Chosen-Ciphertext Security for Nested Proxy Re-Encryption with Restricted Key Exposure

Haotian Yin and Jie Zhang

Xi'an Jiaotong-Liverpool University
haotian.yin23@student.xjtlu.edu.cn, jie.zhang01@xjtlu.edu.cn

Abstract. Proxy re-encryption (PRE) allows an authorised proxy to re-encrypt ciphertexts for another recipient without learning their plaintexts. We construct a single-hop PRE scheme with restricted chosen-ciphertext security (rCCA2). For a given honest delegator, an adversary that obtains an outgoing re-encryption key cannot also obtain re-encryptions to corrupted recipients. We revisit the nested encryption of Ivan and Dodis (NDSS 2003) with the encrypted delivery of the inner secret key described by Ateniese et al. (ACM TISSEC 2006). A recipient can recover this key from one authorised delivery and use it to decrypt inner ciphertexts sent to others. Encrypting each inner ciphertext under an independent public key of its intended recipient gives a scheme secure against restricted honest re-encryption attacks (rHRA). To obtain rCCA2, we use CCA2-secure public-key encryption, authenticate source ciphertexts and encrypted inner secret keys with strongly unforgeable signatures, and protect inner ciphertexts with deterministic authenticated encryption during re-encryption. We instantiate rCCA2 using hybrid public-key encryption with ML-KEM, ML-DSA signatures, and AES-SIV, and give key and ciphertext sizes. A packet-level OpenPGP prototype forwards the session key in a separate envelope and verifies decryption of the same encrypted body by the original and forwarding recipients.

Keywords: Proxy re-encryption · restricted chosen-ciphertext security · rHRA security · post-quantum cryptography · ML-KEM · OpenPGP · email forwarding

1 Introduction

Alice sends an encrypted email to Bob. Later, Bob wants his mail server to forward messages to Charlie while Bob is offline. Proxy re-encryption (PRE) [BBS98] lets Bob authorise the server to do this by giving it a re-encryption key (rekey). In classical OpenPGP, Vial-Prado and Wussler [VPW22] implement forwarding through diverted elliptic-curve Diffie–Hellman exchanges. Post-quantum OpenPGP [WHWN24, KRSW26] protects the session key with both ML-KEM [Nat24b] and ECDH.

What does a compromised proxy learn? The proxy holds long-lived rekeys. Suppose an adversary compromises the proxy and obtains the rekey that Bob issued for an honest recipient. It can apply the proxy algorithm locally to any source ciphertext for Bob. Even with this capability, it should be unable to distinguish which of two messages Alice encrypted to Bob. In the security game, this exposure is a successful query for a rekey from an honest delegator to an honest recipient.

What does an authorised recipient learn? Charlie holds his own secret key and receives messages that the proxy has re-encrypted from Bob to him. He is entitled to open those deliveries. The question is whether he can use what he learns from them to read a different message to Bob, even if he sees that message forwarded to another honest recipient. The game gives Charlie his secret key and the re-encryption outputs addressed to him. The nested-encryption baseline fails this test after a single authorised delivery (Proposition 3.4).

Our restricted models cover both exposures in one experiment. For each honest delegator, the adversary may obtain outgoing rekeys to honest users *or* re-encryptions to corrupted recipients. For example, obtaining Charlie's authorised delivery from Bob and later obtaining Bob's rekey to another honest user falls outside the guarantee. The adversary can combine

the two capabilities across different delegators. Cohen's full honest re-encryption attack (HRA) model [Coh19] also permits the combination for the same delegator. Section 3 shows why this distinction matters for our construction. We believe that, under this restriction, our models capture a useful range of practical attacks by compromised proxies and corrupted recipients.

Our main result is a generic single-hop PRE scheme with restricted chosen-ciphertext security (rCCA2). The adversary can submit ciphertexts of its choice for re-encryption and decryption before and after the challenge, subject to the exposure and challenge exclusions in Definition 2.11. We instantiate the scheme with hybrid public-key encryption (HPKE) using ML-KEM, ML-DSA signatures, and AES-SIV [Har08]. An OpenPGP forwarding experiment measures how many session-key envelopes the proxy can re-encrypt per second.

1.1 Related work

Recent Kyber-based schemes. Sato and Shikata [SS26] claim bounded CCA2 security for a single-hop PRE scheme using a Kyber-based construction and a compiler. Their base scheme omits Kyber compression and replaces CBD sampling with bounded uniform distributions, so it is not a direct instantiation using native ML-KEM algorithms. Section 5.3 gives parameterised size estimates, and Appendix C discusses a query-counting gap in their proof.

Key switching. Lattice PRE constructions based on key switching [PRSV17, BV11, CCG$^+$25], including public-key switching entries used in OpenFHE [Ope26], add error to transformed ciphertexts. Native ML-KEM's fixed modulus and compression leave a limited error budget. Its Fujisaki–Okamoto (FO) transform also requires accounting for randomness recoverable by recipients and for the deterministic re-encryption check. Appendix B gives an illustrative comparison of switching and padding costs.

Bootstrapping and ciphertext sanitisation. Fuchsbauer et al. [FKKP19] construct HRA-secure unbounded multi-hop PRE from FHE and ciphertext sanitisation. Wan et al. [WWXW25] optimise this approach using FHEW-like blind rotation and noise flooding. These approaches require homomorphic evaluation and ciphertext sanitisation, which native ML-KEM does not provide.

Delegation and encryption interfaces. Xagawa's lattice construction [Xag10, Theorem 15.3.3] uses differences of uniform secrets, whose masking argument does not directly apply to small native ML-KEM secrets. Owner-encrypted PRE [YZD$^+$26] requires the secret-key holder, Bob, to generate ciphertexts, whereas our setting allows Alice to encrypt using Bob's public key.

Trapdoor-based PRE. Kirshanova [Kir14] constructs a unidirectional, single-hop lattice PRE scheme under lattice trapdoors. Native ML-KEM keys do not provide the required sampling trapdoor.

Proxy encryption and PRE. Ivan and Dodis (ID03) [ID03] give a generic construction based on two layers of encryption, which we number from the outside in. The proxy holds the outer secret key sk^1 and removes the outer layer; the recipient Charlie uses the inner secret key sk^2 to decrypt the remaining inner ciphertext. Here, sk^2 is the delegator's inner secret key, distinct from Charlie's own long-term secret key. Canetti and Hohenberger [CH07] distinguish proxy *encryption* from proxy *re*-encryption by whether Charlie must separately obtain and maintain this inner secret key or can decrypt using only his own key. Ateniese et al. (AFGH06) [AFGH06, Section 3.1] explicitly describe an adaptation applicable to ID03: encrypt sk^2 under Charlie's public key and deliver this encrypted key alongside the inner ciphertext. Charlie uses his own long-term secret key to recover sk^2, then uses sk^2 to decrypt the second layer. Thus, his own long-term secret key is the only private input he needs to supply. We adopt this structure and use the term PRE throughout.

The original ID03 role-separated CCA guarantee differs from PRE-CCA security. Shao and Cao [SC09, Appendix A] show a rewrapping attack under their PRE-CCA interface. We identify a different limitation of this AFGH06 adaptation of ID03 when the delegator reuses the same inner key across recipients: a recipient can recover the inner key from an ordinary authorised delivery and then decrypt a bare inner ciphertext sent to another, honest recipient. Proposition 3.4 formalises this attack in our rHRA experiment without any rekey or decryption queries. This targets the specified nested-encryption baseline, not all constructions in AFGH06.

1.2 Our contributions

We make four contributions.

1. *Restricted chosen-ciphertext security (rCCA2).* Our main result, Theorem 4.4, gives a generic single-hop PRE scheme that allows re-encryption and decryption queries on adversarial ciphertexts before and after the challenge, subject to the per-delegator exposure restriction. It uses CCA2-secure PKE, strongly unforgeable signatures and deterministic authenticated encryption secure under repeated headers.
2. *A recipient-side attack and a generic repair.* We show that the nested-encryption baseline, when combined with encrypted delivery of the inner secret key and a bare inner ciphertext, fails rHRA even without exposed rekeys (Proposition 3.4). Encrypting the inner ciphertext under an independent receiving public key gives a three-key, single-hop PRE construction with correctness under negligible component decryption error (Proposition 3.1).
3. *Security implications and separations.* We prove that full CCA2 security with the CH07 derivative rule and fixed rekeys implies full HRA security for any constant hop bound, including randomised re-encryption (Theorem 2.12). This addresses the constant-hop case of Cohen's question [Coh19, Remark 3]. We also prove rHRA security of our nested construction from PKE-CPA security (Theorem 3.2), establish the strict hierarchy between full HRA, rHRA, and PRE-CPA (Corollary 2.13), and give a full-HRA attack on our construction.
4. *ML-KEM deployment and an OpenPGP application.* We instantiate rCCA2 using HPKE with ML-KEM, ML-DSA signatures, and AES-SIV, and calculate full and cached ciphertext sizes (Section 5.2). A packet-level prototype tests ordinary decryption by Bob and forwarding to Charlie using a companion PRE envelope. We measure proxy throughput with one, two and four workers (Section 5.4).

Organisation. Section 2 defines the PRE security models and gives their implications and separations. Section 3 presents the nested construction, proves rHRA security, and gives the attacks used for the separations. Section 4 presents the authenticated construction and proves its rCCA2 security. Section 5 instantiates rCCA2 using HPKE with ML-KEM, ML-DSA signatures, and AES-SIV, compares its sizes with the rHRA baseline and other schemes, and reports the OpenPGP forwarding experiment. Appendix A proves the constant-hop CCA2-to-HRA implication, and Appendix B analyses noise budgets for direct key switching.

2 Security models and their relations

The security parameter is supplied in unary as 1^λ. All user and query bounds are polynomial in λ, and $\perp$ denotes rejection. For an encoded message, key, or ciphertext x, write $|x|$ for its bit length. Scheme algorithms and adversaries are classical probabilistic polynomial-time (PPT) algorithms. For a predicate P, the indicator $\mathbf{1}[P]$ equals 1 if P is true and 0 otherwise.

2.1 PRE and component PKE

We use a PRE interface following AFGH06 [AFGH06, Definition 2.2 and Section 3.1]. We write a ciphertext as (h, ct), where ct is the ciphertext payload and h is its hop count; the public

parameters specify a hop bound T. We call ciphertexts with $h = 0$ *source ciphertexts* and those with $h > 0$ *re-encrypted ciphertexts*.

The proxy runs ReEnc and has access to the delegator's and recipient's public keys. We pass these keys explicitly; they could instead be included in the rekey, so this interface adds no restriction to the usual PRE model.

Definition 2.1. *A PRE scheme consists of the following algorithms.*

1. $\mathsf{Setup}(1^\lambda) \to \mathsf{pp}$ *generates public parameters, including a hop bound* T.
2. $\mathsf{KeyGen}(\mathsf{pp}) \to (pk_i, sk_i)$ *generates a user's key pair.*
3. $\mathsf{Enc}(\mathsf{pp}, pk_i, m) \to (0, ct_i)$ *encrypts* m *to user* i*, where* 0 *is the initial hop count.*
4. $\mathsf{ReKeyGen}(\mathsf{pp}, sk_i, pk_j) \to rk_{i\to j}$ *creates a delegation using the delegator's secret and the delegatee's public key.*
5. $\mathsf{ReEnc}(\mathsf{pp}, rk_{i\to j}, (h, ct_i), pk_i, pk_j) \to (h+1, ct_j)$ *or* $\perp$ *re-encrypts a ciphertext. It rejects if* $h \geq T$.
6. $\mathsf{Dec}(\mathsf{pp}, sk_i, (h, ct)) \to m$ *or* $\perp$ *decrypts a source or re-encrypted ciphertext using the recipient's own secret key.*

Decryption is deterministic. We omit pp *when clear. A single-hop scheme has* $T = 1$.

Definition 2.2 (Correctness). *Generate* $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$. *For any valid message* m *and sequence of users* $u_0, \ldots, u_h$*, where* $0 \leq h \leq T$*, generate* $(pk_u, sk_u) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$ *for each distinct user* u *in the sequence. For each distinct ordered pair* (i, j) *occurring in the sequence, sample* $rk_{i\to j} \leftarrow \mathsf{ReKeyGen}(\mathsf{pp}, sk_i, pk_j)$ *once and reuse it whenever that pair occurs again. Let* ct_r *denote the ciphertext payload after* r *hops. Generate* $(0, ct_0) \leftarrow \mathsf{Enc}(\mathsf{pp}, pk_{u_0}, m)$*, and, for each* $r = 1, \ldots, h$*, compute*

$$(r, ct_r) \leftarrow \mathsf{ReEnc}(\mathsf{pp}, rk_{u_{r-1}\to u_r}, (r-1, ct_{r-1}), pk_{u_{r-1}}, pk_{u_r}),$$

using fresh randomness in each call if ReEnc *is randomised. Correctness requires that all calls succeed and*

$$\mathsf{Dec}(\mathsf{pp}, sk_{u_r}, (r, ct_r)) = m \qquad \text{for every } 0 \leq r \leq h,$$

except with negligible probability.

Public-key encryption. PKE can be seen as the zero-hop restriction of this interface: omit $\mathsf{ReKeyGen}$, ReEnc, and the hop count. We write $\mathcal{E} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$, with $\mathsf{KeyGen}(1^\lambda) \to (pk, sk)$, $\mathsf{Enc}(pk, m) \to ct$, and $\mathsf{Dec}(sk, ct) \to m$ or $\perp$. Its correctness is Definition 2.2 with $h = 0$: for every valid message m, honestly generated keys and $ct \leftarrow \mathsf{Enc}(pk, m)$ satisfy $\mathsf{Dec}(sk, ct) = m$, except with negligible probability.

Definition 2.3 (PKE-CPA experiment and security). *For* $\mathcal{E}$*, generate* $(pk, sk) \leftarrow \mathsf{KeyGen}(1^\lambda)$ *and give* $(1^\lambda, pk)$ *to* $\mathcal{A}$*. On the first valid challenge query* (m_0, m_1)*, sample a uniform bit* b *and return* $ct^\star \leftarrow \mathsf{Enc}(pk, m_b)$*. Reject queries with invalid or unequal-length messages and queries after a valid challenge. The adversary outputs* b'*, and the experiment returns* $\mathbf{1}[b' = b]$*. If no valid challenge is made, the experiment returns a uniform bit. Encryption is public, and there is no decryption oracle.*

Denote this experiment by $\mathsf{Exp}^{\mathsf{cpa}}_{\mathcal{E},\mathcal{A}}(1^\lambda)$ *and define* $\mathsf{Adv}^{\mathsf{cpa}}_{\mathcal{E},\mathcal{A}}(1^\lambda) = \left|\Pr[\mathsf{Exp}^{\mathsf{cpa}}_{\mathcal{E},\mathcal{A}}(1^\lambda) = 1] - 1/2\right|$. *The scheme is PKE-CPA-secure if this advantage is negligible in* λ *for every PPT adversary* $\mathcal{A}$.

This is the single-user, single-challenge notion used for the PKE in Section 3.

Definition 2.4 (PKE-CCA2 experiment and security). *Extend Definition 2.3 by adding a decryption oracle. Before the challenge, a query* ct *returns* $\mathsf{Dec}(sk, ct)$*. After the challenge* $ct^\star$ *is generated, the oracle returns* $\perp$ *on* $ct = ct^\star$ *and* $\mathsf{Dec}(sk, ct)$ *otherwise. The adversary may*

make at most q_d adaptive decryption queries in total, before or after the challenge. All other rules, including the equal-length requirement and the treatment of an invalid challenge, remain unchanged.

