(R)CCA Secure Updatable Encryption with Integrity Protection

Michael Klooß[1], Anja Lehmann[2], and Andy Rupp[1]

[1] Karlsruhe Institute for Technology, Germany
{andy.rupp, michael.klooss}@kit.edu
[2] IBM Research – Zurich, Switzerland
anj@zurich.ibm.com

Abstract. An updatable encryption scheme allows a data host to update ciphertexts of a client from an old to a new key, given so-called update tokens from the client. Rotation of the encryption key is a common requirement in practice in order to mitigate the impact of key compromises over time. There are two incarnations of updatable encryption: One is ciphertext-*dependent*, i.e. the data owner has to (partially) download all of his data and derive a dedicated token per ciphertext. Everspaugh et al. (CRYPTO'17) proposed CCA and CTXT secure schemes in this setting. The other, more convenient variant is ciphertext-*independent*, i.e., it allows a single token to update *all* ciphertexts. However, so far, the broader functionality of tokens in this setting comes at the price of considerably weaker security: the existing schemes by Boneh et al. (CRYPTO'13) and Lehmann and Tackmann (EUROCRYPT'18) only achieve CPA security and provide no integrity protection. Arguably, when targeting the scenario of outsourcing data to an untrusted host, plaintext integrity should be a minimal security requirement. Otherwise, the data host may alter or inject ciphertexts arbitrarily. Indeed, the schemes from BLMR13 and LT18 suffer from this weakness, and even EPRS17 only provides integrity against adversaries which cannot arbitrarily inject ciphertexts. In this work, we provide the first ciphertext-*independent* updatable encryption schemes with security beyond CPA, in particular providing strong integrity protection. Our constructions and security proofs of updatable encryption schemes are surprisingly modular. We give a generic transformation that allows key-rotation and confidentiality/integrity of the scheme to be treated almost separately, i.e., security of the updatable scheme is derived from simple properties of its static building blocks. An interesting side effect of our generic approach is that it immediately implies the unlinkability of ciphertext updates that was introduced as an essential additional property of updatable encryption by EPRS17 and LT18.

1 Introduction

Updatable encryption was introduced by Boneh et al. [Bon+13] as a convenient solution to enable key rotation for symmetric encryption. Rotating secret keys is considered good practice to realize proactive security: Periodically changing the cryptographic key that is used to protect the data reduces the risk and impact of keys being compromised over time. For instance, key rotation is mandated when storing encrypted credit card data by the PCI DSS standard [PCI16], and several cloud storage providers, such as Google and Amazon, offer data-at-rest encryption with rotatable keys [Eve+17b].

The challenge with key rotation is how to efficiently update the existing ciphertexts when the underlying secret key is refreshed. The straightforward solution is to decrypt all old ciphertexts and re-encrypt them from scratch using the new key. Clearly, this approach is not practical in the typical cloud storage scenario where data is outsourced to a (potentially untrusted) host, as it would require the full download and upload of all encrypted data.

An *updatable encryption scheme* is a better solution to this problem: it extends a classic symmetric encryption scheme with integrated key rotation and update capabilities. More precisely, these schemes allow to derive a short update token from an old and new key, and provide an additional algorithm that re-encrypts ciphertexts using such a token. A crucial property for updatable encryption is that learning an update token does not impact the confidentiality and also the integrity of the ciphertexts. Thus, the procedure for re-encrypting all existing ciphertexts can be securely outsourced to the data host.

State of the Art. There are two different variants of updatable encryption, depending on whether the update tokens are generated for a specific ciphertext or are ciphertext-independent. The former type – called *ciphertext-dependent* updatable encryption – has been introduced by Boneh et al. [Bon+15] and requires the data owner to (partially) download all outsourced ciphertexts, derive a dedicated token for each ciphertext, and return all tokens to the host. Everspaugh et al. [Eve+17b] provide a systematic treatment for such schemes i.e., defining the desirable security properties and presenting provably secure solutions. Their focus is on *authenticated* encryption schemes, and thus CCA security and ciphertext integrity (CTXT) are required and achieved by their construction.

While ciphertext-dependent schemes allow for fine-grained control of which ciphertexts should be re-encrypted towards the new key, they are clearly far less efficient and convenient for the data owner than *ciphertext-independent* ones. In ciphertext-independent schemes, the update token only depends on the new and old key and allows to re-encrypt *all* ciphertexts. The idea of ciphertext-independent schemes was informally introduced by Boneh et al. [Bon+13] and recently Lehmann and Tackmann [LT18] provided a rigorous treatment of their formal security guarantees. The broader applicability of update tokens in ciphertext-independent schemes is an inherent challenge for achieving strong security properties though: as a single token can be used to update *all* ciphertexts, the corruption of such a token gives the adversary significantly more power than the corruption of a ciphertext-dependent token. As a consequence, the ciphertext-independent schemes proposed so far only achieve CPA security instead of CCA, and did not guarantee any integrity protection [LT18].

	Encrypt-and-MAC (E&M, Sec. 3)	Naor-Yung (NYUAE, Sec. 4)
Confidentiality	CCA	RCCA
Integrity	ciphertext integrity	plaintext integrity
ReEnc algorithm	deterministic	probabilistic
ReEnc oracle	honestly derived ciphertexts only	arbitrary ciphertexts

Fig. 1. Overview of the core differences of our two main schemes and considered settings.

Updatable encryption needs (stronger) integrity protection. Given that updatable encryption targets a cloud-based deployment setting where encrypted data is outsourced to an (untrusted) host, neglecting the integrity protection of the outsourced data is a dangerous shortcoming. For instance, the host might hold encrypted financial or medical data of the data owner. Clearly, a temporary security breach into the host should not allow the adversary to create new and valid ciphertexts that will temper with the owners' records. For the targeted setting of ciphertext-independent schemes no notion of (ciphertext) integrity was proposed so far, and the encryption scheme presented in [LT18] is extremely vulnerable to such attacks: their symmetric updatable encryption scheme (termed RISE) is built from (public-key) ElGamal encryption, which only uses the public key in the update token. However, a single corruption of the update token will allow the data host to create valid ciphertexts of arbitrary messages of his choice.

For the ciphertext-dependent setting, the scheme by Everspaugh et al [Eve+17b] does provide ciphertext-integrity, but only against a weak form of attacks: the security definition for their CTXT notion does not allow the adversary to obtain re-encryptions of *maliciously* formed ciphertexts. That is, the model restricts queries to the re-encryption oracle to honestly generated ciphertexts that the adversary has received from previous (re)encryption oracle queries. Thus, integrity protection is only guaranteed against passive adversaries. Again, given the cloud deployment setting in which updatable encryption is used in, assuming that an adversary that breaks into the host will behave honestly and does not temper with any ciphertexts is a critical assumption.

Our Contributions. In this work we address the aforementioned shortcomings for ciphertext-*independent* updatable encryption and present schemes that provide significantly stronger security than existing solutions. First, we formally define the desirable security properties of (R)CCA security, ciphertext (CTXT) and plaintext integrity (PTXT) for key-evolving encryption schemes. Our definitions allow the adversary to *adaptively* corrupt the secret keys or update tokens of the current and past epochs, as long as it does not empower him to trivially win the respective security game. We then propose two constructions: the first achieves CCA and CTXT security (against passive re-encryption attacks), and the second scheme realizes RCCA and PTXT security against active attacks. Both schemes make use of a generic (proof) strategy that derives the security of the updatable scheme from simple properties of the underlying *static* primitives, which greatly simplifies the design for such updatable encryption schemes. In more detail, our contributions are as follows:

Scheme	Assumption	Ciph. indep.	arbitr. ReEnc	IND	INT	UN-LINK	$\lvert c\rvert$	(Re)Enc	Dec
BLMR [Bon+15]	DDH (+ ROM)	✗	✗	(?)	✗	(?)	$2\mathbb{G}^*$	$2\mathbb{G}$	$2\mathbb{G}$
EPRS [Eve+17b]	DDH + ROM	✗	(✗)	CPA	CTXT	✓	$2\mathbb{G}^*$	$2\mathbb{G}$	$2\mathbb{G}$
RISE [LT18]	DDH	✓	✗	CPA	✗	✓	$2\mathbb{G}$	$2\mathbb{G}$	$2\mathbb{G}$
E&M Sec. 3	DDH + ROM	✓	✗	CCA	CTXT	✓	$3\mathbb{G}$	$3\mathbb{G}$	$3\mathbb{G}$
NYUE Sec. 4	SXDH	✓	✓	RCCA	✗	✓	$(34, 34)$	$(60, 70)$	$22e$
NYUAE Sec. 4	SXDH	✓	✓	RCCA	PTXT	✓	$(58, 44)$	$(110, 90)$	$29e$

Fig. 2. Comparison of ciphertext-independent and -dependent updatable encryption schemes. The second set of columns states the achieved security notions, and whether security against arbitrary (opposed to honest) re-encryption attacks is achieved. For EPRS, security against arbitrary re-encryption attacks is only considered for confidentiality, not for integrity. For BLMR, it was shown that a security proof for confidentiality is unlikely to exist [Eve+17b], and the formal notion of unlinkability of re-encryptions was only introduced later. The final set of columns states the efficiency in terms of ciphertext size and costs for (re-)encryption and decryption in the number of exponentiations and pairings. Tuples (x, y) specify x (resp. y) elements/exponentiations in $\mathbb{G}_1$ (resp. $\mathbb{G}_2$) in case of underlying pairing groups, and a pairing is denoted by e. (Re)encryption and decryption costs for NYUE and NYUAE are approximate. The ciphertext size is given for messages represented as a single group element (in $\mathbb{G}$ or $\mathbb{G}_1$). BLMR and EPRS support encryption of arbitrary size message with the ciphertext size growing linearly with the message blocks.

CCA and CTXT Secure Ciphertext-Independent Updatable Encryption. Our first updatable encryption applies the Encrypt-and-MAC (E&M) transformation to primitives which are key-rotatable and achieves CCA and CTXT security. Using Encrypt-*and*-MAC is crucial for the updatability as we need direct access to both the ciphertext and the MAC. In order to use E&M, which is *not* a secure transformation for authenticated encryption in general, we require a one-to-one mapping between message-randomness pairs and ciphertexts as well as the decryption function to be randomness-recoverable. By applying a PRF on both, the message and the encryption randomness, we obtain the desired ciphertext integrity. Interestingly, we only need the underlying encryption and PRF to be secure w.r.t. their standard, static security notions and derive security for the updatable version of E&M from additional properties we introduce for the update token generation.

An essential property of this first scheme is that its re-encryptions are *deterministic*. This enables us to define and realize a meaningful CCA security notion, as the determinism allows the challenger to keep track of re-encryptions of the challenge ciphertext and prevent decryption of such updates. Similar to the CCA-secure (ciphertext-dependent) scheme of [Eve+17b], we only achieve security against passive re-encryption attacks, i.e., where the re-encryption oracle in the security game can only be queried on honestly generated ciphertexts.

RCCA and PTXT Security Against Malicious Re-Encryption Attacks. Our second scheme then provides strong security against active re-encryption attacks. On a high-level, we use the Naor-Yung approach [NY90] that lifts (public-key) CPA to CCA security by encrypting each message under two public keys and appending a NIZK that both ciphertexts encrypt the same message. The crucial benefit of this approach is that it allows for *public verifiability* of ciphertexts, and thus for any re-encryption it can first be checked that the provided ciphertext is valid — which then limits the power of malicious re-encryption attacks. To lift the approach to an updatable encryption scheme, we rely on the key-rotatable CPA-secure encryption RISE [LT18] and GS proofs [GS12; EG14] that exhibit the malleability necessary for rotating the associated NIZK proof.

A consequence of this approach is that re-encryptions are now probabilistic (as in RISE) and ciphertexts are re-randomizable in general. Therefore, CCA and CTXT are no longer achievable, and we revert to Replayable CCA (RCCA) and plaintext integrity. Informally, RCCA is a relaxed variant of CCA security that ensures confidentiality for all ciphertexts that are not re-randomization of the challenge ciphertext [CKN03]. Plaintext integrity is a weaker notion than ciphertext integrity, as forging ciphertexts is now trivial, but still guarantees that an adversary can not come up with valid ciphertexts for *fresh* messages.

In Fig. 1 we provide an overview of both solutions and their settings, and Fig. 2 gives a compact comparison between our new schemes and the existing ones.

Generic (Proof) Transformation & Unlinkability of Re-Encryption. The security models for updatable encryption are quite involved, which in turn makes proving security in these models rather cumbersome [Eve+17b; LT18]. A core contribution of our work is a generic transformation that yields a surprisingly simple blueprint for building updatable encryption: We show that it is sufficient to consider the underlying encryption and the key-rotation capabilities (almost) separately. That is, we only require the underlying scheme – provided by the $\mathsf{Enc}, \mathsf{Dec}$ algorithms in isolation – to satisfy standard security. In addition we need re-encryption to produce ciphertexts that are indistinguishable from fresh encryption and token generation to be *simulatable*. The latter allows us to produce "fake" tokens when we are dealing with a static CCA/RCCA game, and the former is used to answer re-encryption oracle calls in the security game with decrypt-then-encrypt calls. Further, we leverage the fact that all ciphertext-independent schemes so far are *bi-directional*, i.e., ciphertexts can also be downgraded. This property comes in very handy in the security proof as it essentially allows to embed a static-CCA/RCCA challenger in one epoch, and handle queries in all other epochs by rotating ciphertexts back-and-forth to this "challenge" epoch.

The notion of indistinguishability of re-encryptions and fresh encryptions (termed *perfect re-encryption*) that we define also has another very nice side-effect: it implies the property of re-encryption unlinkability as introduced in [Eve+17b; LT18]. Both works propose a security notion that guarantees that a re-encrypted ciphertext can no longer be linked to its old version, which captures that the full ciphertext must get refreshed during an update. We adapt this unlinkability notion to the CCA and RCCA setting of our work and show that perfect re-encryption (in combination with CCA resp. CPA security) implies such unlinkability. Both of our schemes satisfy this strong security notion.

Other Related Work. Recently, Jarecki et al. [JKR18] proposed an updatable and CCA secure encryption scheme in the context of an Oblivious Key Management Systems (KMS). The KMS is an external service that hosts the secret key, whereas the data owner stores all ciphertexts and adaptively decrypts them with the help of the KMS. Thus, their setting is considerably different to our notion of updatable encryption where the ciphertexts are outsourced, and the secret is managed by the data owner.

Another primitive that is highly related to updatable encryption is proxy re-encryption (PRE). In a recent work, Fuchsbauer et al. [Fuc+18] show how to lift selectively secure PRE to adaptive security without suffering from an exponential loss when using straightforward approaches. Their overall idea is similar to our generic transformation, as it also relies on additional properties of the re-encryption procedure that facilitate the embedding of the static challenger. The different overall setting makes their work rather incomparable to ours: we exploit bi-directional behaviour of updates, whereas [Fuc+18] focuses on uni-directional schemes, and we consider a symmetric key setting whereas the PRE's are public-key primitives. In fact, our security proofs are much tighter (partially due to these differences). We conjecture that our techniques can be applied to obtain adaptive security with polynomial security loss for a class of PRE schemes, cf. App. I. This would improve upon the superpolynomial loss in [Fuc+18].

Organisation. We start our paper by recalling the necessary standard building blocks and the generic syntax of updatable encryption in Sec. 2. In Sec. 3, we then present our formal definitions for CCA and CTXT secure updatable encryption, tailored to our setting of schemes with deterministic re-encryption and covering passive re-encryption attacks. This section also contains our generic transformation for achieving these notions from the static security of the underlying building blocks, and our Encrypt-and-MAC construction that utilizes this generic approach. In Sec. 4 we then introduce RCCA and PTXT security against active re-encryption attacks and present our Naor-Yung inspired scheme. Since our generic transformation immediately implies the unlinkability property UP-REENC introduced in [Eve+17b; LT18] we refer the formal treatment of this notion to App. A.

2 Preliminaries

In this section we introduce our notational conventions and all necessary (standard) building blocks along with their security definitions.

2.1 Notation

We denote the security parameter by κ. All schemes and building blocks in this paper make use of some implicit PPT algorithm $pp \leftarrow \mathsf{GenPP}(1^\kappa)$ which on input of the security parameter 1^κ outputs some public parameters pp. The public parameters, e.g., include a description of the cyclic groups and generators we use. We assume for our security definitions that pp also contains the security parameter. For the sake of simplicity, we omit GenPP in all definitions including security experiments. When composing building blocks as in our Encrypt-*and*-MAC construction, for example, the same GenPP algorithm is assumed for all those building blocks and the output pp is shared between them.

By $\mathbb{G}$ we denote a commutative group and by $(e, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ a pairing group. All groups are of prime order p. The integers modulo p are denoted $\mathbb{F}_p$. We use *additive* notation for groups, in particular the well-established *implicit* representation introduced in [Esc+13]. That is, we write $[1]$ for the generator $g \in \mathbb{G}$ and $[x] = xg$ (in multiplicative notation, g^x). For pairing groups, we write $[1]_1$, $[1]_2$ and $[1]_T$ and we require that $e([1]_1, [1]_2) = [1]_T$. We define $\mathbb{G}^\times := \mathbb{G} \setminus \{[0]\}$. By $\mathrm{supp}(X)$ we denote the support of a random variable X, i.e. the set of outcomes with positive probability.

2.2 Symmetric and Tidy Encryption

We use the following definition of a symmetric encryption scheme, where the existence of a system parameter generation algorithm GenSP reflects the fact, that we partially rely on primitives with public parameters (like a Groth-Sahai CRS) for our constructions.

Definition 1. *A symmetric encryption scheme* $\mathsf{SKE} = (\mathsf{GenSP}, \mathsf{GenKey}, \mathsf{Enc}, \mathsf{Dec})$ *is defined by the following PPT algorithms*

$\mathsf{SKE.GenSP}(pp)$ *returns system parameters sp. We treat sp as implicit inputs for the following algorithms.*
$\mathsf{SKE.GenKey}(sp)$ *returns a key k.*
$\mathsf{SKE.Enc}(k, m; r)$ *returns a ciphertext c for message m, key k and randomness r.*
$\mathsf{SKE.Dec}(k, c)$ *returns the decryption m of c. ($m = \perp$ indicates failure.)*

We assume that the system parameters fix not only the key space $\mathcal{K}_{sp}$, but also the ciphertext space $\mathcal{C}_{sp}$, message space $\mathcal{M}_{sp}$ and randomness space $\mathcal{R}_{sp}$. Also, we assume that membership in $\mathcal{C}_{sp}$ and $\mathcal{M}_{sp}$ can be efficiently tested.

Tidy Encryption. Our construction of an updatable encryption scheme with deterministic reencryption resorts to tidy encryption. For this purpose, we use the following definition which is a slightly adapted version of the definition in [NRS14].

Definition 2. *A symmetric encryption scheme* SKE *is called **randomness-recoverable** if there is an associated efficient deterministic algorithm* $\mathsf{RDec}(k, c)$ *such that*

$$\forall k, m, r\colon \mathsf{RDec}(k, \mathsf{Enc}(k, m; r)) = (m, r). \tag{1}$$

We call a randomness-recoverable SKE ***tidy*** *if*

$$\forall k, c \colon \mathsf{RDec}(k, c) = (m, r) \implies \mathsf{Enc}(k, m; r) = c. \tag{2}$$

In other words, SKE *is tidy if* Enc *and* RDec *are bijections (for a fixed key) between message-randomness pairs and valid ciphertexts (i.e. ciphertexts which do not decrypt to* $\perp$*).*[3]

Indistinguishability Notions. For our constructions, we consider a number of slight variations of the standard security notions IND-CPA and IND-CCA security.

One such variation is IND-RCCA security [CKN03] which relaxes IND-CCA in the sense that it is not considered an attack if a ciphertext can be transformed into a new ciphertext of the same message. Hence, the RCCA decryption oracle refuses to decrypt any ciphertext containing one of the challenge messages.

