right] - \Pr \left[\text{Hybrid}_2(1^\lambda) = 1 \right] \right| \leq q \cdot \text{Adv}_D^{\text{PRG}}(1^\lambda).$$

Hybrid₄: Finally, we modify Hybrid₃ as follows:

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

Assume CKA^Q is assumed to be secure. Then, in Hybrid₃ we replaced K_{aead}^Q for each injection attempt with independent and fresh values. Even if the attacker can cause the receiving party to reuse K_{aead}^C across injections, dual-PRF security of KDF_3 ensures the outputs to be independent and freshly sampled. The analogous argument holds if we assume CKA^C to be secure. Therefore, dual-PRF security ensures that the output is indistinguishable from a uniform random key in either case.

$$\left| \Pr \left[\text{Hybrid}_4(1^\lambda) = 1 \right] - \Pr \left[\text{Hybrid}_3(1^\lambda) = 1 \right] \right| \leq \text{Adv}_E^{\text{dPRF}}(1^\lambda).$$

Distribution $\mathcal{D}_{A,0}^{\text{KeyBaseSim}}$	Distribution $\mathcal{D}_{A,1}^{\text{KeyBaseSim}}$
1 : $(\widehat{ek}_A, \widehat{dk}_A) \stackrel{\$}{\leftarrow} \widehat{\mathcal{D}}_{\text{RKeyGen-A}}$	1 : $(ek_A, dk_A) \stackrel{\$}{\leftarrow} \mathcal{D}_{\text{RKeyGen-A}}$
2 : return $(\widehat{ek}_A, \widehat{dk}_A)$	2 : $(\widehat{ek}_A, \widehat{dk}_A, _) \stackrel{\$}{\leftarrow} \text{RSimKey-A}_1(ek_A, dk_A)$
	3 : return $(\widehat{ek}_A, \widehat{dk}_A)$

Fig. 12: Base Key simulatability.

Distribution $\mathcal{D}_{A,0}^{\text{KeyUpdSim}}$	Distribution $\mathcal{D}_{A,1}^{\text{KeyUpdSim}}$
1 : $(ek_B, dk_B) \stackrel{\$}{\leftarrow} \mathcal{D}_{\text{RKeyGen-B}}\{\text{rand}_0\}$	1 : $(ek_B, dk_B) \stackrel{\$}{\leftarrow} \mathcal{D}_{\text{RKeyGen-B}}\{\text{rand}_0\}$
2 : $(\widehat{ek}_B, \widehat{dk}_B, \text{aux}_0) \stackrel{\$}{\leftarrow} \text{RSimKey-B}_1(ek_B, dk_B)$	2 : $(\widehat{ek}_B, \widehat{dk}_B, \text{aux}_0) \stackrel{\$}{\leftarrow} \text{RSimKey-B}_1(ek_B, dk_B)$
3 : $(ek_A, dk_A) \stackrel{\$}{\leftarrow} \mathcal{D}_{\text{RKeyGen-A}}\{\text{rand}_1\}$	3 : $(ek_A, dk_A) \stackrel{\$}{\leftarrow} \mathcal{D}_{\text{RKeyGen-A}}\{\text{rand}_1\}$
4 : $(ct_B, K, \widehat{dk}_A) \stackrel{\$}{\leftarrow} \text{REnc-A}(\widehat{ek}_B, dk_A; \text{rand}_2)$	4 : $(\widehat{ek}_A, \widehat{dk}_A, \text{aux}_1) \stackrel{\$}{\leftarrow} \text{RSimKey-A}_1(ek_A, dk_A)$
5 : $(K', \widehat{ek}_A) \stackrel{\$}{\leftarrow} \text{RDec-B}(\widehat{dk}_B, ct_B, ek_A)$	5 : $(ct_B, K, K', \text{rand}_2) \stackrel{\$}{\leftarrow} \text{RSimKey-A}_2(\widehat{ek}_B, \widehat{dk}_B, \text{aux}_1)$
6 : return $((\widehat{ek}_B, \widehat{dk}_B), (\widehat{ek}_A, \widehat{dk}_A), ct_B, K, K', \text{aux}_0, \text{rand}_0, \text{rand}_1, \text{rand}_2)$	6 : return $((\widehat{ek}_B, \widehat{dk}_B), (\widehat{ek}_A, \widehat{dk}_A), ct_B, K, K', \text{aux}_0, \text{rand}_0, \text{rand}_1, \text{rand}_2)$

Fig. 13: Updated Key simulatability. The text highlighted in blue denotes the main differences between the two distributions. Recall $D\{\text{rand}\}$ denotes the process of sampling from the distribution D with randomness rand . Above, we assume rand (except for those output by RSimKey-A_2) to be distributed uniformly over their respective domain.

Finally, we consider the probability of \mathcal{A}' winning Hybrid_4 by succeeding with one of the injection attempts. There are three cases to consider:

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

In the first case, the attacker essentially tries to inject to a fresh key K_{aead} (see Hybrid_4) for which they have no information about (in particular not even seen a ciphertext for). 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, AEAD security prevents such an attack. Overall, this probability trivially reduced to AEAD security:

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

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

D Details on the CKA from RKEM

D.1 Omitted Definition of RKEM

We formally define ratchet simulatability of RKEM explained in Sec. 5.1. Notice 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 D.1 (Ratchet Simulatability). For $b \in \{0, 1\}$, let $\mathcal{D}_{A,b}^{\text{KeyBaseSim}}$ be the distributions as defined in Fig. 12, $\mathcal{D}_{A,b}^{\text{KeyUpdSim}}$ be the distributions as defined in Fig. 13, and $\mathcal{D}_{B,b}^{\text{CtxtSim}}$ be the distributions

Distribution $\mathcal{D}_{B,0}^{\text{CtxtSim}}$	Distribution $\mathcal{D}_{B,1}^{\text{CtxtSim}}$
1 : $(ek_A, dk_A) \xleftarrow{\$} \mathcal{D}_{\text{RKeyGen-A}}\{\text{rand}\}$	1 : $(ek_A, dk_A) \xleftarrow{\$} \mathcal{D}_{\text{RKeyGen-A}}\{\text{rand}\}$
2 : $(\widehat{ek}_A, \widehat{dk}_A, \text{aux}) \xleftarrow{\$} \text{RSimKey-A}_1(ek_A, dk_A)$	2 : $(\widehat{ek}_A, \widehat{dk}_A, \text{aux}) \xleftarrow{\$} \text{RSimKey-A}_1(ek_A, dk_A)$
3 : $(ek_B, dk_B) \xleftarrow{\$} \mathcal{D}_{\text{RKeyGen-B}}$	3 : $(\widehat{ek}_B, \widehat{dk}_B) \xleftarrow{\$} \widehat{\mathcal{D}}_{\text{RKeyGen-B}}$
4 : $(ct_A, K, \widehat{dk}_B) \xleftarrow{\$} \text{REnc-B}(\widehat{ek}_A, dk_B)$	4 : $(ct_A, ek_B, K, K') \xleftarrow{\$} \text{RSimCtxt-B}(\widehat{ek}_B, \widehat{ek}_A, \widehat{dk}_A)$
5 : $(K', \widehat{ek}_B) \xleftarrow{\$} \text{RDec-A}(\widehat{dk}_A, ct_A, ek_B)$	5 : return $(\text{aux}, \text{rand}, (\widehat{ek}_A, \widehat{dk}_A), ct_A,$
6 : return $(\text{aux}, \text{rand}, (\widehat{ek}_A, \widehat{dk}_A), ct_A,$	$(ek_B, \widehat{ek}_B), (K, K'))$
$(ek_B, \widehat{ek}_B), (K, K'))$	

Fig. 14: Ciphertext simulatability. The text highlighted in blue denotes the main differences between the two distributions.

as defined in Fig. 14. Moreover, let $\mathcal{D}_{B,b}^{\text{KeyBaseSim}}$ and $\mathcal{D}_{B,b}^{\text{KeyUpdSim}}$ and $\mathcal{D}_{B,b}^{\text{CtxtSim}}$ be defined analogously with the roles of the two parties swapped in the respective experiments. We say a ratcheting KEM Π_{RKEM} is ratchet simulatable if there exists efficient simulators $(\text{RSimKey-P}_1, \text{RSimKey-P}_2, \text{RSimCtxt-P})_{P \in \{A,B\}}$ such that the advantage against base-key simulatability

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

the advantage against updated key simulatability

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

and the advantage against ciphertext simulatability

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

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

D.2 Protocol Description

A formal description of our generic construction of a CKA protocol based on an RKEM is depicted in Fig. 15.

D.3 Security

In this section, we provide the security proof for the generic CKA construction based on RKEM.

Theorem D.1 (Theorem 5.1 restated). *For any correct and forward-secure RKEM, the protocol from Fig. 15 is a correct and secure CKA protocol with $\Delta_{\text{FS}} = 0$ and $\Delta_{\text{PCS}} = 2$. Moreover, if the RKEM is non-forward secure, then the protocol is a secure CKA with $\Delta_{\text{FS}} = 1$ and $\Delta_{\text{PCS}} = 2$.*

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

$$\begin{aligned} \text{Adv}_{\mathcal{A}, \Delta_{\text{FS}}, \Delta_{\text{PCS}}}^{\text{CKA}}(1^\lambda) &\leq q \cdot \epsilon_{\text{corr}}^{\text{RKEM}} + \text{Adv}_{\mathcal{B}}^{\text{KeyBaseSim-B}}(1^\lambda) + (q-1) \cdot \text{Adv}_{\mathcal{C}}^{\text{KeyUpdSim}}(1^\lambda) \\ &\quad + \text{Adv}_{\mathcal{D}}^{\text{CtxtSim}}(1^\lambda) + \text{Adv}_{\mathcal{E}}^{\text{FS-IND-CPA}}(1^\lambda), \end{aligned}$$

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

<p>CKA-Init-KeyGen(1^λ)</p> <hr/> 1 : $\text{par} \xleftarrow{\$} \text{RSetup}(1^\lambda)$ 2 : $(\widehat{\text{ek}}_B, \widehat{\text{dk}}_B) \xleftarrow{\$} \text{RKeyGen-B}(\text{par}, \text{updated})$ 3 : return $\text{lk} := (\widehat{\text{ek}}_B, \widehat{\text{dk}}_B, \text{par})$ <p>CKA-Init-A(lk)</p> <hr/> 1 : parse $(\widehat{\text{ek}}_B, \widehat{\text{dk}}_B, \text{par}) \leftarrow \text{lk}$ 2 : $\text{t}_A := 0$ 3 : $\text{st}_A := (\text{t}_A, \perp, \widehat{\text{ek}}_B, \text{par})$ 4 : return st_A <p>CKA-Init-B(lk)</p> <hr/> 1 : parse $(\widehat{\text{ek}}_B, \widehat{\text{dk}}_B, \text{par}) \leftarrow \text{lk}$ 2 : $\text{t}_B := 0$ 3 : $\text{st}_B := (\text{t}_B, \widehat{\text{dk}}_B, \perp, \text{par})$ 4 : return st_B	<p>CKA-Send-A(st_A)</p> <hr/> 1 : parse $(\text{t}_A, _, \widehat{\text{ek}}_B, \text{par}) \leftarrow \text{st}_A$ 2 : req $\llbracket \text{t}_A \text{ is even} \rrbracket$ 3 : $\text{t}_A += 1$ 4 : $(\text{ek}_A, \text{dk}_A) \xleftarrow{\$} \text{RKeyGen-A}(\text{par})$ 5 : $(\text{ct}_B, \text{K}, \widehat{\text{dk}}_A) \xleftarrow{\$} \text{REnc-A}(\widehat{\text{ek}}_B, \text{dk}_A)$ 6 : $\rho := (\text{t}_A, \text{ek}_A, \text{ct}_B)$ // Send $(\text{ek}_A, \text{ct}_B)$ 7 : $\text{st}_A := (\text{t}_A, \widehat{\text{dk}}_A, \perp, \text{par})$ 8 : return $(\text{K}, \rho, \text{st}_A)$ <p>CKA-Rec-B(st_B, ρ)</p> <hr/> 1 : parse $(\text{t}_B, \widehat{\text{dk}}_B, _, \text{par}) \leftarrow \text{st}_B$ 2 : req $\llbracket \text{t}_B \text{ is even} \rrbracket$ 3 : parse $(\text{t}_A, \text{ek}_A, \text{ct}_B) \leftarrow \rho$ 4 : req $\llbracket \text{t}_A = \text{t}_B + 1 \rrbracket$ 5 : $\text{t}_B += 1$ 6 : $(\text{K}, \widehat{\text{ek}}_A) \xleftarrow{\$} \text{RDec-B}(\widehat{\text{dk}}_B, \text{ct}_B, \text{ek}_A)$ 7 : $\text{st}_B := (\text{t}_B, \perp, \widehat{\text{ek}}_A, \text{par})$ 8 : return (K, st_B)
---	---

Fig. 15: 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.