Denote this experiment by $\mathsf{Exp}^{\mathsf{cca2}}_{\mathcal{E},\mathcal{A}}(1^\lambda)$ *and define* $\mathsf{Adv}^{\mathsf{cca2}}_{\mathcal{E},\mathcal{A}}(1^\lambda; q_d) = \left|\Pr[\mathsf{Exp}^{\mathsf{cca2}}_{\mathcal{E},\mathcal{A}}(1^\lambda) = 1] - 1/2\right|$. *The scheme is PKE-CCA2-secure if this advantage is negligible in λ for every PPT adversary $\mathcal{A}$ and polynomial query bound q_d.*

2.2 Full HRA and PRE-CPA security

We recall Cohen's HRA game [Coh19, Definitions 5–6], which we call full HRA to distinguish it from the restricted model below. The adversary can obtain rekeys and honestly generated re-encryptions in the same experiment, including both outgoing rekeys and ordinary re-encryptions to corrupted recipients for the same honest delegator.

All PRE oracles reject unregistered user identifiers, invalid plaintexts, and malformed inputs with $\perp$ before looking up records or running cryptographic operations. We leave these routine checks implicit in the pseudocode.

Definition 2.5 (HRA experiment and security). *Let $\mathcal{A}$ be a PPT adversary. Figure 1 specifies the experiment.*

Denote this experiment by $\mathsf{Exp}^{\mathsf{hra}}_{\Pi,\mathcal{A}}(1^\lambda)$ *and define* $\mathsf{Adv}^{\mathsf{hra}}_{\Pi,\mathcal{A}}(1^\lambda) = \left|\Pr[\mathsf{Exp}^{\mathsf{hra}}_{\Pi,\mathcal{A}}(1^\lambda) = 1] - 1/2\right|$. *The scheme Π is HRA-secure if this advantage is negligible in λ for every PPT adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$.*

PRE-CPA: plaintext confidentiality against the proxy.

Definition 2.6 (PRE-CPA experiment and security). *Remove the* encrypt *and* reenc *oracles from the experiment in Definition 2.5. All other rules remain unchanged; in particular, $\mathcal{A}_2$ keeps access to* rekey *and* challenge.

Denote this experiment by $\mathsf{Exp}^{\mathsf{cpa}}_{\Pi,\mathcal{A}}(1^\lambda)$ *and define* $\mathsf{Adv}^{\mathsf{cpa}}_{\Pi,\mathcal{A}}(1^\lambda) = \left|\Pr[\mathsf{Exp}^{\mathsf{cpa}}_{\Pi,\mathcal{A}}(1^\lambda) = 1] - 1/2\right|$. *The scheme Π is PRE-CPA-secure if this advantage is negligible in λ for every PPT adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$.*

PRE-CPA security protects plaintexts from an adversary holding exposed rekeys (against the proxy). The HRA experiment additionally supplies ordinary honest-to-corrupted re-encryption outputs (against the recipient). Those outputs may reveal secret information that helps a corrupted recipient decrypt other messages, so PRE-CPA security alone does not imply HRA security [Coh19].

2.3 Restricted HRA security

The two sets below record the per-delegator exposures described in Section 1. Recording one of these exposures prevents the other for the same delegator.[1]

Definition 2.7 (Restricted HRA experiment and security). *The experiment* $\mathsf{Exp}^{\mathsf{rhra}}_{\Pi,\mathcal{A}}(1^\lambda)$ *is obtained from Definition 2.5 by initialising two additional sets* $\mathsf{Exposed} = \mathsf{Converted} = \varnothing$ *and modifying the following oracles:*

1. $\mathsf{rekey}(i, j)$*: if* $i \in \mathsf{Hon}$*,* $j \in \mathsf{Hon}$*, and* $i \in \mathsf{Converted}$*, return* $\perp$*. Otherwise apply the original rules; if the query returns a rekey and* $i \in \mathsf{Hon}$*, add* i *to* Exposed *before returning it.*
2. $\mathsf{reenc}(i, j, k)$*: if* $i \in \mathsf{Hon}$*,* $j \in \mathsf{Cor}$*, and* $i \in \mathsf{Exposed}$*, return* $\perp$*. Otherwise apply the original rules; if the query returns a ciphertext record,* $i \in \mathsf{Hon}$*, and* $j \in \mathsf{Cor}$*, add* i *to* Converted *before returning it.*

[1] For the separate notion of collusion safety, see Yin et al. [YZL+26].

$\underline{\mathsf{Exp}^{\mathsf{hra}}_{\Pi,\mathcal{A}}(1^\lambda)}$
$\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$; $\mathsf{numKeys}, \mathsf{numCt} \leftarrow 0$
$\mathsf{Hon}, \mathsf{Cor}, \mathsf{Deriv} \leftarrow \varnothing$; $\mathsf{Ct}, \mathsf{RK} \leftarrow \emptyset$; $k^\star \leftarrow \bot$
$\mathsf{state} \leftarrow \mathcal{A}_1^{\mathsf{honkeygen},\mathsf{corkeygen}}(\mathsf{pp})$
$\mathsf{Users} \leftarrow \mathsf{Hon} \cup \mathsf{Cor}$
for each $(i,j) \in \mathsf{Users}^2$ **do** $\mathsf{RK}[(i,j)] \leftarrow \mathsf{ReKeyGen}(\mathsf{pp}, sk_i, pk_j)$
$b' \leftarrow \mathcal{A}_2^{\mathsf{rekey},\mathsf{encrypt},\mathsf{challenge},\mathsf{reenc}}(\mathsf{pp}, \mathsf{state})$
if $k^\star = \bot$ **then return** a uniform bit
return $\mathbf{1}[b' = b]$

$\underline{\mathsf{honkeygen}()}$
$\mathsf{numKeys} \leftarrow \mathsf{numKeys} + 1$
$i \leftarrow \mathsf{numKeys}$
$(pk_i, sk_i) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$
$\mathsf{Hon} \leftarrow \mathsf{Hon} \cup \{i\}$
return (i, pk_i)

$\underline{\mathsf{corkeygen}()}$
$\mathsf{numKeys} \leftarrow \mathsf{numKeys} + 1$
$i \leftarrow \mathsf{numKeys}$
$(pk_i, sk_i) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$
$\mathsf{Cor} \leftarrow \mathsf{Cor} \cup \{i\}$
return (i, pk_i, sk_i)

$\underline{\mathsf{encrypt}(i,m)}$
$(0, ct) \leftarrow \mathsf{Enc}(\mathsf{pp}, pk_i, m)$
$\mathsf{numCt} \leftarrow \mathsf{numCt} + 1$
$k \leftarrow \mathsf{numCt}$
$\mathsf{Ct}[(i,k)] \leftarrow (0, ct)$
return $(k, (0, ct))$

$\underline{\mathsf{rekey}(i,j)}$
if $i = j$ **then return** $\bot$
if $i \in \mathsf{Hon} \wedge j \in \mathsf{Cor}$ **then return** $\bot$
return $\mathsf{RK}[(i,j)]$

$\underline{\mathsf{challenge}(i, m_0, m_1)}$
if $i \notin \mathsf{Hon} \vee k^\star \neq \bot$ **then return** $\bot$
if $|m_0| \neq |m_1|$ **then return** $\bot$
$b \leftarrow \{0,1\}$ uniformly
$(0, ct^\star) \leftarrow \mathsf{Enc}(\mathsf{pp}, pk_i, m_b)$
$\mathsf{numCt} \leftarrow \mathsf{numCt} + 1$
$k^\star \leftarrow \mathsf{numCt}$
$\mathsf{Ct}[(i,k^\star)] \leftarrow (0, ct^\star)$
$\mathsf{Deriv} \leftarrow \mathsf{Deriv} \cup \{k^\star\}$
return $(k^\star, (0, ct^\star))$

$\underline{\mathsf{reenc}(i,j,k)}$
if $\mathsf{Ct}[(i,k)]$ is undefined **then return** $\bot$
if $j \in \mathsf{Cor} \wedge k \in \mathsf{Deriv}$ **then return** $\bot$
$(h, ct) \leftarrow \mathsf{Ct}[(i,k)]$
$(h+1, ct') \leftarrow \mathsf{ReEnc}(\mathsf{pp}, \mathsf{RK}[(i,j)], (h, ct), pk_i, pk_j)$
if the call returns $\bot$ **then return** $\bot$
$\mathsf{numCt} \leftarrow \mathsf{numCt} + 1$; $k' \leftarrow \mathsf{numCt}$
$\mathsf{Ct}[(j,k')] \leftarrow (h+1, ct')$
if $k \in \mathsf{Deriv}$ **then** $\mathsf{Deriv} \leftarrow \mathsf{Deriv} \cup \{k'\}$
return $(k', (h+1, ct'))$

Figure 1. HRA experiment and its oracles.

All other rules remain unchanged. Define $\mathsf{Adv}^{\mathsf{rhra}}_{\Pi,\mathcal{A}}(1^\lambda) = \left|\Pr[\mathsf{Exp}^{\mathsf{rhra}}_{\Pi,\mathcal{A}}(1^\lambda) = 1] - 1/2\right|$. *A PRE scheme* Π *is rHRA-secure if this advantage is negligible in* λ *for every PPT adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$.

2.4 Full chosen-ciphertext security

We use the challenge-derivative rule of Canetti and Hohenberger [CH07, Definition 2.4], with Cohen's unidirectional formulation and fixed rekeys [Coh19, full version, Appendix B.1]. We denote this full CCA2 notion by $\mathsf{CCA2}_{\mathsf{CH}}$. The adversary may obtain both an outgoing rekey and re-encryptions to corrupted recipients for the same honest delegator.

Definition 2.8 (CCA2 challenge derivatives). *Here ct denotes a complete ciphertext, including its hop count. Let* $\mathcal{L}$ *record the ordered pairs* (i,j) *for which* $\mathsf{rekey}(i,j)$ *has returned a rekey. Before the challenge, set* $\mathcal{D}_{\mathsf{cca}} = \varnothing$. *After the challenge* $(i^\star, ct^\star)$ *on messages* (m_0, m_1), *define* $\mathcal{D}_{\mathsf{cca}}$ *to be the smallest set closed under the following rules:*

1. *Include* $(i^\star, ct^\star)$.
2. *If a successful query* $\mathsf{reenc}(i,j,ct)$ *returned* ct' *and* $(i, ct) \in \mathcal{D}_{\mathsf{cca}}$, *include* (j, ct').

3. If $(i, ct) \in \mathcal{D}_{\mathsf{cca}}$ *and* $(i, j) \in \mathcal{L}$*, include every* (j, ct') *satisfying* $\mathsf{Dec}(sk_j, ct') \in \{m_0, m_1\}$.

The third rule covers re-encryptions that the adversary can compute using exposed rekeys. It also excludes independently generated ciphertexts that decrypt to either challenge message. For example, after a rekey from the challenge user $i^\star$ to j is exposed, an independent encryption of m_0 under pk_j belongs to $\mathcal{D}_{\mathsf{cca}}$. HRA still permits re-encrypting its ordinary ciphertext record to a corrupted recipient. This difference prevents a direct simulation that simply forwards every HRA query to the CCA2 challenger.

Definition 2.9 (Full CCA2 experiment and security). *Use the registration, cached rekeys, and* rekey *rules of Definition 2.5. Initialise* $\mathcal{L} = \varnothing$ *and record successful rekey queries in it. The* encrypt *and* challenge *oracles use the same validity and encryption rules but return complete ciphertexts without record indices. The first valid challenge fixes* $(i^\star, ct^\star, m_0, m_1)$*; subsequent challenge queries are rejected. Maintain* $\mathcal{D}_{\mathsf{cca}}$ *as in Definition 2.8. Replace* reenc *and add* dec *as follows:*

$\mathsf{reenc}(i, j, ct)$**.**
Accept an arbitrary complete ciphertext ct*. Reject if either user is unregistered, or if* $j \in \mathsf{Cor}$ *and* $(i, ct) \in \mathcal{D}_{\mathsf{cca}}$*. Otherwise compute* $ct' \leftarrow \mathsf{ReEnc}(\mathsf{RK}[(i, j)], ct, pk_i, pk_j)$*. If the call fails, return* $\perp$*. Otherwise record the input and output for the second rule of Definition 2.8, update* $\mathcal{D}_{\mathsf{cca}}$*, and return* ct'*.*

$\mathsf{dec}(i, ct)$**.**
Reject if i *is unregistered or* $(i, ct) \in \mathcal{D}_{\mathsf{cca}}$*. Otherwise return* $\mathsf{Dec}(sk_i, ct)$.

All other rules, including runs without a valid challenge, are as in Definition 2.5. Denote the experiment by $\mathsf{Exp}^{\mathsf{cca2}}_{\Pi,\mathcal{A}}(1^\lambda)$ *and define* $\mathsf{Adv}^{\mathsf{cca2}}_{\Pi,\mathcal{A}}(1^\lambda) = |\Pr[\mathsf{Exp}^{\mathsf{cca2}}_{\Pi,\mathcal{A}}(1^\lambda) = 1] - 1/2|$*. The scheme is CCA2-secure if this advantage is negligible for every PPT adversary.*

2.5 Restricted chosen-ciphertext security

We define rCCA2 for single-hop schemes ($T = 1$) with deterministic ReEnc. Starting from full CCA2 in Definition 2.9, we impose the rHRA restriction: for each honest delegator, the adversary may obtain outgoing rekeys or re-encryptions to corrupted recipients, but not both. Here r stands for restricted.[2]

We also replace the challenge-derivative rule with simple exact ciphertext exclusions, following Gaddam et al. [GLSW21, Section 4.2.2, Definition 6]. Single-hop, deterministic re-encryption lets the challenger compute every challenge re-encryption available through an exposed rekey. The exclusion set contains these outputs, the challenge itself, and its re-encryptions returned by the oracle.

We assume that accidental ciphertext collisions have negligible total probability and include that probability in the security loss.

Definition 2.10 (Challenge exclusion set). *Let* $T = 1$ *and let* ReEnc *be deterministic for fixed inputs. The set* $\mathcal{D}$ *contains pairs* (u, ct) *excluded from decryption.*

Before the challenge, set $\mathcal{D} = \varnothing$*. After the challenge* $(i^\star, ct^\star)$*, include exactly the following pairs:*

1. $(i^\star, ct^\star)$.
2. (j, ct') *for every successful query* $\mathsf{reenc}(i^\star, j, ct^\star)$ *made after the challenge with output* ct'.
3. (j, ct') *for every cached rekey* $rk_{i^\star \to j}$ *returned to the adversary, where*

$$ct' = \mathsf{ReEnc}(\mathsf{RK}[(i^\star, j)], ct^\star, pk_{i^\star}, pk_j) \neq \perp.$$

[2] The r does not refer to replayable CCA [CKN03].

The third item includes rekeys returned before the challenge. Update $\mathcal{D}$ when the challenge is generated and after each successful reenc *or* rekey *query, before the next oracle query.*

Since $T = 1$, every challenge re-encryption has hop count one. We denote by $\mathsf{CCA2}_{\mathsf{exact}}$ the full experiment of Definition 2.9 for single-hop deterministic ReEnc, with dec and reenc both using $\mathcal{D}$ from Definition 2.10. It permits joint exposure for the same delegator and uses the winning rule and advantage of Definition 2.9.

Definition 2.11 (Single-hop restricted chosen-ciphertext security). *Let $T = 1$ and let* ReEnc *be deterministic for fixed inputs. Use the user registration, cached rekeys, and* rekey *rules of Definition 2.7, and initialise* $\mathsf{Exposed} = \mathsf{Converted} = \varnothing$. *Initially set $i^\star = \perp$; the first valid challenge sets $(i^\star, ct^\star)$, and later challenge queries are rejected. Maintain $\mathcal{D}$ as in Definition 2.10.*

Replace reenc *and add* dec *as follows.*

$\mathsf{reenc}(i, j, ct)$.