Furthermore, we consider CPA, CCA, and RCCA security under key-leakage. Here the adversary is additionally given $\mathsf{leak}(k)$ as input, where leak is some function on the key space. This leakage reflects the fact that in one of our constructions (Sec. 4.2), that actually relies on public-key primitives, the corresponding public keys need to be leaked to the adversary. So we would have $k = (sk, pk)$ and $\mathsf{leak}(k) = pk$ in this case. For the deterministic construction in Sec. 3.2 we do not consider key-leakage, i.e., $\mathsf{leak}(k) = \perp$.

Finally, we can define (stronger) real or random variants (IND\$-CPA/CCA) of the former notions. Here, the adversary provides a single challenge message and the challenger responds with either an encryption of this message or a randomly chosen ciphertext.

Def. 3 compactly formalizes the security notions sketched above.

Definition 3. *Let* SKE *be a secret key encryption scheme. Let* $\mathsf{leak} \colon \mathcal{K} \to \mathcal{L}$ *be a leakage function. We call* SKE *IND-X secure, where* $X \in \{CPA, CCA, RCCA\}$*, under key-leakage* leak*, if for every efficient PPT adversary* $\mathcal{A}$*, the advantage*

$$\mathsf{Adv}^{\textit{ind-X}}_{\mathsf{SKE},\mathcal{A}}(\kappa) := \left|\Pr[\mathsf{Exp}^{\textit{ind-X}}_{\mathsf{SKE},\mathcal{A}}(\kappa, 0) = 1] - \Pr[\mathsf{Exp}^{\textit{ind-X}}_{\mathsf{SKE},\mathcal{A}}(\kappa, 1) = 1]\right|$$

in the experiment described in Fig. 3 is negligible. Analogous to IND-X, we define IND\$-X security for $X \in \{CPA, CCA\}$*, with the experiments also described in Fig. 3. IND\$-X is the strictly stronger notion, i.e., it implies IND-X.*

Integrity Notions. We consider both plaintext (PTXT) and ciphertext (CTXT) integrity. In the PTXT experiment, the adversary wins if it is able to output a valid ciphertext for a fresh plaintext, i.e., a ciphertext that decrypts to a plaintext for which it has not queried the encryption oracle before. In the CTXT experiment, in order to win, the adversary just needs to output a valid and fresh

[3] Since encryption of $\perp$ also yields $\perp$, Eq. (2) trivially holds for invalid ciphertexts.

```
**Experiment** $\mathsf{Exp}^{\mathsf{ind\text{-}X}}_{\mathsf{SKE},\mathcal{A}}(\kappa, b)$
    $sp \leftarrow \mathsf{GenSP}(pp)$; $k \leftarrow \mathsf{GenKey}(sp)$;
    $(m_0^*, m_1^*, state) \leftarrow \mathcal{A}^{\mathsf{Enc},\mathsf{Dec}}(pp, sp, \mathsf{leak}(k))$;
    abort if $|m_0^*| \neq |m_1^*|$ or $m_0^*, m_1^* \notin \mathcal{M}_{sp}$
    $c^* \leftarrow \mathsf{Enc}(k, m_b^*)$;
    **return** $\mathcal{A}^{\mathsf{Enc},\mathsf{Dec}}(state, c^*) \stackrel{?}{=} b$
```

```
**Experiment** $\mathsf{Exp}^{\mathsf{ind\$\text{-}X}}_{\mathsf{SKE},\mathcal{A}}(\kappa, b)$
    $sp \leftarrow \mathsf{GenSP}(pp)$; $k \leftarrow \mathsf{GenKey}(sp)$;
    $(m^*, state) \leftarrow \mathcal{A}^{\mathsf{Enc},\mathsf{Dec}}(pp, sp, \mathsf{leak}(k))$;
    abort if $m^* \notin \mathcal{M}_{sp}$
    $c_0^* \leftarrow \mathsf{Enc}(k, m^*)$; $c_1^* \leftarrow_{\mathrm{R}} \mathcal{C}$; $c^* := c_b^*$
    **return** $\mathcal{A}^{\mathsf{Enc},\mathsf{Dec}}(state, c^*) \stackrel{?}{=} b$
```

Fig. 3. The encryption oracle $\mathsf{Enc}(m)$ returns $c \leftarrow_{\mathrm{R}} \mathsf{Enc}(k, m)$. The decryption oracle $\mathsf{Dec}(m)$ computes $m \leftarrow_{\mathrm{R}} \mathsf{Dec}(k, c)$ but then behaves differently depending on the notion. For CPA, $\mathsf{Dec}(c)$ always returns $\perp$. For CCA, $\mathsf{Dec}(c)$ returns m except if $c = c^*$, in which case it returns $\perp$. For RCCA, $\mathsf{Dec}(c)$ returns m except if $m \in \{m_0^*, m_1^*\}$ in which case it returns `invalid`. Note that `invalid` $\neq \perp$, i.e. $\mathcal{A}$ learns that (one of) the challenge messages is encrypted in c. Everything else is unchanged.

ciphertext, i.e., one not resulting from a previous call to the encryption oracle. In both experiments, the adversary is equipped with a decryption oracle instead of an oracle that just tests the validity of ciphertexts. For CTXT, this actually makes no difference. For PTXT, however, there are (pathological) insecure schemes which are only secure w.r.t. validity oracles. Again, we consider variants of these integrity notions under key-leakage. Def. 4 formalizes these notions.

Definition 4. *Let* SKE *be a symmetric encryption scheme. Let* $\mathsf{leak}\colon \mathcal{K} \rightarrow \mathcal{L}$ *be a leakage function. The INT-CTXT as well as the INT-PTXT experiments are defined in Fig. 4. We call* SKE *INT-CTXT secure under (key-)leakage* leak *if the advantage* $\mathsf{Adv}^{\mathit{int\text{-}ctxt}}_{\mathsf{SKE},\mathcal{A}}(\kappa) := \Pr[\mathsf{Exp}^{\mathit{int\text{-}ctxt}}_{\mathsf{SKE},\mathcal{A}}(\kappa) = 1]$ *is negligible. Similarly, we call* SKE *INT-PTXT secure under (key-)leakage* leak *if the advantage* $\mathsf{Adv}^{\mathit{int\text{-}ptxt}}_{\mathsf{SKE},\mathcal{A}}(\kappa) := \Pr[\mathsf{Exp}^{\mathit{int\text{-}ptxt}}_{\mathsf{SKE},\mathcal{A}}(\kappa) = 1]$ *is negligible.*

```
**Experiment** $\mathsf{Exp}^{\mathsf{int\text{-}ptxt}}_{\mathsf{SKE},\mathcal{A}}(\kappa)$
    $\mathbf{M} = \emptyset$; $\mathbf{Q} = \emptyset$;
    $sp \leftarrow \mathsf{GenSP}(pp)$; $k \leftarrow \mathsf{GenKey}(sp)$;
    $c \leftarrow \mathcal{A}^{\mathsf{Enc},\mathsf{Dec}}(pp, sp, \mathsf{leak}(k))$;
    $m \leftarrow \mathsf{Dec}(k, c)$;
    **return** 0 iff $m \in \mathbf{M}$ or $m = \perp$
```

```
**Experiment** $\mathsf{Exp}^{\mathsf{int\text{-}ctxt}}_{\mathsf{SKE},\mathcal{A}}(\kappa)$
    $\mathbf{M} = \emptyset$; $\mathbf{Q} = \emptyset$;
    $sp \leftarrow \mathsf{GenSP}(pp)$; $k \leftarrow \mathsf{GenKey}(sp)$;
    $c \leftarrow \mathcal{A}^{\mathsf{Enc},\mathsf{Dec}}(pp, sp, \mathsf{leak}(k))$;
    $m \leftarrow \mathsf{Dec}(k, c)$;
    **return** 0 iff $c \in \mathbf{Q}$ or $m = \perp$
```

Fig. 4. The INT-PTXT (left) and INT-CTXT (right) games. The encryption oracle $\mathsf{Enc}(m)$ returns $c \leftarrow \mathsf{Enc}(k, m)$ and adds m to the list of queried messages $\mathbf{M}$ and adds c to the list of queried ciphertexts $\mathbf{Q}$. The oracle $\mathsf{Dec}(c)$ returns $\mathsf{Dec}(k, c)$.

2.3 Updatable Encryption

Roughly, an updatable encryption scheme is a symmetric encryption scheme which offers an additional re-encryption functionality that moves ciphertexts from an old to a new key.

The encryption key evolves with *epochs*, and the data is encrypted with respect to a specific epoch e, starting with $e = 0$. When moving from epoch e to epoch $e + 1$, one first creates a new key k_{e+1} via the $\mathsf{UE.GenKey}$ algorithm and then invokes the token generation algorithm $\mathsf{UE.GenTok}$ on the old and new key, k_e and k_{e+1}, to obtain the update token Δ_{e+1}. The update token Δ_{e+1} allows to update all previously received ciphertexts from epoch e to $e + 1$ using the re-encryption algorithm $\mathsf{UE.ReEnc}$.

Definition 5 (Updatable Encryption). *An **updatable encryption scheme** UE is a tuple $(\mathsf{GenSP}, \mathsf{GenKey}, \mathsf{GenTok}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{ReEnc})$ of PPT algorithms defined as:*

$\mathsf{UE.GenSP}(pp)$ *is given the public parameters and returns some system parameters sp. We treat the system parameters as implicit input to all other algorithms.*
$\mathsf{UE.GenKey}(sp)$ *is the key generation algorithm which on input of the system parameters outputs a key $k \in \mathcal{K}_{sp}$.*
$\mathsf{UE.GenTok}(k_e, k_{e+1})$ *is given two keys k_e and k_{e+1} and outputs some update token Δ_{e+1}.*
$\mathsf{UE.Enc}(k_e, m)$ *is given a key k_e and a message $m \in \mathcal{M}_{sp}$ and outputs some ciphertext $c_e \in \mathcal{C}_{sp}$.*
$\mathsf{UE.Dec}(k_e, c_e)$ *is given a key k_e and a ciphertext c_e and outputs some message $m \in \mathcal{M}_{sp}$ or $\perp$.*
$\mathsf{UE.ReEnc}(\Delta_e, c_{e-1})$ *is given an update token Δ_e and a ciphertext c_{e-1} and returns an updated ciphertext c_e.*

*Given UE, we call $\mathsf{SKE} = (\mathsf{GenSP}, \mathsf{GenKey}, \mathsf{Enc}, \mathsf{Dec})$ the **underlying (standard) encryption scheme**.*
*UE is called **correct** if SKE is correct and $\forall sp \leftarrow \mathsf{GenSP}(pp), \forall k^{\text{old}}, k^{\text{new}} \leftarrow \mathsf{GenKey}(sp), \forall \Delta \leftarrow \mathsf{GenTok}(k^{\text{old}}, k^{\text{new}}), \forall c \in \mathcal{C}\colon \mathsf{Dec}(k^{\text{new}}, \mathsf{ReEnc}(\Delta, c)) = \mathsf{Dec}(k^{\text{old}}, c)$.*

We will use both notations, i.e., k_e, k_{e+1} and $k^{\text{old}}, k^{\text{new}}$ synonymous throughout the paper, where the latter omits the explicit epochs e whenever they are not strictly necessary and we simply want to refer to keys for two consecutive epochs.

In our first construction, the re-encryption algorithm $\mathsf{UE.ReEnc}$ will be a deterministic algorithm, whereas for our second scheme the ciphertexts are updated in a probabilistic manner. We define the desired security properties (UP-IND-CCA, UP-INT-CTXT) for updatable encryption schemes with deterministic re-encryption and (UP-IND-RCCA, UP-INT-PTXT) for schemes with a probabilistic $\mathsf{UE.ReEnc}$ algorithm in the following sections.

RISE. In [LT18], Lehmann and Tackmann proposed an updatable encryption scheme called RISE which is essentially (symmetric) ElGamal encryption with added update functionality. We use RISE as a building block in our RCCA and PTXT secure scheme. Please refer to App. F.1 for a description of RISE in our setting.

3 CCA and CTXT Secure Updatable Encryption

In this section, we first introduce the considered confidentiality and integrity definitions for updatable encryption with deterministic re-encryption (Sec. 3.1). This is followed by a generic transformation that allows to realize these notions from simple, static security properties (Sec. 3.2). Finally, we describe a Encrypt-and-MAC construction that can be used in this transformation and give instantiations of its building blocks (Sec. 3.3).

3.1 Security Model

We follow the previous work on updatable encryption and require confidentiality of ciphertexts in the presence of temporary key and token corruption, covering both forward and post-compromise security. This is formalized through the indistinguishability-based security notion UP-IND-CCA which can be seen as the extension of the standard CCA game to the context of updatable encryption. In addition to confidentiality, we also require *integrity* of ciphertexts, which we formulate via our UP-INT-CTXT definition.

Both security notions are defined through experiments run between a challenger and an adversary $\mathcal{A}$. Depending on the notion, the adversary may issue queries to different oracles. At a high level, $\mathcal{A}$ is allowed to adaptively corrupt arbitrary choices of secret keys and update tokens, as long as they do not allow him to trivially win the respective security game.

Oracles and CCA Security. Our UP-IND-CCA notion is essentially the regular IND-CCA definition where the adversary is given additional oracles that capture the functionality inherent to an updatable encryption scheme.

These oracles are defined below and are roughly the same in all our security definitions. We describe the oracles in the context of our UP-IND-CCA security game, which needs some extra restrictions and care in order to prevent a decryption of the challenge ciphertext. When introducing our other security notions, we explain the differences w.r.t. the oracles presented here. An overview of all oracles and their differences in our security games is given in Figure 5.

The oracles may access the global state $(sp, k_e, \Delta_e, \mathbf{Q}, \mathbf{K}, \mathbf{T}, \mathbf{C}^*)$ which is initialized via $\mathsf{Init}(pp)$ as follows:

$\mathsf{Init}(pp)$**:** This initializes the state of the challenger as $(sp, k_0, \Delta_0, \mathbf{Q}, \mathbf{K}, \mathbf{T}, \mathbf{C}^*)$ where $e \leftarrow 0$, $sp \leftarrow_{\mathrm{R}} \mathsf{UE.GenSP}(pp)$ $k_0 \leftarrow_{\mathrm{R}} \mathsf{UE.GenKey}(sp)$, $\Delta_0 \leftarrow \perp$, $\mathbf{Q} \leftarrow \emptyset, \mathbf{K} \leftarrow \emptyset$, $\mathbf{T} \leftarrow \emptyset$ and $\mathbf{C}^* \leftarrow \emptyset$.

The current epoch is denoted as e, and the list $\mathbf{Q}$ contains "honest" ciphertexts which the adversary has obtained entirely through the Enc or ReEnc oracles. The challenger also keeps sets $\mathbf{K}, \mathbf{T}$ and $\mathbf{C}^*$ (all initially set to $\emptyset$) that are used to keep track of the epochs in which $\mathcal{A}$ corrupted a secret key ($\mathbf{K}$), token ($\mathbf{T}$), or obtained a re-encryption of the challenge-ciphertext ($\mathbf{C}^*$). These will later be used to check whether the adversary has made a combination of queries that

	CCA	CTXT	RCCA	PTXT
Next()	moves to the next epoch $e+1$ by generating new key and update token			
Enc(m)	returns encryption c of message m under current epoch key k_e			
	stores ciphertext (e, c) in $\mathbf{Q}$		stores (e, m, c) in $\mathbf{Q}$	stores (e, m) in $\mathbf{Q}$
Dec(c)	returns decryption m of ciphertext c under current epoch key k_e			
	ignores c if it is the challenge c^* or a re-encryption of c^*	—	ignores c if it decrypts to m_0 or m_1	—
ReEnc(i, c)	returns re-encryption c_e of ciphertext c from epoch i into current epoch e			
	allows only ciphertexts in $\mathbf{Q}$ and derivations of c^*	allows only ciphertexts in $\mathbf{Q}$	allows arbitrary ciphertexts	
	if c is c^* or a re-encryption of c^* it adds epoch e to $\mathbf{C}^*$	—	if c decrypts to m_0 or m_1 and $(i, *, c) \notin \mathbf{Q}$ it adds epoch e to $\mathbf{C}^*$	—
Corrupt(x, i)	returns either k_i (if $\mathsf{x} = \mathsf{key}$) or Δ_i (if $\mathsf{x} = \mathsf{token}$) for $0 \le i \le e$			

Fig. 5. Overview of oracles and their restrictions in our different security games. $\mathbf{C}^*$ is the set of challenge-equal epochs used in the CCA and RCCA games, c^* denotes the challenge ciphertext in the CCA game, and m_0, m_1 are the two challenge plaintexts chosen by $\mathcal{A}$ in the RCCA game. $\mathbf{Q}$ is the set of queried (re)encryptions.

trivially allow him to decrypt the challenge ciphertext. For our integrity notions UP-INT-CTXT and UP-INT-PTXT we will omit the set $\mathbf{C}^*$ that is related to the challenge ciphertext. Moreover, the predicate isChallenge, which identifies challenge-related ciphertexts, unnecessary for integrity notions. We implicitly assume that the oracles only proceed when the input is valid, e.g,. for the epoch i it must hold that $0 \le i < e$ for re-encryption queries, and $0 \le i \le e$ for corruption queries. The decryption or re-encryption oracle will only proceed when the input ciphertext is "valid" (which will become clear in the oracle definitions given below). For incorrect inputs, the oracles return `invalid`.

Next(): Runs $k_{e+1} \leftarrow_{\mathrm{R}} \mathsf{UE.GenKey}(sp)$, $\Delta_{e+1} \leftarrow_{\mathrm{R}} \mathsf{UE.GenTok}(k_e, k_{e+1})$, adds (k_{e+1}, Δ_{e+1}) to the global state and updates the current epoch to $e \leftarrow e + 1$.
Enc(m): Returns $c \leftarrow_{\mathrm{R}} \mathsf{UE.Enc}(k_e, m)$ and sets $\mathbf{Q} \leftarrow \mathbf{Q} \cup \{(e, c)\}$.
Dec(c): If $\mathsf{isChallenge}(k_e, c) = \mathtt{false}$, it returns $m \leftarrow \mathsf{UE.Dec}(k_e, c)$.
ReEnc(i, c): The oracle returns the re-encryption of c from the i-th into the current epoch e. That is, it returns c_e that is computed iteratively through $c_\ell \leftarrow \mathsf{UE.ReEnc}(\Delta_\ell, c_{\ell-1})$ for $\ell = i + 1, \ldots, e$ and $c_i \leftarrow c$. The oracle accepts only ciphertexts c that are honestly generated, i.e., either $(i, c) \in \mathbf{Q}$ or $\mathsf{isChallenge}(k_i, c) = \mathtt{true}$. It also updates the global state depending on whether the query is a challenge ciphertext or not:
- If $(i, c) \in \mathbf{Q}$, set $\mathbf{Q} \leftarrow \mathbf{Q} \cup \{(e, c_e)\}$.
- If $\mathsf{isChallenge}(k_i, c) = \mathtt{true}$, set $\mathbf{C}^* \leftarrow \mathbf{C}^* \cup \{e\}$.

$\mathsf{Corrupt}(\{\mathsf{key}, \mathsf{token}\}, i)$**:** This oracle models adaptive and retroactive corruption of keys and tokens, respectively:

- Upon input (key, i), the oracle sets $\mathbf{K} \leftarrow \mathbf{K} \cup \{i\}$ and returns k_i.
- Upon input (token, i), the oracle sets $\mathbf{T} \leftarrow \mathbf{T} \cup \{i\}$ and returns Δ_i.