Proof. Consider the CKA game $\text{Game}_{\mathcal{A}, \text{t}^*}^{\text{CKA}}$ as depicted in Fig. 16 with the protocol from Fig. 15 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 st_P for each operation. Analogously, we keep lk and CKA messages ρ_t in their expanded form, which especially implies that CKA-Init-P which just parses lk 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 t_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 never applies. To this end, observe that by correctness of RKEM, $\text{K}'_1 = \text{K}_1$ and the keypair $(\widehat{\text{ek}}_1, \widehat{\text{dk}}_1)$ is indistinguishable from a fresh one, when only considering the keys themselves. Since the protocol for epoch t only uses the keys $(\widehat{\text{ek}}_{t-1}, \widehat{\text{dk}}_{t-1})$ (and no side information thereof) we can therefore inductively invoke correctness and argue that by correctness $\text{K}'_t = \text{K}_t$ and that the key pair $(\widehat{\text{ek}}_t, \widehat{\text{dk}}_t)$ is indistinguishable from a fresh keypair (when ignoring side information).

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

$\text{Hybrid}_{\mathcal{A}, \text{t}^*, b, 0}^{\text{CKA}}$ to $\text{Hybrid}_{\mathcal{A}, \text{t}^*, b, \text{t}^* - 2}^{\text{CKA}}$: We define a sequence of hybrids $\text{Hybrid}_{\mathcal{A}, \text{t}^*, b, i}^{\text{CKA}}$ for $0 \leq i \leq \text{t}^* - 2$ and $b \in \{0, 1\}$. The hybrids are based on Fig. 16 with modifications described below — they are depicted in Fig. 17.

Initial setup: Instead of directly sampling $(\widehat{\text{ek}}_0, \widehat{\text{dk}}_0)$ using the RKeyGen-B algorithm, all hybrids first sample $(\text{ek}_0, \text{dk}_0)$ instead, and then use RSimKey-B₁ to derive $(\widehat{\text{ek}}_0, \widehat{\text{dk}}_0)$. 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

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

<p>Game$_{\mathcal{A}, t^*}^{\text{CKA}}(1^\lambda)$</p> <hr/> <pre> 1 : $b \xleftarrow{\\$} \{0, 1\}$ 2 : CKA-Init-KeyGen <hr/> $\text{par} \leftarrow \text{RSetup}(1^\lambda)$ $(\widehat{\text{ek}}_0, \widehat{\text{dk}}_0) \xleftarrow{\\$} \text{RKeyGen-B}(\text{par}, \text{updated})$ 3 : for $P \in \{\text{A}, \text{B}\}$ 4 : $t_P := 0$ // CKA-Init-P does nothing 5 : $b' \xleftarrow{\\$} \mathcal{A}(t^*)^{\text{Send-P}(), \text{Receive-P}(), \text{Chall-P}(), \text{Corr-P}()}$ 6 : return $\llbracket b = b' \rrbracket$ </pre> <p>Send-P(rleak)</p> <hr/> <pre> 1 : $t_P \leftarrow t_P + 1$ 2 : $\text{rand} := (\text{rand}_1, \text{rand}_2) \leftarrow \mathcal{R}$ 3 : CKA-Send-P <hr/> $(\text{ek}_{t_P}, \text{dk}_{t_P}) \leftarrow \text{RKeyGen-P}(\text{par}; \text{rand}_1)$ $(\text{ct}_{t_P-1}, \text{K}, \widehat{\text{dk}}_{t_P}) \leftarrow \text{REnc-P}(\widehat{\text{ek}}_{t_P-1}, \text{dk}_{t_P}; \text{rand}_2)$ $\rho := (t_P, \text{ek}_{t_P}, \text{ct}_{t_P-1})$ 4 : if $\llbracket \text{rleak} \rrbracket$ then // Leak randomness // Allow leaking randomness Δ_{PCS}-epoch before t^* 5 : req $\llbracket t_A, t_B \leq t^* - \Delta_{\text{PCS}} \rrbracket$ 6 : else // Secure randomness (for challenge epoch) 7 : $\text{rand} \leftarrow \perp$ 8 : $\text{K}_{t_P} \leftarrow \text{K}$ 9 : return $(\text{K}, \rho, \text{rand})$ </pre>	<p>Chall-P()</p> <hr/> <pre> 1 : $t_P \leftarrow t_P + 1$ 2 : req $\llbracket t_P = \widehat{t}^* \rrbracket$ // Challenge epoch t^* 3 : CKA-Send-P <hr/> $(\text{ek}_{t_P}, \text{dk}_{t_P}) \xleftarrow{\\$} \text{RKeyGen-P}(\text{par})$ $(\text{ct}_{t_P-1}, \text{K}, \widehat{\text{dk}}_{t_P}) \xleftarrow{\\$} \text{REnc-P}(\widehat{\text{ek}}_{t_P-1}, \text{dk}_{t_P})$ $\rho := (t_P, \text{ek}_{t_P}, \text{ct}_{t_P-1})$ 4 : $\text{K}_{t_P} \leftarrow \text{K}$ 5 : if $\llbracket b = 1 \rrbracket$ then 6 : $\text{K} \xleftarrow{\\$} \mathcal{K}$ // Replace with random key 7 : return (K, ρ) </pre> <p>Receive-P()</p> <hr/> <pre> 1 : $t_P \leftarrow t_P + 1$ 2 : CKA-Rec-P <hr/> $(\text{K}, \widehat{\text{ek}}_{t_P}) \xleftarrow{\\$} \text{RDec-P}(\widehat{\text{dk}}_{t_P-1}, \text{ct}_{t_P-1}, \text{ek}_{t_P})$ 3 : assert $\llbracket \text{K} = \text{K}_{t_P} \rrbracket$ // Correctness </pre> <p>Corr-P()</p> <hr/> <pre> 1 : // Allow corrupting Δ_{PCS}-epoch before \widehat{t}^* 2 : req $\llbracket \widehat{t}_A, \widehat{t}_B \leq \widehat{t}^* - \Delta_{\text{PCS}} \rrbracket$ // Allow corrupting Δ_{PCS}-epoch after \widehat{t}^* 3 : req $\llbracket \widehat{t}_P \geq \widehat{t}^* + \Delta_{\text{FS}} \rrbracket$ 4 : Protocol state <hr/> if P is sender in t_P then $\text{st}_P := (t_P, \widehat{\text{dk}}_{t_P}, \perp, \text{par})$ else $\text{st}_P := (t_P, \perp, \widehat{\text{ek}}_{t_P}, \text{par})$ 5 : return st_P </pre>
---	---

Fig. 16: 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.

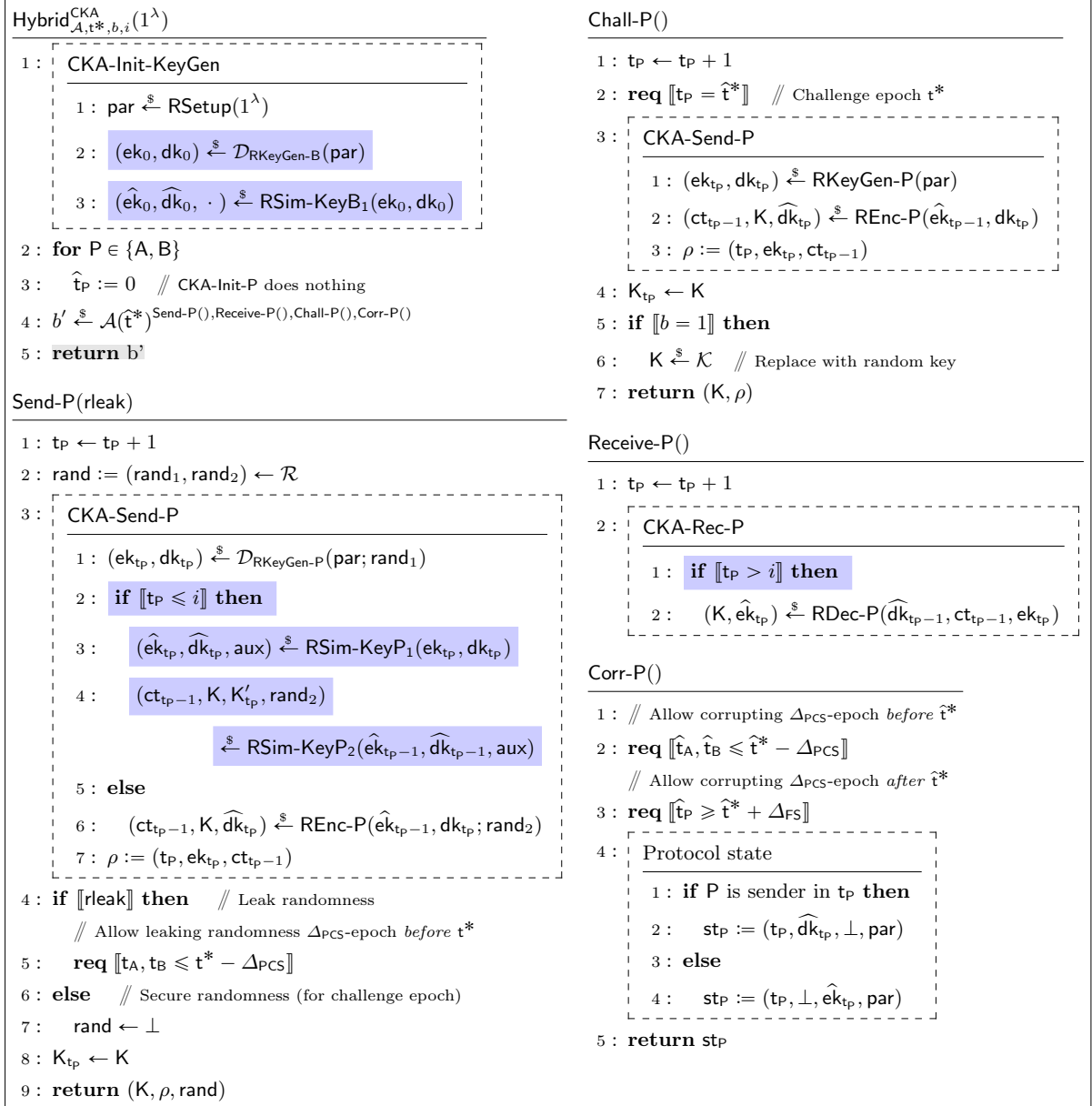


Fig. 17: A sequence of hybrid games for $0 \leq i \leq t^* - 2$. Changes with respect to Fig. 16 are highlighted.