The adversary may submit any ciphertext ct. Reject if i or j is unregistered, or if $j \in \mathsf{Cor}$ and either $(i, ct) \in \mathcal{D}$ or $i \in \mathsf{Exposed}$. Otherwise compute $ct' \leftarrow \mathsf{ReEnc}(\mathsf{RK}[(i, j)], ct, pk_i, pk_j)$. Return $\perp$ if the call fails. If $i \in \mathsf{Hon}$ and $j \in \mathsf{Cor}$, add i to Converted. *Update $\mathcal{D}$ according to Definition 2.10 and return ct'.*

$\mathsf{dec}(u, ct)$.

Reject if u is unregistered, $u \notin \mathsf{Hon}$, or $(u, ct) \in \mathcal{D}$. Otherwise return $\mathsf{Dec}(sk_u, ct)$.

The winning rule, including runs without a valid challenge, is the same as in Definition 2.5. Define $\mathsf{Adv}^{\mathsf{rcca2}}_{\Pi,\mathcal{A}}(1^\lambda) = |\Pr[b' = b] - 1/2|$. The scheme is rCCA2-secure if this advantage is negligible for every PPT adversary.

2.6 Relations among the security notions

Cohen leaves open whether CCA security implies HRA security [Coh19]. Gaddam et al. [GLSW21, Section 4.2.2 and Appendix A] discuss the same obstacle and use exact re-encryption outputs to define derivatives for deterministic ReEnc. Yin et al. [YZL+25] gave an earlier argument for this implication.

Theorem 2.12 (CCA2 implies HRA for constant hop bounds). *Let Π have hop bound $T = O(1)$ and the fixed rekeys of Definition 2.9. Assume correctness holds for polynomially many honest encryptions and re-encryptions. If Π is CCA2-secure in Definition 2.9, then it is HRA-secure in Definition 2.5. This holds for deterministic and randomised* ReEnc.

The reduction delays its CCA2 challenge and prepares enough ordinary encryptions and re-encryptions of m_0, m_1 to answer later HRA queries that CCA2 would reject. For *constant* T, this takes polynomially many calls. Appendix A gives the proof. The theorem answers the constant-hop case of Cohen's question; it makes no claim for growing T.

Corollary 2.13 (Security implications and separations). *Assume a PKE-CPA-secure PKE exists satisfying the encoding and correctness requirements of Section 3.1. The classes of secure schemes satisfy*

$$\mathsf{CCA2}_{\mathsf{CH}} \subsetneq \mathsf{HRA} \subsetneq \mathsf{rHRA} \subsetneq \mathsf{PRE\text{-}CPA},$$
$$\mathsf{CCA2}_{\mathsf{exact}} \subsetneq \mathsf{rCCA2} \subsetneq \mathsf{rHRA}.$$

The first inclusion uses the hop bound and correctness assumption of Theorem 2.12. Its strictness assumes the existence of a full-HRA-secure scheme with those properties. The second chain concerns single-hop schemes with deterministic ReEnc *and uses the assumptions of Theorem 4.4.*

Proof. Theorem 2.12 gives $\mathsf{CCA2}_{\mathsf{CH}} \subseteq \mathsf{HRA}$. The HRA-to-rHRA and rHRA-to-PRE-CPA inclusions follow by restricting queries. For rCCA2-to-rHRA, maintain the HRA record table

and submit the recorded ciphertexts to reenc, with negligible loss from accidental collisions as in Section 2.5. To simulate rCCA2 in the $\mathsf{CCA2}_{\mathsf{exact}}$ game, enforce the per-delegator exposure restriction locally, reject decryption queries for corrupted recipients, and forward all other allowed queries. Both games exclude the same pairs in $\mathcal{D}$.

The HRA/rHRA separation follows from Theorem 3.2 and Proposition 3.5. For the rHRA/PRE-CPA separation, consider the variant Π' of Proposition 3.4. Only ReEnc and Dec differ from the construction of Corollary 3.3. The PRE-CPA experiment calls neither algorithm, so the two schemes give the adversary identically distributed views. Thus Π' is PRE-CPA-secure, and Proposition 3.4 shows that it fails rHRA.

For the separation between $\mathsf{CCA2}_{\mathsf{CH}}$ and HRA, append a zero bit to every ciphertext output of the assumed HRA-secure scheme, and have Dec and ReEnc ignore it on input. This preserves HRA security because its oracle only re-encrypts recorded ciphertexts. Applying the same modification to the construction of Theorem 4.4 preserves rHRA security and deterministic re-encryption, giving the rCCA2/rHRA separation.

Finally, Theorem 4.4 gives rCCA2 security for the authenticated construction. In the $\mathsf{CCA2}_{\mathsf{exact}}$ game, the ordinary delivery and exposed rekey in Proposition 3.5 reveal the same delegator's inner and outer secret keys, proving strictness. □

3 Revisiting the ID03–AFGH06 Framework

We revisit the nested-encryption construction of Ivan and Dodis [ID03] and the encrypted-secret delivery method described by Ateniese et al. [AFGH06, Section 3.1]. Alice encrypts a message for Bob under two nested keys. Bob gives the proxy the outer secret key and supplies the inner secret key encrypted to Charlie. The proxy removes Bob's outer layer, encrypts the inner ciphertext to Charlie, and sends it together with the encrypted inner secret key. Charlie opens these two ciphertexts using his own secret key and then recovers the message. The proxy never receives the inner secret key in the clear.

3.1 Construction

Underlying encryption. Let $\mathcal{E} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ be PKE-CPA-secure with negligible probability of decryption failure, and support encryption of messages, inner ciphertexts, and inner secret keys.

Keys and ciphertexts. Each user i generates three independent key pairs (pk_i^a, sk_i^a), for $a \in \{0, 1, 2\}$. Their roles are:

- *Receiving key* (pk_i^0, sk_i^0): for receiving re-encrypted inner ciphertexts and encrypted inner secret keys from delegators.
- *Outer key* (pk_i^1, sk_i^1): protects the outer layer of source ciphertexts for i; sk_i^1 is included in outgoing rekeys.
- *Inner key* (pk_i^2, sk_i^2): protects the inner layer of source ciphertexts for i; an authorised recipient obtains sk_i^2 through an encryption under the recipient's receiving public key.

Write $pk_i = (pk_i^0, pk_i^1, pk_i^2)$ and $sk_i = (sk_i^0, sk_i^1, sk_i^2)$. The subscripts identify users in the notation, and superscripts identify the three key roles. We write $ct_{i \to j}^{\text{key}}$ for the encryption of sk_i^2 generated for delegation (i, j). The unindexed symbol ct^{key} denotes the encrypted-key component parsed from an input ciphertext.

Algorithms. Figure 2 gives the six algorithms of Definition 2.1 with $T = 1$. Any decoding or component decryption failure returns $\perp$; these checks are implicit in the pseudocode.

$\underline{\mathsf{Setup}(1^\lambda)}$

$1:$ Select $\mathcal{E}$ and the encodings; set $T = 1$.
$2:$ **return** pp containing these parameters

$\underline{\mathsf{KeyGen}(\mathsf{pp})}$

$1:$ **for** $a \in \{0,1,2\}$ **do**
$2:$ $\quad (pk_i^a, sk_i^a) \leftarrow \mathcal{E}.\mathsf{KeyGen}(1^\lambda)$
$3:$ $pk_i \leftarrow (pk_i^0, pk_i^1, pk_i^2)$
$4:$ $sk_i \leftarrow (sk_i^0, sk_i^1, sk_i^2)$
$5:$ **return** (pk_i, sk_i)

$\underline{\mathsf{ReKeyGen}(sk_i, pk_j)}$

$1:$ $ct_{i\to j}^{\mathrm{key}} \leftarrow \mathcal{E}.\mathsf{Enc}(pk_j^0, sk_i^2)$
$2:$ **return** $rk_{i\to j} = (sk_i^1, ct_{i\to j}^{\mathrm{key}})$

$\underline{\mathsf{ReEnc}(rk_{i\to j}, ct_i, pk_i, pk_j)}$

$1:$ $(sk_i^1, ct_{i\to j}^{\mathrm{key}}) \leftarrow rk_{i\to j}$
$2:$ **if** ct_i does not match $(0, ct_i^1)$ **then return** $\perp$
$3:$ $ct_i^2 \leftarrow \mathcal{E}.\mathsf{Dec}(sk_i^1, ct_i^1)$
$4:$ $ct_{i\to j}^0 \leftarrow \mathcal{E}.\mathsf{Enc}(pk_j^0, ct_i^2)$
$5:$ **return** $(1, ct_{i\to j}^0, ct_{i\to j}^{\mathrm{key}})$

$\underline{\mathsf{Enc}(pk_i, m)}$

$1:$ $ct_i^2 \leftarrow \mathcal{E}.\mathsf{Enc}(pk_i^2, m)$
$2:$ $ct_i^1 \leftarrow \mathcal{E}.\mathsf{Enc}(pk_i^1, ct_i^2)$
$3:$ **return** $(0, ct_i^1)$

$\underline{\mathsf{Dec}(sk_u, ct)}$

$1:$ **if** $ct = (0, ct^1)$ **then**
$2:$ $\quad ct^2 \leftarrow \mathcal{E}.\mathsf{Dec}(sk_u^1, ct^1)$
$3:$ $\quad$ **return** $\mathcal{E}.\mathsf{Dec}(sk_u^2, ct^2)$
$4:$ **else if** $ct = (1, ct^0, ct^{\mathrm{key}})$ **then**
$5:$ $\quad sk^{2\prime} \leftarrow \mathcal{E}.\mathsf{Dec}(sk_u^0, ct^{\mathrm{key}})$
$6:$ $\quad ct' \leftarrow \mathcal{E}.\mathsf{Dec}(sk_u^0, ct^0)$
$7:$ $\quad$ **return** $\mathcal{E}.\mathsf{Dec}(sk^{2\prime}, ct')$
$8:$ **else return** $\perp$

Figure 2. Generic single-hop PRE construction.

3.2 Correctness

Proposition 3.1 (Correctness of the single-hop formulation). *Under the correctness assumption in Section 3.1, the construction satisfies single-hop correctness.*

This follows by undoing the encryption layers in order and applying a union bound to the negligible failure probabilities of the decryptions.

3.3 rHRA security

We prove rHRA security in Definition 2.7. The proof uses PKE-CPA security of the underlying PKE $\mathcal{E}$.

Let N bound the number of registered users. Let Q_{key} bound the number of distinct encrypted inner secret keys addressed to honest recipients that appear in returned rekeys or re-encryption outputs, counting each ordered pair only once; in particular, $Q_{\mathrm{key}} \le N^2$. Let Q_{HH} bound the number of successful honest-to-honest re-encryption queries. Following Definition 2.3, let $\epsilon = \epsilon(\lambda)$ satisfy $\mathsf{Adv}^{\mathsf{cpa}}_{\mathcal{E},\mathcal{B}}(1^\lambda) \le \epsilon$ for each PPT reduction $\mathcal{B}$ used below.

Let δ_{dec} bound the probability of at least one outer decryption error in the real experiment.

Theorem 3.2 (Direct rHRA security). *For every PPT adversary $\mathcal{A}$ in Definition 2.7, the construction in Section 3.1 satisfies*

$$\mathsf{Adv}^{\mathsf{rhra}}_{\Pi,\mathcal{A}}(1^\lambda) \le \delta_{\mathrm{dec}} + 2N(Q_{\mathrm{key}} + Q_{\mathrm{HH}} + 1)\epsilon. \tag{1}$$

Consequently, it is rHRA-secure when $\mathcal{E}$ is PKE-CPA-secure and δ_{dec} is negligible.

Proof. Write G_t^b for the adversary-visible transcript and final guess in game t with challenge bit fixed to $b \in \{0,1\}$. Let $p_t^b = \Pr[G_t^b \text{ outputs } 1]$. Game G_0^b is the real experiment, so its advantage is $\mathsf{Adv}^{\mathsf{rhra}}_{\Pi,\mathcal{A}}(1^\lambda) = |p_0^1 - p_0^0|/2$. When δ_{dec} is negligible, the bounds below give

$$G_0^b \approx_s G_1^b \approx_c G_2^b \approx_c G_3^b.$$

Here $\approx_s$ denotes statistical indistinguishability, meaning that the statistical distance is negligible in λ. The notation $\approx_c$ denotes computational indistinguishability, meaning that no PPT distinguisher has non-negligible distinguishing advantage. The games respectively use real re-encryption, recorded inner ciphertexts, dummy encrypted inner secret keys, and dummy deliveries between honest users. We then compare G_3^0 and G_3^1.

Lazy generation of rekeys. The HRA experiment precomputes $\mathsf{RK}[(i,j)]$ for all registered users. We equivalently generate each entry when an allowed rekey or re-encryption query first needs it and cache it for later queries.

Game G_1^b: use recorded inner ciphertexts. When generating each ordinary encryption and the challenge, privately record the inner ciphertext ct^2 used to form $ct^1 \leftarrow \mathcal{E}.\mathsf{Enc}(pk_i^1, ct^2)$. In G_1^b, answer re-encryption queries by reading this recorded value instead of computing $\mathcal{E}.\mathsf{Dec}(sk_i^1, ct^1)$. This lets later reductions answer queries without the outer secret key.

Given an adversary, couple G_0^b and G_1^b using the same random coins for the challenger and the adversary. Until an outer decryption fails, $\mathcal{E}.\mathsf{Dec}(sk_i^1, ct^1) = ct^2$, so every response and subsequent state are identical in the two games. Their transcript and final-guess distributions therefore have statistical distance at most δ_{dec}, implying $|p_0^b - p_1^b| \le \delta_{\mathrm{dec}}$. When δ_{dec} is negligible, this gives $G_0^b \approx_s G_1^b$. With perfect correctness, the two games have identical distributions.

Game G_2^b: encrypt zeros instead of inner secret keys for honest recipients. Starting from G_1^b, replace $\mathcal{E}.\mathsf{Enc}(pk_j^0, sk_i^2)$ by $\mathcal{E}.\mathsf{Enc}(pk_j^0, 0^{|sk_i^2|})$ for each pair (i,j) with an honest recipient j. Use the replacement in both rekey and re-encryption responses, caching one encrypted key per pair.

Replace these encrypted keys one at a time, in the order in which they are first returned. For one replacement, a PKE-CPA reduction guesses the recipient j, uses its challenge public key as pk_j^0, and generates all other keys itself. It submits sk_i^2 and $0^{|sk_i^2|}$ as its challenge messages and uses the returned ciphertext as $ct_{i\to j}^{\mathrm{key}}$ for the selected delegation. It can compute all remaining responses using these keys and the records from G_1^b, without sk_j^0. PKE-CPA security and the guess of j bound each change in output probability by $2N\epsilon$. Summing over at most Q_{key} replacements gives $|p_1^b - p_2^b| \le 2NQ_{\mathrm{key}}\epsilon$, and hence $G_1^b \approx_c G_2^b$.

Game G_3^b: encrypt zeros instead of inner ciphertexts between honest users. Starting from G_2^b, replace $ct_{i\to j}^0 = \mathcal{E}.\mathsf{Enc}(pk_j^0, ct_i^2)$ by $\mathcal{E}.\mathsf{Enc}(pk_j^0, 0^{|ct_i^2|})$ in each successful re-encryption query between honest users. This applies to both ordinary ciphertexts and the challenge ciphertext.

Replace these output encryptions one at a time, in query order. Use the reduction for G_2^b, now submitting ct_i^2 and $0^{|ct_i^2|}$ as its PKE-CPA challenge messages and using the returned ciphertext as $ct_{i\to j}^0$. All other responses are computed in the same way. Each replacement changes the output probability by at most $2N\epsilon$, so $|p_2^b - p_3^b| \le 2NQ_{\mathrm{HH}}\epsilon$, and hence $G_2^b \approx_c G_3^b$.