Finally, we define UP-IND-CCA security as follows, requesting the adversary after engaging with the oracles defined above, to detect whether the challenge ciphertext $c^* \leftarrow_{\mathrm{R}} \mathsf{UE.Enc}(k_e, m_b)$ is an encryption of m_0 or m_1. The adversary wins if he correctly guesses the challenge bit b and has not corrupted the secret key in any challenge-equal epoch. In the following we explain how we define the set of challenge-equal epochs $\widehat{\mathbf{C}}^*$ and prevent trivial wins.

Definition 6. *An updatable encryption scheme* UE *(with deterministic re-encryption) is called UP-IND-CCA secure if for any PPT adversary $\mathcal{A}$ the advantage*

$$\mathsf{Adv}^{\textit{up-ind-cca}}_{\mathsf{UE},\mathcal{A}}(pp) := \left|\Pr[\mathsf{Exp}^{\textit{up-ind-cca}}_{\mathsf{UE},\mathcal{A}}(pp, 0) = 1] - \Pr[\mathsf{Exp}^{\textit{up-ind-cca}}_{\mathsf{UE},\mathcal{A}}(pp, 1) = 1]\right|$$

is negligible in κ.

```
Experiment $\mathsf{Exp}^{\mathsf{up\text{-}ind\text{-}cca}}_{\mathsf{UE},\mathcal{A}}(pp, b)$
    $(sp, k_0, \Delta_0, \mathbf{Q}, \mathbf{K}, \mathbf{T}, \mathbf{C}^*) \leftarrow \mathsf{Init}(pp)$
    $(m_0, m_1, state) \leftarrow_{\mathrm{R}} \mathcal{A}^{\mathsf{Enc},\mathsf{Dec},\mathsf{Next},\mathsf{ReEnc},\mathsf{Corrupt}}(sp)$
    proceed only if $|m_0| = |m_1|$ and $m_0, m_1 \in \mathcal{M}_{sp}$
    $c^* \leftarrow_{\mathrm{R}} \mathsf{UE.Enc}(k_e, m_b)$, $\mathbf{C}^* \leftarrow \{e\}$, $e^* \leftarrow e$
    $b' \leftarrow_{\mathrm{R}} \mathcal{A}^{\mathsf{Enc},\mathsf{Dec},\mathsf{Next},\mathsf{ReEnc},\mathsf{Corrupt}}(c^*, state)$
    return $b'$ if $\mathbf{K} \cap \widehat{\mathbf{C}}^* = \emptyset$, i.e. $\mathcal{A}$ did not trivially win. (Else abort.)
```

Preventing decryption of an updated challenge ciphertext. We use a predicate $\mathsf{isChallenge}(k_i, c)$ to detect attempts of decrypting the challenge ciphertext c^* or a re-encryption thereof. Whether a given ciphertext is a re-encryption of the challenge c^* can be tested efficiently by exploiting the deterministic behaviour of the re-encryption algorithm, and the fact that all secret keys and token are known to the challenger. This approach has also been used to define CCA-security for ciphertext-*dependent* schemes by Everspaugh et al. [Eve+17b].

For the following definition, recall that c^* is the challenge ciphertext obtained in epoch e^*, or $c^* = \perp$ if the adversary has not made the challenge query yet.

$\mathsf{isChallenge}(k_i, c)$:

- If $i = e^*$ and $c^* = c$, return `true`.
- If $i > e^*$ and $c^* \neq \perp$, return `true` if $c_i^* = c$ where c_i^* for epoch i is computed iteratively as $c_\ell^* \leftarrow \mathsf{UE.ReEnc}(\Delta_{\ell+1}, c_\ell^*)$ for $\ell = e^*, \ldots, i$.
- Else return `false`.

Defining trivial wins. A crucial part of the definition is to capture the information the adversary has learned through his oracle queries. In particular, any corruption of the token Δ_{e+1} in an epoch after where the adversary has learned the challenge ciphertext c_e^* (directly or via a re-encryption) will enable to adversary to further update the challenge ciphertext into the next epoch $e + 1$. The goal of capturing this inferable information, is to exclude adversaries following a trivial winning

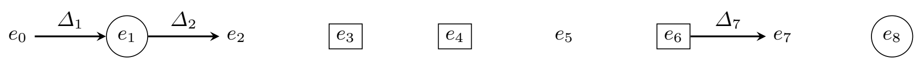

Fig. 6. Example of corrupted tokens, keys (boxed) and challenge-equal epochs (circled) in a UP-IND-CCA game. Corrupting Δ_3 and Δ_8 is forbidden, as they would allow to re-encrypt the challenge ciphertext into an epoch where $\mathcal{A}$ knows the secret key.

strategy such as, e.g., corrupting a key under which a given challenge ciphertext has been (re-)encrypted.

We use the notation from [LT18] to define the information the adversary may trivially derive. We focus on schemes that are bi-directional, i.e., we assume up and downgrades of ciphertexts. That is, we assume that a token Δ_e may enable *downgrades* of ciphertexts from epoch e into epoch $e-1$. While bi-directional security and schemes are not preferable from a security point of view, all currently known (efficient) solutions exhibit this additional property.[4] Thus, for the sake of simplicity we state all our definitions for this setting. As a consequence, it is sufficient to consider only the inferable information w.r.t. ciphertexts: [LT18] also formulate inference of keys, which in the case of bi-directional schemes has no effect on the security notions though.

For the (R)CCA game, we need to capture all the epochs in which the adversary knows a version of the challenge ciphertext, which we define through the set $\widehat{\mathbf{C}}^*$ containing all *challenge-equal* epochs. Recall that $\mathbf{C}^*$ denotes the set of epochs in which the adversary has obtained an updated version of the ciphertext via the challenge query or by updating the challenge ciphertext via the ReEnc oracle. The set $\mathbf{T}$ contains all epochs in which the adversary has corrupted the update tokens, and e_{end} denotes the last epoch of the game. The set $\widehat{\mathbf{C}}^*$ of all challenge-equal ciphertexts is defined via the recursive predicate $\mathsf{challenge\text{-}equal}$:

$$\begin{aligned}\widehat{\mathbf{C}}^* \leftarrow \{e \in \{0,\dots,e_{\mathsf{end}}\} \mid \mathsf{challenge\text{-}equal}(e) = \mathtt{true}\}\\ \text{and } \mathtt{true} \leftarrow \mathsf{challenge\text{-}equal}(e) \text{ iff: } (e \in \mathbf{C}^*) \vee\\ (\mathsf{challenge\text{-}equal}(e-1) \wedge e \in \mathbf{T}) \ \vee\ (\mathsf{challenge\text{-}equal}(e+1) \wedge e+1 \in \mathbf{T})\end{aligned}$$

Note that $\widehat{\mathbf{C}}^*$ is efficiently computable (e.g. via fixpoint iteration).

Re-encryptions of the challenge ciphertext. Note that we allow ReEnc to *skip* keys, as we let $\mathcal{A}$ give the starting epoch i as an additional parameter and return the re-encryption from any old key k_i to the current one. This is crucial for obtaining a meaningful security model: any ReEnc query where the input ciphertext is a derivation of the challenge ciphertext (that the adversary will receive in the CCA game), marks the current target epoch e as challenge-equal by adding e to $\mathbf{C}^*$. In our UP-IND-CCA security game defined below we disallow the adversary from corrupting the key of any challenge-equal epoch to prevent trivial wins. Calling the ReEnc oracle for a re-encryption of the challenge ciphertext from some epoch i to e will still allow $\mathcal{A}$ to corrupt keys between i and e.

[4] Note that bi-directionality is a property of the *security model*, not the scheme per se. That is, uni-directional schemes are evidently also bi-directional secure, even though they do *not* allow ciphertext downgrades.

Ciphertext Integrity. Updatable encryption should also protect the integrity of ciphertexts. That is, an adversary should not be able to produce a ciphertext himself that correctly decrypts to a message $m \neq \perp$. Our definition adapts he classic INT-CTXT notion to the setting of updatable encryption. We use the same oracles as in the UP-IND-CCA game defined above, but where isChallenge always returns false (as there is no challenge ciphertext). Again, the tricky part of the definition is to capture the set of trivial wins – in this case trivial forgeries – that the adversary can make given the secret keys and update tokens he corrupts. For simplicity, we only consider forgeries that the adversary makes in the current and final epoch e_{end}, but not in the past. This matches the idea of updatable encryption where the secret keys and update tokens of old epochs will (ideally) be deleted, and thus a forgery for an old key is meaningless anyway.

Clearly, when the adversary corrupted the secret key at some previous epoch and since then learned *all* update tokens until the final epoch e_{end}, then all ciphertexts in this last epoch can easily be forged. This is captured by the first case in the definition of UP-INT-CTXT security.

Definition 7. *An updatable encryption scheme* UE *is called UP-INT-CTXT secure if for any PPT adversary* $\mathcal{A}$ *the following advantage is negligible in* κ*:* $\mathsf{Adv}^{\textit{up-int-ctxt}}_{\mathsf{UE},\mathcal{A}}(pp) := \Pr[\mathsf{Exp}^{\textit{up-int-ctxt}}_{\mathsf{UE},\mathcal{A}}(pp) = 1]$.

```
Experiment $\mathsf{Exp}^{\mathsf{up\text{-}int\text{-}ctxt}}_{\mathsf{UE},\mathcal{A}}(pp)$
    $(sp, k_0, \Delta_0, \mathbf{Q}, \mathbf{K}, \mathbf{T}) \leftarrow \mathsf{Init}(pp)$
    $c^* \leftarrow_{\mathrm{R}} \mathcal{A}^{\mathsf{Next},\mathsf{Enc},\mathsf{Dec},\mathsf{ReEnc},\mathsf{Corrupt}}(sp)$
    return 1 if $\mathsf{UE.Dec}(k_{e_{\mathsf{end}}}, c^*) \neq \perp$ and $(e_{\mathsf{end}}, c^*) \notin \mathbf{Q}^*$ and
        $\nexists e \in \mathbf{K}$ where $i \in \mathbf{T}$ for $i = e$ to $e_{\mathsf{end}}$; i.e. $\mathcal{A}$ did not trivially win.
```

Defining trivial ciphertext updates. When defining the set of trivial ciphertexts $\mathbf{Q}^*$ for the UP-INT-CTXT game defined above, we now move from general epochs to concrete ciphertexts, i.e., we capture all ciphertexts that the adversary could know, either through queries to the Enc or ReEnc oracle or through updating such ciphertexts himself. We exploit that ReEnc is deterministic to define the set of trivial forgeries $\mathbf{Q}^*$ as narrow as possible. More precisely, $\mathbf{Q}^*$ is defined by going through the ciphertexts $(e, c) \in \mathbf{Q}$ the adversary has received through oracle queries and iteratively update them into the next epoch $e + 1$ whenever the adversary has corrupted Δ_{e+1}. The latter information is captured in the set $\mathbf{T}$ that contains all epochs in which the adversary learned the update token. We start with $\mathbf{Q}^* \leftarrow \emptyset$ and amend the set as follows:

```
for each $(e, c) \in \mathbf{Q}$:
    set $\mathbf{Q}^* \leftarrow \mathbf{Q}^* \cup (e, c)$, and $i \leftarrow e + 1$, $c_{i-1} \leftarrow c$
    while $i \in \mathbf{T}$:
        set $\mathbf{Q}^* \leftarrow \mathbf{Q}^* \cup (i, c_i)$ where $c_i \leftarrow \mathsf{UE.ReEnc}(\Delta_i, c_{i-1})$, and $i \leftarrow i + 1$
```

On the Necessity of the "Queried Restriction". Restricting ReEnc queries to honestly generated ciphertexts seems somewhat unavoidable, as the ability of ciphertext-independent key-rotation seems to require homomorphic properties on the encryption. In our construction, an adversary could exploit this homomorphism to "blind" the challenge ciphertext before sending it to the ReEnc oracle, and later "unblind" the re-encrypted ciphertext. This blinding would prevent us from recognizing that the challenge ciphertext was re-encrypted, and thus the target epoch would no longer be marked as challenge-equal, allowing the adversary to corrupt the secret key in the new epoch and trivially win by decrypting the re-encrypted challenge. A similar restriction is used in the CTXT definition for ciphertext-dependent schemes in [Eve+17b] as well.[5] In Sec. 4 we overcome this challenge by making ciphertexts publicly verifiable. The above "blinding" trick then no longer works as it would invalidate the proof of ciphertext correctness.

3.2 Generic Transformation for Secure Updatable Encryption

In the following we prove UP-IND-CCA and UP-INT-CTXT security for a class of updatable encryption schemes satisfying some mild requirements. The goal is that given an updatable encryption scheme $\mathsf{UE} = (\mathsf{Gen}, \mathsf{GenKey}, \mathsf{GenTok}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{ReEnc})$, we can prove the security of UE based only on classical security of the underlying encryption scheme $\mathsf{SKE} = (\mathsf{Gen}, \mathsf{GenKey}, \mathsf{Enc}, \mathsf{Dec})$ and simple properties satisfied by GenTok and ReEnc.

Properties of the (Re-)Encryption and Token Generation. Now we define the additional properties that are needed to lift static IND-CCA and INT-CTXT security to their updatable version with adaptive key and token corruptions as just defined.

Tidy Encryption & Strong CCA/CTXT. When re-encryptions are deterministic, we need the underlying standard encryption scheme SKE of an updatable scheme to be tidy (cf. Def. 2), so there is a one-to-one correspondence between ciphertexts and message-randomness pairs. Further, we need slightly stronger variants of the standard security definitions IND-CCA and INT-CTXT in the deterministic setting where the encryption oracle additionally reveals the used encryption randomness. We denote these stronger experiments by S-IND-CCA and S-INT-CTXT, or simply by saying **strong** IND-CCA/INT-CTXT.

Definition 8. *An (updatable) encryption scheme is called* ***strong*** *IND-X (or S-IND-X) secure for $X \in \{CPA, CCA\}$ if it remains secure even if the encryption oracle returns the encryption randomness, including the (purported) encryption randomness of the challenge ciphertext. For IND\$, in case of a random ciphertext, a purported randomness is drawn. (We assume that encryption randomness is uniform in the randomness space $\mathcal{R}$.) Analogously, we define* ***strong*** *INT-CTXT (S-INT-CTXT).*

[5] The CTXT definition in the proceedings version of their paper did not have such a restriction, however the revised ePrint version [Eve+17a] later showed that the original notion is not achievable and a weaker CTXT definition is introduced instead.

Simulatable & Reverse Tokens. We need further properties (Definitions 9 and 10) that are concerned with the token generation of an updatable encryption scheme. It should be possible to simulate perfectly indistinguishable tokens as well as reverse tokens, inverting the effect of the former ones, without knowing any key.

Definition 9. *We call a token Δ' a **reverse token** of a token Δ if for every pair of keys $k^{\mathrm{old}}, k^{\mathrm{new}} \in \mathcal{K}$ with $\Delta \in \operatorname{supp}(\mathsf{UE.GenTok}(k^{\mathrm{old}}, k^{\mathrm{new}}))$ we have $\Delta' \in \operatorname{supp}(\mathsf{UE.GenTok}(k^{\mathrm{new}}, k^{\mathrm{old}}))$.*

Definition 10. *Let UE be an updatable encryption scheme. We say that UE has **simulatable** token generation if it has the following property: There is a PPT algorithm $\mathsf{SimTok}(sp)$ which samples a pair (Δ, Δ') of token and reverse token. Furthermore, for arbitrary (fixed) $k^{\mathrm{old}} \leftarrow \mathsf{UE.GenKey}(sp)$ following distributions of Δ are identical: The distribution of Δ*

- *induced by $(\Delta, \_) \leftarrow_R \mathsf{SimTok}(sp)$.*
- *induced by $\Delta \leftarrow_R \mathsf{UE.GenTok}(k^{\mathrm{old}}, k^{\mathrm{new}})$ where $k^{\mathrm{new}} \leftarrow_R \mathsf{UE.GenKey}(sp)$.*

In other words, honest token generation and token simulation are perfectly indistinguishable.

Re-encryption = decrypt-then-encrypt. The final requirement states that the re-encryption of a ciphertext $c = \mathsf{UE.Enc}(k^{\mathrm{old}}, m; r)$ looks like a fresh encryption of m under k^{new} where $\mathsf{UE.Enc}$ uses the same randomness r. To formalize this, we make use of $\mathsf{UE.RDec}$, the randomness-recoverable decryption algorithm of the underlying encryption scheme (Def. 2), where we have $(m, r) \leftarrow \mathsf{UE.RDec}(k, c)$ for $c \leftarrow \mathsf{UE.Enc}(k, m; r)$.

Definition 11. *Let UE be an updatable encryption scheme with deterministic re-encryption. We say that re-encryption (for UE) is **randomness-preserving** if the following holds: First, as usually assumed, UE encrypts with* uniformly chosen randomness *(i.e., $\mathsf{UE.Enc}(k, m)$ and $\mathsf{UE.Enc}(k, m; r)$ for uniformly chosen r are identically distributed). Second, for all $sp \leftarrow_R \mathsf{UE.GenSP}(pp)$, all keys $k^{\mathrm{old}}, k^{\mathrm{new}} \leftarrow_R \mathsf{UE.GenKey}(sp)$, tokens $\Delta \leftarrow_R \mathsf{UE.GenTok}(k^{\mathrm{old}}, k^{\mathrm{new}})$, and all valid ciphertexts c under k^{old}, we have*

$$\mathsf{UE.Enc}(k^{\mathrm{new}}, \mathsf{UE.RDec}(k^{\mathrm{old}}, c)) = \mathsf{UE.ReEnc}(\Delta, c).$$

More precisely, $\mathsf{UE.Enc}(k^{\mathrm{new}}, \mathsf{UE.RDec}(k^{\mathrm{old}}, c))$ is defined as $\mathsf{UE.Enc}(k^{\mathrm{new}}, m; r)$ where $(m, r) \leftarrow \mathsf{UE.RDec}(k^{\mathrm{old}}, c)$.

In App. A, we argue that this randomness-preserving property additionally guarantees unlinkability of re-encrypted ciphertexts (UP-REENC security) as considered by prior work [LT18; Eve+17b].

UP-IND-CCA and UP-INT-CTXT Security. We are now ready to state our generic transformation for achieving security of the updatable encryption scheme. The proofs for both properties are very similar, and below we describe the core ideas of our proof strategy. The detailed proofs are given in App. B.

Theorem 1. *Let* $\mathsf{UE} = (\mathsf{Gen}, \mathsf{GenKey}, \mathsf{GenTok}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{ReEnc})$ *be an updatable encryption scheme with deterministic re-encryption. Suppose that* UE *has* randomness-preserving *re-encryption and* simulatable *token generation and the underlying encryption scheme* $\mathsf{SKE} = (\mathsf{Gen}, \mathsf{GenKey}, \mathsf{Enc}, \mathsf{Dec})$ *is tidy.*

- *If* SKE *is S-IND-CCA-secure, then* UE *is UP-IND-CCA-secure.*
- *If* SKE *is S-INT-CTXT-secure, then* UE *is UP-INT-CTXT-secure.*

Proof (sketch). In the following, we illustrate the main challenges occurring in our security proofs as well as how we can cope with these using the properties we just introduced. Let us consider the problems that arise when we embed a static challenge, say an IND-CCA challenge, into an UP-IND-CCA game. Let us assume the UP-IND-CCA adversary $\mathcal{A}$ asks for its challenge under key k_{e^*} and we want to embed our IND-CCA challenge there. Then k_{e^*} is unknown to us but we can answer $\mathcal{A}$'s encryption and decryption queries under k_{e^*} using our own IND-CCA oracles.