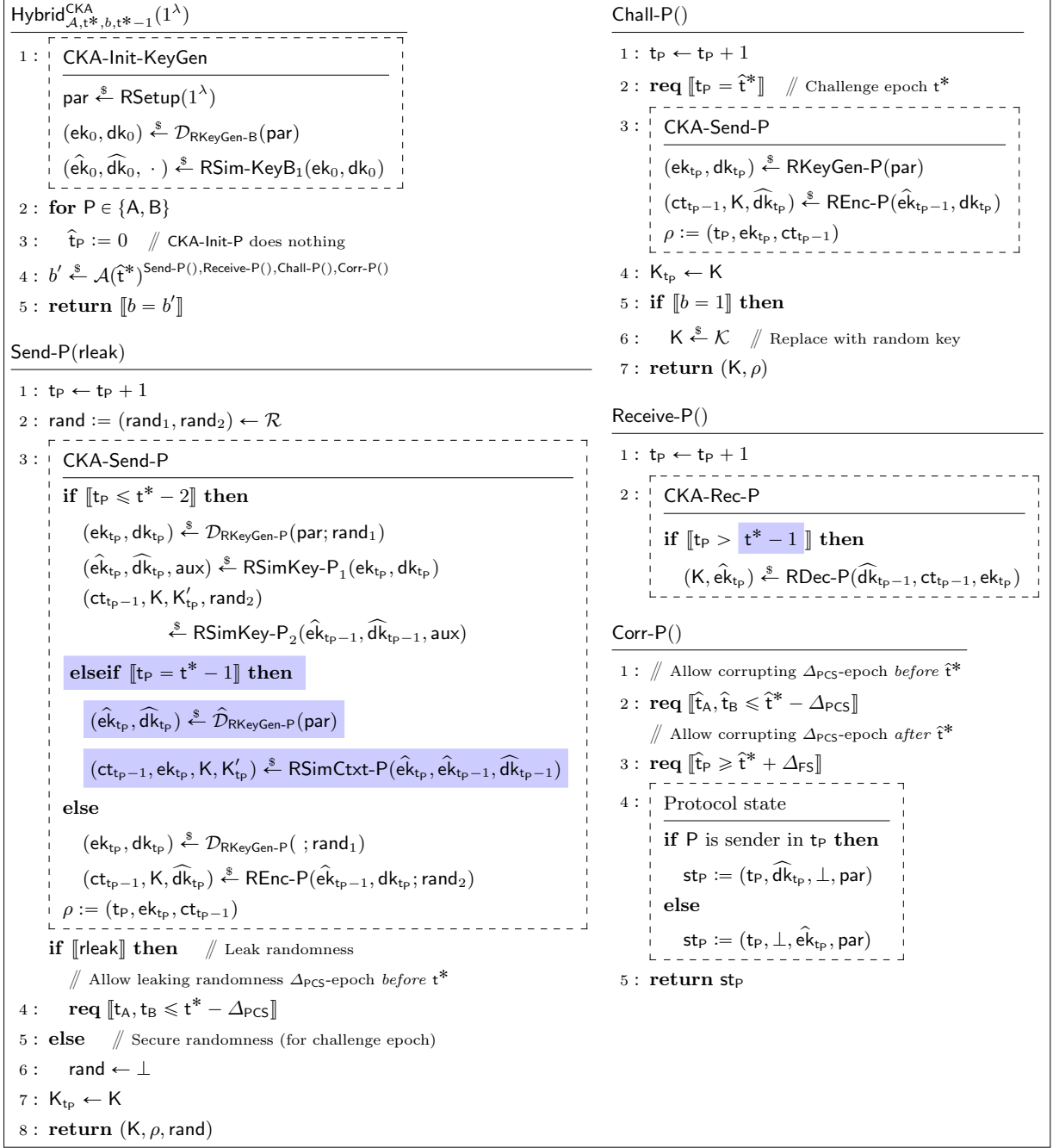


Fig. 18: An additional hybrid game. Changes with respect to $\text{Hybrid}_{\mathcal{A}, t^*, b, t^* - 2}^{\text{CKA}}$ are highlighted.

Sending and receiving: For epochs $1 \leq t_P \leq i$, we moreover change CKA-Send-P to use RSimKey-P_1 and RSimKey-P_2 to generate the key pair $(\hat{ek}_{t_P}, \hat{dk}_{t_P})$ as well as the ciphertext ct_{t_P-1} and key K_{t_P} for epoch t_P . Note that the updated decryption key \hat{dk}_{t_P} is already generated by the simulator and, thus, we skip RDec-P in CKA-Rec-P for those epoch. (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 $\mathcal{D}_{P,1}^{\text{KeyUpdSim}}$ while the regular protocol behavior exactly corresponds to $\mathcal{D}_{P,0}^{\text{KeyUpdSim}}$. Therefore, there exists a simple reduction to updated-key simulatability, i.e.,

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

where $P = A$ for odd t_P and $P = B$ for even t_P .

$\text{Hybrid}_{\mathcal{A},t^*,b,t^*-1}^{\text{CKA}}$: Next, consider a hybrid depicted in Fig. 18 that changes how the keys for epoch $t^* - 1$ are sampled. More concretely, it samples the key pair $(\widehat{\text{ek}}_{t^*}, \widehat{\text{dk}}_{t^*})$ freshly and then uses the simulator RSimCtxt-P to simulate ek_{t^*} , the key K_{t^*} and the ciphertext ct_{t^*-1} . (The private key dk_{t^*} is not needed by the hybrid.) In addition, we also emit the decryption for epoch $t^* - 1$ as $\widehat{\text{ek}}_{t^*}$ has already been produced. Observe that this matches the sampling strategy of $\mathcal{D}_{P,1}^{\text{CtxtSim}}$, while the old strategy of $\text{Hybrid}_{\mathcal{A},t^*,b,t^*-2}^{\text{CKA}}$ matches $\mathcal{D}_{P,0}^{\text{CtxtSim}}$. Therefore, we obtain

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

for an appropriate reduction \mathcal{D} .

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

- $(\widehat{\text{ek}}_{t^*-1}, \widehat{\text{dk}}_{t^*-1})$ is a fresh key pair drawn from the same distribution the key generation algorithm produces. Moreover, with $\Delta_{\text{PCS}} = 2$, the adversary is not allowed to leak the key pair's randomness.
- The only place $\widehat{\text{dk}}_{t^*-1}$ is used in the game is in CKA-Rec-P to update ek_{t^*} to $\widehat{\text{ek}}_{t^*}$.
- The only place $\widehat{\text{ek}}_{t^*-1}$ 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

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

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

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

Using the sequence of hybrids

$$\begin{aligned} \text{Game}_{\mathcal{A},t^*,b=0}^{\text{CKA}} &\rightarrow \text{Hybrid}_{\mathcal{A},t^*,b=0,0}^{\text{CKA}} \rightarrow \dots \rightarrow \text{Hybrid}_{\mathcal{A},t^*,b=0,t^*-1}^{\text{CKA}} \\ &\rightarrow \text{Hybrid}_{\mathcal{A},t^*,b=1,t^*-1}^{\text{CKA}} \rightarrow \dots \rightarrow \text{Hybrid}_{\mathcal{A},t^*,b=1,0}^{\text{CKA}} \rightarrow \text{Game}_{\mathcal{A},t^*,b=1}^{\text{CKA}} \end{aligned}$$

then yields the desired bound. \square

E More Details on Our Lattice-based RKEM Katana

In this section, we provide the correctness and security proofs omitted in Sec. 6. We further provide omitted details on the optimized RKEM using bit-dropping.

E.1 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 χ , $\widehat{\chi}$, and $\widetilde{\chi}$ (e.g., discrete Gaussian distributions), we leave it general to allow any distribution. See Sections 6.1 and 6.4 for more detail.

Lemma E.1 (Correctness). *Our RKEM Katana is correct assuming*

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

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

Proof. Recalling that $\hat{\chi}$ is defined as $[2] \cdot \chi$ (i.e., convolution of two independent copies of χ), 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

$$\begin{aligned} \text{ct}_A - \text{ek}_B^\top \cdot \widehat{\text{dk}}_A &= \widehat{\text{ek}}_A^\top \cdot \mathbf{s}_B + \tilde{e}_B + m - \text{ek}_B^\top \cdot \widehat{\text{dk}}_A \\ &= (\mathbf{D}\widehat{\mathbf{s}}_A + \widehat{\mathbf{e}}_A)^\top \cdot \mathbf{s}_B + \tilde{e}_B + m - (\mathbf{D}^\top \mathbf{s}_B + \mathbf{e}_B)^\top \cdot \widehat{\mathbf{s}}_A \\ &= m + \underbrace{\widehat{\mathbf{e}}_A^\top \cdot \mathbf{s}_B - \mathbf{e}_B^\top \cdot \widehat{\mathbf{s}}_A}_{=:z} + \tilde{e}_B, \end{aligned}$$

where $(\widehat{\mathbf{s}}_A, \widehat{\mathbf{e}}_A) \stackrel{\$}{\leftarrow} \hat{\chi} \times \hat{\chi}$ and $(\mathbf{s}_B, \mathbf{e}_B, \tilde{e}_B) \stackrel{\$}{\leftarrow} \chi \times \chi \times \tilde{\chi}$. If each coefficient of $z \in R_q$ is smaller than $q/4$ (i.e., $\|z\|_\infty \leq q/4$), Decode will correctly decode to m as desired. \square

E.2 FS-IND-CPA Security

We recall the theorem statement establishing the FS-IND-CPA security of our RKEM Katana.

Theorem E.1 (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 $\mathcal{B}_{\text{MLWE}}$ against the $\text{MLWE}_{q,k,\chi}$ problem and adversaries $\mathcal{B}_{\text{hint-MLWE},1}$ and $\mathcal{B}_{\text{hint-MLWE},2}$ against the $\text{hint-MLWE}_{q,k,2k,\chi,\chi,\mathcal{F}_{\text{cpa}}}$ problem with $\mathcal{F}_{\text{cpa}} := \mathcal{U}(\{\mathbf{I}_{2k \times 2k}\})$ such that

$$\begin{aligned} \text{Adv}_{\mathcal{A}}^{\text{FS-IND-CPA-A}}(1^\lambda) &\leq \text{Adv}_{\mathcal{B}_{\text{MLWE}}}^{\text{MLWE}}(1^\lambda) + \text{Adv}_{\mathcal{B}_{\text{hint-MLWE},1}}^{\text{hint-MLWE}}(1^\lambda) \\ &\quad + 2 \cdot \text{Adv}_{\mathcal{B}_{\text{hint-MLWE},2}}^{\text{hint-MLWE}}(1^\lambda) + \epsilon_{\text{corr}} + \frac{Q}{2^{\lambda-1}}, \end{aligned}$$

where ϵ_{corr} is the probability that correctness with updated keys fails (cf. Def. 5.2).

Proof. Due to the symmetry of users A and B, we only focus on bounding the advantage $\text{Adv}_{\mathcal{A}}^{\text{FS-IND-CPA-A}}(1^\lambda)$ (cf. Def. 5.3). The theorem is proven in a sequence of hybrid games given in Figs. 19 and 20. The first Game_0 is the real FS-IND-CPA security game, where 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, ϵ_i denotes the advantage of \mathcal{A} in Game_i and Q denotes the number of random oracle queries performed by \mathcal{A} .

Game₀: This is the real FS-IND-CPA security game. For reference, in Fig. 19, we provide the full details of the game.

Game₁: In this game, the challenger reuses $\mathbf{K}, \mathbf{s}, \mathbf{e}$ 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

$$|\epsilon_0 - \epsilon_1| \leq \epsilon_{\text{corr}},$$

where ϵ_{corr} is the probability that correctness with updated keys fails (cf. Def. 5.2).

Game₂: In this game, the challenger samples a random $\widehat{\mathbf{u}}_B$ from \mathcal{R}_q^k as opposed to generating them as an MLWE instance. It is straight forward to see that the Game_2 is indistinguishable from Game_1 under the MLWE assumption. Formally, we can construct an adversary $\mathcal{B}_{\text{MLWE}}$ against the $\text{MLWE}_{q,k,\chi}$ problem such that

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

Game₃: In this game, the challenger first samples $(\mathbf{K}_0, \mathbf{s}, \mathbf{e}) \stackrel{\$}{\leftarrow} \{0, 1\}^\lambda \times \chi \times \chi$ and later programs the random oracle H on input $(\text{ek}_A, \text{seed})$. In case the input is already queried (i.e., $Q_{\mathsf{H}}[\text{ek}_A, \text{seed}] \neq \perp$), then the challenger declares the adversary \mathcal{A} wins and outputs 1 as the output of the game. This game is identical to Game_2 as long as $Q_{\mathsf{H}}[\text{ek}_A, \text{seed}] \neq \perp$. Since seed is sampled uniformly random over $\{0, 1\}^\lambda$, the probability of this occurring is $Q/2^\lambda$.