Final games: compare G_3^0 and G_3^1. We now compare the games with challenge messages m_0 and m_1. In both games, re-encrypting the challenge to an honest recipient returns encryptions of zeros for both the inner ciphertext and the inner secret key. Re-encrypting the challenge to a corrupted recipient is forbidden. We therefore only need to change the source ciphertext returned by the challenge oracle, while still answering all other permitted queries.

Let $i^\star$ be the honest challenge delegator. Split runs with a valid challenge into two events: E, where $i^\star \in \mathsf{Exposed}$ at the end of the run, and U, where $i^\star \notin \mathsf{Exposed}$. Write $p_b^E = \Pr[G_3^b \text{ outputs } 1 \ \wedge E]$ and $p_b^U = \Pr[G_3^b \text{ outputs } 1 \ \wedge U]$. Hence

$$|p_3^1 - p_3^0| \le \left|p_1^E - p_0^E\right| + \left|p_1^U - p_0^U\right|.$$

We bound the two differences using separate reductions to PKE-CPA security. Each reduction guesses the challenge delegator's index in advance from $\{1, \ldots, N\}$.

Event E: use the PKE-CPA challenge for the inner encryption. Let $\hat{i}$ be the guessed challenge delegator. Use the PKE-CPA challenge public key as $pk^2_{\hat{i}}$, and generate the receiving and outer key pairs normally. If the PRE challenge query names another user, stop and output zero. Otherwise submit m_0 and m_1 as the PKE-CPA challenge messages. Use the returned ciphertext as the inner ciphertext and encrypt it under $pk^1_{\hat{i}}$. The reduction can store this inner ciphertext even though it does not know $sk^2_{\hat{i}}$. Encryptions of this user's inner secret key to honest recipients are already replaced by encryptions of zeros, and the reduction knows the outer secret key $sk^1_{\hat{i}}$ needed for outgoing rekeys. It can generate incoming rekeys and answer queries for other users using the receiving key pair and the other keys it generated. For ordinary encryptions to $\hat{i}$, it uses the public keys and records the inner ciphertexts it generates.

The reduction stops and outputs zero on any otherwise valid re-encryption query from $\hat{i}$ to a corrupted recipient: answering it would rule out E when $\hat{i} = i^\star$.

At the end, output one exactly when the challenge delegator is $\hat{i}$, event E occurs, and the adversary outputs one; otherwise output zero. Every early stop gives the same zero output as this rule. Guessing the challenge delegator adds a factor of at most N, giving $\left|p^E_1 - p^E_0\right| \le 2N\epsilon$.

Event U: use the PKE-CPA challenge for the outer encryption. Guess $\hat{i}$ as for event E and stop with output zero if the PRE challenge names another user. Use the PKE-CPA challenge public key as $pk^1_{\hat{i}}$ and generate all other keys locally. Independently compute $ct^2_a \leftarrow \mathcal{E}.\mathsf{Enc}(pk^2_{\hat{i}}, m_a)$ for $a \in \{0,1\}$, submit these equal-length ciphertexts as the PKE-CPA challenge messages, and use the returned ciphertext as the PRE challenge's outer ciphertext.

Answer permitted challenge re-encryptions with the encryptions of zeros specified in G^b_3. For ordinary re-encryptions from $\hat{i}$, the records from G^b_1 and the known inner secret key $sk^2_{\hat{i}}$ suffice to compute the G^b_3 responses, including deliveries to corrupted recipients. Incoming rekeys and queries for other users are handled as for event E.

The reduction stops and outputs zero on any otherwise valid outgoing rekey query for $\hat{i}$: answering it would rule out U when $\hat{i} = i^\star$. Use the same final-output rule as for event E, with U in place of E, giving $\left|p^U_1 - p^U_0\right| \le 2N\epsilon$.

Adding the bounds. Adding the bounds for E and U gives $|p^1_3 - p^0_3| \le 4N\epsilon$. Together with the earlier game comparisons, this gives

$$G^0_0 \approx_c G^0_3 \approx_c G^1_3 \approx_c G^1_0,$$

where each comparison between G^b_0 and G^b_3 goes through G^b_1 and G^b_2. Adding the probability differences by the triangle inequality gives

$$\begin{aligned}
\mathsf{Adv}^{\mathsf{rhra}}_{\Pi,\mathcal{A}}(1^\lambda) &= \frac{1}{2}|p^1_0 - p^0_0| \\
&\le \frac{1}{2}\sum_{b\in\{0,1\}}\sum_{t=0}^{2}|p^b_t - p^b_{t+1}| + \frac{1}{2}|p^1_3 - p^0_3| \\
&\le \delta_{\mathrm{dec}} + 2NQ_{\mathrm{key}}\epsilon + 2NQ_{\mathrm{HH}}\epsilon + 2N\epsilon.
\end{aligned}$$

This proves Equation (1). □

The receiving secret keys (sk^0) are independent of the outer and inner key pairs and are never encrypted by the construction, so the simulation does not require circular or key-dependent-message security.

Corollary 3.3 (PRE-CPA as a consequence). *Under the hypotheses of Theorem 3.2, the construction is PRE-CPA-secure. Using the same notation, the direct proof gives* $\mathsf{Adv}^{\mathsf{cpa}}_{\Pi,\mathcal{A}}(1^\lambda) \le 2N(Q_{\mathrm{key}} + 1)\epsilon$.

In a direct PKE-CPA proof, $Q_{\mathrm{HH}} = 0$ and $\delta_{\mathrm{dec}} = 0$, since there are no re-encryption oracle queries or challenger-side outer decryptions.

3.4 Protecting the inner ciphertext

AFGH06 [AFGH06, Section 3.1] describes encrypting the key needed for final decryption under the recipient's public key and sending it alongside the partially decrypted ciphertext. For the nested encryption of ID03 [ID03, Section 4.1], consider a variant Π' of our construction: after recovering ct_i^2, the proxy returns $(1, ct_i^2, ct_{i\to j}^{\text{key}})$ directly, omitting the encryption $ct_{i\to j}^0 \leftarrow \mathcal{E}.\mathsf{Enc}(pk_j^0, ct_i^2)$. Key generation, encryption, and rekey generation are unchanged. The recipient uses sk_j^0 to recover sk_i^2 from the encrypted key, then decrypts ct_i^2.

The problem is that sk_i^2 is the same for every recipient of user i. Once Charlie learns this key, he can decrypt any inner ciphertext for Bob that he later sees, including one re-encrypted to an honest user.

Proposition 3.4 (An rHRA attack without rekey queries). *Assume the component correctness condition in Section 3.1. Against the variant Π', an rHRA adversary making no rekey queries has advantage at least* $1/2 - \mathrm{negl}(\lambda)$.

Proof. In the experiment of Definition 2.7, register honest users Bob B and H, and a corrupted user Charlie C. The adversary knows sk_C, including the receiving secret key sk_C^0.

1. Query $\mathsf{encrypt}(B, m)$ for an ordinary message m, obtaining $(k, (0, ct_B^1))$, then query $\mathsf{reenc}(B, C, k)$. The returned ciphertext is $(1, ct_B^2, ct_{B\to C}^{\text{key}})$. Compute $\mathcal{E}.\mathsf{Dec}(sk_C^0, ct_{B\to C}^{\text{key}})$, which recovers Bob's inner secret key sk_B^2 except with negligible probability.
2. Choose distinct equal-length messages m_0, m_1 and query $\mathsf{challenge}(B, m_0, m_1)$, obtaining $(k^\star, (0, ct_B^{1,\star}))$. Query $\mathsf{reenc}(B, H, k^\star)$. Its output includes the challenge's inner ciphertext $ct_B^{2,\star}$.
3. Compute $m' \leftarrow \mathcal{E}.\mathsf{Dec}(sk^{2\prime}, ct_B^{2,\star})$ and output the bit corresponding to m_0 or m_1. Output a uniform bit if any required call fails or m' is neither message.

These queries obey Definition 2.7: the re-encryption to corrupted Charlie uses an ordinary ciphertext, and the challenge is re-encrypted only to honest user H. □

This explains why our proxy encrypts the inner ciphertext under the recipient's receiving public key. A challenge re-encrypted to H contains $ct_{B\to H}^{0,\star} = \mathcal{E}.\mathsf{Enc}(pk_H^0, ct_B^{2,\star})$, so Charlie cannot directly apply the recovered sk_B^2 to the challenge's inner ciphertext. This prevents the attack above.

3.5 Full HRA and the need for independent receiving keys

Proposition 3.5 (Full HRA attack with exposed rekeys). *Under the component correctness assumption in Section 3.1, an HRA adversary in Definition 2.5 has advantage at least* $1/2 - \mathrm{negl}(\lambda)$ *against the construction.*

Proof. Register honest users B, H and a corrupted recipient C. Query $\mathsf{encrypt}(B, m)$ for an ordinary message, then query $\mathsf{reenc}(B, C, k)$ on the returned record. Use sk_C^0 to decrypt the encrypted inner secret key and recover sk_B^2. Next, query $\mathsf{rekey}(B, H)$. The returned rekey contains sk_B^1.

Choose distinct equal-length messages m_0, m_1 and request a challenge for B. Use sk_B^1 to decrypt its outer layer and sk_B^2 to decrypt its inner layer, then output the bit of the recovered message. □

The rHRA experiment in Definition 2.7 forbids combining these two queries for B. If the re-encryption query comes first, it adds B to $\mathsf{Converted}$, so the rekey query is rejected. If the rekey query comes first, it adds B to $\mathsf{Exposed}$, so the re-encryption query is rejected.

Why must the receiving key be independent? Proposition 3.4 shows why the proxy must encrypt the inner ciphertext for the recipient in this construction. We now keep this encryption step and consider using the recipient's outer or inner key pair as its receiving key pair. Each choice allows an attack in the current rHRA model:

- If $(pk_i^0, sk_i^0) = (pk_i^1, sk_i^1)$, register honest users B, C, H and expose $rk_{B\to C}$ and $rk_{C\to H}$. The second rekey reveals $sk_C^1 = sk_C^0$, which opens the encrypted key in the first rekey to recover sk_B^2. The first rekey already reveals sk_B^1. Thus both layers of Bob's challenge can be decrypted using only honest-to-honest rekeys. This variant fails even PRE-CPA security.
- If $(pk_i^0, sk_i^0) = (pk_i^2, sk_i^2)$, register honest users B, C and a corrupted user Z. Expose $rk_{B\to C}$, obtaining sk_B^1 and an encryption of sk_B^2 under pk_C^2. An ordinary re-encryption from C to Z reveals sk_C^2. This opens Bob's encrypted key and yields sk_B^2, so the adversary can decrypt Bob's challenge. The queries are allowed by rHRA: B has only exposed outgoing rekeys, and C has only ordinary outgoing re-encryptions to a corrupted user.

These attacks explain why this construction uses an independent receiving key pair.

4 An authenticated construction for rCCA2 security

Using PKE-CCA2-secure $\mathcal{E}$ in the rHRA construction of Section 3.1 does not prevent the following chosen-ciphertext attack. An adversary can obtain a rekey between two honest users, learn the delegator's outer secret key, and remove the challenge's outer encryption. It then encrypts the same inner ciphertext under the same outer public key with fresh randomness and submits the result to the PRE decryption oracle. Except with negligible probability, this is a different source ciphertext, so the exclusion rule in Definition 2.10 permits the query and the oracle reveals the challenge message. The construction below therefore adds authentication and binds the ciphertext components to prevent this repackaging.

4.1 Authenticated nested construction

We use a PKE $\mathcal{E}$ with deterministic decryption and PKE-CCA2 security as in Definition 2.4. We also use a strongly unforgeable digital signature scheme to authenticate the ciphertext $ct_{i\to j}^{\text{key}}$ carrying delegator i's inner secret key and the DAE key for delegation (i, j), a strongly unforgeable one-time signature scheme to authenticate each source ciphertext, and a deterministic authenticated encryption (DAE) scheme to protect the confidentiality and integrity of the inner ciphertext during re-encryption. We specify the required properties below.

4.1.1 Components

Definition 4.1 (Digital signatures and strong unforgeability [BSW06]). *A digital signature scheme* Sig *consists of three polynomial-time algorithms:*

- $(V, S) \leftarrow \mathsf{Sig.KeyGen}(1^\lambda)$ *generates a public verification key* V *and a secret signing key* S.
- $\eta \leftarrow \mathsf{Sig.Sign}(S, x)$ *signs a message* x.
- $\mathsf{Sig.Verify}(V, x, \eta) \in \{0, 1\}$ *deterministically accepts or rejects a claimed signature* η *on* x.

For every valid message x*, correctness requires an honestly generated signature to verify except with negligible probability.*

In the strong unforgeability game, the challenger generates (V, S)*, gives* $(1^\lambda, V)$ *to* $\mathcal{A}$*, and initialises* $\mathcal{Q} = \varnothing$*. The adversary may make at most* q_s *adaptive signing queries. On a valid query* x*, the challenger returns* $\eta \leftarrow \mathsf{Sig.Sign}(S, x)$ *and adds* (x, η) *to* $\mathcal{Q}$*. The adversary wins if its output* $(x^\star, \eta^\star)$ *satisfies*

$$\mathsf{Sig.Verify}(V, x^\star, \eta^\star) = 1 \quad \textit{and} \quad (x^\star, \eta^\star) \notin \mathcal{Q}.$$

Define $\mathsf{Adv}^{\mathsf{suf}}_{\mathsf{Sig},\mathcal{A}}(1^\lambda; q_s)$ *as this winning probability. The scheme is strongly unforgeable under chosen-message attack if the advantage is negligible for every PPT adversary and polynomial query bound* q_s*.*

Definition 4.2 (Strongly unforgeable one-time signatures [BCHK07]). *A one-time signature scheme* OTS *has the same algorithms and correctness requirement as in Definition 4.1. We write* $(v, s) \leftarrow \mathsf{OTS.KeyGen}(1^\lambda)$*,* $\sigma \leftarrow \mathsf{OTS.Sign}(s, x)$*, and* $\mathsf{OTS.Verify}(v, x, \sigma) \in \{0, 1\}$*. Strong one-time unforgeability uses the same game with* OTS *in place of* Sig *and at most one signing query, including the case of no signing query. Its advantage* $\mathsf{Adv}^{\mathsf{suf}}_{\mathsf{OTS},\mathcal{A}}(1^\lambda; 1)$ *must be negligible for every PPT adversary.*

We use DAE so that re-encrypting the same source ciphertext with the same rekey and public keys returns the same ciphertext.[3] For each delegation (i, j), sample one DAE key $k_{i\to j}$ and include it in $rk_{i\to j}$. DAE encryption uses no fresh randomness: its output is determined by this key, the header, and the inner ciphertext. The inverse property below also ensures that no second DAE ciphertext decrypts to the same inner ciphertext under that key and header, even when the key is exposed. This prevents the adversary from obtaining a different accepted DAE encryption of the challenge's inner ciphertext to evade the exact-ciphertext exclusions in Definition 2.10.