However, the token Δ_{e^*+1} might be corrupted by $\mathcal{A}$. Note that in this case, k_{e^*+1} cannot be corrupted, since $\mathcal{A}$ could trivially win. Now, the question is how Δ_{e^*+1} can be generated without knowing k_{e^*}. For this purpose, we make use of the simulatable token generation property (Def. 10) that ensures that well-distributed tokens can be generated even without knowing keys. So we can hand over a simulated Δ_{e^*+1} to $\mathcal{A}$ if it asks for it. But when simulating tokens in this way, we do not know the corresponding keys. This is a potential problem as we need to be able to answer encryption and decryption queries under the unknown key k_{e^*+1}. To cope with this problem, we use the corresponding IND-CCA oracle for k_{e^*} and update or downgrade the ciphertexts from/to epoch e^*. That means, if we are asked to encrypt under k_{e^*+1}, we actually encrypt under k_{e^*} and update the resulting ciphertext to epoch $e^* + 1$ using Δ_{e^*+1}. Now, we need to ensure that ciphertexts created in this way look like freshly encrypted ciphertexts under key k_{e^*+1}. This is what Def. 11 requires. Similarly, if we are asked to decrypt under k_{e^*+1}, we downgrade the ciphertext using the reverse token Δ'_{e^*+1} (Def. 9) that was generated along with Δ_{e^*+1} (Def. 10). Note that in this case, we do not need the downgraded ciphertext to look like a fresh one as $\mathcal{A}$ never sees it. Assuming the next token Δ_{e^*+2} gets also corrupted we can do the same to handle encryption and decryption queries for epoch $e^* + 2$.

Now let us assume that not the token Δ_{e^*+1} but the key k_{e^*+1} gets corrupted. In this case we can neither generate Δ_{e^*+1} regularly as we do not know k_{e^*} nor simulate it as k_{e^*+1} is known to the adversary. As we know k_{e^*+1}, we have no problems in handling encryption and decryption queries for epoch e^*+1. But it is not clear how we can re-encrypt a (non-challenge) ciphertext c freshly generated in epoch e^* to $e^* + 1$ without knowing Δ_{e^*+1}. As we called our IND-CCA encryption oracle to generate c, we certainly know the contained message m. So we could just encrypt m under key k_{e^*+1} yielding ciphertext c'. However, now the freshly encrypted ciphertext c' and a ciphertext c'' resulting from regularly updating c' to epoch $e^* + 1$ may look different as they involve different randomness. To circumvent this problem, we require the IND-CCA encryption oracle to additionally output the randomness r which has been used to generate

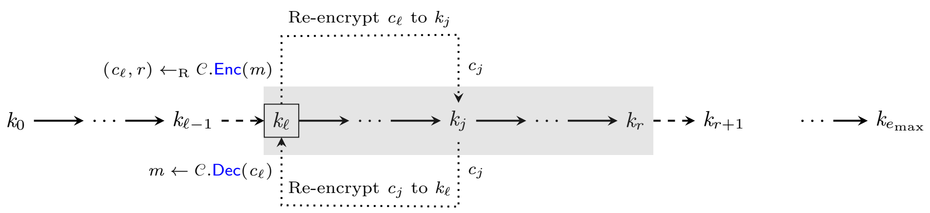
Re-encrypt c_ℓ to k_j
$(c_\ell, r) \leftarrow_{\mathrm{R}} \mathcal{C}.\mathsf{Enc}(m)$
c_j
k_0
$k_{\ell-1}$
k_ℓ
k_j
k_r
k_{r+1}
$k_{e_{\max}}$
$m \leftarrow \mathcal{C}.\mathsf{Dec}(c_\ell)$
c_j
Re-encrypt c_j to k_ℓ

Fig. 7. Encryption and decryption in the insulated region. The keys in the grey area (k_ℓ to k_r) are not known in the reduction. Encryption and decryption for other keys is unchanged. The S-INT-CTXT resp. S-IND-CCA challenger $\mathcal{C}$ is embedded in epoch ℓ.

c. Computing c' using randomness r then yields perfect indistinguishability assuming Def. 11. Hence, we need SKE to be S-IND-CCA (and S-INT-CTXT) and not only IND-CCA (and INT-CTXT) secure.

Finally, let us consider how to handle queries to the left of the challenge epoch. For this, let us assume that k_{e^*-1} gets corrupted and Δ_{e^*} is uncorrupted but unknown to us. Then again we can easily handle encrypt/decrypt queries for epoch $e^* - 1$ but cannot re-encrypt a ciphertext c from epoch $e^* - 1$ to e^* in a straightforward manner. Now, as c needs to result from a previous query the corresponding message-randomness pair (m, r) (due to tidyness there is only one such pair) is known. So, as before, we would like to replace the re-encryption by a fresh encryption under key k_{e^*}. Unfortunately, the S-IND-CCA encryption oracle we would use for this purpose only accepts the message but not the randomness as input. We cope with this as follows: when we are asked to encrypt a message m under key k_{e^*-1} (or prior keys), we will always first call the S-IND-CCA oracle to encrypt m yielding a ciphertext c' and randomness r. Then we would encrypt (m, r) under key k_{e^*-1} yielding c. The ciphertext c' can then be stored until a re-encryption of c is needed. Again Def. 11 ensures perfect indistinguishability from a real re-encryption. (Here, we use that encryption randomness is chosen uniformly, independent of the key.)

Note that the case that Δ_{e^*} is corrupted could actually be handled analogous to the case that Δ_{e^*+1} is corrupted by additionally demanding randomness-preserving re-encryption for reverse tokens but we can get around this.

Overall, this solves the main challenges when embedding an S-IND-CCA challenge into an UP-IND-CCA game.

Key Insulation. Our key insulation technique aims at coping with the problems when embedding challenges and follows the ideas just described. However, instead of guessing the challenge epoch *and* the region to the left and to the right in which the adversary corrupted all of the tokens (and none of keys) and embed our S-IND-CCA/S-INT-CTXT challenge there, we rather do the following: we only guess the boundaries of this region $\{\ell, \ldots, r\}$ (containing the challenge epoch) and embed the S-IND-CCA/S-INT-CTXT challenge at epoch ℓ. Note that the tokens Δ_ℓ and Δ_r entering and leaving the boundaries of this "insulated" region are not corrupted.

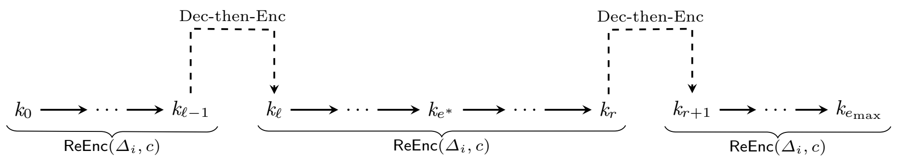
Dec-then-Enc
Dec-then-Enc
$k_0 \longrightarrow \cdots \longrightarrow k_{\ell-1}$
$k_\ell \longrightarrow \cdots \longrightarrow k_{e^*} \longrightarrow \cdots \longrightarrow k_r$
$k_{r+1} \longrightarrow \cdots \longrightarrow k_{e_{\max}}$
$\mathsf{ReEnc}(\Delta_i, c)$
$\mathsf{ReEnc}(\Delta_i, c)$
$\mathsf{ReEnc}(\Delta_i, c)$

Fig. 8. Entering and leaving the insulated region. Re-encryption in the underbraced regions is done using the known tokens. The two missing tokens are "emulated" by decrypt-then-encrypt.

Now we change the inner workings in this region and the way it can be entered from the left using the ideas described before. Namely, only key k_ℓ in the region is generated. Recall, in the reduction we have S-IND-CCA/S-INT-CTXT oracles at our disposal to replace this key. The tokens $\Delta_{\ell+1}, \ldots, \Delta_{r+1}$ along with corresponding reverse tokens are generated using SimTok (cf. Def. 10). For encryption in the region, we encrypt under k_ℓ and update the ciphertext to the desired epoch. For decryption, we the use reverse tokens to downgrade the ciphertext to k_ℓ and decrypt with this key. This is illustrated in Fig. 7. Leaving and entering the region which was originally done by re-encryption, is now essentially done by retrieving the plaintext and randomness of the ciphertext that should be reencrypted (so we sort of decrypt the queried ciphertext) and use it to generate a fresh ciphertext inside or outside the region by encryption. This is depicted in Fig. 8.

3.3 An Encrypt-and-MAC Construction

We construct an UE scheme with *deterministic* re-encryption that achieves UP-IND-CCA, UP-REENC, and UP-INT-CTXT security. For this, we use generic building blocks which can be securely instantiated from the DDH assumption.

High-Level Idea. Our idea is to do a Encrypt-and-MAC (E&M) construction with primitives which are key-rotatable. Using Encrypt-*and*-MAC instead of the more standard Encrypt-*then*-MAC approach is crucial for the updatability as we need "direct access" to both the ciphertext and the MAC.

It is well-known that, in general, E&M is *not* a secure transformation for authenticated encryption, as the MAC could leak information about the plaintext and does not authenticate the ciphertext. However, when using a *tidy* encryption scheme SKE (cf. Def. 2) and a pseudorandom function PRF as MAC, then E&M *does* provide (static) CCA and CTXT security. Recall that tidy encryption means that decryption is randomness-recoverable, i.e., it also outputs the randomness r used in the encryption. This allows to apply the PRF on both, the message and the randomness r, which then guarantees the integrity of ciphertexts.

We start with such tidy E&M for static primitives but also require that SKE and PRF support key-rotation and updates of ciphertexts and PRF values. Then, for yielding the updatable version of the E&M transform, one simply relies on the key-rotation capabilities of SKE and PRF and updates the individual parts of the authenticated ciphertext. Security of the UE scheme obtained in this way follows since the properties from Sec. 3.2 are satisfied.

Encrypt-and-MAC. First we recall the E&M transformation and its security for tidy (randomness recoverable) encryption. To make it clear that decryption recovers the encryption randomness we write RDec for decryption and make the randomness chosen in the encryption explicit as $\mathsf{Enc}(k, m; r)$. Let $\mathsf{SKE} = (\mathsf{GenSP}, \mathsf{GenKey}, \mathsf{Enc}, \mathsf{RDec})$ be a *tidy* encryption scheme and $\mathsf{PRF} = (\mathsf{GenSP}, \mathsf{GenKey}, \mathsf{Eval})$ be a pseudorandom function, then the E&M transform of SKE and PRF is defined as follows:

- $\mathsf{AE.GenSP}(pp)$ returns $sp = (sp_{\mathsf{SKE}}, sp_{\mathsf{PRF}})$ where $sp_{\mathsf{SKE}} \leftarrow_{\mathrm{R}} \mathsf{SKE.GenSP}(pp)$ and $sp_{\mathsf{PRF}} \leftarrow_{\mathrm{R}} \mathsf{PRF.GenSP}(pp)$.
- $\mathsf{AE.GenKey}(sp)$ returns $k = (k_{\mathsf{SKE}}, k_{\mathsf{PRF}})$, where $k_{\mathsf{SKE}} \leftarrow_{\mathrm{R}} \mathsf{SKE.GenKey}(sp_{\mathsf{SKE}})$ and $k_{\mathsf{PRF}} \leftarrow_{\mathrm{R}} \mathsf{PRF.GenKey}(sp_{\mathsf{PRF}})$.
- $\mathsf{AE.Enc}(k, m; r)$ returns (c, τ) where $c \leftarrow \mathsf{SKE.Enc}(k_{\mathsf{SKE}}, m; r)$ and $\tau \leftarrow \mathsf{PRF.Eval}(k_{\mathsf{PRF}}, (m, r))$.
- $\mathsf{AE.RDec}(k, (c, \tau))$ computes $(m, r) \leftarrow \mathsf{SKE.RDec}(k_{\mathsf{SKE}}, c)$. It returns (m, r) if $\mathsf{PRF.Eval}(k_{\mathsf{PRF}}, (m, r)) = \tau$, and $\perp$ otherwise.

Lemma 1 essentially follows from [NRS14] where, however, a slightly different definition of tidy was used. But the adaption to our setting is straightforward.

Lemma 1. *If* SKE *is a* tidy *encryption scheme satisfying S-IND-CPA security, and* PRF *is a secure pseudorandom function (with domain* $\mathcal{M} \times \mathcal{R}$*), then* AE *as defined above is a S-IND-CCA and S-INT-CTXT secure tidy encryption scheme. The same holds for IND$ instead of IND.*

Updatable Encrypt-and-MAC. To make this E&M construction a secure updatable encryption scheme, we need that both underlying primitives support key-rotation satisfying certain properties. That means, for SKE we assume that additional algorithms $\mathsf{GenTok}(k^{\mathrm{old}}, k^{\mathrm{new}})$ and $\mathsf{ReEnc}(\Delta, c)$ as in Def. 5 are given satisfying simulatable token generation (Def. 10) and randomness-preserving re-encryption (Def. 11). Likewise, we need similar algorithms $\mathsf{GenTok}(k^{\mathrm{old}}, k^{\mathrm{new}})$ and $\mathsf{Upd}(\Delta, \tau)$ for the PRF satisfying similar properties, i.e., a straightforward adaption of simulatable token generation (see Def. 20) and correctness in the sense that $\mathsf{Upd}(\Delta, \mathsf{Eval}(k^{\mathrm{old}}, (m, r))) = \mathsf{Eval}(k^{\mathrm{new}}, (m, r))$.

We now obtain our secure UE scheme by extending the AE scheme defined above with the following GenTok and ReEnc algorithms:

- $\mathsf{AE.GenTok}(k^{\mathrm{old}}, k^{\mathrm{new}})$ computes $\Delta_{\mathsf{SKE}} \leftarrow_{\mathrm{R}} \mathsf{SKE.GenTok}(k^{\mathrm{old}}_{\mathsf{SKE}}, k^{\mathrm{new}}_{\mathsf{SKE}})$ and $\Delta_{\mathsf{PRF}} \leftarrow_{\mathrm{R}} \mathsf{PRF.GenTok}(k^{\mathrm{old}}_{\mathsf{PRF}}, k^{\mathrm{new}}_{\mathsf{PRF}})$ and returns $\Delta := (\Delta_{\mathsf{SKE}}, \Delta_{\mathsf{PRF}})$.
- $\mathsf{AE.ReEnc}(\Delta, (c, \tau))$ computes $c' \leftarrow \mathsf{SKE.ReEnc}(\Delta_{\mathsf{SKE}}, c)$ and $\tau' \leftarrow \mathsf{PRF.Upd}(\Delta_{\mathsf{PRF}}, \tau)$ and returns (c', τ').

UP-IND-CCA and UP-INT-CTXT security directly follows from Thm. 1 and UP-REENC-CCA follows from Thm. 5 (where we also state the definition for UP-REENC security adapted to the CCA setting).

Corollary 1. *Suppose* AE *is the E&M construction as in Lemma 1, in particular S-IND-CCA and S-INT-CTXT secure. Suppose* AE *supports randomness-preserving reencryption and simulatable token generation as described above, i.e.* AE *constitutes an updatable encryption scheme. Then* AE *is UP-IND-CCA and UP-INT-CTXT secure. Moreover, if* AE *is S-IND$-CCA secure, then it is also UP-REENC-CCA secure.*

Instantiating the key-rotatable building blocks. We now show how the key-rotatable building blocks SKE and PRF can be securely instantiated. First we construct the encryption scheme which is S-IND$-CPA secure under the DDH assumption and also tidy. Then we present the key-rotatable PRF that is secure under the DDH assumption in the random oracle model.

$\mathsf{SKE}_{\mathsf{DDH}}$. Since we need a tidy, and hence randomness recoverable encryption scheme, we must pick the encryption randomness $[r] \leftarrow_{\mathrm{R}} \mathbb{G}$ from $\mathbb{G}$ if discrete logarithms are hard. A straightforward choice is to use $[r]sk$ instead of $r[pk]$ in RISE/ElGamal. However, our result which gives UP-REENC security (i.e., the unlinkability of re-encryptions) for free, c.f. Thm. 5, requires *strong* IND$-CCA security. Thus, we instead use following variation of the mentioned approach:

$\mathsf{SKE}_{\mathsf{DDH}}.\mathsf{GenSP}(pp)$ does nothing. That is, it returns $sp = pp$.
$\mathsf{SKE}_{\mathsf{DDH}}.\mathsf{GenKey}(sp)$ returns $k = (k_1, k_2) \leftarrow_{\mathrm{R}} \mathbb{F}_p^* \times \mathbb{F}_p = \mathcal{K}$.
$\mathsf{SKE}_{\mathsf{DDH}}.\mathsf{GenTok}(k^{\mathrm{old}}, k^{\mathrm{new}})$ returns $\Delta = (\Delta_1, \Delta_2) = (\frac{k_1^{\mathrm{new}}}{k_1^{\mathrm{old}}}, \frac{k_2^{\mathrm{new}} - k_2^{\mathrm{old}}}{k_1^{\mathrm{old}}}) \in \mathcal{D} = \mathcal{K}$.
$\mathsf{SKE}_{\mathsf{DDH}}.\mathsf{Enc}(k, [m]; [r])$ returns $[c]$, encryption of a message $[m] \in \mathbb{G}$ with randomness $[r] \leftarrow_{\mathrm{R}} \mathbb{G}$ as $[c] = (k_1[r], k_2[r] + [m]) \in \mathbb{G}^2 = \mathcal{C}$.
$\mathsf{SKE}_{\mathsf{DDH}}.\mathsf{RDec}(k, [c])$ returns $([r], [m])^\top$ via $[r] = \frac{1}{k_1}[c_1]$, $[m] = [c_2] - k_2[r]$.
$\mathsf{SKE}_{\mathsf{DDH}}.\mathsf{ReEnc}(\Delta, [c^{\mathrm{old}}])$ returns $[c^{\mathrm{new}}] = [\Delta_1 c_1^{\mathrm{old}}, \Delta_2 c_1^{\mathrm{old}} + c_2^{\mathrm{old}}]$.

It is easy to see that the scheme is correct with deterministic re-encryption.

Lemma 2. *The scheme* $\mathsf{SKE}_{\mathsf{DDH}}$ *is tidy, has simulatable token generation, and randomness-preserving deterministic re-encryption. The underlying encryption of* $\mathsf{SKE}_{\mathsf{DDH}}$ *is strong IND$-CPA secure under the DDH assumption over* $\mathbb{G}$*.*

It is evident, that the scheme is tidy. Randomness-preserving re-encryption follows from straightforward calculations. For simulatable token generation, note that any two of k^{old}, Δ, k^{new}, determine the third uniquely (and it is efficiently computable). Moreover, if we define $\mathsf{invert}((\Delta_1, \Delta_2)) = (\frac{1}{\Delta_1}, -\frac{\Delta_2}{\Delta_1})$ then $\mathsf{invert}(\Delta)$ is a token which downgrades ciphertexts from k^{new} to k^{old} With this, token simulation is easy to see. S-IND$-CPA security follows from a straightforward adaptation of the standard ElGamal security proof. Note that we do not allow key-leakage, i.e. $\mathsf{leak}(k) = \perp$.

$\mathsf{PRF}_{\mathsf{DDH}}$. Using a hash function $\mathsf{H}\colon \{0,1\}^* \to \mathbb{G}$, we instantiate the key-rotatable PRF as $\mathsf{PRF}_{\mathsf{DDH}}\colon \mathbb{F}_p^\times \times \{0,1\}^* \to \mathbb{G}$. The core part of the PRF is the classical DDH-based construction from [NPR99; Bon+15]. We show that it can also be extended to allow for key-rotation for which it enjoys token simulation.

$\mathsf{PRF}_{\mathsf{DDH}}.\mathsf{GenSP}(pp)$ does nothing, i.e. returns $sp = pp$.
$\mathsf{PRF}_{\mathsf{DDH}}.\mathsf{GenKey}(sp)$ returns $k \leftarrow_{\mathrm{R}} \mathbb{F}_p = \mathcal{K}$.
$\mathsf{PRF}_{\mathsf{DDH}}.\mathsf{GenTok}(k^{\mathrm{old}}, k^{\mathrm{new}})$ returns $\Delta = \frac{k^{\mathrm{new}}}{k^{\mathrm{old}}}$.
$\mathsf{PRF}_{\mathsf{DDH}}.\mathsf{Eval}(k, x)$ returns $[\tau] = k\,\mathsf{H}(x) \in \mathbb{G}$.
$\mathsf{PRF}_{\mathsf{DDH}}.\mathsf{Upd}(\Delta, [\tau])$ returns $\Delta[\tau]$.