Hence, we have

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

<p>Game₀ // Original FS-IND-CPA security game</p> <hr/> 1 : $Q_H[\cdot] := \perp$ // Prepare empty RO 2 : $b \xleftarrow{\$} \{0, 1\}$ 3 : $K_1 \xleftarrow{\$} \{0, 1\}^\lambda$ // Sample from $\widehat{\mathcal{D}}_{\text{RKeyGen-B}}$ 4 : $(\widehat{\mathbf{s}}_B, \widehat{\mathbf{e}}_B) \xleftarrow{\$} \widehat{\chi} \times \widehat{\chi}$ 5 : $\widehat{\mathbf{u}}_B := \mathbf{D}^\top \cdot \widehat{\mathbf{s}}_B + \widehat{\mathbf{e}}_B \in R_q^k$ 6 : $(\widehat{\mathbf{ek}}_B, \widehat{\mathbf{dk}}_B) := (\widehat{\mathbf{u}}_B, \widehat{\mathbf{s}}_B)$ // Sample from $\mathcal{D}_{\text{RKeyGen-A}}$ 7 : $(\mathbf{s}_A, \mathbf{e}_A) \xleftarrow{\$} \chi \times \chi$ 8 : $\mathbf{u}_A := \mathbf{D} \cdot \mathbf{s}_A + \mathbf{e}_A \in R_q^k$ 9 : $(\mathbf{ek}_A, \mathbf{dk}_A) := (\mathbf{u}_A, (\mathbf{u}_A, \mathbf{s}_A))$ // Run $\text{REnc-A}(\widehat{\mathbf{ek}}_B, \mathbf{dk}_A)$ 10 : $\text{seed} \xleftarrow{\$} \{0, 1\}^\lambda$ 11 : $m \leftarrow \text{Encode}(\text{seed})$ // $m \in R_q$ 12 : $(K_0, \mathbf{s}, \mathbf{e}) := \text{H}(\mathbf{u}_A, \text{seed})$ 13 : $\tilde{e}_A \xleftarrow{\$} \tilde{\chi}$ 14 : $v_B := \widehat{\mathbf{ek}}_B^\top \cdot \mathbf{s}_A + \tilde{e}_A + m \in R_q$ 15 : $\text{ct}_B := v_B$ 16 : $\widehat{\mathbf{dk}}_A := \mathbf{s}_A + \mathbf{s} \in R_q^k$ // Update and erase \mathbf{dk}_A // Run $\text{RDec-B}(\widehat{\mathbf{dk}}_B, \text{ct}_B, \mathbf{ek}_A)$ 17 : $m' := \text{ct}_B - \widehat{\mathbf{ek}}_A^\top \cdot \widehat{\mathbf{dk}}_B$ 18 : $\text{seed}' := \text{Decode}(m')$ 19 : $(K', \mathbf{s}', \mathbf{e}') := \text{H}(\widehat{\mathbf{ek}}_A, \text{seed}')$ 20 : $\widehat{\mathbf{ek}}_A := \mathbf{ek}_A + \mathbf{D} \cdot \mathbf{s}' + \mathbf{e}'$ // Run adversary \mathcal{A} 21 : $b' \xleftarrow{\$} \mathcal{A}(\mathbf{ek}_A, \widehat{\mathbf{ek}}_A, \widehat{\mathbf{ek}}_B, \text{ct}_B, \widehat{\mathbf{dk}}_A, K_b)$ 22 : return $[b = b']$ <hr/> <p>Game₁</p> // Same up till Game ₀ , line 16 16 : $\widehat{\mathbf{ek}}_A := \mathbf{ek}_A + \mathbf{D} \cdot \mathbf{s} + \mathbf{e}$ // Reuse $K, \mathbf{s}, \mathbf{e}$ from REnc-A 17 : $b' \xleftarrow{\$} \mathcal{A}(\mathbf{ek}_A, \widehat{\mathbf{ek}}_A, \widehat{\mathbf{ek}}_B, \text{ct}_B, \widehat{\mathbf{dk}}_A, K_b)$ 18 : return $[b = b']$	<p>Game₂</p> <hr/> 1 : $Q_H[\cdot] := \perp$ 2 : $b \xleftarrow{\$} \{0, 1\}$ 3 : $K_1 \xleftarrow{\$} \{0, 1\}^\lambda$ 4 : $\widehat{\mathbf{u}}_B \xleftarrow{\$} R_q^k$ 5 : $\widehat{\mathbf{ek}}_B := \widehat{\mathbf{u}}_B$ // Remove $\widehat{\mathbf{dk}}_B$ // Sample from $\mathcal{D}_{\text{RKeyGen-A}}$ // Same from Game ₁ , line 7 <hr/> <p>Game₃</p> <hr/> 1 : $Q_H[\cdot] := \perp$ 2 : $b \xleftarrow{\$} \{0, 1\}$ 3 : $(K_0, \mathbf{s}, \mathbf{e}) \xleftarrow{\$} \{0, 1\}^\lambda \times \chi \times \chi$ // Sample w/o RO 4 : $K_1 \xleftarrow{\$} \{0, 1\}^\lambda$ 5 : $\widehat{\mathbf{ek}}_B \xleftarrow{\$} \mathcal{R}_q^k$ 6 : $(\mathbf{s}_A, \mathbf{e}_A) \xleftarrow{\$} \chi \times \chi$ 7 : $\mathbf{u}_A := \mathbf{D} \cdot \mathbf{s}_A + \mathbf{e}_A \in R_q^k$ 8 : $(\mathbf{ek}_A, \mathbf{dk}_A) := (\mathbf{u}_A, (\mathbf{u}_A, \mathbf{s}_A))$ 9 : $\text{seed} \xleftarrow{\$} \{0, 1\}^\lambda$ 10 : $m \leftarrow \text{Encode}(\text{seed})$ 11 : if $[Q_H[\mathbf{u}_A, \text{seed}] \neq \perp]$ then 12 : return 1 // Declare \mathcal{A} wins 13 : $Q_H[\mathbf{u}_A, \text{seed}] := (K_0, \mathbf{s}, \mathbf{e})$ // Program RO 14 : $\tilde{e}_A \xleftarrow{\$} \tilde{\chi}$ 15 : $v_B := \widehat{\mathbf{ek}}_B^\top \cdot \mathbf{s}_A + \tilde{e}_A + m \in R_q$ 16 : $\text{ct}_B := v_B$ 17 : $\widehat{\mathbf{dk}}_A := \mathbf{s}_A + \mathbf{s} \in R_q^k$ 18 : $\widehat{\mathbf{ek}}_A := \mathbf{ek}_A + \mathbf{D} \cdot \mathbf{s} + \mathbf{e}$ 19 : $b' \xleftarrow{\$} \mathcal{A}(\mathbf{ek}_A, \widehat{\mathbf{ek}}_A, \widehat{\mathbf{ek}}_B, \text{ct}_B, \widehat{\mathbf{dk}}_A, K_b)$ 20 : return $[b = b']$ <hr/> <p>H($\mathbf{u}_A, \text{seed}$) // Used by Game₀ to Game₃</p> <hr/> 1 : if $[Q_H[\mathbf{u}, \text{seed}] = \perp]$ then 2 : $(K_0, \mathbf{s}, \mathbf{e}) \xleftarrow{\$} \{0, 1\}^\lambda \times \chi \times \chi$ 3 : $Q_H[\mathbf{u}, \text{seed}] \leftarrow (K_0, \mathbf{s}, \mathbf{e})$ 4 : return $Q_H[\mathbf{u}, \text{seed}]$
--	---

Fig. 19: Hybrid games Game₀ to Game₃ used for the proof of FS-IND-CPA. The text highlighted in blue denotes the main difference between the previous hybrid.

Game₄: 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 $(\mathbf{u}_A^*, \text{seed}^*)$. We denote this event by E_4 . Clearly, as long as event E_4 does not occur, Game₃ and Game₄ proceed identically. Hence, we have

$$|\epsilon_3 - \epsilon_4| \leq \Pr[E_4].$$

Game ₄	Game ₅
1 : $Q_H[\cdot] := \perp$ 2 : $(\mathbf{u}_A^*, \text{seed}^*) := (\perp, \perp)$ 3 : $b \xleftarrow{\$} \{0, 1\}$ 4 : $(K_0, \mathbf{s}, \mathbf{e}) \xleftarrow{\$} \{0, 1\}^\lambda \times \chi \times \chi$ // Sample w/o RO 5 : $K_1 \xleftarrow{\$} \{0, 1\}^\lambda$ 6 : $\hat{\mathbf{e}}_{K_B} \xleftarrow{\$} \mathcal{R}_q^k$ 7 : $(\mathbf{s}_A, \mathbf{e}_A) \xleftarrow{\$} \chi \times \chi$ 8 : $\mathbf{u}_A := \mathbf{D} \cdot \mathbf{s}_A + \mathbf{e}_A \in R_q^k$ 9 : $(\mathbf{ek}_A, \mathbf{dk}_A) := (\mathbf{u}_A, (\mathbf{u}_A, \mathbf{s}_A))$ 10 : $\text{seed} \xleftarrow{\$} \{0, 1\}^\lambda$ 11 : $m \leftarrow \text{Encode}(\text{seed})$ 12 : if $[[Q_H[\mathbf{u}_A, \text{seed}] \neq \perp]]$ then 13 : return 1 // Declare \mathcal{A} wins 14 : $(\mathbf{u}_A^*, \text{seed}^*) \leftarrow (\mathbf{u}_A, \text{seed})$ 15 : $\tilde{\mathbf{e}}_A \xleftarrow{\$} \tilde{\chi}$ 16 : $v_B := \hat{\mathbf{e}}_{K_B}^\top \cdot \mathbf{s}_A + \tilde{\mathbf{e}}_A + m \in R_q$ 17 : $\text{ct}_B := v_B$ 18 : $\widehat{\mathbf{dk}}_A := \mathbf{s}_A + \mathbf{s} \in R_q^k$ 19 : $\hat{\mathbf{e}}_{K_A} := \mathbf{ek}_A + \mathbf{D} \cdot \mathbf{s} + \mathbf{e}$ 20 : $b' \xleftarrow{\$} \mathcal{A}(\mathbf{ek}_A, \hat{\mathbf{e}}_{K_A}, \hat{\mathbf{e}}_{K_B}, \text{ct}_B, \widehat{\mathbf{dk}}_A, K_b)$ 21 : return $[[b = b']]$ H (\mathbf{u}, seed) // Used by Game ₄ to Game ₆ 1 : if $[[\mathbf{u}_A^*, \text{seed}^*] \neq (\perp, \perp)]$ then 2 : if $[[\mathbf{u}, \text{seed}] = (\mathbf{u}_A^*, \text{seed}^*)]$ then 3 : abort 4 : if $[[Q_H[\mathbf{u}, \text{seed}] = \perp]]$ then 5 : $(K_0, \mathbf{s}, \mathbf{e}) \xleftarrow{\$} \{0, 1\}^\lambda \times \chi \times \chi$ 6 : $Q_H[\mathbf{u}, \text{seed}] \leftarrow (K_0, \mathbf{s}, \mathbf{e})$ 7 : return $Q_H[\mathbf{u}, \text{seed}]$	// Same up till Game ₄ , line 17 20 : $\widehat{\mathbf{dk}}_A := \mathbf{s}_A + \mathbf{s} \in R_q^k$ 21 : $\hat{\mathbf{e}}_A := \mathbf{e}_A + \mathbf{e} \in R_q^k$ 22 : $\hat{\mathbf{e}}_{K_A} := \mathbf{D} \cdot \widehat{\mathbf{dk}}_A + \hat{\mathbf{e}}_A$ 23 : $b' \xleftarrow{\$} \mathcal{A}(\mathbf{ek}_A, \hat{\mathbf{e}}_{K_A}, \hat{\mathbf{e}}_{K_B}, \text{ct}_B, \widehat{\mathbf{dk}}_A, K_b)$ 24 : return $[[b = b']]$ Game₆ 1 : $Q_H[\cdot] := \perp$ 2 : $(\mathbf{u}_A^*, \text{seed}^*) := (\perp, \perp)$ 3 : $b \xleftarrow{\$} \{0, 1\}$ 4 : $(K_0, \mathbf{s}, \mathbf{e}) \xleftarrow{\$} \{0, 1\}^\lambda \times \chi \times \chi$ // Sample w/o RO 5 : $K_1 \xleftarrow{\$} \{0, 1\}^\lambda$ 6 : $\hat{\mathbf{e}}_{K_B} \xleftarrow{\$} \mathcal{R}_q^k$ 7 : $(\mathbf{s}_A, \mathbf{e}_A) \xleftarrow{\$} \chi \times \chi$ 8 : $\mathbf{u}_A \xleftarrow{\$} R_q^k$ 9 : $\mathbf{ek}_A := \mathbf{u}_A$ // Remove \mathbf{dk}_A 10 : $\text{seed} \xleftarrow{\$} \{0, 1\}^\lambda$ 11 : $m \leftarrow \text{Encode}(\text{seed})$ 12 : if $[[Q_H[\mathbf{u}_A, \text{seed}] \neq \perp]]$ then 13 : return 1 // Declare \mathcal{A} wins 14 : else 15 : $(\mathbf{u}_A^*, \text{seed}^*) \leftarrow (\mathbf{u}_A, \text{seed})$ 16 : $v'_B \xleftarrow{\$} R_q$ 17 : $v_B := v'_B + m$ 18 : $\text{ct}_B := v_B$ 19 : $\widehat{\mathbf{dk}}_A := \mathbf{s}_A + \mathbf{s} \in R_q^k$ 20 : $\hat{\mathbf{e}}_A := \mathbf{e}_A + \mathbf{e} \in R_q^k$ 21 : $\hat{\mathbf{e}}_{K_A} := \mathbf{D} \cdot \widehat{\mathbf{dk}}_A + \hat{\mathbf{e}}_A$ 22 : $b' \xleftarrow{\$} \mathcal{A}(\mathbf{ek}_A, \hat{\mathbf{e}}_{K_A}, \hat{\mathbf{e}}_{K_B}, \text{ct}_B, \widehat{\mathbf{dk}}_A, K_b)$ 23 : return $[[b = b']]$

Fig. 20: Hybrid games Game₄ to Game₆ 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.