Definition 4.3 (Deterministic authenticated encryption [RS06]). *We follow the DAE syntax and security definition of Rogaway and Shrimpton. A deterministic authenticated encryption scheme* DAE *consists of three polynomial-time algorithms:*

- $k \leftarrow \mathsf{DAE.KeyGen}(1^\lambda)$ *samples a secret symmetric key* k*.*
- $ct = \mathsf{DAE.Seal}(k, ad, x)$ *deterministically encrypts a byte string* x *under* k *with header* ad*.*
- $\mathsf{DAE.Open}(k, ad, ct)$ *deterministically returns the recovered string* x*, or* $\perp$ *if authentication fails.*

The header ad *is public associated data. The ciphertext length depends only on the public parameters, header, and plaintext length. Correctness requires* $\mathsf{DAE.Open}(k, ad, \mathsf{DAE.Seal}(k, ad, x)) = x$ *for every generated key* k *and valid pair* (ad, x)*. We also require the inverse property:* $\mathsf{DAE.Open}(k, ad, ct) = x \neq \perp$ *to imply* $\mathsf{DAE.Seal}(k, ad, x) = ct$*.*

Security is defined by the following real-or-ideal experiment. An adversary $\mathcal{A}(1^\lambda)$ *makes at most* q_e *sealing queries and* q_d *opening queries, adaptively and in any order:*

- *In the real experiment, sample* $k \leftarrow \mathsf{DAE.KeyGen}(1^\lambda)$*. Answer sealing queries* (ad, x) *with* $\mathsf{DAE.Seal}(k, ad, x)$ *and opening queries* (ad, ct) *with* $\mathsf{DAE.Open}(k, ad, ct)$*.*
- *In the ideal experiment, each new valid sealing input* (ad, x) *receives an independent uniform string of the required ciphertext length. Repeated inputs receive the same stored output. Every opening query returns* $\perp$*.*

The adversary must not open a pair (ad, ct) *previously returned by the sealing oracle, or seal a pair* (ad, x) *previously recovered by a successful opening query. Unsupported inputs return* $\perp$ *in both experiments. Let* p_{real} *and* p_{ideal} *be the respective probabilities that* $\mathcal{A}$ *outputs one. Define*

$$\mathsf{Adv}^{\mathsf{dae}}_{\mathsf{DAE},\mathcal{A}}(1^\lambda; q_e, q_d) = |p_{\text{real}} - p_{\text{ideal}}|.$$

The scheme is secure if this advantage is negligible for every PPT adversary obeying these restrictions and polynomial query bounds q_e, q_d*.*

We use Sig, OTS, and DAE satisfying these security definitions. Assume that collisions among independently generated verification keys have negligible probability.

[3] The fixed padding in owner-encrypted PRE [YZD+26] serves a different purpose. There, the proxy uses a private PRF key to derive one padding value for each source ciphertext and reuses it across recipients. Repeated queries therefore do not supply independent padding samples that can be averaged away.

4.1.2 Full construction By abuse of notation, pk_u denotes both user u's full public key and its canonical encoding. Tuples also denote their unambiguous canonical encodings. Within an algorithm, we write ct^1, ct^2, and ct^0 for the outer ciphertext, inner ciphertext, and re-encrypted inner ciphertext, respectively. For a delegation (i, j), the ciphertext $ct^{\text{key}}_{i\to j}$ encrypts $(pk_i, sk_i^2, k_{i\to j})$ under pk_j^0. As in Section 3.1, the unindexed symbol ct^{key} denotes the encrypted-key component parsed from an input ciphertext.

Figure 3 gives the six PRE algorithms. The equality check on v' uses the outer plaintext in ReEnc and the inner plaintext in Dec. A dot denotes an ignored tuple component. Any failed parsing, decryption, or check returns $\perp$. All key generations use independent randomness. The game caches one rekey per ordered pair as in Figure 1.

$\underline{\mathsf{Setup}(1^\lambda)}$

1 : Select $\mathcal{E}, \mathsf{OTS}, \mathsf{Sig}, \mathsf{DAE}$ and encodings; set $T = 1$.
2 : **return** the public parameters

$\underline{\mathsf{KeyGen}()}$

1 : **for** $a \in \{0, 1, 2\}$ **do**
2 : $\quad (pk_i^a, sk_i^a) \leftarrow \mathcal{E}.\mathsf{KeyGen}(1^\lambda)$
3 : $(V_i, S_i) \leftarrow \mathsf{Sig.KeyGen}(1^\lambda)$
4 : $pk_i \leftarrow (pk_i^0, pk_i^1, pk_i^2, V_i)$
5 : $sk_i \leftarrow (pk_i, sk_i^0, sk_i^1, sk_i^2, S_i)$
6 : **return** (pk_i, sk_i)

$\underline{\mathsf{ReKeyGen}(sk_i, pk_j)}$

1 : $(pk_i, sk_i^0, sk_i^1, sk_i^2, S_i) \leftarrow sk_i$
2 : $k_{i\to j} \leftarrow \mathsf{DAE.KeyGen}(1^\lambda)$
3 : $ct^{\text{key}}_{i\to j} \leftarrow \mathcal{E}.\mathsf{Enc}(pk_j^0, (pk_i, sk_i^2, k_{i\to j}))$
4 : $\eta \leftarrow \mathsf{Sig.Sign}(S_i, (pk_i, pk_j, ct^{\text{key}}_{i\to j}))$
5 : $B_{i\to j} \leftarrow (ct^{\text{key}}_{i\to j}, \eta)$
6 : **return** $rk_{i\to j} = (sk_i^1, k_{i\to j}, B_{i\to j})$

$\underline{\mathsf{ReEnc}(rk_{i\to j}, ct, pk_i, pk_j)}$

1 : $(sk_i^1, k_{i\to j}, B_{i\to j}) \leftarrow rk_{i\to j}$
2 : $(0, v, ct^1, \sigma) \leftarrow ct$
3 : **check** $\mathsf{OTS.Verify}(v, (v, ct^1), \sigma) = 1$
4 : $(v', ct^2) \leftarrow \mathcal{E}.\mathsf{Dec}(sk_i^1, ct^1)$
5 : **check** $v' = v$
6 : $ad \leftarrow (pk_i, pk_j, v)$
7 : $ct^0 \leftarrow \mathsf{DAE.Seal}(k_{i\to j}, ad, ct^2)$
8 : **return** $(1, pk_i, v, ct^0, B_{i\to j})$

$\underline{\mathsf{Enc}(pk_i, m)}$

1 : $(v, s) \leftarrow \mathsf{OTS.KeyGen}(1^\lambda)$
2 : $ct^2 \leftarrow \mathcal{E}.\mathsf{Enc}(pk_i^2, (v, m))$
3 : $ct^1 \leftarrow \mathcal{E}.\mathsf{Enc}(pk_i^1, (v, ct^2))$
4 : $\sigma \leftarrow \mathsf{OTS.Sign}(s, (v, ct^1))$
5 : **return** $(0, v, ct^1, \sigma)$

$\underline{\mathsf{Dec}(sk_u, ct)}$

1 : $(pk_u, sk_u^0, sk_u^1, sk_u^2, S_u) \leftarrow sk_u$
2 : **if** $ct = (0, v, ct^1, \sigma)$ **then**
3 : $\quad$ **check** $\mathsf{OTS.Verify}(v, (v, ct^1), \sigma) = 1$
4 : $\quad (\cdot, ct^2) \leftarrow \mathcal{E}.\mathsf{Dec}(sk_u^1, ct^1)$
5 : $\quad (v', m) \leftarrow \mathcal{E}.\mathsf{Dec}(sk_u^2, ct^2)$
6 : $\quad$ **check** $v' = v$
7 : $\quad$ **return** m
8 : **else if** $ct = (1, pk_i, v, ct^0, B)$ **then**
9 : $\quad (ct^{\text{key}}, \eta) \leftarrow B$
10 : $\quad$ **check** $\mathsf{Sig.Verify}(V_i, (pk_i, pk_u, ct^{\text{key}}), \eta) = 1$
11 : $\quad (pk_i', sk_i^2, k_{i\to u}) \leftarrow \mathcal{E}.\mathsf{Dec}(sk_u^0, ct^{\text{key}})$
12 : $\quad$ **check** $pk_i' = pk_i$
13 : $\quad ad \leftarrow (pk_i, pk_u, v)$
14 : $\quad ct^2 \leftarrow \mathsf{DAE.Open}(k_{i\to u}, ad, ct^0)$
15 : $\quad (v', m) \leftarrow \mathcal{E}.\mathsf{Dec}(sk_i^2, ct^2)$
16 : $\quad$ **check** $v' = v$
17 : $\quad$ **return** m
18 : **else return** $\perp$

Figure 3. Authenticated single-hop PRE construction for rCCA2 security.

Correctness. For an honestly generated source ciphertext, decrypting ct^1 gives (v, ct^2), and decrypting ct^2 gives (v, m). For an honest re-encryption from i to j, decrypting $ct^{\text{key}}_{i\to j}$ gives $(pk_i, sk_i^2, k_{i\to j})$, and decrypting ct^0 gives the same ct^2. Honest signatures verify and all equality checks pass, so both decryption paths return m, except on component correctness errors. A union bound over these errors proves correctness.

4.2 Security

Security intuition. Strong one-time unforgeability prevents changing the challenge ciphertext while keeping $v^\star$, and the v checks prevent reuse of its components under another verification key. For honest i, the long-term signature and the check $pk'_i = pk_i$ bind the input component ct^{key} to the delegation (i, j).

As in the rHRA proof, the inner PKE protects the message when an outgoing rekey is exposed; otherwise, the outer PKE and DAE hide the inner ciphertext. For honest recipients, DAE authentication prevents new accepted ciphertexts under an unexposed delegation key. When that key is exposed, the DAE inverse property ensures that any accepted encryption of the challenge's inner ciphertext under the same header is its exact re-encryption, which the experiment excludes from decryption.

Theorem 4.4 (Restricted chosen-ciphertext security). *Assume the primitive and encoding requirements of Section 4.1. For the polynomially many honestly generated component inputs used in the proof, assume that the probability of any correctness error is negligible. Then the authenticated nested construction is rCCA2-secure in Definition 2.11.*

Proof. Fix the challenge bit $b \in \{0, 1\}$ and write $p_t^b = \Pr[G_t^b \text{ outputs } 1]$. Each game outputs the adversary's guess, or zero if it stops early. Game G_0^b runs the experiment in Definition 2.11. Generate and cache rekeys as in Theorem 3.2. On each successful $\mathsf{reenc}(i, j, ct)$ query, record the DAE plaintext and ciphertext in an initially empty table:

$$\mathcal{T}_{i,j}[(ad, ct^0)] \leftarrow ct^2, \qquad ad = (pk_i, pk_j, v).$$

In every game, decryption first rejects pairs in $\mathcal{D}$, which is updated from the returned ciphertexts and rekeys as in Definition 2.10. We prove

$$G_0^b \approx_c G_1^b \approx_c G_2^b \approx_c G_3^b \approx_c G_4^b,$$

then compare G_4^0 and G_4^1.

Game G_1^b: compare the encrypted key and signature with earlier oracle responses. On a re-encrypted ciphertext, $\mathsf{dec}(j, ct)$ reads the full delegator public key and the pair (ct^{key}, η) from ct. If the delegator public key carried in ct equals the registered public key of an honest user i, compare this input pair (ct^{key}, η) with the recorded $B_{i\to j}$ returned by rekey or reenc for (i, j). Reject if no pair was returned or either component differs. For a match, use $(sk_i^2, k_{i\to j})$ from the record. For any other delegator public key, run the decryption algorithm in Figure 3.

A different accepted pair gives a strong forgery under V_i. The reduction guesses the honest signer and requests signatures only for pairs returned to the adversary. For a matching pair, correctness gives the recorded keys. Thus strong unforgeability and correctness make $|p_0^b - p_1^b|$ negligible.

Game G_2^b: encrypt zeros instead of the secret keys. For each honest pair (i, j), replace the plaintext $(pk_i, sk_i^2, k_{i\to j})$ of $ct_{i\to j}^{\text{key}}$ by $(pk_i, 0^{|sk_i^2|}, 0^{|k_{i\to j}|})$. Sign the new ciphertext and reuse it for that pair. Keep the real keys in rekeys and private records, so decryption uses them as in G_1^b.

Replace one pair at a time, guessing the recipient as in the rHRA proof. Use the PKE challenge public key as pk_j^0, submit the two tuples above, and use the returned ciphertext as $ct_{i\to j}^{\text{key}}$. Generate all other keys locally. A query $\mathsf{dec}(j, ct)$ whose delegator public key matches a registered honest user's public key uses the G_1^b record check. For any other delegator public key, follow Dec in Figure 3, using the PKE decryption oracle when $ct^{\text{key}} \neq ct_{i\to j}^{\text{key}}$. Otherwise, return $\perp$: both challenge plaintexts contain pk_i, which differs from the public key carried in ct, so the scheme's public-key check rejects. For other recipients the reduction knows the receiving secret key. PKE-CCA2 security gives negligible $|p_1^b - p_2^b|$.

Game G_3^b: replace DAE decryption with table lookup. For re-encrypted ciphertexts from honest i to honest j, while $rk_{i\to j}$ is unexposed, change $\mathsf{dec}(j, ct)$ after the G_1^b check. Form $ad = (pk_i, pk_j, v)$ and look up (ad, ct^0) in $\mathcal{T}_{i,j}$. Reject if absent; otherwise use the saved ct^2, then perform inner decryption and the v check as before.

For (ad, ct^0) already recorded in $\mathcal{T}_{i,j}$, correctness makes the answer unchanged. A difference requires DAE.Open to accept an unrecorded pair before its rekey is exposed. To bound this event, guess (i, j) and the index of the first such query. Use the DAE sealing oracle for re-encryptions and reject earlier unrecorded opening queries. Submit the guessed pair to the DAE opening oracle, output whether it accepts, and stop. If the rekey would be returned first, stop with output zero. Game G_2^b lets the reduction generate $ct_{i\to j}^{\text{key}}$ without the DAE key. The ideal opening oracle always rejects, so DAE security bounds this event, with polynomial guessing loss. Hence $|p_2^b - p_3^b|$ is negligible.

Game G_4^b: check source ciphertexts that use $v^\star$. Generate the challenge OTS key before the experiment. Stop with output zero if a source ciphertext verifies under $v^\star$ before the challenge, or afterwards differs from $ct^\star$ and verifies under $v^\star$. Also stop if two independently generated honest OTS verification keys coincide.

To bound the probability of a source query that makes G_4^b stop, an OTS reduction simulates G_3^b, using its verification key as $v^\star$. If a source query $(0, v^\star, ct^1, \sigma)$ passes signature verification before the challenge, the reduction stops its simulation and submits the message $(v^\star, ct^1)$ and signature σ to its OTS challenger as a forgery without making a signing query. At the challenge, it queries its signing oracle once on $(v^\star, ct^{1,\star})$. If a source query $(0, v^\star, ct^1, \sigma)$ passes signature verification after the challenge but differs from $ct^\star$, canonical encoding implies that ct^1 or σ differs from its challenge value. The reduction stops its simulation and submits $(v^\star, ct^1)$ and σ to its OTS challenger as a strong forgery. Games G_3^b and G_4^b agree until a stopping event occurs, so strong one-time unforgeability and the assumed negligible collision probability make $|p_3^b - p_4^b|$ negligible. In runs that do not stop, every verifying source query after the challenge with $v = v^\star$ contains the same bytes $ct^\star$. The query specifies the user separately.

Compare G_4^0 and G_4^1 according to whether a rekey is returned. Use the events from Theorem 3.2: E means an outgoing rekey of the challenge delegator is returned, and U means none is returned. For $F \in \{E, U\}$, write $p_b^F = \Pr[G_4^b \text{ outputs } 1 \wedge F]$. Runs without a valid challenge contribute equally for both bits, so

$$|p_4^1 - p_4^0| \le \left|p_1^E - p_0^E\right| + \left|p_1^U - p_0^U\right|.$$

Use the same user guess i and output rule as in that proof. For E, a permitted re-encryption request from i to a corrupted recipient first checks the signature, outer decryption and outer tag. Return $\perp$ on failure. On success, stop with output zero before generating the delivery. For U, stop before returning an outgoing rekey for i.

Event E: use the inner PKE's CCA2 security. Use the PKE challenge public key as pk_i^2, submit $(v^\star, m_0)$ and $(v^\star, m_1)$, and use the returned ciphertext as $ct^{2,\star}$. Generate the outer encryption and signature normally. Game G_2^b replaces encryptions of the unknown sk_i^2 by encryptions of zeros; known outer and DAE keys suffice for re-encryption queries before the stopping point.