Lemma 3. *The* $\mathsf{PRF}_{\mathsf{DDH}} = (\mathsf{GenSP}, \mathsf{GenKey}, \mathsf{Eval})$ *scheme defined above (without* GenTok *and* Upd*) is secure under the DDH assumption on* $\mathbb{G}$ *if* H *is a (programmable) random oracle.* $\mathsf{PRF}_{\mathsf{DDH}}$ *has simulatable token generation.*

The security of $\mathsf{PRF}_{\mathsf{DDH}}$ was shown in [NPR99], and the simulatable properties of the token generation follow from the same observations as for $\mathsf{SKE}_{\mathsf{DDH}}$.

4 RCCA and PTXT Secure Updatable Encryption

In this section, we first define RCCA and PTXT security for updatable encryption under active re-encryption attacks (Sec. 4.1). In Sec. 4.2 we then present our Naor-Yung inspired scheme that satisfies these strong security notions.

4.1 Security Model

We now present our definitions for updatable encryption with Replayable CCA (RCCA) security and plaintext integrity (PTXT). The oracles used in these definitions are mostly equivalent to the ones introduced for CCA security in Section 3.1, and thus we focus on the parts that have changed.

The most important difference is that the ReEnc oracle can be invoked on *arbitrary* ciphertexts in both definitions, whereas our CCA and CTXT definitions only allowed re-encryptions of ciphertexts that had been obtained through oracle queries themselves. This strengthening to arbitrary inputs is much closer to the reality of updatable encryption, where ciphertexts and the update procedure are outsourced to potentially untrusted data hosts. All previous definitions cover only passive corruptions of such a host, whereas our notions in this section even guarantee security against active adversaries.

RCCA Security. Standard RCCA is a relaxed variant of CCA security which is identical to CCA with the exception that the decryption oracle will not respond with `invalid` whenever a ciphertext decrypts to either of the challenge messages m_0 or m_1. This includes ciphertexts that are different from the challenge ciphertext c^* the adversary has obtained. RCCA is a suitable definition in particular for schemes where ciphertexts can be re-randomized, and thus cannot achieve the standard CCA notion. Our setting allows similar public re-randomization as ciphertext updates are now *probabilistic* instead of deterministic. Thus, as soon as the adversary has corrupted an update token we can no longer trace re-encryptions of the challenge ciphertexts (as we did in the UP-IND-CCA definition for deterministic schemes) in order to prevent the adversary from decrypting the challenge ciphertext.

Thus, instead of tracing the challenge ciphertext we now follow the RCCA approach. Our definition of UP-IND-RCCA security is essentially the standard RCCA definition adapted for updatable encryption by giving the adversary access to a re-encryption oracle and allowing him to adaptively corrupt secret keys and tokens in the current or any past epoch.

In Enc and ReEnc described below we still keep track of honestly generated ciphertexts (and their plaintexts) which allows us to be less restrictive when a ciphertext-query can be traced down to a non-challenge ciphertext. We explain this modelling choice in more detail below.

Next(), Corrupt($\{\mathsf{key}, \mathsf{token}\}, i$): as in CCA game
Enc(m): Returns $c \leftarrow_{\mathrm{R}} \mathsf{UE.Enc}(k_e, m)$ and sets $\mathbf{Q} \leftarrow \mathbf{Q} \cup \{(e, m, c)\}$.
Dec(c): If $\mathsf{isChallenge}(k_e, c) = \mathtt{false}$, the oracle returns $m \leftarrow \mathsf{UE.Dec}(k_e, c)$.
ReEnc(c, i): The oracle returns c_e which it iteratively computes as $c_\ell \leftarrow_{\mathrm{R}} \mathsf{UE.ReEnc}(\Delta_\ell, c_{\ell-1})$ for $\ell = i+1, \ldots, e$ and $c_i \leftarrow c$. It also updates the global state depending on whether the queried ciphertext is the challenge ciphertext or not:
- If $(i, m, c) \in \mathbf{Q}$ (for some m), then set $\mathbf{Q} \leftarrow \mathbf{Q} \cup \{(e, m, c_e)\}$.
- Else, if $\mathsf{isChallenge}(k_i, c) = \mathtt{true}$, then set $\mathbf{C}^* \leftarrow \mathbf{C}^* \cup \{e\}$.

As for UP-IND-CCA security, the challenge is to prevent trivial wins where an adversary tries to exploit the update capabilities of such schemes. We again achieve this by capturing the indirect knowledge of the adversary through the recursive predicate that defines all challenge-equal epochs $\widehat{\mathbf{C}}^*$. This set (which is as defined in Section 3.1) contains all epochs in which the adversary trivially knows a version of the challenge ciphertext, either through oracle queries or by up/downgrading the challenge ciphertext himself. The adversary wins UP-IND-RCCA if he can determine the challenge bit b used to compute $c^* \leftarrow_{\mathrm{R}} \mathsf{UE.Enc}(k_e, m_b)$ and does not corrupt the secret key in any challenge-equal epoch.

Definition 12. *An updatable encryption scheme* UE *is called UP-IND-RCCA secure if for any PPT adversary $\mathcal{A}$ the following advantage is negligible in κ:* $\mathsf{Adv}^{\mathit{up\text{-}ind\text{-}rcca}}_{\mathsf{UE},\mathcal{A}}(pp) := \left|\Pr[\mathsf{Exp}^{\mathit{up\text{-}ind\text{-}rcca}}_{\mathsf{UE},\mathcal{A}}(pp, 0) = 1] - \Pr[\mathsf{Exp}^{\mathit{up\text{-}ind\text{-}rcca}}_{\mathsf{UE},\mathcal{A}}(pp, 1) = 1]\right|$.

```
Experiment $\mathsf{Exp}^{\mathsf{up\text{-}ind\text{-}rcca}}_{\mathsf{UE},\mathcal{A}}(pp, b)$
    $(sp, k_0, \Delta_0, \mathbf{Q}, \mathbf{K}, \mathbf{T}, \mathbf{C}^*) \leftarrow \mathsf{Init}(pp)$
    $(m_0, m_1, state) \leftarrow_{\mathrm{R}} \mathcal{A}^{\mathsf{Enc},\mathsf{Dec},\mathsf{Next},\mathsf{ReEnc},\mathsf{Corrupt}}(sp)$
    proceed only if $|m_0| = |m_1|$ and $m_0, m_1 \in \mathcal{M}_{sp}$
    $c^* \leftarrow_{\mathrm{R}} \mathsf{UE.Enc}(k_e, m_b)$, $\mathrm{M}^* \leftarrow (m_0, m_1)$, $\mathbf{C}^* \leftarrow \{e\}$, $e^* \leftarrow e$
    $b' \leftarrow_{\mathrm{R}} \mathcal{A}^{\mathsf{Enc},\mathsf{Dec},\mathsf{Next},\mathsf{ReEnc},\mathsf{Corrupt}}(c^*, state)$
    return $b'$ if $\mathbf{K} \cap \widehat{\mathbf{C}}^* = \emptyset$, i.e. $\mathcal{A}$ did not trivially win. (Else abort.)
```

Handling queries of (potential) challenge ciphertexts. As in the standard RCCA definition, we do not allow any decryption of ciphertexts that decrypts to either of the two challenge plaintexts m_0, or m_1. This is expressed via the isChallenge predicate that is checked for every Dec and ReEnc query and is defined as follows:

$\mathsf{isChallenge}(k_i, c)$:

– If $\mathsf{UE.Dec}(k_i, c) = m_b$ where $m_b \in \mathrm{M}^*$, return `true`. Else, return `false`.

Whereas the decryption oracle will ignore any query where $\mathsf{isChallenge}(k_e, c) =$ `true`, the re-encryption oracle is more generous: When ReEnc is invoked on (i, c) where $\mathsf{isChallenge}(k_i, c) =$ `true`, it will still update the ciphertext into the current epoch e. The oracle might mark the epoch e as challenge-equal though, preventing the adversary from corrupting the secret key of epoch e. However, this is only done when c is *not* a previous oracle response from an encryption query (or re-encryption of such a response). That is, the re-encryption oracle will treat ciphertexts normally when they can be traced down to a honest encryption query, even when they encrypt one of the challenge messages. This added "generosity" is crucial for re-encryptions, as otherwise an adversary would not be able to see *any* re-encryption from a ciphertext that encrypts the same message as the challenge *and* corrupt the secret key in such an epoch.

Plaintext Integrity. Another impact of having a probabilistic instead of a deterministic ReEnc algorithm is that ciphertext integrity can no longer be guaranteed: When the adversary has corrupted an update token it can create various valid ciphertexts by updating an old ciphertext into the new epoch. Thus, instead we aim for the notion of plaintext integrity and request the adversary to produce a ciphertext that decrypts to a message for which he does not trivially know an encryption of.

The oracles used in this game are as in the UP-IND-RCCA definition above, except that we no longer need the $\mathsf{isChallenge}$ predicate and the set of honest queries $\mathbf{Q}$ only records the plaintexts but not the ciphertexts.

$\mathsf{Next}()$, $\mathsf{Corrupt}(\{\mathsf{key}, \mathsf{token}\}, i)$: as in CCA game
$\mathsf{Enc}(m)$: Returns $c \leftarrow_{\mathrm{R}} \mathsf{UE.Enc}(k_e, m)$ and sets $\mathbf{Q} \leftarrow \mathbf{Q} \cup \{(e, m)\}$.
$\mathsf{Dec}(c)$: Returns $m \leftarrow \mathsf{UE.Dec}(k_e, c)$ and sets $\mathbf{Q} \leftarrow \mathbf{Q} \cup \{(e, m)\}$.
$\mathsf{ReEnc}(c, i)$: Returns c_e, the re-encryption of c from epoch i to the current epoch e. It also sets $\mathbf{Q} \leftarrow \mathbf{Q} \cup \{(e, m)\}$ where $m \leftarrow \mathsf{UE.Dec}(k_e, c_e)$.

As in our definition of UP-INT-CTXT, we have to capture all plaintexts for which the adversary can easily create ciphertexts, based on the information he learned through the oracles and by exploiting his knowledge of some of the secret keys and update tokens. Again, the first case in our definition of UP-INT-PTXT security excludes adversaries that have corrupted a secret key and all tokens from then on, as this allows to create valid ciphertexts for *all* plaintexts

Definition 13. *An updatable encryption scheme* UE *is called UP-INT-PTXT secure if for any PPT adversary* $\mathcal{A}$ *the following advantage is negligible in* κ*:* $\mathsf{Adv}^{\mathit{up\text{-}int\text{-}ptxt}}_{\mathsf{UE},\mathcal{A}}(pp) := \Pr[\mathsf{Exp}^{\mathit{up\text{-}int\text{-}ptxt}}_{\mathsf{UE},\mathcal{A}}(pp) = 1]$.

```
Experiment $\mathsf{Exp}^{\mathsf{up\text{-}int\text{-}ptxt}}_{\mathsf{UE},\mathcal{A}}(pp)$
    $(sp, k_0, \Delta_0, \mathbf{Q}, \mathbf{K}, \mathbf{T}) \leftarrow \mathsf{Init}(pp)$
    $c^* \leftarrow_{\mathrm{R}} \mathcal{A}^{\mathsf{Enc},\mathsf{Dec},\mathsf{Next},\mathsf{ReEnc},\mathsf{Corrupt}}(sp)$
    return 1 if $\mathsf{UE.Dec}(k_{e_{\mathsf{end}}}, c^*) = m^* \neq \perp$ and $(e_{\mathsf{end}}, m^*) \notin \mathbf{Q}^*$,
      and $\nexists e \in \mathbf{K}$ where $i \in \mathbf{T}$ for $i = e$ to $e_{\mathsf{end}}$; i.e. if $\mathcal{A}$ does not trivially win.
```

Our definition of trivial plaintext forgeries $\mathbf{Q}^*$ is to the one for CTXT security. That is, when the adversary has received a ciphertext for a message m in epoch e (which is recorded in $\mathbf{Q}$) and the update token Δ_{e+1} (which is recorded in $\mathbf{T}$) for the following epoch, then we (iteratively) declare m to be a trivial forgery for epoch $e+1$ as well. We start with $\mathbf{Q}^* \leftarrow \emptyset$ and amend the set as follows:

```
for each $(e, m) \in \mathbf{Q}$:
    set $\mathbf{Q}^* \leftarrow \mathbf{Q}^* \cup (e, m)$, and $i \leftarrow e + 1$
    while $i \in \mathbf{T}$: set $\mathbf{Q}^* \leftarrow \mathbf{Q}^* \cup (i, m)$ and $i \leftarrow i + 1$
```

4.2 RCCA and PTXT Secure Construction

We now present our construction with probabilistic re-encryption that achieves our definition of RCCA and PTXT security (under leakage). The main idea is to use the Naor-Yung (NY) CCA-transform [NY90] (for public-key schemes). That is, a message is encrypted under two (public) keys of a CPA-secure encryption scheme and accompanied with a NIZK proof that both ciphertexts indeed encrypt the same message. By relying on building blocks that support key-rotation, we then lift this approach into the setting of updatable encryption. For the key-rotatable CPA-secure encryption we use the RISE scheme as presented in [LT18], and NIZKs are realized with Groth–Sahai (GS) proofs which provide the malleability capabilities that are necessary for key rotation. As in the case of our deterministic scheme presented in Sec. 3, we prove the full security of the updatable scheme based on static properties of the underlying building blocks and simulation-based properties of their token generation and update procedures.

A downside of this NY approach is that it yields a *public key* encryption scheme in disguise. That is, we expose the resulting public key scheme in a symmetric key style and only use the "public key" for key rotation. However, the corruption of an update token then allows the adversary to create valid ciphertexts for messages of his choice. Thus, this scheme would not achieve the desirable PTXT security yet. We therefore extend the NY approach and let each encryption also contain a proof that one knows a valid signature on the underlying plaintext. This combined scheme then satisfies both RCCA and PTXT security.

The crucial feature of this overall approach is that it allows for *public verifiability* of well-formedness of ciphertexts, and thus provides security under arbitrary (as opposed to queried) re-encryption attacks.

Structure of the rest of this section. We start with an overview of GS proofs systems and their essential properties. We continue with *perfect* re-encryption, a stronger definition than randomness-preserving. Then, we give give the intuition and definition of the basic NY-based RCCA-secure updatable encryption scheme. Finally we describe how to add plaintext integrity.

Linearly malleable proofs. As our proof system, we use Groth–Sahai proofs which is a so-called *commit-and-prove system* [GS12; EG14]. That is, one (first) commits to a witness w (with randomness r) and then proves statement(s) *stmt* about the committed witness by running $\pi \leftarrow \mathsf{Prove}(crs, stmt, w, r)$. The statement(s) *stmt* are "quadratic" equations, e.g. pairing product equations. See App. D for details.

Groth–Sahai proofs are a so-called *dual-mode* proof system, which has two setups: $\mathsf{GS.SetupH}(pp)$ (resp. $\mathsf{GS.SetupB}(pp)$) generates a hiding (resp. binding) *crs* for which commitments are perfectly hiding (resp. perfectly binding) and the proof π is perfectly zero-knowledge (resp. perfectly sound). Moreover, binding commitments to groups are *extractable.*

Groth–Sahai proofs offer extra-functionality. They are perfectly *rerandomisable*, i.e. the commitments and proofs can be re-randomised. Also, they are *linearly malleable.* Roughly, given a set of "quadratic" equations, one can apply (certain) linear transformations to the witness and statement (i.e. the constants in the equation), which map satisfying assignments to satisfying assignments, and compute adapted commitments and proofs. In particular, the commitments are homomorphic. See App. D or [Cha+12; Fuc11] for more.

Perfect re-encryption. Perfect re-encryption is a strengthening of randomness-preserving re-encryption. It assures that decrypt-then-encrypt has the same distribution as re-encryption, without any exceptions. In particular, it does neither require the encryption randomness, nor is it restricted to valid ciphertexts.

Definition 14. *Let* UE *be an updatable encryption scheme where* $\mathsf{UE.ReEnc}$ *is* probabilistic. *We say that re-encryption (of* UE*) is* ***perfect****, if for all* $sp \leftarrow_R \mathsf{UE.GenSP}(pp)$*, all keys* $k^{\text{old}}, k^{\text{new}} \leftarrow_R \mathsf{UE.GenKey}(sp)$*, token* $\Delta \leftarrow_R \mathsf{UE.GenTok}(k^{\text{old}}, k^{\text{new}})$*, and all ciphertexts* c*, we have*

$$\mathsf{UE.Enc}(k^{\text{new}}, \mathsf{UE.Dec}(k^{\text{old}}, c)) \overset{\text{dist}}{\equiv} \mathsf{UE.ReEnc}(\Delta, c).$$

Note that $\mathsf{Enc}(k, \bot) = \bot$ *by definition.*

The General Idea: RCCA Security via NY Transform. Our first goal is to build a UP-IND-RCCA-secure updatable encryption scheme. which we achieve via the double-encryption technique of Naor-Yung[NY90] using key-rotatable building blocks: we use a linear encryption with a *linearly malleable* NIZK, namely RISE (i.e. ElGamal-based updatable encryption) with Groth–Sahai proofs[GS12]. The malleability and re-randomizability of GS proofs allow for key rotation and ciphertext re-randomisation (as part of the re-encryption procedure).

A double-encryption with a *simulation sound* consistency proof (as formalized in [Sah99; Gro06]) is too rigid and yields CCA security. We must allow certain transformations of the ciphertext, namely re-randomisation and re-encryption. Thus, we weaken our security to RCCA and rely on a relaxation of simulation soundness, which still ensures that the adversary cannot maul the message, but allows re-randomisation and re-encryption.

We achieve this property by the following variation of a standard technique, which was previously used in conjunction with Groth–Sahai proofs, e.g. in [HJ12]. Our NIZK proves that either the NY statement holds, i.e., two ciphertexts

$c_1 = \mathsf{Enc}(pk_1, m_1)$ and $c_2 = \mathsf{Enc}(pk_2, m_2)$ encrypt the same message $m_1 = m_2$, or m_1, m_2 (possibly being different) are signed under a signature verification key which is part of the system parameters. In the security proof the simulator will be privy of the signing key and thus can produce valid NIZK proofs for inconsistent ciphertexts. Further, the signature scheme is *structure-preserving*, which allows to hide the signature σ and its verification $\mathsf{Verify}(vk, m_1, m_2, \sigma)$ in the NIZK proof. Note that the signature scheme does *not* have to be key-rotatable as the key is fixed throughout all epochs.