As we cannot bound $\Pr[E_4]$ yet, we postpone bounding it to later.

Game₅: In this game, the challenger computes user A's updated key $\hat{\mathbf{e}}_{K_A}$ directly without using \mathbf{ek}_A . Since this is only a conceptual change, we have

$$\epsilon_4 = \epsilon_5.$$

Moreover, denoting E_5 the event that the adversary triggers the abort condition in Game₅, we also have

$$\Pr[E_4] = \Pr[E_5].$$

Game₆: In the final game, the challenger samples a random \mathbf{u}_A from \mathcal{R}_q^k and sets user A's key as $\mathbf{ek}_A := \mathbf{u}_A$. Moreover, it samples random v'_B from \mathcal{R}_q and sets the ciphertext as $\text{ct}_B := v'_B + m$. Recall in the

previous game, \mathbf{u}_A and v_B were set as MLWE instances. Since \mathbf{s}_A and \mathbf{e}_A are partially leaked to the adversary \mathcal{A} via $\widehat{\mathbf{d}}\mathbf{k}_A = \mathbf{s}_A + \mathbf{s}$ and $\widehat{\mathbf{e}}\mathbf{k}_A$, we cannot rely on the standard MLWE assumption to argue indistinguishability of the two games. However, noticing that $\mathbf{s}, \mathbf{e} \stackrel{\$}{\leftarrow} \chi$ are information theoretically hidden to \mathcal{A} conditioning on the game not aborting, we can rely instead on the *hint* MLWE assumption. Let E_6 denote the event that \mathcal{A} triggers an abort in Game_6 and let Win_i denote the event that \mathcal{A} wins in Game_i . Then, we can construct an adversary $\mathcal{B}_{\text{hint-MLWE},1}$ against the $\text{hint-MLWE}_{q,k,2k,\chi,\chi,\mathcal{F}_{\text{cpa}}}$ problem with $\mathcal{F} := \mathcal{U}(\{\mathbf{I}_{2k \times 2k}\})$ such that

$$|\Pr[\text{Win}_5 \wedge \neg E_5] - \Pr[\text{Win}_6 \wedge \neg E_6]| \leq \text{Adv}_{\mathcal{B}_{\text{hint-MLWE},1}}^{\text{hint-MLWE}}(1^\lambda).$$

Indeed, the reduction is straightforward as $\mathcal{B}_{\text{hint-MLWE},1}$ receives as hints $\widehat{\mathbf{d}}\mathbf{k}_A$ and $\widehat{\mathbf{e}}\mathbf{k}_A$ and can efficiently simulate Game_5 or Game_6 to \mathcal{A} depending on whether it receives a random or valid 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

$$\begin{aligned} & |\epsilon_5 - \epsilon_6| \\ &= |(\Pr[E_5] \cdot \Pr[\text{Win}_5|E_5] + \Pr[\text{Win}_5 \wedge \neg E_5]) - (\Pr[E_6] \cdot \Pr[\text{Win}_6|E_6] + \Pr[\text{Win}_6 \wedge \neg E_6])| \\ &\leq \Pr[E_5] + \Pr[E_6] + |\Pr[\text{Win}_5 \wedge \neg E_5] - \Pr[\text{Win}_6 \wedge \neg E_6]| \\ &\leq \Pr[E_5] + \Pr[E_6] + \text{Adv}_{\mathcal{B}_{\text{hint-MLWE},1}}^{\text{hint-MLWE}}(1^\lambda), \end{aligned}$$

where we use the fact that $\Pr[\text{Win}_i|E_i] = 1$ for $i \in \{5, 6\}$. In addition, it can be checked that the differences between $\Pr[E_5]$ and $\Pr[E_6]$ are negligible assuming the hardness of the hint-MLWE problem. Formally, we can construct an adversary $\mathcal{B}_{\text{hint-MLWE},2}$ against the $\text{hint-MLWE}_{q,k,2k,\chi,\chi,\mathcal{F}_{\text{cpa}}}$ problem such that

$$|\Pr[E_5] - \Pr[E_6]| \leq \text{Adv}_{\mathcal{B}_{\text{hint-MLWE},2}}^{\text{hint-MLWE}}(1^\lambda).$$

Lastly, observe that in the final game, m is information theoretically hidden from \mathcal{A} as v'_B is sampled uniformly at random. Put differently, seed is hidden from \mathcal{A} , and thus, we have $\Pr[E_6] \leq Q/2^\lambda$. Combining everything together, we have

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

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

E.3 Ratchet Simulatability

We recall the theorem statement establishing the ratchet simulatability of our RKEM. Below, we rely on the $\text{hint-MLWE}_{q,k,1,\chi,\tilde{\chi},\mathcal{F}_{\text{sim}}}$ problem, where \mathcal{F}_{sim} is a distribution over $R_q^{1 \times 2k}$ that outputs $[-\widehat{\mathbf{s}}^\top | \widehat{\mathbf{e}}^\top]$ with $\widehat{\mathbf{s}}, \widehat{\mathbf{e}} \stackrel{\$}{\leftarrow} \widehat{\chi}$.

Theorem E.2 (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 \mathbf{H} , there exists adversary $\mathcal{B}_{\text{MLWE}}$ against the $\text{MLWE}_{q,k,\widehat{\chi}}$ problem and adversary $\mathcal{B}_{\text{hint-MLWE}}$ against the $\text{hint-MLWE}_{q,k,1,\chi,\tilde{\chi},\mathcal{F}_{\text{sim}}}$ problem such that

$$\text{Adv}_{\mathcal{A}}^{\text{KeyBaseSim-P}}(1^\lambda) = 0, \quad \text{Adv}_{\mathcal{A}}^{\text{KeyUpdSim-P}}(1^\lambda) \leq \epsilon_{\text{corr}}$$

and

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

where ϵ_{corr} is the probability that correctness with updated keys fails (cf. Def. 5.2).

Proof. We first provide the simulators ($\text{RSimKey-P}_1, \text{RSimKey-P}_2, \text{RSimCtxt-P}$) $_{P \in \{A, B\}}$ 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.

RSimKey-P ₁ (ek _P , dk _P)	RSimKey-P ₂ (ek _{P̄} , dk _{P̄} , aux ₁)	RSimCtxt-P(ek _P , ek _{P̄} , dk _{P̄})
1 : (u _P , s _P) := dk _P	1 : (rand _{seed} , dk _P) := aux ₁	1 : (K, s, e) $\stackrel{\$}{\leftarrow}$ {0, 1} ^λ × χ × χ
2 : seed $\stackrel{\$}{\leftarrow}$ U({0, 1} ^λ){rand _{seed} }	2 : (u _P , s _P) := dk _P	2 : if [P = A] then
3 : (K, s, e) := H(u _P , seed)	3 : seed $\stackrel{\$}{\leftarrow}$ U({0, 1} ^λ){rand _{seed} }	3 : ek _A := ek _A - D · s - e // ek before update
4 : if [P = A] then	4 : (K, s, e) := H(u _P , seed)	4 : ek _B := ek _B - D ^T · dk _B // MLWE noise in ek
5 : ek _A := u _A + D · s + e	5 : m ← Encode(seed) // m ∈ R _q	5 : else // P = B
6 : else // P = B	6 : e _{P̄} $\stackrel{\$}{\leftarrow}$ χ̃{rand _ε }	6 : ek _B := ek _B - D ^T · s - e
7 : ek _B := u _B + D ^T · s + e	7 : v _{P̄} := ek _{P̄} ^T · s _P + e _{P̄} + m ∈ R _q	7 : ek _A := ek _A - D · dk _A
8 : dk _P := s _P + s ∈ R _q ^k	8 : ct _{P̄} := v _{P̄}	8 : seed $\stackrel{\$}{\leftarrow}$ {0, 1} ^λ
9 : aux ₁ := (rand _{seed} , dk _P)	9 : rand ₂ := (rand _{seed} , rand _ε)	9 : H(ek _P , seed) := (K, s, e) // Program RO
10 : return (ek _P , dk _P , aux ₁)	10 : return (ct _{P̄} , K, K, rand ₂)	10 : m ← Encode(seed) // m ∈ R _q
		11 : (s _P , e _P , e _{P̄}) $\stackrel{\$}{\leftarrow}$ χ̃ × χ̃ × χ̃
		12 : v _{P̄} := s _{P̄} ^T · ek _P - s _{P̄} ^T · e _P + e _{P̄} ^T · s _P + e _{P̄} + m
		13 : ct _{P̄} := v _{P̄} // Simulate v _{P̄} w/o user P secret
		14 : return (ct _{P̄} , ek _P , K, K)

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

Updated key simulatability. Due to the symmetry of users A and B, we only focus on bounding the distinguishing advantage of distributions $\mathcal{D}_{A,0}^{\text{KeyUpdSim}}$ and $\mathcal{D}_{A,1}^{\text{KeyUpdSim}}$ (cf. Def. D.1). 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 $\mathcal{D}_{A,i}$ for $i \in \{0, 1, 2\}$ and outputs a bit. Let ϵ_i denote the probability that \mathcal{A} outputs 1 given a sample from $\mathcal{D}_{A,i}$. The goal is then to bound the difference between ϵ_0 and ϵ_2 .

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

Distribution $\mathcal{D}_{A,1}$: The only difference between the prior distribution is that we reuse K, s, 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

$$|\epsilon_0 - \epsilon_1| \leq \epsilon_{\text{corr}},$$

where ϵ_{corr} is the probability that correctness with updated keys fails (cf. Def. 5.2).

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

$$\epsilon_1 = \epsilon_2.$$

Lastly, observe that $\mathcal{D}_{A,2}$ implicitly defines the desired RSimKey-A₁ and RSimKey-A₂. Hence, $\mathcal{D}_{A,2}$ has the same distribution as $\mathcal{D}_{A,1}^{\text{KeyUpdSim}}$. This completes the proof of updated key simulatability.

Ciphertext simulatability. Similarly to above, we only focus on the distinguishing advantage of the distributions $\mathcal{D}_{B,0}^{\text{CtxtSim}}$ and $\mathcal{D}_{B,1}^{\text{CtxtSim}}$. This is proven in a sequence of hybrid distributions given in Figs. 23 and 24. Again, let ϵ_i denote the probability that \mathcal{A} outputs 1 given a sample from $\mathcal{D}_{B,i}$. The goal is then to bound the difference between ϵ_0 and ϵ_6 .

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

Distribution $\mathcal{D}_{B,1}$: The only difference between the prior distribution is that we reuse K, s, e from algorithm REnc-B as opposed to generating them through executing RDec-A. This follows from the