The reduction uses its PKE decryption oracle for source decryption under i and for re-encrypted ciphertexts that carry pk_i as the delegator public key and whose signed pair passes the G_1^b check. Before the challenge it may submit any ct^2. Afterwards it submits $ct^2 \neq ct^{2,\star}$ and checks the recovered v' as usual. If $ct^2 = ct^{2,\star}$, the inner plaintext contains $v^\star$, so reject when $v \neq v^\star$. For $v = v^\star$, a verifying source ciphertext under i is $ct^\star$ by G_4^b. On an unexposed edge (i, j), a matching table entry with this header comes from $\mathsf{reenc}(i, j, ct^\star)$. On an exposed edge, G_1^b fixes the signed pair and the DAE inverse property forces ct^0 to equal the DAE component of $\mathsf{ReEnc}(rk_{i\to j}, ct^\star, pk_i, pk_j)$. Definition 2.10 excludes each of these ciphertexts for its recipient.

For a source query under $u \neq i$, even on the bytes of $ct^\star$, the reduction uses u's locally generated keys. For re-encrypted ciphertexts carrying any other delegator public key, it handles decryption as in G_1^b. Thus it answers every permitted query before the stopping point, and PKE-CCA2 security gives negligible $|p_1^E - p_0^E|$.

Event U: replace $ct^{2,\star}$ by zeros, then use the outer PKE's CCA2 security. Fix an honest recipient j. On the edge (i, j), replace the DAE plaintext in $\mathsf{reenc}(i, j, ct^\star)$ by $0^{|ct^{2,\star}|}$. Record the zero string in $\mathcal{T}_{i,j}$ and reuse the sealed output on repeats. On this edge, G_4^b ensures that only this query uses the header $(pk_i, pk_j, v^\star)$. The real and zero inputs therefore give the same lengths and repetitions to the ideal DAE oracle. Game G_2^b lets the reduction generate $ct_{i\to j}^{\text{key}}$ without the DAE key, and G_3^b answers decryption by lookup. Each modified response is excluded under j, and G_1^b rejects a different signed pair for this delegation. Thus no permitted decryption reads the stored zero string. DAE security bounds the change, one honest recipient at a time.

Now use the PKE challenge public key as pk_i^1. Submit $(v^\star, ct^{2,0})$ and $(v^\star, ct^{2,1})$, where $ct^{2,a} \leftarrow \mathcal{E}.\mathsf{Enc}(pk_i^2, (v^\star, m_a))$ for $a \in \{0, 1\}$. Use the returned ciphertext as $ct^{1,\star}$ and sign $(v^\star, ct^{1,\star})$.

For a source decryption or re-encryption query under i, verify the OTS signature. Use the PKE decryption oracle for the outer layer before the challenge and for $ct^1 \neq ct^{1,\star}$ afterwards. If $ct^1 = ct^{1,\star}$ and $v \neq v^\star$, reject. Both candidate outer plaintexts carry the tag $v^\star$ checked by ReEnc, and both inner plaintexts carry the same tag checked by source Dec. If $v = v^\star$, G_4^b forces $ct = ct^\star$. Reject its decryption under i and its re-encryption from i to a corrupted recipient. For an honest recipient j, return the modified response above. A source query under $u \neq i$ uses sk_u^1, including when its bytes equal $ct^\star$. The reduction knows all remaining keys, including sk_i^2, and can complete the simulation before the stopping point. The DAE replacements and PKE-CCA2 security give negligible $|p_1^U - p_0^U|$.

Add the probability differences. By the triangle inequality,

$$\mathsf{Adv}^{\mathsf{rcca2}}_{\Pi,\mathcal{A}}(1^\lambda) \leq \frac{1}{2} \sum_{b\in\{0,1\}} \sum_{t=0}^{3} |p_t^b - p_{t+1}^b| + \frac{1}{2} \sum_{F\in\{E,U\}} |p_1^F - p_0^F|.$$

Each difference is negligible. Correctness and collision errors are negligible by assumption, and all guessing losses are polynomial. Hence the rCCA2 advantage is negligible. □

5 ML-KEM Deployment

We instantiate the rCCA2 construction using HPKE with ML-KEM, ML-DSA signatures, and AES-SIV. The rHRA construction provides a smaller baseline in the size comparison. The OpenPGP application below uses the authenticated rCCA2 construction.

5.1 HPKE with ML-KEM

ML-KEM generates a shared key rather than encrypting an input message [Nat24b]. Both PRE constructions use the single-shot Base mode of HPKE [BBLW22, Section 6.1], with ML-KEM-768 or ML-KEM-1024, HKDF-SHA-512 [KE10], and AES-256-GCM. We use the ML-KEM interface and key encodings in `draft-ietf-hpke-pq-05` [BC26]. Its $\mathsf{GenerateKeyPair}$ procedure returns the decapsulation key as a 64-byte seed; decapsulation expands this seed using ML-KEM key generation.

Let ek and dk be the ML-KEM public and secret keys generated by this procedure. Fix info = `PRE-HPKE` and use the empty string ε as associated data. Encryption calls $\mathsf{HPKE.SealBase}(\mathsf{ek}, \mathsf{info}, \varepsilon, m)$ and returns the canonical encoding of its output (enc, d). Decryption calls $\mathsf{HPKE.OpenBase}(\mathsf{enc}, \mathsf{dk}, \mathsf{info}, \varepsilon, d)$.

Here enc is the c-byte ML-KEM encapsulation, and d is the AES-256-GCM ciphertext of length $|m| + 16$ bytes, including its 16-byte authentication tag. HPKE derives the nonce from the shared secret and context, so it is not transmitted [BBLW22]. Excluding framing, the total ciphertext length is therefore $|m| + \omega$ bytes, where $\omega = c + 16$.

We assume IND-CCA security of ML-KEM, pseudorandomness of the HPKE key schedule, and confidentiality and ciphertext integrity of AES-256-GCM, as required by HPKE [BBLW22]. Under these assumptions, we use the IND-CCA2 guarantee stated for ML-KEM-based HPKE in [BC26, Section 7]. This supplies the component PKE security required by both PRE constructions.

5.2 Instantiating the rCCA2 construction

Use three independent ML-KEM key pairs per user, indexed by $0, 1, 2$ as in Figure 3. We pair ML-KEM-768 with ML-DSA-65, and ML-KEM-1024 with ML-DSA-87. ML-DSA implements both Sig and OTS: each user has a long-term signing key, and each source ciphertext uses a fresh, independent signing key that signs only once. Use the FIPS 204 signing and verification algorithms [Nat24a]. Under ML-DSA strong unforgeability, these choices satisfy both signature requirements in Section 4.1.1.

For DAE, use AES-SIV with two AES-256 keys, hence a 64-byte key and a 16-byte ciphertext overhead [Har08]. Apply its deterministic mode without a nonce, with the canonical encoding of (pk_i, pk_j, v) as associated data [Har08, Section 4]. SIV opening recomputes the full synthetic IV from the recovered plaintext and header and checks equality. Encrypting an accepted plaintext again therefore reproduces both the IV and the CTR ciphertext, giving the inverse property in Definition 4.3. Its DAE security follows from the SIV analysis [RS06]. Together with the HPKE security assumptions in Section 5.1 and negligible correctness error, these assumptions give rCCA2 security by Theorem 4.4.

Operations. A source encryption uses two ML-KEM encapsulations, one fresh ML-DSA key generation, and one signature. Rekey generation uses one encapsulation and one long-term signature. The proxy verifies the source signature, decapsulates and opens the outer PKE layer, and applies AES-SIV to the inner ciphertext. A recipient verifies the delegation signature, decrypts the auxiliary key, opens AES-SIV, and decrypts the inner ciphertext.

Key and ciphertext lengths. All lengths in this section are in bytes. Let p, s, c be the encoded ML-KEM public-key, secret-key, and encapsulation lengths: $(1184, 64, 1088)$ for ML-KEM-768 and $(1568, 64, 1568)$ for ML-KEM-1024 [BC26, Table 2]. Here $s = 64$ counts the seed encoding used wherever a component secret key is stored or encrypted. The HPKE overhead ω is 1,104 or 1,584 bytes, respectively. We use the full ML-DSA secret-key encodings: for ML-DSA-65, $(|V|, |S|, |\eta|) = (1952, 4032, 3309)$; for ML-DSA-87, these lengths are $(2592, 4896, 4627)$ [Nat24a, Table 2]. The fresh one-time signing keys use the same profile, so $|v| = |V|$ and $|\sigma| = |\eta|$.

The full PRE public key has length $|pk| = 3p + |V|$. The inner ciphertext and signed encrypted auxiliary key have lengths

$$|ct^2| = |m| + |v| + \omega, \qquad |B_{i\to j}| = |pk| + s + 64 + \omega + |\eta|.$$

Write $|B| = |B_{i\to j}|$ for this common length. Table 1 counts every cryptographic field in Figure 3.

The secret-key sizes in Table 1 include a copy of pk for use by ReKeyGen and Dec. If this public key is stored separately, the remaining secret-key fields occupy $3s + |S|$ bytes: 4,224 or 5,088 for the two profiles, respectively.

Table 1. rCCA2 sizes (in bytes) for the two ML-KEM/ML-DSA profiles.

Object	Length	768/65	1024/87
Public key	$3p + \|V\|$	5,504	7,296
Secret key	$\|pk\| + 3s + \|S\|$	9,728	12,384
Source ciphertext	$\|ct^2\| + 2\|v\| + \omega + \|\sigma\|$	$\|m\| + 11{,}373$	$\|m\| + 15{,}571$
Signed encrypted key	$\|B\|$	10,045	13,635
Rekey	$s + 64 + \|B\|$	10,173	13,763
Re-encrypted ciphertext	$\|pk\| + \|v\| + \|ct^2\| + 16 + \|B\|$	$\|m\| + 20{,}573$	$\|m\| + 27{,}715$
With cached delegation	$\|v\| + \|ct^2\| + 16$	$\|m\| + 5{,}024$	$\|m\| + 6{,}784$

Table 2. Rekey and re-encrypted ciphertext sizes (in bytes). For Sato–Shikata, † marks their claimed security; see Appendix C.

Configuration	Security	Rekey	Re-encrypted ciphertext
This work, ML-KEM-768/ML-DSA-65	rCCA2	10,173	20,605
This work, ML-KEM-1024/ML-DSA-87	rCCA2	13,763	27,747
This work, ML-KEM-768	rHRA	1,232	3,408
This work, ML-KEM-1024	rHRA	1,712	4,848
Polyakov et al., BV-PRE	IND-CPA (DAG)	36,992	2,176
Cohen et al., single-hop	HRA (mixed)	$\approx 111{,}616$	$\approx 66{,}560$
Sato–Shikata	Bounded CCA2†	$k\ell(k+u^2)b$	$(k+1)b + \|\mathsf{vk}\| + \|\sigma\|$

Transport-level caching. For a fixed delegation and key epoch, the recipient can cache the verified pair $(pk_i, B_{i\to j})$, costing 15,549 or 20,931 bytes. Subsequent deliveries carry (v, ct^0) and identify that cache entry; the recipient reconstructs the complete ciphertext before decryption. For M messages of equal length, the amortised cryptographic payload is $|m|+2|v|+\omega+16+(|pk|+|B|)/M$ bytes.

5.3 Design and size comparison

We compare our two constructions with the BV-PRE construction of Polyakov et al. [PRSV17], the HRA-secure construction of Cohen et al. [CCG$^+$25], and the lattice construction of Sato and Shikata [SS26]. These schemes make different choices of delegation interface, security guarantee, and functionality. BV-PRE proves IND-CPA security for directed acyclic re-encryption graphs (DAGs), whereas Cohen et al. provide HRA security with a mixed statistical–computational analysis and support multiple hops and homomorphic evaluation with suitable parameters. Sato and Shikata claim bounded CCA2 security for a single-hop construction with preset bounds on decryption and re-encryption queries. Our generic constructions provide single-hop rHRA or rCCA2 security through ordinary public-key encryption interfaces.

Delegation interface. BV-PRE uses recipient-generated preprocessing material in addition to the ordinary encryption public key. This material must remain unavailable to an adversary holding the corresponding rekey, as noted by Davidson et al. [DDLM19, Section 5.2 and Appendix C]. Indeed, its rekey entries satisfy $\gamma_i = \theta_i - s_A 2^{ri}$, so joint access to the preprocessing value θ_0 and rekey value γ_0 reveals $s_A = \theta_0 - \gamma_0 \pmod q$.

Selected configurations. Table 2 compares rekey and re-encrypted ciphertext sizes. The numerical rows use a 32-byte key. We use the published BV-PRE parameters and the ring dimension and modulus size reported for Cohen et al.'s single-hop HRA configuration. The security levels and encoding conventions differ: our and BV-PRE sizes follow the stated encodings; for Cohen et al., we estimate a packed rekey and use their reported serialised ciphertext size. For Sato–Shikata, we leave the query bounds, lattice parameters, and OTS lengths explicit, as defined below.

For rCCA2, setting $|m| = 32$ in Table 1 gives the stated lengths. For the rHRA baseline of Section 3, the rekey has length $2s + \omega$. The full re-encrypted ciphertext has length $|m| + s + 3\omega$. For BV-PRE, the published configuration has $n = 512$, $b = \lceil \log_2 q \rceil = 17$, plaintext modulus 2, and decomposition window $r = 1$ [PRSV17, Table 1 and Section 5]. Its 512-bit plaintext space accommodates the key with zero padding. One fully encoded polynomial occupies $nb/8 = 1{,}088$ bytes, so a ciphertext occupies $2nb/8 = 2{,}176$ bytes. A rekey has $\ell = \lfloor \log_2 q/r \rfloor + 1 = 17$ polynomial pairs, giving $2\ell B = 36{,}992$ bytes.

Cohen et al. [CCG+25, Table 1 and Section 6.1] report a single-hop re-encrypted ciphertext size corresponding to approximately 66,560 bytes in their HRA-secure mode, using $N = 4096$, $\log_2(QP) \simeq 109$, plaintext modulus 2, statistical parameter $\nu = 48$, and query bound $\tau = 2^{18}$. Here P is the auxiliary modulus for hybrid key switching. The byte estimate follows from their rounded ciphertext expansion factor, $520.0 \cdot (1024/8) = 66{,}560$, rather than an exact byte count. Their experiment uses a 1,024-bit payload, which accommodates the 32-byte key with padding, and measures ciphertext size using OpenFHE binary serialisation, including its overhead.

For our rekey estimate, we use one decomposition block, giving two polynomials modulo PQ [CCG+25, Definition 11]. Their packed length is $2N\lceil \log_2(PQ) \rceil/8 \simeq 111{,}616$ bytes.

For Sato–Shikata, let t_d, t_r bound the decryption and re-encryption queries. Our estimate uses the combined budget $t_d + t_r$ to account for both types of query. Choose a $(t_d + t_r)$-disjunct matrix for the OTS verification-key space, and let u be its row count. The compiler uses one base rekey per row pair, giving u^2 rekeys [SS26, preproceedings, Sections 2.4 and 3.2]. Let N be the ring degree, k the module rank, $\ell = \lceil \log_2 q \rceil$, and $b = \lceil N\ell/8 \rceil$ the byte length of one packed polynomial in $R_q = \mathbb{Z}_q[X]/(X^N + 1)$.