Definition 15 (NYUE). *Our Naor–Yung-like transformation* NYUE *of the key-rotatable encryption* RISE*, using GS proofs and a structure-preserving signature* SIG*, is defined as:*

$\mathsf{NYUE.GenSP}(pp)$**:** *Run* $\mathrm{crs}_{\mathrm{GS}} \leftarrow_R \mathsf{GS.SetupH}(pp)$*,* $sp_{\mathsf{Enc}} \leftarrow_R \mathsf{RISE.GenSP}(pp)$*,* $sp_{\mathsf{SIG}} \leftarrow_R \mathsf{SIG.GenSP}(pp)$ *and* $(\_, vk_{\mathsf{SIG}}) \leftarrow_R \mathsf{SIG.GenKey}(sp_{\mathsf{SIG}})$*. Return* $sp = (\mathrm{crs}_{\mathrm{GS}}, sp_{\mathsf{Enc}}, (sp_{\mathsf{SIG}}, vk_{\mathsf{SIG}}))$*.*

$\mathsf{NYUE.GenKey}(sp)$**:** *Run* $k_i \leftarrow_R \mathsf{RISE.GenKey}(sp_{\mathsf{Enc}})$ *for* $i = 1, 2$ *and parse* $k_i = (sk_i, pk_i)$*. Let* $sk = (sk_1, sk_2)$ *and* $pk = (pk_1, pk_2)$*. Return* $k = (sk, pk)$*.*

$\mathsf{NYUE.Enc}(k, m; r_1, r_2)$**:** *Parse* $k = (sk, pk)$*. Compute* $c_i = \mathsf{RISE.Enc}(pk_i, m; r_i)$ *for* $i = 1, 2$ *and the following proof* $\pi \leftarrow_R \mathsf{NIZK}(\mathsf{OR}(S_{\mathrm{NY}}, S_{\mathsf{SIG}}))$ *with common input* sp, pk_1, pk_2, c_1, c_2 *where*[6]
- S_{NY}*:* $\exists \widehat{m}, \widehat{r_1}, \widehat{r_2}\colon \mathsf{RISE.Enc}(pk_1, \widehat{m}; \widehat{r_1}) = c_1, \wedge \mathsf{RISE.Enc}(pk_2, \widehat{m}; \widehat{r_2}) = c_2$
- S_{SIG}*:* $\exists \widehat{m_1}, \widehat{m_2}, \widehat{r_1}, \widehat{r_2}, \widehat{\sigma}\colon \mathsf{RISE.Enc}(pk_1, \widehat{m_1}; \widehat{r_2}) = c_1 \wedge \mathsf{RISE.Enc}(pk_2, \widehat{m_2}; \widehat{r_2}) = c_2 \wedge \mathsf{SIG.Verify}(vk_{\mathsf{SIG}}, (\widehat{m_1}, \widehat{m_2}), \widehat{\sigma}) = 1$

Return (c_1, c_2, π)*.*

$\mathsf{NYUE.Dec}(k, (c_1, c_2, \pi))$**:** *Parse* $k = (sk, pk)$ *and verify the proof* π *w.r.t.* $pk = (pk_1, pk_2)$*. If* π *is valid, return* $\mathsf{RISE.Dec}(sk_1, c_1)$*, and* $\perp$ *otherwise.*

$\mathsf{NYUE.GenTok}(k^{\mathrm{old}}, k^{\mathrm{new}})$**:** *Compute* $\Delta_i \leftarrow_R \mathsf{RISE.GenTok}(k_i^{\mathrm{old}}, k_i^{\mathrm{new}})$ *for* $i = 1, 2$ *where* k^{old} *and* k^{new} *is parsed as in* NYUE.GenKey*. Return* $\Delta = (\Delta_1, \Delta_2)$*.*

$\mathsf{NYUE.ReEnc}(\Delta, c)$**:** *is sketched below.*

We use a hiding $\mathrm{crs}_{\mathrm{GS}}$ in the above construction to attain *perfect* re-encryption. just like RISE, c.f. Rem. 2.

For the ease of exposition, we use RISE for both encryptions in the NY transform. If one follows the classical NY approach that immediately deletes sk_2 (in epoch 0), it would be sufficient to require key-rotatable encryption only for c_1, whereas encryption for c_2 merely needs to be re-randomizable (as we also aim for UP-REENC security).

Re-encryption for NYUE. The high-level idea of the re-encryption is using the linear malleability, and re-randomisability of RISE and GS proofs. For $\mathsf{NYUE.ReEnc}(\Delta, c)$ with $c = (c_1, c_2, \pi)$ we proceed in four steps. Steps 2 and 3 constitute a computation of RISE.ReEnc, separated into key-rotation and re-randomisation, c.f. Rem. 2.

[6] Here we exploit the public key nature of the construction, i.e., we only need pk_i (not sk_i) for verifying consistency proofs.

(1) Verify ciphertext. Note that the re-encryption tokens of RISE (and therefore NYUE) contain essentially the old and new public keys. We use this to let NYUE.ReEnc first verify the consistency proof of a ciphertext before starting the update procedure. Thus, re-encryption only works for well-formed, *decryptable* ciphertexts.

(2) Key rotation. We use the key rotation of RISE on the ciphertexts parts c_1 and c_2 of $c = (c_1, c_2, \pi)$, but *without* the implicit *re-randomisation*. Additionally, we use malleability of GS proofs to adapt the proof π.

(3) Re-randomise c_1, c_2. We re-randomise the RISE (i.e. ElGamal) ciphertexts c_1, c_2, thus completing the computation of $\mathsf{RISE.ReEnc}(\Delta_i, c_i)$ for $i = 1, 2$. Additionally, we use malleability of GS proofs to adapt the proof π.

(4) Re-randomise π. We re-randomise the proof π using re-randomisability of GS proofs.

Thus, we first switch to the new key, and then ensure that the ciphertext is distributed identically to a fresh encryption by re-randomising the RISE ciphertexts and the GS proofs (both of which are perfectly re-randomisable).

UP-IND-RCCA Security of NYUE. We now argue how NYUE achieves our notion of UP-IND-RCCA security that captures arbitrary re-encryption attacks. First, we observe that NYUE has *perfect* re-encryption, i.e., a re-encrypted ciphertext (c_1', c_2', π') has the same distribution as a fresh encryption (Def. 14). This follows because RISE has perfect re-encryption and GS proofs with hiding CRS have perfect re-randomisation. Furthermore, NYUE satisfies simulatable token generation *under (key-)leakage*, see App. C.

Lemma 4. *The updatable encryption scheme* NYUE *has perfect re-encryption and simulatable token generation under leakage* $\mathsf{leak}(k) = pk$*, c.f. App. C.*

Lemma 4 follows easily from token simulation for RISE, see App. G. The UP-IND-RCCA security of NYUE is shown analogous to UP-IND-CCA security in Thm. 1. That is, we bootstrap the UP-IND-RCCA security from the (static) IND-RCCA security of NYUE, perfect re-encryption and token simulation. By a standard reduction, the underlying encryption of NYUE is IND-RCCA secure (under leakage $\mathsf{leak}(k) = pk$), see App. G.1. There are three major differences compared to UP-IND-CCA:

First, NYUE.ReEnc uses the public verifiability of ciphertexts to reject invalid inputs, i.e., it updates only ciphertexts for which NYUE.Dec will not return $\perp$. Hence, the decrypt-then-encrypt strategy (used in the proof of Thm. 1) is not impacted by allowing arbitrary requests in the ReEnc oracle. Consequently, the queried restriction is not giving the adversary any additional advantage.

Second, re-encryption is *perfect*, which is stronger than randomness-preserving re-encryption. This simplifies the proof strategy slightly. Third, $\mathsf{leak}(k) = pk$ is non-trivial, unlike for the deterministic schemes. All in all, we obtain:

Proposition 1 (App. G). *Suppose the SXDH assumption holds in* $(e, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$*, and* SIG *is (one-time) EUF-CMA secure. Then the updatable encryption scheme* NYUE *from Def. 15 is UP-IND-RCCA secure.*

The SXDH assumption guarantees the security of RISE and GS proofs.

NYUAE Construction: Adding PTXT Security. As discussed, NYUE is a public key encryption scheme in disguise (with the public key "hidden" in the update token). Thus, a (corrupt) data host can trivially create new ciphertext to chosen messages, and thus we do not achieve the desired PTXT security yet.

To obtain such plaintext integrity, using a structure-preserving *key-rotatable* MAC [KPW15] on the plaintext seems a straightforward solution. However, for proving security against arbitrary re-encryption attacks, we need that ciphertext validity is *publicly verifiable.* Thus, we use the signature from [KPW15] instead (which is constructed from the MAC). Furthermore, we hide the signature (and its verification) behind a GS proof, to ensure confidentiality.

Updatable Signatures. Opposed to the signature SIG used in NYUE for the simulatability of the main GS proof, we need the signature scheme which ensures integrity of the plaintext to be key-rotatable and updatable as well. The definition of an updatable signature scheme USIG is straightforward and given in App. E. We stress that we will not require USIG to be secure in the updatable setting, but only need standard static (one-time) EUF-CMA security in combination with generic properties of the token generation (c.f. Def. 10).

We now incorporate plaintext integrity into the NYUE construction using such a key-rotatable signature USIG. For encryption, we additionally sign the plaintext with USIG and include this signature in the main NY statement of the GS proof π. That is, $S_{\mathrm{NY+I}}$ now asserts that c_1 and c_2 encrypt the same USIG-*signed* message. As before, we use concrete instantiations of all key-rotatable building blocks to avoid a cumbersome abstraction of malleability properties. We use the one-time signature OTS from [KPW15, Fig. 2] for USIG with simulatable token generation and malleable signature verification. In App. E we recall their scheme, define its key-rotation capabilities, and show that it satisfies all required properties (OTS is one-time EUF-CMA secure under the SXDH assumption).

In the following we describe our final construction NYUAE. For the sake of brevity, we refer to the NYUE scheme whenever we use it in an unchanged way.

Definition 16 (NYUAE). *The Naor-Yung transformation with plaintext integrity from key-rotatable encryption* RISE*, GS proofs and structure-preserving signature* SIG *(with* RISE *and* SIG *being abstracted away in the* NYUE *scheme), and a key-rotatable structure-preserving signature* OTS *(c.f. Def. 26) is defined as follows:*

$\mathsf{NYUAE.GenSP}(pp)$**:** *Run* $sp_{\mathsf{NYUE}} \leftarrow_R \mathsf{NYUE.GenSP}(pp)$*, and* $sp_{\mathsf{OTS}} \leftarrow_R \mathsf{OTS.GenSP}(pp)$*. Return* $sp = (sp_{\mathsf{NYUE}}, sp_{\mathsf{OTS}})$*.*

$\mathsf{NYUAE.GenKey}(sp)$**:** *Run* $k_{\mathsf{NYUE}} \leftarrow_R \mathsf{NYUE.GenKey}(sp_{\mathsf{NYUE}})$*, and* $(sk_{\mathsf{OTS}}, vk) \leftarrow_R \mathsf{OTS.GenKey}(sp_{\mathsf{OTS}})$*. Let* $sk = (sk_{\mathsf{NYUE}}, sk_{\mathsf{OTS}})$*,* $pk = (pk_{\mathsf{NYUE}}, vk)$*. Return* $k = (sk, pk)$*.*

$\mathsf{NYUAE.Enc}(k, m; r_1, r_2)$**:** *Parse* $k = ((sk_{\mathsf{NYUE}}, sk_{\mathsf{OTS}}), (pk_{\mathsf{NYUE}}, vk))$ *compute* c_1, c_2 *as in* NYUE*,* $\sigma \leftarrow \mathsf{OTS.Sign}(sk_{\mathsf{OTS}}, m)$ *and a proof* $\pi \leftarrow_R \mathsf{NIZK}(\mathsf{OR}(S_{\mathrm{NY+I}}, S_{\mathsf{SIG}}))$ *where*

– $S_{\mathrm{NY+I}}$*:* $\exists\, \widehat{m}, \widehat{r_1}, \widehat{r_2}, \widehat{\sigma} \colon \mathsf{OTS.Verify}(pk_{\mathsf{OTS}}, \widehat{m}, \widehat{\sigma}) = 1 \ \wedge\ S_{\mathrm{NY}}$
and with $S_{\mathrm{NY}}, S_{\mathsf{SIG}}$ *defined as in* NYUE *(Def. 15). Return* (c_1, c_2, π)*.*

$\mathsf{NYUAE.Dec}(k, (c_1, c_2, \pi))$**:** *If* π *is valid, return* $\mathsf{RISE.Dec}(k_1, c_1)$*, and* $\perp$ *else.*

$\mathsf{NYUAE.GenTok}(k^{\mathrm{old}}, k^{\mathrm{new}})$**:** *Run* $\Delta_{\mathsf{NYUE}} \leftarrow_R \mathsf{NYUE.GenTok}(k^{\mathrm{old}}_{\mathsf{NYUE}}, k^{\mathrm{new}}_{\mathsf{NYUE}})$ *and* $\Delta_{\mathsf{OTS}} \leftarrow_R \mathsf{OTS.GenTok}(k^{\mathrm{old}}_{\mathsf{OTS}}, k^{\mathrm{new}}_{\mathsf{OTS}})$*. Return* $\Delta = (\Delta_{\mathsf{NYUE}}, \Delta_{\mathsf{OTS}})$*.*

$\mathsf{NYUAE.ReEnc}(\Delta, c)$**:** *is as* NYUE.ReEnc *(Def. 15), but also adapts the proof of knowledge of an* OTS*-signature.*

The details for generating, verifying and updating the proof π are given in App. F. The proof of security as an updatable encryption scheme follows the usual blueprint. As for NYUE, UP-REENC security follows from Thm. 3.

Theorem 2. *Suppose* SIG *is* unbounded *EUF-CMA secure, and SXDH holds in* $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$*. Then* NYUAE *is UP-IND-RCCA and UP-INT-PTXT secure.*

Acknowledgments. We thank Kenny Paterson for fruitful discussions at early stages of this work. We also thank the reviewers for helpful suggestions. The first author is supported by the German Federal Ministry of Education and Research within the framework of the project "Sicherheit kritischer Infrastrukturen (SKI)" in the Competence Center for Applied Security Technology (KASTEL). The second author was supported by the European Union's Horizon 2020 research and innovation program under Grant Agreement No. 786725 (OLYMPUS). The third author is supported by DFG grant RU 1664/3-1 and KASTEL.

References

[Abe+10] Masayuki Abe, Georg Fuchsbauer, Jens Groth, Kristiyan Haralambiev, and Miyako Ohkubo. "Structure-Preserving Signatures and Commitments to Group Elements". In: *CRYPTO 2010*. Ed. by Tal Rabin. Vol. 6223. Aug. 2010, pp. 209–236.

[Bel+09] Mira Belenkiy, Jan Camenisch, Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Hovav Shacham. "Randomizable Proofs and Delegatable Anonymous Credentials". In: *CRYPTO 2009*. Ed. by Shai Halevi. Vol. 5677. Aug. 2009, pp. 108–125.

[Bon+13] Dan Boneh, Kevin Lewi, Hart William Montgomery, and Ananth Raghunathan. "Key Homomorphic PRFs and Their Applications". In: *CRYPTO 2013, Part I*. Ed. by Ran Canetti and Juan A. Garay. Vol. 8042. Aug. 2013, pp. 410–428.

[Bon+15] Dan Boneh, Kevin Lewi, Hart Montgomery, and Ananth Raghunathan. *Key Homomorphic PRFs and Their Applications*. Cryptology ePrint Archive, Report 2015/220. `http://eprint.iacr.org/2015/220`. 2015.

[Cha+12] Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Sarah Meiklejohn. *Malleable Proof Systems and Applications*. Cryptology ePrint Archive, Report 2012/012. `http://eprint.iacr.org/2012/012`. 2012.

[CKN03] Ran Canetti, Hugo Krawczyk, and Jesper Buus Nielsen. "Relaxing Chosen-Ciphertext Security". In: *CRYPTO 2003*. Ed. by Dan Boneh. Vol. 2729. Aug. 2003, pp. 565–582.

[Coh17] Aloni Cohen. *What about Bob? The Inadequacy of CPA Security for Proxy Reencryption*. Cryptology ePrint Archive, Report 2017/785. `http://eprint.iacr.org/2017/785`. 2017.

[EG14] Alex Escala and Jens Groth. "Fine-Tuning Groth-Sahai Proofs". In: *PKC 2014*. Ed. by Hugo Krawczyk. Vol. 8383. Mar. 2014, pp. 630–649.

[Esc+13] Alex Escala, Gottfried Herold, Eike Kiltz, Carla Ràfols, and Jorge Villar. "An Algebraic Framework for Diffie-Hellman Assumptions". In: *CRYPTO 2013, Part II*. Ed. by Ran Canetti and Juan A. Garay. Vol. 8043. Aug. 2013, pp. 129–147.

[Eve+17a] Adam Everspaugh, Kenneth Paterson, Thomas Ristenpart, and Sam Scott. *Key Rotation for Authenticated Encryption*. Cryptology ePrint Archive, Report 2017/527. `http://eprint.iacr.org/2017/527`. 2017.

[Eve+17b] Adam Everspaugh, Kenneth G. Paterson, Thomas Ristenpart, and Samuel Scott. "Key Rotation for Authenticated Encryption". In: *CRYPTO 2017, Part III*. Ed. by Jonathan Katz and Hovav Shacham. Vol. 10403. Aug. 2017, pp. 98–129.

[Fuc+18] Georg Fuchsbauer, Chethan Kamath, Karen Klein, and Krzysztof Pietrzak. *Adaptively Secure Proxy Re-encryption*. Cryptology ePrint Archive, Report 2018/426. `https://eprint.iacr.org/2018/426`. 2018.

[Fuc11] Georg Fuchsbauer. "Commuting Signatures and Verifiable Encryption". In: *EUROCRYPT 2011*. Ed. by Kenneth G. Paterson. Vol. 6632. May 2011, pp. 224–245.

[Gro06] Jens Groth. "Simulation-Sound NIZK Proofs for a Practical Language and Constant Size Group Signatures". In: *ASIACRYPT 2006*. Ed. by Xuejia Lai and Kefei Chen. Vol. 4284. Dec. 2006, pp. 444–459.

[GS12] Jens Groth and Amit Sahai. "Efficient Noninteractive Proof Systems for Bilinear Groups". In: *SIAM J. Comput.* 41.5 (2012), pp. 1193–1232.

[GW11] Craig Gentry and Daniel Wichs. "Separating succinct non-interactive arguments from all falsifiable assumptions". In: *43rd ACM STOC*. Ed. by Lance Fortnow and Salil P. Vadhan. June 2011, pp. 99–108.

[Her+17] Gottfried Herold, Max Hoffmann, Michael Klooß, Carla Ràfols, and Andy Rupp. "New Techniques for Structural Batch Verification in Bilinear Groups with Applications to Groth-Sahai Proofs". In: *ACM CCS 17*. Ed. by Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu. 2017, pp. 1547–1564.

[HJ12] Dennis Hofheinz and Tibor Jager. "Tightly Secure Signatures and Public-Key Encryption". In: *CRYPTO 2012*. Ed. by Reihaneh Safavi-Naini and Ran Canetti. Vol. 7417. Aug. 2012, pp. 590–607.

[Hof16] Dennis Hofheinz. "Algebraic Partitioning: Fully Compact and (almost) Tightly Secure Cryptography". In: *TCC 2016-A, Part I.* Ed. by Eyal Kushilevitz and Tal Malkin. Vol. 9562. Jan. 2016, pp. 251–281.

[Jaf+17] Zahra Jafargholi, Chethan Kamath, Karen Klein, Ilan Komargodski, Krzysztof Pietrzak, and Daniel Wichs. "Be Adaptive, Avoid Overcommitting". In: *CRYPTO 2017, Part I.* Ed. by Jonathan Katz and Hovav Shacham. Vol. 10401. Aug. 2017, pp. 133–163.

[JKR18] Stanislaw Jarecki, Hugo Krawczyk, and Jason Resch. *Threshold Partially-Oblivious PRFs with Applications to Key Management.* Cryptology ePrint Archive, Report 2018/733. `https://eprint.iacr.org/2018/733`. 2018.

[KPW15] Eike Kiltz, Jiaxin Pan, and Hoeteck Wee. "Structure-Preserving Signatures from Standard Assumptions, Revisited". In: *CRYPTO 2015, Part II.* Ed. by Rosario Gennaro and Matthew J. B. Robshaw. Vol. 9216. Aug. 2015, pp. 275–295.