Distribution $\mathcal{D}_{A,0} := \mathcal{D}_{A,0}^{\text{KeyUpdSim}}$	Distribution $\mathcal{D}_{A,1} := \mathcal{D}_{A,1}^{\text{KeyUpdSim}}$
<pre> 1 : $(\text{ek}_B, \text{dk}_B) \xleftarrow{\\$} \mathcal{D}_{\text{RKeyGen-B}}\{\text{rand}_0\}$ 2 : $(\widehat{\text{ek}}_B, \widehat{\text{dk}}_B, \text{aux}_0) \xleftarrow{\\$} \text{RSimKey-B}_1(\text{ek}_B, \text{dk}_B)$ 3 : $(\text{ek}_A, \text{dk}_A) \xleftarrow{\\$} \mathcal{D}_{\text{RKeyGen-A}}\{\text{rand}_1\}$ // REnc-A in full detail 4 : $(\text{rand}_{\text{seed}}, \text{rand}_{\tilde{e}}) := \text{rand}_2$ 5 : $(\mathbf{u}_A, \mathbf{s}_A) := \text{dk}_A$ 6 : $\text{seed} \xleftarrow{\\$} \mathcal{U}(\{0, 1\}^\lambda)\{\text{rand}_{\text{seed}}\}$ 7 : $m \leftarrow \text{Encode}(\text{seed})$ // $m \in R_q$ 8 : $(\mathbf{K}, \mathbf{s}, \mathbf{e}) := \text{H}(\mathbf{u}_A, \text{seed})$ 9 : $\tilde{e}_A \xleftarrow{\\$} \tilde{\chi}\{\text{rand}_{\tilde{e}}\}$ 10 : $v_B := \widehat{\text{ek}}_B^\top \cdot \mathbf{s}_A + \tilde{e}_A + m \in R_q$ 11 : $\text{ct}_B := v_B$ 12 : $\widehat{\text{dk}}_A := \mathbf{s}_A + \mathbf{s} \in R_q^k$ // RDec-B in full detail 13 : $m' := \text{ct}_B - \widehat{\text{ek}}_B^\top \cdot \widehat{\text{dk}}_A$ 14 : $\text{seed}' := \text{Decode}(m')$ 15 : $(\mathbf{K}', \mathbf{s}', \mathbf{e}') := \text{H}(\widehat{\text{ek}}_B, \text{seed}')$ 16 : $\widehat{\text{ek}}_A := \text{ek}_A + \mathbf{D} \cdot \mathbf{s}' + \mathbf{e}'$ 17 : return $((\widehat{\text{ek}}_B, \widehat{\text{dk}}_B), (\widehat{\text{ek}}_A, \widehat{\text{dk}}_A), \text{ct}_B, \mathbf{K}, \mathbf{K}',$ $\text{aux}_0, \text{rand}_0, \text{rand}_1, \text{rand}_2)$ </pre>	<pre> // Same up till $\mathcal{D}_0^{\text{KeyUpdSim}}$, line 12 13 : $\widehat{\text{ek}}_A := \text{ek}_A + \mathbf{D} \cdot \mathbf{s} + \mathbf{e}$ // Reuse $(\mathbf{K}, \mathbf{s}, \mathbf{e})$ from REnc-A 14 : return $((\widehat{\text{ek}}_B, \widehat{\text{dk}}_B), (\widehat{\text{ek}}_A, \widehat{\text{dk}}_A), \text{ct}_B, \mathbf{K}, \mathbf{K},$ $\text{aux}_0, \text{rand}_0, \text{rand}_1, \text{rand}_2)$ </pre> <hr/> <pre> Distribution $\mathcal{D}_{A,2}$ // Same up till $\mathcal{D}_0^{\text{KeyUpdSim}}$, line 3 // Run $\text{RSimKey-A}_1(\text{ek}_A, \text{dk}_A)$ 4 : $(\mathbf{u}_A, \mathbf{s}_A) := \text{dk}_A$ // $\text{ek}_A = \mathbf{u}_A$ 5 : $\text{seed} \xleftarrow{\\$} \mathcal{U}(\{0, 1\}^\lambda)\{\text{rand}_{\text{seed}}\}$ // Sample <i>only</i> $\text{rand}_{\text{seed}}$ 6 : $(\mathbf{K}, \mathbf{s}, \mathbf{e}) := \text{H}(\mathbf{u}_A, \text{seed})$ 7 : $\widehat{\text{ek}}_A := \text{ek}_A + \mathbf{D} \cdot \mathbf{s} + \mathbf{e}$ // Compute at the beginning 8 : $\widehat{\text{dk}}_A := \mathbf{s}_A + \mathbf{s} \in R_q^k$ 9 : $\text{aux}_1 := (\text{rand}_{\text{seed}}, \text{dk}_A)$ // Run $\text{RSimKey-A}_2(\widehat{\text{ek}}_B, \widehat{\text{dk}}_B, \text{aux}_1)$, ignoring the initial steps 10 : $m \leftarrow \text{Encode}(\text{seed})$ // $m \in R_q$ 11 : $\tilde{e}_A \xleftarrow{\\$} \tilde{\chi}\{\text{rand}_{\tilde{e}}\}$ 12 : $v_B := \widehat{\text{ek}}_B^\top \cdot \mathbf{s}_A + \tilde{e}_A + m \in R_q$ 13 : $\text{ct}_B := v_B$ 14 : $\text{rand}_2 := (\text{rand}_{\text{seed}}, \text{rand}_{\tilde{e}})$ // Set rand_2 at the end 15 : return $((\widehat{\text{ek}}_B, \widehat{\text{dk}}_B), (\widehat{\text{ek}}_A, \widehat{\text{dk}}_A), \text{ct}_B, \mathbf{K}, \mathbf{K},$ $\text{aux}_0, \text{rand}_0, \text{rand}_1, \text{rand}_2)$ </pre>

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

same argument made to prove correctness: m used during REnc-B and m' generated during RDec-A are the same with all but a negligible probability. Hence, we have

$$|\epsilon_0 - \epsilon_1| \leq \epsilon_{\text{corr}},$$

where ϵ_{corr} is the probability that correctness with updated keys fails (cf. Def. 5.2).

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

$$\begin{aligned}
v_A - m &= \widehat{\text{ek}}_A^\top \cdot \mathbf{s}_B + \tilde{e}_B \\
&= (\mathbf{D} \cdot \widehat{\mathbf{s}}_A + \widehat{\mathbf{e}}_A)^\top \cdot \mathbf{s}_B + \tilde{e}_B \\
&= \widehat{\mathbf{s}}_A^\top \cdot (\mathbf{D}^\top \cdot \mathbf{s}_B + \mathbf{e}_B) - \widehat{\mathbf{s}}_A^\top \cdot \mathbf{e}_B + \widehat{\mathbf{e}}_A^\top \cdot \mathbf{s}_B + \tilde{e}_B \\
&= \widehat{\mathbf{s}}_A^\top \cdot \text{ek}_B - \widehat{\mathbf{s}}_A^\top \cdot \mathbf{e}_B + \widehat{\mathbf{e}}_A^\top \cdot \mathbf{s}_B + \tilde{e}_B
\end{aligned}$$

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

$$\epsilon_1 = \epsilon_2.$$

Distribution $\mathcal{D}_{B,3}$: The difference between the prior distribution is that we sample $(\mathbf{K}_0, \mathbf{s}, \mathbf{e}) \xleftarrow{\$} \{0, 1\}^\lambda \times \chi \times \chi$ and later program the random oracle H on input $(\text{ek}_A, \text{seed})$. In case the input is already queried

Distribution $\mathcal{D}_{B,0} := \mathcal{D}_{B,0}^{\text{CtxtSim}}$	Distribution $\mathcal{D}_{B,2}$
<pre> 1 : $Q_H[\cdot] := \perp$ // Prepare empty RO 2 : $(ek_A, dk_A) \xleftarrow{\\$} \mathcal{D}_{\text{RKeyGen-A}}\{\text{rand}\}$ 3 : $(\hat{ek}_A, \hat{dk}_A, \text{aux}) \xleftarrow{\\$} \text{RSimKey-A}_1(ek_A, dk_A)$ // Real user B procedure in full detail // $\mathcal{D}_{\text{RKeyGen-B}}$ in full detail 4 : $(s_B, e_B) \xleftarrow{\\$} \chi \times \chi$ 5 : $u_B := \mathbf{D}^\top \cdot s_B + e_B \in R_q^k$ 6 : $(ek_B, dk_B) := (u_B, (u_B, s_B))$ // $\text{REnc-B}(\hat{ek}_A, dk_B)$ in full detail 7 : $\text{seed} \xleftarrow{\\$} \{0, 1\}^\lambda$ 8 : $m \leftarrow \text{Encode}(\text{seed})$ // $m \in R_q$ 9 : $(K, s, e) := H(ek_B, \text{seed})$ 10 : $\tilde{e}_B \xleftarrow{\\$} \tilde{\chi}$ 11 : $v_A := \hat{ek}_A^\top \cdot s_B + \tilde{e}_B + m \in R_q$ 12 : $ct_A := v_A$ 13 : $\hat{dk}_B := s_B + s \in R_q^k$ // Update and erase dk_B // $\text{RDec-A}(\hat{dk}_A, ct_A, ek_B)$ in full detail 14 : $m' := ct_A - ek_B^\top \cdot \hat{dk}_A$ 15 : $\text{seed}' := \text{Decode}(m')$ 16 : $(K', s', e') := H(ek_B, \text{seed}')$ 17 : $\hat{ek}_B := ek_B + \mathbf{D}^\top \cdot s' + e'$ 18 : return $(\text{aux}, \text{rand}, (\hat{ek}_A, \hat{dk}_A),$ $ct_A, (ek_B, \hat{ek}_B), (K, K'))$ </pre>	<pre> // Same up till $\mathcal{D}_{B,0}$, line 3 4 : $\hat{s}_A := \hat{dk}_A$ 5 : $\hat{e}_A := \hat{ek}_A - \mathbf{D} \cdot \hat{s}_A$ // MLWE noise in \hat{ek}_A 6 : $(s_B, e_B, \tilde{e}_B) \xleftarrow{\\$} \chi \times \chi \times \tilde{\chi}$ 7 : $u_B := \mathbf{D}^\top \cdot s_B + e_B \in R_q^k$ 8 : $ek_B := u_B$ // Remove dk_B 9 : $\text{seed} \xleftarrow{\\$} \{0, 1\}^\lambda$ 10 : $(K, s, e) := H(ek_B, \text{seed})$ 11 : $\hat{ek}_B := ek_B + \mathbf{D}^\top \cdot s + e$ // Update immediately 12 : $m \leftarrow \text{Encode}(\text{seed})$ // Generate ct_A w/o dk_B 13 : $v_A := \hat{s}_A^\top \cdot ek_B - \hat{s}_A^\top \cdot e_B + \hat{e}_A^\top \cdot s_B + \tilde{e}_B + m$ 14 : $ct_A := v_A$ 15 : return $(\text{aux}, \text{rand}, (\hat{ek}_A, \hat{dk}_A),$ $ct_A, (ek_B, \hat{ek}_B), (K, K))$ </pre>
<p>Distribution $\mathcal{D}_{B,1}$</p> <pre> // Same up till $\mathcal{D}_{B,0}$, line 13 13 : $\hat{ek}_B := ek_B + \mathbf{D}^\top \cdot s + e$ // Reuse (K, s, e) from REnc-B 14 : return $(\text{aux}, \text{rand}, (\hat{ek}_A, \hat{dk}_A),$ $ct_A, (ek_B, \hat{ek}_B), (K, K))$ </pre> <p>$H(u, \text{seed})$ // Used by all distributions</p> <pre> 1 : if $[Q_H[u, \text{seed}] = \perp]$ then 2 : $(K_0, s, e) \xleftarrow{\\$} \{0, 1\}^\lambda \times \chi \times \chi$ 3 : $Q_H[u, \text{seed}] \leftarrow (K_0, s, e)$ 4 : return $Q_H[u, \text{seed}]$ </pre>	<p>Distribution $\mathcal{D}_{B,3}$</p> <pre> // Same up till $\mathcal{D}_{B,0}$, line 3 4 : $(K, s, e) \xleftarrow{\\$} \{0, 1\}^\lambda \times \chi \times \chi$ // Sample w/o RO 5 : $\hat{s}_A := \hat{dk}_A$ 6 : $\hat{e}_A := \hat{ek}_A - \mathbf{D} \cdot \hat{s}_A$ 7 : $(s_B, e_B, \tilde{e}_B) \xleftarrow{\\$} \chi \times \chi \times \tilde{\chi}$ 8 : $u_B := \mathbf{D}^\top \cdot s_B + e_B \in R_q^k$ 9 : $ek_B := u_B$ 10 : $\hat{ek}_B := ek_B + \mathbf{D}^\top \cdot s + e$ 11 : $\text{seed} \xleftarrow{\\$} \{0, 1\}^\lambda$ 12 : if $[Q_H[ek_B, \text{seed}] \neq \perp]$ then 13 : return \top // Output special symbol \top 14 : $Q_H[ek_B, \text{seed}] := (K_0, s, e)$ // Program RO 15 : $m \leftarrow \text{Encode}(\text{seed})$ 16 : $v_A := \hat{s}_A^\top \cdot ek_B - \hat{s}_A^\top \cdot e_B + \hat{e}_A^\top \cdot s_B + \tilde{e}_B + m$ 17 : $ct_A := v_A$ 18 : return $(\text{aux}, \text{rand}, (\hat{ek}_A, \hat{dk}_A),$ $ct_A, (ek_B, \hat{ek}_B), (K, K))$ </pre>

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

(i.e., $Q_H[ek_A, \text{seed}] \neq \perp$), the distribution outputs a special symbol \top . This distribution is identical to the previous one as long as $Q_H[ek_A, \text{seed}] \neq \perp$. Since seed is sampled uniformly random over $\{0, 1\}^\lambda$, the probability of this occurring is $Q/2^\lambda$.