Their lattice construction shares one matrix $\mathbf{U} \in R_q^{k \times k\ell}$ across these rekeys, with a separate vector in $R_q^{k\ell}$ for each row pair [SS26, preproceedings, Section 4]. Storing $\mathbf{U}$ once gives $k^2\ell + u^2 k\ell$ polynomials. The full ciphertext contains $k + 1$ polynomials, an OTS verification key vk, and a signature σ. Thus,

$$|rk| = k\ell(k + u^2)b, \qquad |\mathsf{ct}| = (k + 1)b + |\mathsf{vk}| + |\sigma|.$$

The lattice parameters and noise bound must satisfy the module-LWE assumption and keep every re-encryption error coefficient below $q/4$ in absolute value, except with negligible probability. This error also depends on the disjunct matrix's column weight [SS26].

5.4 Application to OpenPGP email forwarding

We use the rCCA2 instantiation of Section 5.2 to forward the OpenPGP session key while keeping the encrypted message body unchanged. OpenPGP encrypts the body under a symmetric session key and delivers that key through a Public-Key Encrypted Session Key (PKESK) packet [WHWN24]. RFC 9980 protects the session key using both ML-KEM and ECDH shared secrets [KRSW26]. Alice adds a separate PRE encryption of the same session key. Bob recovers it from the ordinary PKESK, and Charlie recovers it from the PRE envelope. Anyone who recovers the session key through either path can decrypt the body, so its confidentiality also depends on the ML-KEM-based PRE envelope.

Alice prepares the message. Alice creates the ordinary OpenPGP message with session key K_{sess}. Let P_{SEIPD} be the exact byte string of its complete encoded Symmetrically Encrypted and Integrity Protected Data (SEIPD) packet, including packet and length headers. The proxy preserves these bytes; the binding does not normalise or reserialise the packet. Let desc specify the content-encryption suite, SEIPD version, and forwarding-extension version, and let fpr_B identify Bob's authenticated OpenPGP encryption key. Alice binds the session key to this encrypted body by computing

$$\begin{aligned} h_{\text{body}} &= \mathsf{H}(\texttt{OpenPGP-PRE}, \mathsf{desc}, P_{\text{SEIPD}}), \\ m &= (\texttt{OpenPGP-PRE}, \mathsf{fpr}_B, \mathsf{desc}, h_{\text{body}}, K_{\text{sess}}). \end{aligned} \tag{2}$$

Here H is collision-resistant. The tuples denote their unambiguous encodings in a fixed format with a length prefix for each field. Alice computes $ct_B \leftarrow \mathsf{PRE.Enc}(pk_B, m)$ using Bob's separate PRE public key, authenticated as belonging to the same OpenPGP identity. She sends ct_B and the public descriptors alongside the ordinary OpenPGP message. Its PKESK packets and encrypted body remain unchanged.

Bob authorises the proxy. Bob generates $rk_{B\to C} \leftarrow \mathsf{PRE.ReKeyGen}(sk_B, pk_C)$ for Charlie's authenticated PRE public key and supplies it to the proxy through an authenticated authorisation channel. He can then remain offline. For each message, the proxy computes $ct_C \leftarrow \mathsf{PRE.ReEnc}(rk_{B\to C}, ct_B, pk_B, pk_C)$ and sends Charlie ct_C, the unchanged P_{SEIPD}, and the public descriptors.

Charlie decrypts the message. Charlie checks that ct_C has hop count one and carries Bob's PRE public key from the authenticated binding for fpr_B, then computes $m \leftarrow \mathsf{PRE.Dec}(sk_C, ct_C)$. He checks the fingerprint, descriptor and session-key length in m, and compares the recovered digest with the value recomputed from the received descriptor and SEIPD packet using Eq. (2). If all checks pass, he decrypts and authenticates the SEIPD packet with K_{sess}.

Local experiment setup. We run the experiment on a local Intel Core i5-12500 with six physical cores, twelve logical CPUs and 7 GiB RAM, using Linux and Go 1.27.4. Other programs may run on this machine, and its CPU frequency can change. We implement the rCCA2 construction of Section 5.2 in Go, using CIRCL 1.6.3 [FHK19] for ML-KEM-768 and ML-DSA-65, and ProtonMail/go-crypto 1.5.2 [Pro26] for OpenPGP packet processing. We implement the HPKE Base-mode key schedule and AES-SIV using Go's cryptographic primitives, with HKDF expansion provided by `golang.org/x/crypto/hkdf`. We prepare user keys and source ciphertexts before timing. Each re-encryption verifies the source signature, decrypts the outer HPKE ciphertext, encrypts the inner ciphertext with AES-SIV, and encodes the full PRE output. We repeat these steps for every request, including expanding the required ML-KEM secret seed. Recipient decryption checks are outside the timers.

Proxy throughput. We measure how many re-encryptions the proxy completes per second. We prepare 1,024 distinct source ciphertexts for 32 delegations, using a fresh session key for each encryption of the test email. The email has a 12-byte UTF-8 message and occupies 189 bytes including headers and line endings. Each source ciphertext encrypts the 131-byte value m in Eq. (2), which contains the session key and the information tying it to Bob and the encrypted email.[4]

A worker is a software task that repeatedly re-encrypts ciphertexts from this prepared set. We test one, two and four workers, allowing up to that many re-encryptions to run in parallel. The operating system chooses which CPU cores run them. For each setting, we warm up for five seconds, then count completed re-encryptions over 60 seconds. We repeat the measurements in three rounds, using worker counts in the orders $(1, 2, 4)$, $(2, 4, 1)$ and $(4, 1, 2)$, respectively. This test times re-encryption and PRE output encoding.

Figure 4 shows the median of the three rates for each worker count. The proxy completes 6,023, 11,468 and 22,276 re-encryptions per second with one, two and four workers, respectively. Four workers are 3.70 times as fast as one worker. Median CPU usage across all threads of the benchmark process is 100.3%, 199.6% and 393.3%; 100% means fully using one logical CPU.

Before timing, we check that each prepared input can be re-encrypted and decrypted correctly. After each run, we decrypt each worker's last output and compare it with the expected plaintext. All checks passed.

[4] The five fields contain the 11-byte string `OpenPGP-PRE`, a 32-byte key fingerprint, a four-byte descriptor, a 32-byte SHA-256 digest and a 32-byte session key. Each field has a four-byte length prefix, giving $11 + 32 + 4 + 32 + 32 + 5 \times 4 = 131$ bytes.

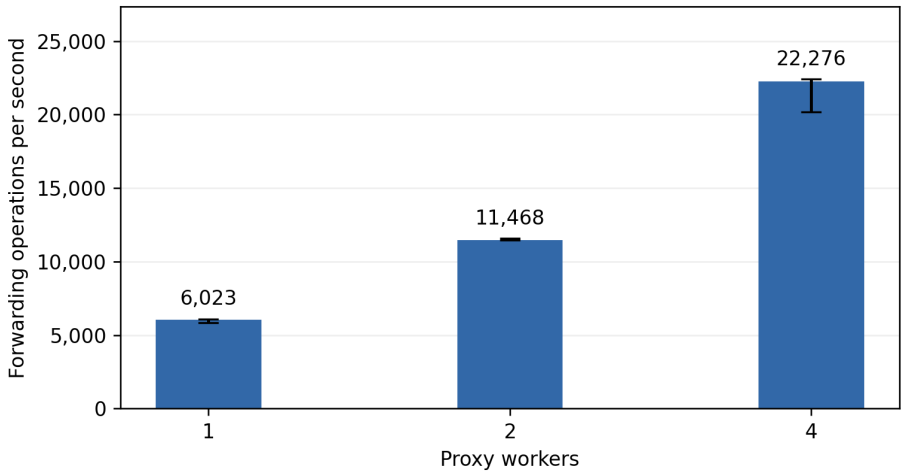


Figure 4. Re-encryptions completed per second. Each bar shows the median rate over three 60-second runs. Error bars show the lowest and highest rates.

Acknowledgements

The authors thank the Proton team for pointing out their 2022 work on OpenPGP email forwarding [VPW22]. The authors thank Didi Liu for helpful discussions.

Haotian Yin used OpenAI Codex to assist with checking formatting, proofs, grammar, and references. The authors also used Codex to run the benchmark and its tests. The authors directed the experiments, checked the results, and are responsible for all content, including the correctness of the mathematical arguments and the accuracy of the references and experimental results.

References

[AFGH06] Ateniese, G., Fu, K., Green, M., Hohenberger, S.: Improved proxy re-encryption schemes with applications to secure distributed storage. ACM Transactions on Information and System Security **9**(1), 1–30 (Feb 2006). https://doi.org/10.1145/1127345.1127346

[BBLW22] Barnes, R., Bhargavan, K., Lipp, B., Wood, C.A.: Hybrid public key encryption. RFC 9180, RFC Editor (2022). https://doi.org/10.17487/RFC9180

[BC26] Barnes, R., Connolly, D.: Post-quantum and post-quantum/traditional hybrid algorithms for HPKE. Internet-Draft draft-ietf-hpke-pq-05, Internet Engineering Task Force (Jul 2026), https://datatracker.ietf.org/doc/html/draft-ietf-hpke-pq-05, work in progress, 6 July 2026

[BBS98] Blaze, M., Bleumer, G., Strauss, M.: Divertible protocols and atomic proxy cryptography. In: Nyberg, K. (ed.) EUROCRYPT'98. LNCS, vol. 1403, pp. 127–144. Springer, Berlin, Heidelberg (1998). https://doi.org/10.1007/BFb0054122

[BCHK07] Boneh, D., Canetti, R., Halevi, S., Katz, J.: Chosen-ciphertext security from identity-based encryption. SIAM Journal on Computing **36**(5), 1301–1328 (2007). https://doi.org/10.1137/S009753970544713X, https://crypto.stanford.edu/~dabo/pubs/papers/ccaibejour.pdf, full version dated 13 June 2006

[BSW06] Boneh, D., Shen, E., Waters, B.: Strongly unforgeable signatures based on computational Diffie-Hellman. In: Yung, M., Dodis, Y., Kiayias, A., Malkin, T. (eds.) PKC 2006. LNCS, vol. 3958, pp. 229–240. Springer, Berlin, Heidelberg (2006). https://doi.org/10.1007/11745853_15

[BV11] Brakerski, Z., Vaikuntanathan, V.: Fully homomorphic encryption from ring-LWE and security for key dependent messages. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 505–524. Springer, Berlin, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22792-9_29

[CH07] Canetti, R., Hohenberger, S.: Chosen-ciphertext secure proxy re-encryption. In: Ning, P., De Capitani di Vimercati, S., Syverson, P.F. (eds.) ACM CCS 2007. pp. 185–194. ACM (2007). https://doi.org/10.1145/1315245.1315269

[CKN03] Canetti, R., Krawczyk, H., Nielsen, J.B.: Relaxing chosen-ciphertext security. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 565–582. Springer, Berlin, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45146-4_33

[Coh19] Cohen, A.: What about Bob? The inadequacy of CPA security for proxy reencryption. In: Lin, D., Sako, K. (eds.) PKC 2019, Part II. LNCS, vol. 11443, pp. 287–316. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17259-6_10, full version at https://eprint.iacr.org/2017/785

[CCG+25] Cohen, A., Cousins, D.B., Genise, N., Kline, E., Polyakov, Y., RV, S.: HRA-secure homomorphic lattice-based proxy re-encryption with tight security. IACR Communications in Cryptology **2**(1) (2025). https://doi.org/10.62056/ab0l5wol7

[DDLM19] Davidson, A., Deo, A., Lee, E., Martin, K.: Strong post-compromise secure proxy re-encryption. In: Jang-Jaccard, J., Guo, F. (eds.) ACISP 2019. LNCS, vol. 11547, pp. 58–77. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-21548-4_4, https://eprint.iacr.org/2019/368, full version, Appendix C

[FHK19] Faz-Hernandez, A., Kwiatkowski, K.: Introducing CIRCL: An Advanced Cryptographic Library. Cloudflare (2019), https://github.com/cloudflare/circl/tree/v1.6.3, version 1.6.3, accessed 14 November 2026

[FKKP19] Fuchsbauer, G., Kamath, C., Klein, K., Pietrzak, K.: Adaptively secure proxy re-encryption. In: Lin, D., Sako, K. (eds.) PKC 2019, Part II. LNCS, vol. 11443, pp. 317–346. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17259-6_11

[GLSW21] Gaddam, S., Luykx, A., Sinha, R., Watson, G.: Reducing HSM reliance in payments through proxy re-encryption. In: 30th USENIX Security Symposium. pp. 4061–4078. USENIX Association (2021), https://www.usenix.org/conference/usenixsecurity21/presentation/gaddam

[Har08] Harkins, D.: Synthetic initialization vector (SIV) authenticated encryption using the advanced encryption standard (AES). RFC 5297, RFC Editor (2008). https://doi.org/10.17487/RFC5297

[ID03] Ivan, A., Dodis, Y.: Proxy cryptography revisited. In: NDSS 2003. The Internet Society (2003), https://www.ndss-symposium.org/wp-content/uploads/2017/09/Proxy-Cryptography-Revisited-Anca-Ivan.pdf

[Kir14] Kirshanova, E.: Proxy re-encryption from lattices. In: Krawczyk, H. (ed.) PKC 2014. LNCS, vol. 8383, pp. 77–94. Springer, Berlin, Heidelberg (2014). https://doi.org/10.1007/978-3-642-54631-0_5

[KRSW26] Kousidis, S., Roth, J., Strenzke, F., Wussler, A.: Post-quantum cryptography in OpenPGP. RFC 9980, RFC Editor (2026). https://doi.org/10.17487/RFC9980

[KE10] Krawczyk, H., Eronen, P.: HMAC-based extract-and-expand key derivation function (HKDF). RFC 5869, RFC Editor (2010). https://doi.org/10.17487/RFC5869

[LSW25] Lai, R.W.F., Swarnakar, M., Woo, I.K.Y.: Leaky LWE: Learning with errors with semi-adaptive secret- and error-leakage. IACR Communications in Cryptology **2**(3) (2025). https://doi.org/10.62056/ah89ksuc2

[LV08] Libert, B., Vergnaud, D.: Unidirectional chosen-ciphertext secure proxy re-encryption. In: Cramer, R. (ed.) PKC 2008. LNCS, vol. 4939, pp. 360–379. Springer, Berlin, Heidelberg (2008). https://doi.org/10.1007/978-3-540-78440-1_21

[MP12] Micciancio, D., Peikert, C.: Trapdoors for lattices: Simpler, tighter, faster, smaller. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 700–718. Springer, Berlin, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29011-4_41

[MS25] Micciancio, D., Suhl, A.: Simulation-secure threshold PKE from LWE with polynomial modulus. IACR Communications in Cryptology **1**(4) (2025). https://doi.org/10.62056/a0zogy4e-

[Nat24a] National Institute of Standards and Technology: Module-lattice-based digital signature standard. FIPS 204, National Institute of Standards and Technology (2024). https://doi.org/10.6028/NIST.FIPS.204

[Nat24b] National Institute of Standards and Technology: Module-lattice-based key-encapsulation mechanism standard. FIPS 203, National Institute of Standards and Technology (2024). https://doi.org/10.6028/NIST.FIPS.203

[Ope26] OpenFHE Project: OpenFHE v1.6.1: Proxy re-encryption implementation. Code snapshot (2026), https://github.com/openfheorg/openfhe-development/tree/b59e1f291965f14fbee5172cd83a20f5b856204f, commit b59e1f2, inspected 28 October 2026

[PRSV17] Polyakov, Y., Rohloff, K., Sahu, G., Vaikuntanathan, V.: Fast proxy re-encryption for publish/subscribe systems. Cryptology ePrint Archive, Report 2017/410 (2017), https://web.njit.edu/~rohloff/papers/2017/410.pdf, full version dated 6 September 2017

[Pro26] ProtonMail: go-crypto. Software, version 1.5.2 (2026), https://github.com/ProtonMail/go-crypto/releases/tag/v1.5.2, accessed 14 November 2026