[LT18] Anja Lehmann and Björn Tackmann. "Updatable Encryption with Post-Compromise Security". In: *EUROCRYPT 2018, Part III.* Ed. by Jesper Buus Nielsen and Vincent Rijmen. Vol. 10822. 2018, pp. 685–716.

[MRV16] Paz Morillo, Carla Ràfols, and Jorge Luis Villar. "The Kernel Matrix Diffie-Hellman Assumption". In: *ASIACRYPT 2016, Part I.* Ed. by Jung Hee Cheon and Tsuyoshi Takagi. Vol. 10031. Dec. 2016, pp. 729–758.

[NPR99] Moni Naor, Benny Pinkas, and Omer Reingold. "Distributed Pseudorandom Functions and KDCs". In: *EUROCRYPT'99.* Ed. by Jacques Stern. Vol. 1592. May 1999, pp. 327–346.

[NRS14] Chanathip Namprempre, Phillip Rogaway, and Thomas Shrimpton. "Reconsidering Generic Composition". In: *EUROCRYPT 2014.* Ed. by Phong Q. Nguyen and Elisabeth Oswald. Vol. 8441. May 2014, pp. 257–274.

[NY90] Moni Naor and Moti Yung. "Public-key Cryptosystems Provably Secure against Chosen Ciphertext Attacks". In: *22nd ACM STOC.* May 1990, pp. 427–437.

[Ràf15] Carla Ràfols. "Stretching Groth-Sahai: NIZK Proofs of Partial Satisfiability". In: *TCC 2015, Part II.* Ed. by Yevgeniy Dodis and Jesper Buus Nielsen. Vol. 9015. Mar. 2015, pp. 247–276.

[Sah99] Amit Sahai. "Non-Malleable Non-Interactive Zero Knowledge and Adaptive Chosen-Ciphertext Security". In: *40th FOCS.* Oct. 1999, pp. 543–553.

[PCI16] PCI Security Standards Council. *Requirements and security assessment procedures.* PCI DSS v3.2. 2016.

A Unlinkability of Re-Encryptions

Our security models UP-IND-CCA and UP-IND-RCCA lift the classical CCA and RCCA notions to the setting of updatable encryption. As argued in [Eve+17b] and [LT18], these definitions would allow parts of the ciphertext to remain static which somehow contradicts the idea that full security can be re-gained when moving a ciphertext into a new epoch. Both aforementioned works therefore introduce an additional security notion that guarantees that a re-encrypted ciphertext can no longer be linked to its old version, thus capturing that the full ciphertext must get refreshed in an update.

We follow their argumentation and adapt the notion of unlinkability of re-encryptions from [LT18] to the CCA and RCCA setting of our work (which we call UP-REENC). We also point out that an even stronger notion might be desirable: namely that fresh and re-encrypted ciphertexts are indistinguishable (which is not guaranteed by UP-REENC). Interestingly, defining these stronger security notions (Definitions 11 and 14) is much simpler than the adaptive UP-REENC notion, and (in combination with UP-IND-RCCA resp. UP-IND$-CCA security) imply UP-REENC security.

A.1 Defining UP-REENC Security

We now present our definitions for updatable encryption with unlinkability of re-encryptions. This notion, called UP-REENC, has a very similar security game to UP-IND-CCA and UP-IND-RCCA. The major difference is, that it deals with re-encryptions of ciphertexts instead of encryptions of plaintexts. That is, the challenge *messages* are replaced by challenge *ciphertexts*, and the challenge *encryption* is replaced by a challenge *re-encryption*. We only allow *valid* ciphertext as challenge, i.e. ciphertexts which do not decrypt to $\perp$.

UP-REENC necessarily comes in multiple variants. Namely, the re-encryption oracle may restrict to queried re-encryptions or allow arbitrary re-encryptions. The decryption (and re-encryption) oracle may reject ciphertexts, depending on the definition of $\mathsf{isChallenge}$ and we need some definition of trivial wins. A suitable combination of these properties yields the UP-REENC notions for deterministic (resp. probabilistic) re-encryption and CCA (resp. RCCA) security. We will only consider the combinations of deterministic queried re-encryption plus CCA security and probabilistic arbitrary re-encryption plus RCCA security. We abbreviate these cases simply by referring to CCA and RCCA respectively.[7]

Oracles. The oracles and game are essentially the same as in the UP-IND-CCA and UP-IND-RCCA games:

$\mathsf{Next}()$: as always.

[7] These are the relevant cases for us. Other combinations, are straightforward to define. For example, (probabilistic) queried re-encryption plus CPA security corresponds to the definition of UP-REENC in [LT18].

$\mathsf{Enc}(m)$**:** Returns $c \leftarrow_{\mathrm{R}} \mathsf{UE.Enc}(k_{e_{\mathrm{cur}}}, m)$. Adds the information to $\mathbf{Q}$ as specified in the CCA resp. RCCA game.

$\mathsf{Dec}(c)$**:** If $\mathsf{isChallenge}(e_{\mathrm{cur}}, c) = \mathtt{false}$, the oracle returns $m \leftarrow \mathsf{UE.Dec}(k_{e_{\mathrm{cur}}}, c)$. The definition of $\mathsf{isChallenge}$ depends whether a CCA resp. RCCA variant is played.

$\mathsf{ReEnc}(c, i)$**:** Works as specified in the CCA resp. RCCA game. In particular, it updates the respective sets $\mathbf{Q}$ and $\mathbf{C}^*$ and checks $\mathsf{isChallenge}$. (Also the queried re-encryptions restriction is enforced.)

$\mathsf{Corrupt}(\{\mathsf{key}, \mathsf{token}\}, i)$**:** as always.

The challenge in an UP-REENC experiment are two ciphertexts c_0, c_1, which are re-encrypted from the previous epoch $e_{\mathrm{cur}} - 1$ to the next epoch e_{cur}. Let c_0^*, c_1^* be the re-encryptions. The adversary obtains c_b^* (for $b \leftarrow_{\mathrm{R}} \{0,1\}$) and must guess b. Whereas in the RCCA variant, both ciphertexts can be chosen arbitrarily by the adversary (as long as both decrypt to some message $m_b \neq \bot$), we limit the adversary to queried-ciphertexts in the CCA game.

Trivial wins for the UP-REENC games are defined exactly as in the UP-IND-CCA and UP-IND-RCCA notions, building the inferable set $\widehat{\mathbf{C}}^*$ of challenge-equal epochs and requiring the adversary not to corrupt a key (captured in the set $\mathbf{K}$) in a challenge-equal epoch. Clearly, there is one additional and unavoidable restriction when re-encryption is deterministic: An adversary which corrupts Δ_{e^*} loses due to trivial win.

Now, we give explicit security games for the two relevant combinations: UP-REENC-CCA security for schemes for deterministic re-encryption, and UP-REENC-RCCA security for schemes that re-encrypt probabilistically.

Definition 17. *An updatable encryption scheme* UE *is called UP-REENC-X secure for* $X \in \{CCA, RCCA\}$ *if for any PPT adversary* $\mathcal{A}$ *the following advantage is negligible in* κ*:*

$$\mathsf{Adv}^{\textit{up-reenc-x}}_{\mathsf{UE},\mathcal{A}}(pp) := \left| \Pr[\mathsf{Exp}^{\textit{up-reenc-x}}_{\mathsf{UE},\mathcal{A}}(pp, 0) = 1] - \Pr[\mathsf{Exp}^{\textit{up-reenc-x}}_{\mathsf{UE},\mathcal{A}}(pp, 1) = 1] \right| .$$

```
Experiment $\mathsf{Exp}^{\text{up-reenc-cca}}_{\mathsf{UE},\mathcal{A}}(pp, b)$
  $(sp, k_0, \Delta_0, \mathbf{Q}, \mathbf{K}, \mathbf{T}, \mathbf{C}^*) \leftarrow \mathsf{Init}(pp)$
  $(c_0, c_1, state) \leftarrow_{\mathrm{R}} \mathcal{A}^{\mathsf{Enc},\mathsf{Dec},\mathsf{Next},\mathsf{ReEnc},\mathsf{Corrupt}}(sp)$
  proceed only if $|c_0| = |c_1|$ and $(e_{\mathrm{cur}} - 1, c_0) \in \mathbf{Q}$ and $(e_{\mathrm{cur}} - 1, c_1) \in \mathbf{Q}$
  $c^* \leftarrow_{\mathrm{R}} \mathsf{UE.ReEnc}(\Delta_{e_{\mathrm{cur}}}, c_b)$, $\mathbf{C}^* \leftarrow \{e_{\mathrm{cur}}\}$
  $b' \leftarrow_{\mathrm{R}} \mathcal{A}^{\mathsf{Enc},\mathsf{Dec},\mathsf{Next},\mathsf{ReEnc},\mathsf{Corrupt}}(c^*, state)$
  return $b$ and $\mathbf{K} \cap \widehat{\mathbf{C}}^* = \emptyset$ and $e^* \notin \mathbf{T}$, i.e. if $\mathcal{A}$ did not trivially win.
```

```
Experiment $\mathsf{Exp}^{\text{up-reenc-rcca}}_{\mathsf{UE},\mathcal{A}}(pp, b)$
  $(sp, k_0, \Delta_0, \mathbf{Q}, \mathbf{K}, \mathbf{T}, \mathbf{C}^*) \leftarrow \mathsf{Init}(pp)$
  $(c_0, c_1, state) \leftarrow_{\mathrm{R}} \mathcal{A}^{\mathsf{Enc},\mathsf{Dec},\mathsf{Next},\mathsf{ReEnc},\mathsf{Corrupt}}(sp)$
  proceed only if $|c_0| = |c_1|$ and $m_0 \neq \bot$ and $m_1 \neq \bot$ where
    $m_0 \leftarrow \mathsf{Dec}(k_{e_{\mathrm{cur}}-1}, c_0)$ and $m_1 \leftarrow \mathsf{Dec}(k_{e_{\mathrm{cur}}-1}, c_1)$
  $c^* \leftarrow_{\mathrm{R}} \mathsf{UE.ReEnc}(\Delta_{e_{\mathrm{cur}}}, c_b)$, $\mathrm{M}^* \leftarrow (m_0, m_1)$, $\mathbf{C}^* \leftarrow \{e_{\mathrm{cur}}\}$
  $b' \leftarrow_{\mathrm{R}} \mathcal{A}^{\mathsf{Enc},\mathsf{Dec},\mathsf{Next},\mathsf{ReEnc},\mathsf{Corrupt}}(c^*, state)$
  return 1 if $b' = b$ and $\mathbf{K} \cap \widehat{\mathbf{C}}^* = \emptyset$, i.e. if $\mathcal{A}$ did not trivially win
```

Indistinguishability of fresh vs. re-encrypted ciphertexts. Note that UP-REENC guarantees unlinkability of updated ciphertexts, but it does not ensure that updated and fresh ones are indistinguishable. For instance, an updatable encryption scheme could simply set a "is-fresh" bit to 1 for freshly encrypted ciphertexts (i.e. Enc) and set it to 0 upon re-encryption (i.e. ReEnc). Thus, "fresh" ciphertexts are trivially distinguishable from "old" ones (but the scheme would still be be UP-REENC secure). A stronger notion would also prevent such leakage.

We observe that a very simple approach can be used for expressing this additional security: Perfect and randomness-preserving re-encryption (as defined in Definitions 11 and 14) guarantee that such a leak does not exist, and imply (in combination with UP-IND-RCCA resp. UP-IND\$-CCA security) the UP-REENC notions defined above.

A.2 UP-REENC-RCCA from perfect re-encryption

In this section, we show that UP-REENC-RCCA is actually implied by perfect re-encryption and a decryption oracle.

Theorem 3. *Let* UE *be an updatable encryption scheme with perfect re-encryption. Suppose that* UE *is UP-IND-RCCA secure. Then* UE *is UP-REENC-RCCA secure (via a tight reduction to UP-IND-RCCA).*

Proof. Let $\mathcal{C}$ be the UP-IND-RCCA challenger who plays with $\mathcal{B}$, and let $\mathcal{A}$ be the UP-REENC-RCCA adversary (who plays against $\mathcal{B}$). By construction $\mathcal{B}$, simply forwards (and records) all oracle calls of $\mathcal{A}$ to $\mathcal{C}$.[8] In particular, Dec behaves exactly the same in both games. Naturally, there is one exception: In epoch e^*, the adversary sends the ciphertext pair c_0, c_1. By assumption c_0 and c_1 are valid ciphertexts under epoch $e^* - 1$, hence $\mathcal{B}$ can decrypt them using $\mathcal{C}$. More concretely, $\mathcal{B}$ requests $\mathcal{C}$ to rotate c_i to epoch e^*, obtaining c_i' and then requests the decryption $m_i \leftarrow \mathcal{C}.\mathsf{Dec}(c_i')$. (Remember that the decryption oracle works (implicitly) under the current epoch e_{cur}, which is e^* in this case.) Note that $m_i \neq \perp$ (otherwise $\mathcal{A}$ loses and $\mathcal{B}$ can abort). Now, $\mathcal{B}$ issues m_0, m_1 as its own challenge messages to $\mathcal{C}$ (under epoch e^*). Thus, $\mathcal{B}$ obtains $c^* = \mathsf{Enc}(k_{e^*}, m_b)$, where $\mathcal{C}$ picks $b \leftarrow_{\text{R}} \{0, 1\}$. The distribution of c^* is *perfectly indistinguishable* from $\mathsf{ReEnc}(\Delta_{e^*}, c_b)$ because UE has perfect re-encryption. Consequently, $\mathcal{B}$ perfectly simulates the UP-REENC-RCCA game for $\mathcal{A}$ up to this point.

Continuing from this point onwards, it is evident that only the simulation of the Dec-oracle could be problematic. Fortunately, the UP-IND-RCCA challenger $\mathcal{C}$ rejects a decryption query (only) if $\mathsf{Dec}(k_{e_{\text{cur}}}, c) \in \{m_0, m_1\}$ and c was not queried. This is the same as in the UP-REENC-RCCA game. Consequently, $\mathcal{B}$ perfectly simulates the UP-REENC-RCCA game for $\mathcal{A}$. Finally, $\mathcal{B}$ simply forwards $\mathcal{A}$'s guess as its own guess. Due to perfect simulation $\mathcal{B}$ wins the UP-IND-RCCA if and only if $\mathcal{A}$ wins UP-REENC-RCCA.

[8] Note that the UP-IND-RCCA game allows corruption of the same keys as the UP-REENC-RCCA game, but possibly more tokens (for the deterministic re-encryption).

In fact, even the UP-REENC security of RISE follows as above, by considering a "trivial" decryption oracle, which "decrypts" by a table-lookpu of ciphertexts which were previously encrypted via Enc or rotated. (Remember that RISE only guarantees security against queried re-encryptions, not against arbitrary. Thus, the challenge ciphertext c_0, c_1 are honestly generated and their plaintext is known.)

A.3 UP-REENC-CCA from randomness-preserving reencryption

Unlike perfect re-encryption which easily implies UP-REENC-RCCA, the situation for randomness-preserving re-encryption is less convenient. Nevertheless, we give a sufficient criterion for this implication to hold. For this we define the following.

Definition 18. *The UP-IND\$-CCA security of an updatable encryption scheme is defined exactly like the UP-IND-CCA security, except for the following: The challenge is only a single message m^*. The challenge ciphertext c^* is either an encryption of m^* or is drawn uniformly at random from $\mathcal{C}$.*

The security definition of strong *UP-IND\$-CCA is the same as for UP-IND\$-CCA, except that encryption oracle calls (and the challenge encryption) return the (purported) encryption randomness.*[9] *Similarly, we define* strong *UP-IND-CCA security.*

Evidently, UP-IND\$-CCA implies UP-IND-CCA. Furthermore, the results from Thm. 1 work without change for IND\$ instead of IND. And it is easy to see that *strong* UP-IND-CCA and *strong* UP-IND\$-CCA are also implied by Thm. 1, since the underlying encryption is assumed *strongly* secure anyway. The extension of *strong* security to updatable schemes in done in the obvious way: All encryptions (including the challenge encryption) leak the (purported) encryption randomness to the adversary.

Theorem 4. *Let* UE *be an updatable encryption scheme with deterministic reencryption. Suppose that* UE *has* randomness-preserving *reencryption and* simulatable *token generation and the underlying encryption scheme* SKE *is* tidy. *Then, under queried reencryption,* UE *is*

- *strongly UP-IND-CCA-secure if* SKE *is strongly IND-CCA-secure*
- *strongly UP-IND\$-CCA-secure if* SKE *is strongly IND\$-CCA-secure*

Proof. The proof of Thm. 1 works essentially unchanged. For achieving *strong* UP-IND-CCA security, note that we already took care of all the encryption randomness in the that proof. Indeed, the embedded challenger was assumed to be *strong* IND-CCA-secure. For IND\$-CCA, the changes to the embedding of the challenger are straightforward.

[9] Here we assume that the encryption randomness can be chosen "independent" of a ciphertext. This is the case in all of our schemes.

With this, we can state our theorem.

Theorem 5. *Let* UE *be an updatable encryption scheme with deterministic* randomness-preserving *reencryption. Suppose that* UE *is* strong *UP-IND\$-CCA secure (under queried reencryption). Then* UE *is (strong) UP-REENC-CCA secure (under queried reencryption).*

The reason for IND\$ instead of IND is trouble with consistency of encryption randomness. In UP-REENC-CCA games, we have *two* challenge ciphertexts c_0, c_1. They are embedded in UP-IND\$-CCA by guessing which messages m_0, m_1 they encrypt (plus randomness). Here we crucially rely on the *queried* restriction, as well as *deterministic* reencryption, to be able to guess the adversary's *ciphertexts* by guessing from which $\mathsf{Enc}(m)$ they originate. When embedding the CCA challenge in these messages, we ensure *consistent encryption randomness*, as we did in the proof of Thm. 1. Herein lies the reason for IND\$. If we use IND-security, we can replace challenge reencryptions c^* of (the respective ciphertexts of) m_0 and m_1 by encryptions of 0. But the encryption *randomness* for c^*, i.e. r_0 or r_1, would still be consistent w.r.t. the challenge ciphertexts prior to the challenge reencryption (even though the encrypted message is inconsistent). We do not know how to finish the reduction in this case. Our solution is to rely on IND\$-security, where the ciphertext is replaced by a random ciphertext. Thus, it bears no connection to the purported reencryption randomness.[10]

There are some smaller technical issues with reducing UP-REENC-CCA to UP-IND\$-CCA. For example, the initial encryptions of m_0 and m_1 may be in different epochs and long before the actual challenge epoch. But the solutions to these problems are straightforward.

Proof. We only give a proof sketch which leaves out many formal details, e.g. tracking all (re)encryptions as done in Thm. 1. All details are straightforward to fill in by consulting earlier proofs.

Game 1: Let $\mathcal{B}$ be the UP-REENC-CCA challenger, playing with $\mathcal{A}$. Let $b_{\mathcal{B}}$ be $\mathcal{B}$'s challenge bit. We hook an UP-IND\$-CCA challenger $\mathcal{C}$ up to our UP-REENC-CCA game. For now, this is only conceptual. More precisely, we make $\mathcal{B}$ against an adversary against UP-IND\$-CCA. In this game, we merely proxy all requests, i.e. all oracles calls. This is perfectly "simulates" UP-REENC-CCA for $\mathcal{A}$. (The output of $\mathcal{B}$ does not matter at the moment, but suppose that $\mathcal{B}$ forwards $\mathcal{A}$'s guess as it's own. Since $\mathcal{B}$ never requests a challenge from $\mathcal{C}$, its "advantage" is 0.)