Hence, we have

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

Distribution $\mathcal{D}_{B,4}$	Distribution $\mathcal{D}_{B,6} := \mathcal{D}_{B,1}^{\text{CtxtSim}}$
<p>// Same up till $\mathcal{D}_{B,0}$, line 3</p> <p>4 : $(K, s, e) \xleftarrow{\\$} \{0, 1\}^\lambda \times \chi \times \chi$</p> <p>5 : $\hat{s}_A := \widehat{dk}_A$</p> <p>6 : $\hat{e}_A := \hat{ek}_A - \mathbf{D} \cdot \hat{s}_A$</p> <p>7 : $(s_B, e_B, \tilde{e}_B) \xleftarrow{\\$} \chi \times \chi \times \tilde{\chi}$</p> <p>8 : $\mathbf{u}_B \xleftarrow{\\$} R_q^k$</p> <p>9 : $ek_B := \mathbf{u}_B$</p> <p>10 : $\hat{ek}_B := ek_B + \mathbf{D}^\top \cdot s + e$</p> <p>11 : $\text{seed} \xleftarrow{\\$} \{0, 1\}^\lambda$</p> <p>12 : if $\llbracket Q_H[ek_B, \text{seed}] \neq \perp \rrbracket$ then</p> <p>13 : return \top</p> <p>14 : $Q_H[ek_B, \text{seed}] := (K_0, s, e)$</p> <p>15 : $m \leftarrow \text{Encode}(\text{seed})$</p> <p>16 : $v_A := \hat{s}_A^\top \cdot ek_B - \hat{s}_A^\top \cdot e_B + \hat{e}_A^\top \cdot s_B + \tilde{e}_B + m$</p> <p>17 : $ct_A := v_A$</p> <p>18 : return $(\text{aux}, \text{rand}, (\hat{ek}_A, \widehat{dk}_A),$ $ct_A, (ek_B, \hat{ek}_B), (K, K))$</p>	<p>// Same up till $\mathcal{D}_{B,0}$, line 3</p> <p>// Sample from $\hat{\mathcal{D}}_{\text{RKeyGen-B}}$</p> <p>4 : $(\hat{s}_B, \hat{e}_B) \xleftarrow{\\$} \tilde{\chi} \times \tilde{\chi}$</p> <p>5 : $\hat{\mathbf{u}}_B := \mathbf{D}^\top \cdot \hat{s}_B + \hat{e}_B \in R_q^k$</p> <p>6 : $\hat{ek}_B := \hat{\mathbf{u}}_B$ // Implicitly $\widehat{dk}_B := \hat{s}_B$ // Run $\text{RSimCtxt-B}(\hat{ek}_B, \hat{ek}_A, \widehat{dk}_A)$</p> <p>7 : $(K, s, e) \xleftarrow{\\$} \{0, 1\}^\lambda \times \chi \times \chi$</p> <p>8 : $\hat{s}_A := \widehat{dk}_A$</p> <p>9 : $\hat{e}_A := \hat{ek}_A - \mathbf{D} \cdot \hat{s}_A$</p> <p>10 : $ek_B := \hat{ek}_B - \mathbf{D}^\top \cdot s - e$</p> <p>11 : $\text{seed} \xleftarrow{\\$} \{0, 1\}^\lambda$</p> <p>12 : if $\llbracket Q_H[ek_B, \text{seed}] \neq \perp \rrbracket$ then</p> <p>13 : return \top</p> <p>14 : $Q_H[ek_B, \text{seed}] := (K_0, s, e)$</p> <p>15 : $m \leftarrow \text{Encode}(\text{seed})$</p> <p>16 : $(s_B, e_B, \tilde{e}_B) \xleftarrow{\\$} \chi \times \chi \times \tilde{\chi}$</p> <p>17 : $v_A := \hat{s}_A^\top \cdot ek_B - \hat{s}_A^\top \cdot e_B + \hat{e}_A^\top \cdot s_B + \tilde{e}_B + m$</p> <p>18 : $ct_A := v_A$</p> <p>19 : return $(\text{aux}, \text{rand}, (\hat{ek}_A, \widehat{dk}_A),$ $ct_A, (ek_B, \hat{ek}_B), (K, K))$</p>
<p>Distribution $\mathcal{D}_{B,5}$</p> <p>// Same up till $\mathcal{D}_{B,0}$, line 3</p> <p>4 : $(K, s, e) \xleftarrow{\\$} \{0, 1\}^\lambda \times \chi \times \chi$</p> <p>5 : $\hat{s}_A := \widehat{dk}_A$</p> <p>6 : $\hat{e}_A := \hat{ek}_A - \mathbf{D} \cdot \hat{s}_A$</p> <p>7 : $\hat{\mathbf{u}}_B \xleftarrow{\\$} R_q^k$</p> <p>8 : $\hat{ek}_B := \hat{\mathbf{u}}_B$</p> <p>9 : $ek_B := \hat{ek}_B - \mathbf{D}^\top \cdot s - e$</p> <p>10 : $\text{seed} \xleftarrow{\\$} \{0, 1\}^\lambda$</p> <p>11 : if $\llbracket Q_H[ek_B, \text{seed}] \neq \perp \rrbracket$ then</p> <p>12 : return \top</p> <p>13 : $Q_H[ek_B, \text{seed}] := (K_0, s, e)$</p> <p>14 : $m \leftarrow \text{Encode}(\text{seed})$</p> <p>15 : $(s_B, e_B, \tilde{e}_B) \xleftarrow{\\$} \chi \times \chi \times \tilde{\chi}$</p> <p>16 : $v_A := \hat{s}_A^\top \cdot ek_B - \hat{s}_A^\top \cdot e_B + \hat{e}_A^\top \cdot s_B + \tilde{e}_B + m$</p> <p>17 : $ct_A := v_A$</p> <p>18 : return $(\text{aux}, \text{rand}, (\hat{ek}_A, \widehat{dk}_A),$ $ct_A, (ek_B, \hat{ek}_B), (K, K))$</p>	

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

Distribution $\mathcal{D}_{B,4}$: The difference between the prior distribution is that we sample a random \mathbf{u}_B from \mathcal{R}_q^k and sets user B's key as $ek_B := \mathbf{u}_B$. Recall in the previous game, \mathbf{u}_B was set as MLWE instances. Since s_B and e_B are partially leaked via $ct_A = v_A$ and $\widehat{dk}_A = \hat{s}_A$, we cannot rely on the standard

Notations	Explanation
d	Amount of bit dropping performed on ciphertext such that $d < \lceil \log_2(q) \rceil$
q_d	Rounded modulus satisfying $q_d := 2^d$
Compress_q Decompress_q	Rounding operations from Kyber [33]

Table 5: Parameters and notations regarding bit-dropping optimization. See App. A.2 for the definitions of Compress_q and Decompress_q .

MLWE assumption to argue indistinguishability of the two distributions.¹⁸ However, noticing that $\tilde{e} \stackrel{s}{\leftarrow} \tilde{\chi}$ is information theoretically hidden to \mathcal{A} conditioning on the game not aborting, we can rely instead on the *hint* MLWE assumption. Concretely, we can construct an adversary $\mathcal{B}_{\text{hint-MLWE}}$ against the $\text{hint-MLWE}_{q,k,1,\chi,\tilde{\chi},\mathcal{F}_{\text{sim}}}$ problem such that

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

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

Distribution $\mathcal{D}_{B,5}$: The only difference from the prior distribution is that we sample user B's updated key $\hat{\mathbf{e}}_B = \hat{\mathbf{u}}_B$ uniformly sample and then set \mathbf{e}_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 $\mathcal{D}_{B,6}$: The only difference from the prior distribution is that we sample user B's updated key $\hat{\mathbf{e}}_B = \hat{\mathbf{u}}_B$ as a valid MLWE instance. It is straight forward to see that the two distributions is indistinguishable under the MLWE assumption. Formally, we can construct an adversary $\mathcal{B}_{\text{MLWE}}$ against the $\text{MLWE}_{q,k,\hat{\chi}}$ problem such that

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

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

E.4 Details on Optimized RKEM

We minimize the size of the ciphertext in our RKEM by performing bit-dropping, similarly to Kyber [33]. The additional notations we use and our optimized construction is provide in Table 5 and Fig. 25, respectively.

Lemma 6.1 (Correctness with bit-dropping). *Our optimized RKEM Katana is correct assuming*

$$\Pr[\|\hat{\mathbf{s}}^\top \cdot \mathbf{e} - \hat{\mathbf{e}}^\top \cdot \mathbf{s} + \tilde{e} + \delta\|_\infty \leq q/4] = 1 - \text{negl}(\lambda), \quad (1)$$

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

Proof. Since the encapsulation key is unchanged, correctness of update key distribution follows from Lemma E.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

$$v_A = \hat{\mathbf{e}}_A^\top \cdot \mathbf{s}_B + \tilde{e}_B + m$$

¹⁸ It is worth noting that at a high level, this is the argument where [1] made a mistake by arguing indistinguishability solely on MLWE.

REnc-P(\widehat{ek}_P, dk_P)	RDec-P($\widehat{dk}_P, ct_P, ek_P$)
1 : $(\mathbf{u}_P, \mathbf{s}_P) := dk_P$	1 : $ct'_P := \text{Decompress}_q(ct_P, d) \in R_q$
2 : $\text{seed} \xleftarrow{\$} \{0, 1\}^\lambda$	2 : $m := ct'_P - ek_P^\top \cdot \widehat{dk}_P \in R_q$
3 : $m \leftarrow \text{Encode}(\text{seed}) \quad // m \in R_q$	3 : $\text{seed} := \text{Decode}(m)$
4 : $(\mathbf{K}, \mathbf{s}, \mathbf{e}) := \text{H}(\mathbf{u}_P, \text{seed})$	4 : $(\mathbf{K}, \mathbf{s}, \mathbf{e}) := \text{H}(ek_P, \text{seed})$
5 : $\tilde{e}_P \xleftarrow{\$} \tilde{\chi}$	$// \text{Update } ek_P$
6 : $v_P := \widehat{ek}_P^\top \cdot \mathbf{s}_P + \tilde{e}_P + m \in R_q$	5 : if $[[\mathbf{P} = \mathbf{A}]]$ then
7 : $ct_P := \text{Compress}_q(v_P, d) \in R_{qd}$	6 : $\widehat{ek}_B := ek_B + \mathbf{D}^\top \cdot \mathbf{s} + \mathbf{e}$
8 : $\widehat{dk}_P := \mathbf{s}_P + \mathbf{s} \in R_q^k \quad // \text{Update and erase } dk_P$	7 : else $// \mathbf{P} = \mathbf{B}$
9 : return $(ct_P, \mathbf{K}, \widehat{dk}_P)$	8 : $\widehat{ek}_A := ek_A + \mathbf{D} \cdot \mathbf{s} + \mathbf{e}$
	9 : return $(\mathbf{K}, \widehat{ek}_B)$

Fig. 25: Our RKEM Π_{RKEM} with the bit-dropping optimization. The differences are highlighted in blue. RSetup and RKeyGen-P are defined identically to those in Fig. 6.

$$\begin{aligned}
&= (\mathbf{D} \cdot \widehat{\mathbf{s}}_A + \widehat{\mathbf{e}}_A)^\top \cdot \mathbf{s}_B + \tilde{e}_B + m \\
&= (\mathbf{D}^\top \cdot \widehat{\mathbf{s}}_B + \mathbf{e}_B)^\top \cdot \mathbf{s}_A + \widehat{\mathbf{e}}_A^\top \cdot \mathbf{s}_B - \mathbf{e}_B^\top \cdot \widehat{\mathbf{s}}_A + \tilde{e}_B + m \\
&= ek_B^\top \cdot \widehat{dk}_A + \underbrace{\widehat{\mathbf{e}}_A^\top \cdot \mathbf{s}_B - \mathbf{e}_B^\top \cdot \widehat{\mathbf{s}}_A}_{=:z} + \tilde{e}_B + m
\end{aligned}$$

Plugging this into the decryption equation, we have

$$\begin{aligned}
\text{Decompress}_q(ct_A, d) - ek_B^\top \cdot \widehat{dk}_A &= \text{Decompress}_q(\text{Compress}_q(v_A), d) - v_A + z + m \\
&= m + z + \delta,
\end{aligned}$$

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

E.5 Details on Concrete Parameter Selection

Let us explain the omitted details behind our parameter selection.