[RS06] Rogaway, P., Shrimpton, T.: A provable-security treatment of the key-wrap problem. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 373–390. Springer, Berlin, Heidelberg (2006). https://doi.org/10.1007/11761679_23, full version: https://eprint.iacr.org/2006/221

[SS26] Sato, S., Shikata, J.: Bounded CCA2-secure proxy re-encryption from lattices. In: Boura, C., Mashatan, A., Miri, A. (eds.) Selected Areas in Cryptography – SAC 2025. LNCS, vol. 16207, pp. 425–455. Springer, Cham (2026). https://doi.org/10.1007/978-3-032-10536-3_16,

preproceedings: https://www.sacworkshop.org/SAC25/preproceedings/sac2025-2-paper20.pdf
[SC09] Shao, J., Cao, Z.: CCA-secure proxy re-encryption without pairings. In: Jarecki, S., Tsudik, G. (eds.) PKC 2009. LNCS, vol. 5443, pp. 357–376. Springer, Berlin, Heidelberg (2009). https://doi.org/10.1007/978-3-642-00468-1_20, https://www.iacr.org/archive/pkc2009/54430361/54430361.pdf
[VPW22] Vial-Prado, F., Wussler, A.: OpenPGP email forwarding via diverted elliptic curve Diffie-Hellman key exchanges. arXiv:2211.06992 (2022). https://doi.org/10.48550/arXiv.2211.06992, presented at ICMC 2021
[WWXW25] Wan, X., Wang, Y., Xue, H., Wang, M.: Unbounded multi-hop proxy re-encryption with HRA security: An LWE-based optimization. In: Susilo, W., Pieprzyk, J. (eds.) ACISP 2025, Part II. LNCS, vol. 15659, pp. 124–144. Springer, Singapore (2025). https://doi.org/10.1007/978-981-96-9098-5_7
[WHWN24] Wouters, P., Huigens, D., Winter, J., Niibe, Y.: OpenPGP. RFC 9580, RFC Editor (2024). https://doi.org/10.17487/RFC9580
[Xag10] Xagawa, K.: Cryptography with Lattices. Ph.D. thesis, Tokyo Institute of Technology (2010), https://xagawa.net/pdf/2010Thesis.pdf, chapter 15
[YZL+25] Yin, H., Zhang, J., Li, W., Dong, Y., Lim, E.G., Wojtczak, D.: Reducing honest re-encryption attack to chosen ciphertext attack. Cryptology ePrint Archive, Paper 2025/704, early version under this title, first posted 18 April 2025. Version history: https://eprint.iacr.org/archive/versions/2025/704
[YZL+26] Yin, H., Zhang, J., Li, W., Dong, Y., Lim, E.G., Wojtczak, D.: Collusion-safe proxy re-encryption. In: Financial Cryptography and Data Security (FC 2026) (2026), https://eprint.iacr.org/2025/1423, conference version; available preprint
[YZD+26] Yin, H., Zhang, J., Dong, Y., Wojtczak, D., Lim, E.G.: Single-hop HRA-secure owner-encrypted lattice-based proxy re-encryption without statistical noise flooding. Cryptology ePrint Archive, Paper 2026/1113 (2026), https://eprint.iacr.org/2026/1113

A CCA2 implies HRA for constant hop bounds

Proof (Theorem 2.12). Let $\mathcal{A}$ be an HRA adversary with at most n users and q oracle queries, where $n, q \geq 1$ are polynomial. The CCA2 adversary $\mathcal{B}$ follows the HRA record rules and stores the plaintext of each ordinary record. It forwards registration and rekey queries. Before the challenge, it answers encryption queries locally and forwards valid re-encryption queries, since $\mathcal{D}_{\mathsf{cca}}$ is still empty.

When $\mathcal{A}$ submits its first valid challenge $(i^\star, m_0, m_1)$, $\mathcal{B}$ delays its CCA2 challenge. Under each user's public key, it generates q source ciphertexts of each challenge message. Starting from these ciphertexts and every existing ordinary record of m_0 or m_1, it queries the re-encryption oracle q times for each possible recipient, then repeats from each output until the hop count reaches T. Each child is an oracle output on that particular parent under the cached rekey for the chosen edge. Before the challenge, $\mathcal{D}_{\mathsf{cca}}$ is empty, so these queries are allowed even for corrupted recipients. If a preparation call fails, $\mathcal{B}$ aborts with a random guess. The resulting trees contain at most $(2n+1)q\sum_{d=0}^{T}(nq)^d$ nodes, which is polynomial for constant T. The reduction keeps these outputs hidden, submits the challenge, and returns the challenge ciphertext to $\mathcal{A}$.

After the challenge, $\mathcal{B}$ answers ordinary encryption and re-encryption queries for m_0 or m_1 with the next unused root or child for the requested user. The q copies at each step suffice for all of $\mathcal{A}$'s queries. For other messages, it encrypts locally and forwards re-encryption queries. It also forwards re-encryptions of challenge-derived records to honest users, and rejects those to corrupted users as HRA requires. All HRA validity and hop checks remain in place.

Except on a correctness error, every pair in $\mathcal{D}_{\mathsf{cca}}$ decrypts to m_0 or m_1: the challenge and exposed-rekey rules have this property, and every re-encryption queried by $\mathcal{B}$, including the hidden preparation queries, preserves its honest input's plaintext. Thus ordinary records of other messages remain outside $\mathcal{D}_{\mathsf{cca}}$, and forwarding their re-encryption queries is allowed. Queries on ordinary records of the challenge messages are answered from the prepared trees, while challenge-derived queries go only to honest users.

Fix the registered keys and cached rekeys. Couple each ordinary HRA response for an encryption of m_0 or m_1 to the next prepared root, and each ordinary re-encryption of either

message to the next prepared child of its actual parent for the requested recipient. Each such node uses fresh coins drawn for that call. Which node is revealed depends only on the visible transcript. Thus the visible responses agree up to the first failure of honest generation or decryption. All queried parents, including the unshown branches, are honestly generated, and the total number of calls is polynomial. Hence

$$\mathsf{Adv}^{\mathsf{hra}}_{\Pi,\mathcal{A}}(1^\lambda) \le \mathsf{Adv}^{\mathsf{cca2}}_{\Pi,\mathcal{B}}(1^\lambda) + \mathrm{negl}(\lambda).$$

Since $\mathcal{B}$ runs in polynomial time for constant T, CCA2 security implies HRA security. □

B Noise budgets for ML-KEM parameters

This appendix explains why key-switching its lattice ciphertexts is problematic. We compare the error from public-key-generated switching entries with the native decoding margin, then examine statistical flooding and computational padding for hiding key-dependent information from recipients.

B.1 Decoding margin and switching error

Work over $R_q = \mathbb{Z}_q[X]/(X^{256}+1)$, with $q = 3329$. ML-KEM-768 and ML-KEM-1024 use module ranks $k = 3, 4$, respectively, and secrets and errors whose coefficients are sampled from CBD_2, each with variance one [Nat24b]. Write the delegator's public key as $t = As + e$. Before compression, an encryption has $u = A^\top y + e_1$ and $v = t^\top y + e_2 + \mu(m)$, where $\mu(m)$ encodes each message bit as 0 or 1665. Let $\bar{u}$ and $\bar{v}$ be the values obtained by compressing and then decompressing u and v. The compression errors $\delta_u = \bar{u} - u$ and $\delta_v = \bar{v} - v$ are taken coefficientwise as centred differences modulo q, so $\bar{u} = u + \delta_u$ and $\bar{v} = v + \delta_v$ modulo q. These errors arise from rounding in lossy compression and are determined by u, v and the compression parameters. Their decryption phase is

$$\bar{v} - s^\top \bar{u} = \mu(m) + \nu_{\mathrm{orig}}, \qquad \nu_{\mathrm{orig}} = y^\top e - s^\top(e_1 + \delta_u) + e_2 + \delta_v.$$

The compression errors satisfy $(\|\delta_u\|_\infty, \|\delta_v\|_\infty) \le (2, 104)$ for ML-KEM-768 and $(1, 52)$ for ML-KEM-1024.

To decode each bit, ML-KEM applies $\mathrm{Compress}_1(x) = \lfloor 2x/q + 1/2 \rfloor \bmod 2$ to the corresponding decryption coefficient represented in $\{0, \ldots, q-1\}$ [Nat24b, Section 4.2.1 and Algorithm 15]. For $q = 3329$, this returns one exactly when $833 \le x \le 2496$. Thus the correct centred integer errors around the encodings 0 and 1665 lie in $[-832, 832]$ and $[-832, 831]$, respectively. The uniform symmetric bound $\|\nu\|_\infty \le 831$ therefore suffices to decode all 256 bits correctly.

For a correctness benchmark, model each coefficient of the total error before reduction modulo q as a centred Gaussian with standard deviation at most σ. A Gaussian tail bound and a union bound over the 256 coefficients give

$$\Pr[\text{bit-decoding failure}] \le \Pr[\|\nu\|_\infty > 831] \le 512 \exp\left(-\frac{831^2}{2\sigma^2}\right). \tag{3}$$

The factor $512 = 2 \cdot 256$ accounts for both tails of each coefficient. Even for the relaxed target failure probability 2^{-40}, making this upper bound at most the target requires $\sigma \le 831/\sqrt{2\ln(512 \cdot 2^{40})} \simeq 100.83$.

Consider binary gadget decomposition of $\bar{u}$ into $L = k\lceil \log_2 q \rceil = 12k$ polynomials $d_1, \ldots, d_L$, with coefficients in $\{0, 1\}$. Let $W(d)$ count all their nonzero coefficients. A public-key-generated switching entry has decryption error $\xi_j = e_C^\top r_j - s_C^\top e_{1,j} + e_{2,j}$, where s_C, e_C belong to an independent recipient key and the other terms are fresh encryption coins. A secret-key-generated entry can instead have a single error polynomial h_j. The respective added errors are $\nu_{\mathrm{PK}} = -\sum_j d_j \xi_j$ and $\nu_{\mathrm{SK}} = -\sum_j d_j h_j$.

Proposition B.1. *Fix d independently of the recipient key and switching randomness. Suppose all coefficients of $s_C, e_C, r_j, e_{1,j}, e_{2,j}, h_j$ are independent* CBD_2 *samples, with s_C, e_C reused across entries. For each integer coefficient before reduction modulo q,*

$$\mathrm{Var}(\nu_{\mathrm{PK},a} \mid d) = (2kn+1)W(d), \qquad \mathrm{Var}(\nu_{\mathrm{SK},a} \mid d) = W(d), \qquad n = 256.$$

These variances average over the recipient key as well as the switching randomness.

Proof. Expanding the sum and collecting terms gives the public-key error as $-e_C^\top \sum_j d_j r_j + s_C^\top \sum_j d_j e_{1,j} - \sum_j d_j e_{2,j}$. Each coefficient of a sum $\sum_j d_j r_j$ has variance $W(d)$. Independence and zero means give variance $knW(d)$ for each of the first two terms, variance $W(d)$ for the last, and zero cross moments. The same calculation for $-\sum_j d_j h_j$ gives $W(d)$. □

For scale, setting $W(d) = Ln/2 = 6kn$ gives the following values. This is an illustrative digit weight, not the exact expected weight of ML-KEM ciphertexts.

Parameters	$W(d)$	Public-key error SD	Secret-key error SD
ML-KEM-768	4608	2661.3	67.9
ML-KEM-1024	6144	3548.1	78.4

The public-key switching error standard deviations are much larger than the Gaussian benchmark of 100.83 for a target failure probability of 2^{-40}.

B.2 Statistical flooding

Statistical flooding adds independent noise to hide the key-dependent error seen by a recipient. For the secret-key switch above, this error includes $g = y^\top e - (e_1 + \delta_u)^\top s - d^\top h$. Adding a pad F changes the decryption phase to $\mu(m) + g + e_2 + \delta_v + F \pmod q$, so the same pad must both hide the shift and allow correct decoding.

For an integer shift with $\|g\|_2 \le T$, the discrete Gaussian calibration of Cohen et al. [CCG+25, Lemma 3 and Theorems 2–3] uses nominal coefficient standard deviation

$$\sigma_{\mathrm{fl}} = T\sqrt{12\tau}\, 2^{\nu/2}, \tag{4}$$

where τ is the query bound and ν is their statistical security parameter. Any nonzero integer shift has norm at least one. Thus, even with one query and $\nu = 40$, this calibration gives $\sigma_{\mathrm{fl}} \ge \sqrt{12}\, 2^{20} \simeq 3.63 \cdot 10^6$. At such a width, the independent Gaussian pad reduced modulo q is nearly uniform.

Consequently, this statistical-flooding calibration is incompatible with correct decoding at the native ML-KEM-768 and ML-KEM-1024 parameters. It therefore does not provide a route to correct HRA-secure PRE while keeping that modulus and decoder.

B.3 Computational padding

Computational padding can avoid the exponential factor in statistical flooding, but it still needs a leakage proof and a correctness bound. The Gaussian techniques of Micciancio and Suhl [MS25] and Lai, Swarnakar, and Woo [LSW25] do not directly apply to native CBD secrets and errors merely by matching their variance.

For a simple parameter comparison, let M map the coefficients of (e, s) to $(y_i^\top e - f_i^\top s)_{i=1}^Q$, with $f_i = e_{1,i} + \delta_{u,i}$. In the Gaussian calibration of Lai et al. [LSW25, Condition 1 and Example 1], let s_*, s_x, and s_p denote the underlying LWE width, the common width of the delegator's

secret and public-key error, and the pad width, respectively. With Gaussian probability mass proportional to $\exp(-\pi\|z\|_2^2/s^2)$, the conditions include

$$s_* \geq \sqrt{2}\eta, \qquad C = 2\eta^2 + s_*^2, \qquad s_x^2 > C, \qquad s_p^2 \geq \frac{\|M\|_2^2}{C^{-1} - s_x^{-2}} > C\|M\|_2^2, \tag{5}$$

for nonzero M, where η is a smoothing upper bound for $\mathbb{Z}^{2kn}$. Thus the nominal pad standard deviation satisfies $\sigma_p = s_p/\sqrt{2\pi} > \sqrt{C/(2\pi)}\,\|M\|_2$.

For ML-KEM-768 dimensions, take $\varepsilon = 2^{-128}$ for the smoothing bound, $s_* = \sqrt{2\pi}\,3.2$, and the sufficient bound $\eta = \sqrt{\log(2 \cdot 1536(1 + 1/\varepsilon))/\pi} \simeq 5.550$. Then $C/(2\pi) \simeq 20.0431$. For illustration, consider one encryption with $\|y\|_2^2 = kn = 768$, the expected squared norm under independent CBD_2 sampling. Since a row of M contains the coefficients of y, $\|M\|_2 \geq \|y\|_2$, giving $\sigma_p > \sqrt{768C/(2\pi)} \simeq 124.07$. Applying the Gaussian benchmark in (3) to the pad alone, even the relaxed target failure probability 2^{-40} gives the sufficient threshold $\sigma_p \lesssim 100.83$. The illustrative lower bound 124.07 already exceeds this threshold before including the original and switching errors. This comparison shows that the example does not meet that sufficient criterion.

C Decryption and re-encryption query bounds in the Sato–Shikata proof

The proof of Theorem 1 in the Sato–Shikata preproceedings uses a t-disjunct matrix to claim (t, t)-CCA2 security, although the game allows t decryption queries and t re-encryption queries separately [SS26, preproceedings, Definition 5 and Section 3.3].

Both oracle simulations abort when a query for the challenge user selects a row set containing the same hidden index j^*; the query's verification key selects this set. Even considering only verification keys different from the challenge key, the index must avoid the union of up to $2t$ sets. A t-disjunct matrix does not guarantee that such an index exists, so the claimed non-abort probability does not follow. This is a gap in the proof.

The size estimate in Section 5.3 uses the combined budget $t_d + t_r$ to account for both types of query.