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:

$$y_i \sim [2kn] \cdot \widehat{\chi}_0 \cdot \chi_0 + \tilde{\chi}_0, \quad (4)$$

where $\widehat{\chi}_0$ (resp. χ_0 , resp. $\tilde{\chi}_0$) is the distribution of each integer coefficient of $\widehat{\chi}$ (resp. χ , resp. $\tilde{\chi}$). Now, let us note χ_{round} the distribution entailed by rounding v and $\chi_{0,\text{round}}$ as the distribution of each integer coefficient. Since, we use exactly the same rounding method as in Kyber, we may re-employ their analysis, [33, see Eqs. (7) and (8)] in order to characterize the resulting distributions. Eq. (4) may then be adapted as follows:

$$y_i \sim [2kn] \cdot \widehat{\chi}_0 \cdot \chi_0 + \tilde{\chi}_0 + \chi_{0,\text{round}}. \quad (5)$$

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 $\leq 2^{-\lambda}/n$). Finally, we use the union bound to ensure that $\|\widehat{\mathbf{s}}^\top \cdot \mathbf{e} - \widehat{\mathbf{e}}^\top \cdot \mathbf{s} + \tilde{e} + \delta\|_\infty \leq q/4$ with overwhelming probability.

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,

RKeyGen-P(par, mode)	REnc-P(\widehat{ek}_P, dk_P)	RDec-P($\widehat{dk}_P, ct_P, ek_P$)
1 : $(ek_P, dk_P) \leftarrow \text{KeyGen}(1^\lambda)$	1 : $(ct, K) \xleftarrow{\$} \text{Enc}(ek_P)$	1 : parse (\widehat{ek}_P, ct) $\leftarrow ct_P$
2 : return (ek_P, dk_P)	2 : $(\widehat{ek}_P, \widehat{dk}_P) \leftarrow \text{KeyGen}(1^\lambda)$	2 : $K \leftarrow \text{Dec}(\widehat{dk}_P, ct)$
	3 : $ct_P := (\widehat{ek}_P, ct)$	3 : return (K, \widehat{ek}_P)
	4 : return $(ct_P, K, \widehat{dk}_P)$	

Fig. 26: A generic forward-secure RKEM based on a KEM = (KeyGen, Enc, Dec). When using the RKEM to instantiate our generic CKA construction (cf. Sec. 5.2) this exactly yields the generic CKA construction analyzed by Alwen et al. [1].

under Heuristics 1 and 2, we may say that $\text{hint-MLWE}_{q,k,1,\chi,\tilde{\chi},\mathcal{F}_{\text{sim}}}$ is at least as hard as $\text{MLWE}_{q,k,2k,\chi'}$, where χ' is the discrete Gaussian of parameter σ_0 :¹⁹

$$\frac{1}{\sigma_0^2} = \frac{1}{\sigma^2} + \frac{s_1(\mathbf{M})^2}{\tilde{\sigma}^2} \quad (6)$$

where $\mathbf{M} = [-\widehat{\mathbf{s}}^\top \mid \widehat{\mathbf{e}}^\top]$, σ (resp. $\tilde{\sigma}$, resp. $\hat{\sigma}$) is the standard variation of χ (resp. $\tilde{\chi}$, resp. $\hat{\chi}$), and χ (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 $\mathbb{E} \left[\|\mathbf{M} \cdot \mathbf{z}\|^2 \right] = 2kn^2\sigma^2\hat{\sigma}^2$ and $\mathbb{E} \left[\|\mathbf{z}\|^2 \right] = 2kn\sigma^2$. Therefore we heuristically estimate:

$$\mathbb{E} \left[\frac{\|\mathbf{M} \cdot \mathbf{z}\|^2}{\|\mathbf{z}\|^2} \right] \approx \frac{\mathbb{E} \left[\|\mathbf{M} \cdot \mathbf{z}\|^2 \right]}{\mathbb{E} \left[\|\mathbf{z}\|^2 \right]} = n\hat{\sigma}^2 \quad (7)$$

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. (6) by $n\hat{\sigma}^2$. Since $\hat{\sigma} = 2\sigma$, Eq. (6) becomes:

$$\frac{1}{\sigma_0^2} = \frac{1}{\sigma^2} + \frac{4n\sigma^2}{\tilde{\sigma}^2}. \quad (8)$$

Eq. (8) 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 σ_0 as the one obtained in the study of FS-IND-CPA security. Again, we use the lattice estimator to estimate the hardness of the underlying MLWE assumption.

F Additional RKEM instantiation

F.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 F.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 $(\widehat{ek}_P, \widehat{dk}_P)$ are just fresh keys unrelated to this round's KEM encapsulation, leaking \widehat{dk}_P does not affect FS-IND-CPA security. \square

¹⁹ Recall that $s_1(\mathbf{M}^t \cdot \mathbf{M}) = s_1(\mathbf{M})^2$.

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

RSimKey-P ₁ (ek _P , dk _P)	RSimKey-P ₂ (ek _{P̄} , dk _{P̄} , aux ₁)	RSimCtxt-P(ek _P , ek _{P̄} , dk _{P̄})
1 : (dk _P , ek _P) $\xleftarrow{\$}$ KeyGen(1 ^λ ; rand)	1 : parse (ek _P , rand) \leftarrow aux ₁	1 : (ct, K) $\xleftarrow{\$}$ Enc(ek _{P̄})
2 : aux ₁ := (ek _P , rand)	2 : (ct, K) $\xleftarrow{\$}$ Enc(ek _{P̄} ; rand')	2 : K' \leftarrow Dec(dk _{P̄} , ct)
3 : return (dk _P , ek _P , aux ₂)	3 : K' \leftarrow Dec(dk _{P̄} , ct)	3 : ct _{P̄} := (ek _P , ct)
	4 : ct _{P̄} := (ek _P , ct)	4 : ek _P := ek _P return (ct _{P̄} , ek _P , K, K')
	5 : rand ₂ := (aux ₁ , rand')	
	6 : return (ct _{P̄} , K, K, rand ₂)	

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

F.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 5.1). The difference is that we modify the REnc-P algorithm so that $H(\mathbf{u}_P, \text{seed})$ outputs only K rather than $(K, \mathbf{s}, \mathbf{e})$, and skips the updating of dk_P . Moreover, the RDec-P algorithm is modified similarly where $ek_{P̄}$ is no longer updated.

It is easy to check that correctness (cf. Lemma E.1) holds, where the updated key distribution $\hat{\chi}$ is replaced by the non-updated one χ . 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 Game_5 and Game_6 in the proof of Theorem 6.1. The difference stems from the fact that the reduction no longer requires to simulate the updated decapsulation key $dk_{P̄}$. Lastly, ratcheting simulatability remains intact as well. It is worth highlighting that we still need hint-MLWE to simulate the ciphertext in distribution $\mathcal{D}_{B,2}$ in the proof of Theorem 6.2 as this argument does not stem from forward security.

F.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. [9] can be viewed as an instantiation of RKEM.²⁰

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 $K_i = (Y_{i-1})^{x_i}$ as the CKA into the key chain. For the next message, Bob then sends $Y_{i+1} = g^{y_{i+1}}$ and outputs $K_{i+1} = X_i^{y_{i+1}}$ — therefore reuses X_i . In the forward-secure modification, Bob instead computes $\widehat{X}_i := X_i^{H(K_i)}$, upon receiving X_i , and then encapsulates to that public key instead. Upon sending X_i , Alice updates her secret key $\hat{x}_i := x_i \cdot H(K_i)$ analogously. Simply put, the idea is that leaking \hat{x}_i no longer exposed K_i , which can be proven in the ROM. This neatly fits the RKEM abstraction as used by our generic CKA construction in Sec. 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.

²⁰ Bienstock et al. [9] dubbed their forward secure protocol the “Triple Ratchet”. To avoid confusion with our hybrid SM protocol, we omit this term.

RKeyGen-P(par, mode)	REnc-P($\widehat{ek}_P := Y, dk_P := x$)	RDec-P($\widehat{dk}_P := x, ct_P, ek_{\bar{P}} := Y$)
1 : $x \xleftarrow{\$} \mathbb{Z}_q$	1 : $ct_P := ()$ // empty ciphertext	1 : $K := Y^x$
2 : $(ek_P, dk_P) \leftarrow (g^x, x)$	2 : $K := Y^x$	2 : $\widehat{ek}_{\bar{P}} := Y \cdot H(K)$
3 : return (ek_P, dk_P)	3 : $\widehat{dk}_P := x \cdot H(K)$	3 : return $(K, \widehat{ek}_{\bar{P}})$
	4 : return $(ct_P, K, \widehat{dk}_P)$	

Fig. 28: Diffie-Hellman based RKEM. When using the base protocol to instantiate our generic CKA construction (cf. Sec. 5.2) this exactly yields the Double Ratchet CKA as analyzed by Alwen et al. [1]. When making the RKEM forward secure, this yields a forward-secure KEM, as analyzed by Bienstock et al. [9]. Note that for the forward secure variant, key generation is the same irrespective of the mode.

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

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

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

Theorem F.4. *The non-forward secure RKEM construction from Fig. 28 is IND-CPA secure under the DDH assumption. When modelling H as a random oracle, the forward secure variant is FS-IND-CPA secure under the DDH assumption.*

Proof. Let $(ek_A, dk_A) := (X, x)$ and $(\widehat{ek}_B, \widehat{dk}_B) := (Y, y)$, 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 $ek_A = X$, $\widehat{ek}_B = Y$ and either the real key $K_0 = g^{xy}$ or an uniform random and independent key $K_1 \in G$. This directly corresponds to a DDH instance. In the forward secure protocol, the adversary is given the following:

- $ek_A = X$
- $\widehat{ek}_A = X^{H(K_0)}$
- $\widehat{ek}_B = Y$
- $\widehat{dk}_A = x \cdot H(K_0)$
- K_b , i.e., either K_0 or K_1 depending on the bit b .

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

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{P} instead of the one of P . Due to the symmetry of Diffie-Hellman, this nevertheless allows to compute the correct key. The forward-secure variant is slightly more elaborate and involves $R\text{SimKey-}P_1$ actually choosing a fresh key pair and then $R\text{SimKey-}P_2$ programming the ROM to make this appear consist.

Theorem F.5. *The forward-secure 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 from 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 $H(K)$ 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. \square

$\text{RSimKey-P}_1(\text{ek}_P := X, \text{dk}_P := x)$	$\text{RSimKey-P}_2(\widehat{\text{ek}}_P := \widehat{Y}, \widehat{\text{dk}}_P := \widehat{y}, \text{aux}_1)$	$\text{RSimCtxt-P}(\widehat{\text{ek}}_P := \widehat{X}, \widehat{\text{ek}}_P := \widehat{Y}, \widehat{\text{dk}}_P := \widehat{y})$
1 : $(\widehat{x}, \widehat{X}) \xleftarrow{\$} \text{RKeyGen-P}()$	1 : parse $(x, \widehat{x}) \leftarrow \text{aux}_1$	1 : $z \xleftarrow{\$} \mathbb{Z}_q$
2 : $\text{aux}_1 := (x, \widehat{x})$	2 : $K := (\widehat{Y})^x$	2 : $X := \widehat{X}^{-z}$
3 : return $(\widehat{x}, \widehat{X}, \text{aux}_1)$	3 : $\text{H}(K) := x^{-1} \cdot \widehat{x} \quad // \text{Program RO}$	3 : $K := X^{\widehat{y}}$
	4 : $\text{ct}_P, \text{rand}_2 := ()$	4 : $\text{ct}_P := ()$
	5 : return $(\text{ct}_P, K, K, \text{rand}_2)$	5 : return (ct_P, X, K, K)

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

$\text{RSimKey-P}_1(\text{ek}_P := X, \text{dk}_P := x)$	$\text{RSimKey-P}_2(\widehat{\text{ek}}_P := Y, \widehat{\text{dk}}_P := y, \text{aux}_1 := x)$	$\text{RSimCtxt-P}(\widehat{\text{ek}}_P := X, \widehat{\text{ek}}_P := Y, \widehat{\text{dk}}_P := y)$
1 : $\text{aux}_1 := x$	1 : $K = Y^x$	1 : $K = X^y$
2 : return (x, X, aux_1)	2 : $\text{ct}_P := (); \text{rand}_2 := ()$	2 : $\text{ct}_P := ()$
	3 : return $(\text{ct}_P, K, K, \text{rand}_2)$	3 : return (ct_P, X, K, K)

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

G 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 [2,21], 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. [13,6,23], considers “good” (i.e., uniformly sampled) randomness that can be revealed to the adversary.
- **Adversarial randomness.** Another line of work, e.g. [1,20,9,3] assumes that the adversary gets to fully control the randomness.

As argued by Balli et al. [5], 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 [1], 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 [20] the authors generate the randomness via an output of a hash function. In the ROM, one can then prove, using an union bound argument, that the probability of the adversary finding bad randomness triggering a decryption failure is negligible (cf. [20, 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 [5]. Finally, randomness hardening should preferably be performed at the operating system level and not the at the level of an individual cryptographic protocol.