Game 2: $\mathcal{B}$ guesses the challenge epoch e^* and the encryption requests ℓ_0, ℓ_1 which correspond to the ciphertexts c_0, c_1 which $\mathcal{A}$ will use as its challenge, i.e. c_i is an update of $\mathsf{Enc}(m_{\ell_i})$, where the messages from requested encryptions

[10] Weaker security notions, e.g. a definition of "very strong" (UP)-RoR-CCA where the challenge (c, r) either satisfies $c = \mathsf{Enc}(k, m, r)$ or $c = \mathsf{Enc}(k, s_1, s_2)$ for random r, s_1, s_2 would suffice as well. But are even more tailored for our purpose, hence we chose IND\$ for the presentation. (This is stronger than the natural "strong" notion of RoR-CCA. The purported encryption randomness r is now also inconsistent.)

are enumerated as $m_0, m_1, \ldots$. Note that this is possible, since the adversary is restricted to queried reencryption. We assume correct guesses in the following.

Game 3: $\mathcal{B}$ immediately "skips" to epoch e^* (by calling $\mathcal{C}.\mathsf{Next}()$ e^* times). Also, $\mathcal{B}$ corrupts the keys $k_0, \ldots, k_{e^*-1}$ and tokens $\{\Delta_1, \ldots, \Delta_{e^*-1}\}$ before the challenge epoch e^*. For all epochs $0, \ldots, e^*-1$, $\mathcal{B}$ channels encryptions through $\mathcal{C}$ (in epoch e^*) to obtain the encryption randomness, which $\mathcal{B}$ uses to answer encryption requests using the corrupted key. Here, we use that $\mathcal{B}$ plays a *strong* UP-IND\$-CCA game. This works just as in Thm. 1 and exploits randomness-preserving reencryption. For all later epochs, $\mathcal{B}$ does everything through $\mathcal{C}$ again. This is change of behaviour of internal workings is perfectly indistinguishable.

Note that corrupting these epochs and tokens will not pose a problem, since the UP-IND-CCA challenge will be embedded in e^*. Furthermore, since $\mathcal{A}$ plays UP-REENC-CCA with *deterministic* reencryption, we can assume that $\mathcal{A}$ does not corrupt token Δ_{e^*} (as this implies immediate loss of the game).

Game 4 replaces the reencryption c_0^* by randomness. Here, we finally embed a challenge. Instead of honestly encrypting message m_{ℓ_0} via $\mathcal{C}.\mathsf{Enc}(m_{\ell_0})$, $\mathcal{B}$ passes m_{ℓ_0} as its challenge to $\mathcal{C}$. As answer, $\mathcal{B}$ obtains purported encryption randomness r_0 and ciphertext c^*, which $\mathcal{B}$ treats as c_0^* for $\mathcal{A}$. From now on, $\mathcal{B}$ continues honestly, i.e. it encrypts m_{ℓ_0} with randomness r_0 in the epoch $\mathcal{A}$ originally requested. Everything else works exactly as in Game 3.

If $\mathcal{A}$ distinguishes between Games 3 and 4, $\mathcal{B}$ distinguishes between the real encryption and the random ciphertext by simply forwarding $\mathcal{A}$'s guess.

Game 5 replaces the reencryption c_1^* by randomness. This is done analogous to Game 4.

Game 6: The challenge reencryption for $\mathcal{A}$ is c_b^*, where $c_0^*, c_1^* \leftarrow_{\mathrm{R}} \mathcal{C}$ are drawn uniformly at random by $\mathcal{B}$. Therefore, the view of $\mathcal{A}$ is independent of the bit $b_{\mathcal{B}}$. Hence $\mathcal{A}$ has advantage 0.

Note again that, if c_0^*, c_1^* are replaced by encryptions of 0, but with *consistent* randomness r_0, r_1, then the view of $\mathcal{A}$ *still* depends on $b_{\mathcal{B}}$. Thus, to "randomise the randomness", we required IND\$ instead of IND. Also note that giving $\mathcal{A}$ the encryption randomness for all of its encryption calls is not a problem, hence the proof actually shows *strong* UP-REENC-CCA security.

Unlike the reduction for UP-IND-RCCA, Thm. 3 suffers from a significant loss in advantage of $(e_{\max}+1)^{-3}$ due to guessing.

B Generic security proof for deterministic reencryption

In this section, we give the proofs of the generic security theorems for deterministic reencryption. Recall that we say a scheme is **strong** IND-CPA, IND-CCA, INT-CTXT, etc., secure if it is secure under following modification: The encryption oracle Enc additionally leaks the encryption randomness to the adversary.

We start with a useful lemma.

Lemma 5. *Suppose* UE *has deterministic randomness-preserving reencryption. Let* $k_0, \ldots, k_e$ *be a sequence of keys and* $\Delta_i \in \mathrm{supp}(\mathsf{GenTok}(k_{i-1}, k_i))$ *a sequence* *of update tokens connecting them. Then for any ciphertext $c_e^* = \mathsf{Enc}(k_e, m; r)$ we have that $c_0 := \mathsf{Enc}(k_0, m; r)$ yields reencryptions consistent with c_e^* in the sense that $c_e = c_e^*$, where $c_i := \mathsf{ReEnc}(\Delta_i, c_{i-1})$*

Suppose that Δ_i' are reverse tokens for Δ. Then the reverse of the above holds, i.e. let $c_0^* = \mathsf{Enc}(k_0, m, r)$ be arbitrary. Then we have that $c_e := \mathsf{Enc}(k_e, m; r)$ yields reencryptions consistent with c_e^* in the sense that $c_e = c_e^*$, where $c_{i-1} := \mathsf{ReEnc}(\Delta_i', c_i)$

Proof. The proof for the first statement is a simple consequence of the definition of randomness-preserving reencryption. Namely, it guarantees that $\mathsf{ReEnc}(\Delta, \mathsf{Enc}(k^{\text{old}}, m, r)) = \mathsf{Enc}(k^{\text{new}}, m, r)$ for any key and token choice. Thus, naturally, the claim holds for $e = 1$. By induction, the claim holds for arbitrary e.

The proof of the second statement follows from the first one, because our definition of reverse tokens requires that Δ' is a token itsself.

Note that the second statement need not hold for relaxed definitions of reverse token, i.e. definitions where reverse token information need not be of the form of an actual token and an algorithm different from ReEnc is allowed as "reverse reencryption". This relaxation is quite natural, but fortunately, all of our examples satisfy our stronger notion. From this lemma, we obtain a useful corollary.

Corollary 2. *Suppose the* UE *has tidy encryption. Then a ciphertext is uniquely determined by the message and encryption randomness. Hence in the situation of Lemma 5, the ciphertexts c_i are the* unique *ciphertexts which are compatible with c_0^* (resp. c_e^*). In other words, all valid ciphertext for (connected) keys are in bijection via* ReEnc*.*

Note that the previous lemma was concerned with ciphertexts of the form $\mathsf{Enc}(k, m, r)$. But there may be more valid, i.e. decryptable ciphertexts. Tidiness ensures that all valid ciphertexts are indeed of the form $\mathsf{Enc}(k, m, r)$.

B.1 UP-INT-CTXT

We begin with the simpler proof, the one for ciphertext integrity.

Theorem 6. *Let $\mathsf{UE} = (\mathsf{Gen}, \mathsf{GenKey}, \mathsf{GenTok}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{ReEnc})$ be an updatable encryption scheme with deterministic reencryption. Let us assume that* UE *has* randomness-preserving *reencryption and* simulatable *token generation and the underlying encryption scheme $\mathsf{SKE} := (\mathsf{Gen}, \mathsf{GenKey}, \mathsf{Enc}, \mathsf{Dec})$ is* tidy*. Then* UE *is UP-INT-CTXT-secure if* SKE *is S-INT-CTXT-secure.*

Proof. Let $\mathcal{A}$ be an adversary against UE in the UP-INT-CTXT game which makes at most $e_{\max}$ calls to the Next oracle. Starting from Game 1, which equals UP-INT-CTXT, we gradually introduce Game 3 in which $\mathcal{A}$'s advantage is reduced by a polynomial factor. Let $\mathsf{Adv}^{\text{Game}_i}_{\mathsf{UE},\mathcal{A}}(\kappa) := \Pr[\mathsf{Exp}^{\text{Game}_i}_{\mathsf{UE},\mathcal{A}}(\text{sec}) = 1]$ denote the advantage of $\mathcal{A}$ in Game i. Then, we have

$$\mathsf{Adv}^{\text{Game}_3}_{\mathsf{UE},\mathcal{A}}(\kappa) \geq \frac{1}{(e_{\max}+1)^2} \mathsf{Adv}^{\text{Game}_1}_{\mathsf{UE},\mathcal{A}}(\kappa) \tag{3}$$

Finally, we show that an adversary $\mathcal{A}$ in Game 3 can be turned directly into an S-INT-CTXT adversary $\mathcal{B}$ against the underlying encryption scheme SKE such that

$$\mathsf{Adv}^{\mathrm{Game}_3}_{\mathsf{UE},\mathcal{A}}(\kappa) = \mathsf{Adv}^{\mathsf{s\text{-}int\text{-}ctxt}}_{\mathsf{SKE},\mathcal{B}}(\kappa) \quad (4)$$

Game 1: This is the standard UP-INT-CTXT game for UE (with reencrytion for queried ciphertexts only).

Game 2: This game equals Game 1, except that we try to guess the region of epochs $\{\ell, \ldots, r\}$ for which the following properties are satisfied:

– the challenge epoch e^* in which $\mathcal{A}$ outputs its forgery c^* is contained in $\{\ell, \ldots, r\}$.
– $\mathcal{A}$ does not corrupt any of the keys $k_\ell, \ldots, k_r$.
– $\mathcal{A}$ corrupts the tokens $\Delta_{\ell+1}, \ldots, \Delta_r$.
– $\mathcal{A}$ corrupts neither Δ_ℓ nor Δ_{r+1}.

Note that such a region exists in any execution of Game 1 in which $\mathcal{A}$ does not (trivially) lose due to trivial win conditions being true. We guess this region by simply drawing $\ell \leftarrow_{\mathrm{R}} \{0, \ldots, e_{\max}\}$ and $r \leftarrow_{\mathrm{R}} \{\ell, \ldots, e_{\max}\}$ uniformly at random. Thus, this guess is correct with probability at least $\frac{1}{(e_{\max}+1)^2}$. (If the guess turns out to be wrong, we abort.)

Game 3: This game implements the **key insulation** technique described in Sec. 3.2 (Thm. 1 and Figs. 7 and 8) and is perfectly indistinguishable from Game 2 assuming UE has *randomness-preserving* reencryption and *simulatable* token generation and SKE is tidy. In summary, in this game we apply the following modifications:

– The challenger does
  - not generate keys $k_{\ell+1}, \ldots, k_r$,
  - not generate tokens Δ_ℓ and Δ_{r+1},
  - generate tokens $\Delta_{\ell+1}, \ldots, \Delta_r$ along with reverse tokens $\Delta'_{\ell+1}, \ldots, \Delta'_r$ using SimTok.
– An $\mathsf{Enc}(m)$ call in epoch
  - $\ell < e_{\mathrm{cur}} \leq r$ is handled by first computing $c_\ell \leftarrow_{\mathrm{R}} \mathsf{UE.Enc}(k_\ell, m)$ and then $c_{e_{\mathrm{cur}}}$ via $c_j \leftarrow \mathsf{UE.ReEnc}(\Delta_j, c_{j-1})$ for $\ell + 1 \leq j \leq e_{\mathrm{cur}}$.
  - $0 \leq e_{\mathrm{cur}} < \ell$ is handled by first computing $(c_\ell, r) \leftarrow_{\mathrm{R}} \mathsf{UE.Enc}(k_\ell, m)$ and then $c_{e_{\mathrm{cur}}} \leftarrow \mathsf{UE.Enc}(k_{e_{\mathrm{cur}}}, m; r)$.
– A $\mathsf{Dec}(c)$ call in epoch $\ell < e_{\mathrm{cur}} \leq r$ is handled by first computing c_ℓ via $c_{j-1} \leftarrow \mathsf{UE.ReEnc}(\Delta'_j, c_j)$ for $e_{\mathrm{cur}} \geq j > \ell$ and then $m \leftarrow \mathsf{UE.Dec}(k_\ell, c_\ell)$.
– A $\mathsf{ReEnc}(c_i, i)$ call in epoch $e_{\mathrm{cur}} > i$ is handled by computing a series of ciphertexts c_j for $i < j \leq e_{\mathrm{cur}}$, where
  - $c_j \leftarrow \mathsf{UE.ReEnc}(\Delta_j, c_{j-1})$ for $j \neq \ell, r+1$ (as usual).
  - c_ℓ is computed by looking up the message-randomness pair (m, r) corresponding to c_i and then set to the ciphertext for (m, r) under key k_ℓ which has been automatically created when calling $\mathsf{Enc}(m)$ in some epoch $i' \leq i \leq \ell$ (cf. modification to Enc). Since ReEnc only accepts *queried* ciphertexts, the lookup of (m, r) succeeds.

- c_{r+1} is computed by looking up the message-randomness pair (m, r) corresponding to c_i (possible since queried before) and then computing $c_{e_{\text{cur}}} \leftarrow \mathsf{UE.Enc}(k_{r+1}, m; r)$.

To convince ourselves that all the changes described above cannot be noticed by $\mathcal{A}$, we may consider the following sequence of games.

Game 3.1: In contrast to Game 2, we simply generate all keys $k_0, \ldots, k_{e_{\max}}$ using $\mathsf{UE.GenKey}(sp)$ and all tokens $\Delta_1, \ldots, \Delta_{e_{\max}}$, where $\Delta_{i+1} \leftarrow_{\text{R}} \mathsf{UE.GenTok}(k_i, k_{i+1})$, at the beginning of the game. Clearly, this is only a conceptual change.

Game 3.2: In this game we get rid of the keys in region $\{\ell, \ldots, r\}$ while still ensuring that encryption, decryption, and reencryption in the region works perfectly fine.

More precisely, we generate $k_0, \ldots, k_\ell$ and $k_{r+1}, \ldots, k_{e_{\max}}$ as in the previous game, but we omit generating $k_{\ell+1}, \ldots, k_r$. Consequently, we also need to generate Δ_i for $\ell + 1 \leq i \leq r$ differently, as we are missing the corresponding keys. To this end, we make use of UE's simulatable token generation property which allows to sample tokens without knowing the corresponding keys. So we run $(\Delta_i, \Delta'_i) \leftarrow_{\text{R}} \mathsf{SimTok}(sp)$ for $\ell + 1 \leq i \leq r$. Note that this property also ensures that Δ_i in this and the previous game are identically distributed. So a ciphertext updated with one of these tokens will look the same in both games. (We do not need this property for the reverse tokens Δ'_i as ciphertexts resulting from applying a reverse token will not be handed to the adversary.)

As another consequence of not knowing the keys $k_{\ell+1}, \ldots, k_r$ anymore, we need to redefine how encryption and decryption under these key should work now. For this, we simply do encryptions (resp. decryptions) only under key k_ℓ and update to (resp. from) epoch $e_{\text{cur}} \in \{\ell + 1, \ldots, r\}$ using the corresponding (reverse) tokens. More precisely, we do the following:

– Upon an $\mathsf{Enc}(m)$ call in epoch $\ell + 1 \leq e_{\text{cur}} \leq r$, we first compute $c_\ell \leftarrow_{\text{R}} \mathsf{UE.Enc}(k_\ell, m; r)$ (for random r). Then compute $c_i \leftarrow \mathsf{ReEnc}(\Delta_i, c_{i-1})$ for $\ell + 1 \leq i \leq e_{\text{cur}}$ and return $c_{e_{\text{cur}}}$.
– Upon an $\mathsf{Dec}(c_{e_{\text{cur}}})$ call in epoch $\ell + 1 \leq e_{\text{cur}} \leq r$, we first compute $c_{i-1} \leftarrow \mathsf{ReEnc}(\Delta'_i, c_i)$ for $e_{\text{cur}} \geq i \geq \ell + 1$. Then we call $\mathsf{UE.Dec}(k_\ell, c_\ell)$ and return whatever Dec returns.

Due to *randomness-preserving* reencryption, Enc is identical to Enc in the previous game.[11] Since the reverse tokens generated by SimTok guarantee that the messages contained in the ciphertexts they are applied to stay the same, Dec is identical to Enc in the previous game. Note that encryption and decryption for epochs in $\{\ell, \ldots, r\}$ are now done in a "centralised" way using k_ℓ. As also the challenge key only exists implicitly now, we need to redefine when the ciphertext that $\mathcal{A}$ outputs decrypts correctly under this key. Consequently, we say that it decrypts correctly when the updated ciphertext in epoch ℓ decrypts correctly. By Cor. 2 this is equivalent to the game before.

[11] Note again that choosing encryption randomness indepent from keys is an integral part of the definition of randomness-preserving reencryption.

Game 3.3: In this game, we only add some bookkeeping done by the challenger. We maintain a list $\mathbf{L}$ with all trivial knowledge about ciphertexts the challenger generates.

- For every execution of $c \leftarrow_{\mathrm{R}} \mathsf{UE.Enc}(k_{e_{\mathrm{cur}}}, m; r)$ (as part of an Enc or ReEnc call), we add $((c, e_{\mathrm{cur}}), m, r)$ to $\mathbf{L}$.
- For every execution of $c' \leftarrow \mathsf{UE.ReEnc}(\Delta_i, c)$ (as part of an Enc or ReEnc call), we look up $((c', i-1), m, r) \in \mathbf{L}$ and add $((c', i), m, r)$ to $\mathbf{L}$. Note that since reencryption is randomness-preserving, it holds that $c' = \mathsf{UE.Enc}(k_i, m; r)$.

Game 3.4: In this game, we change the way how encryption calls for epochs "left" to our region $\{\ell, \ldots, r\}$ are handled (in order to realize reencryptions into this region when the "entry" token Δ_ℓ to the region is missing later on). When receiving an $\mathsf{Enc}(m)$ call in epoch $e_{\mathrm{cur}} < \ell$, we run $c' \leftarrow_{\mathrm{R}} \mathsf{UE.Enc}(k_\ell, m)$ also yielding encryption randomness r. Then we use r to encrypt m under key $k_{e_{\mathrm{cur}}}$, i.e. we run $c \leftarrow \mathsf{UE.Enc}(k_{e_{\mathrm{cur}}}, m; r)$ and return the resulting ciphertext c to the adversary. Note that this also results in two entries being added to list $\mathbf{L}$. Due to randomness-preserving reencryption, Game 3.4 and Game 3.3 are perfectly indistinguishable.

Game 3.5: In this game, we get rid of the token Δ_ℓ and, consequently, need to change the way ReEnc calls are handled. In the previous game, a $\mathsf{ReEnc}(c, i)$ call in epoch e_{cur} (for $e > i$) was handled by a series of $c_j \leftarrow \mathsf{UE.ReEnc}(\Delta_j, c_{j-1})$ computations for $i \le j \le e_{\mathrm{cur}}$. Since Δ_ℓ is missing now, we need to compute c_ℓ differently: We simply lookup the entry $((c_{\ell-1}, \ell - 1), m, r)$ in $\mathbf{L}$. Since we have a bijection between ciphertexts and message-randomness pairs, there is only one entry containing $c_{\ell-1}$. Then we look up the entry containing ℓ, m, r yielding some ciphertext c'. This entry exists due to the change we made in the previous game. We set $c_\ell := c'$ and go on as usual in a reencryption call. Perfect indistinguishability follows from randomness-preserving reencryption.

Game 3.6: In this game, we additionally get rid of the token Δ_{r+1}. To this end, we again slightly change the way ReEnc calls are handled. Similar to the previous game, we only need to replace the $\mathsf{UE.ReEnc}$ computation involving Δ_{r+1}: We simply lookup the entry $((c_r, r), m, r)$ in $\mathbf{L}$ and set $c_{r+1} \leftarrow \mathsf{UE.Enc}(k_{r+1}, m; r)$. Perfect indistinguishability follows from randomness-preserving reencryption.

INT-CTXT Adversary $\mathcal{B}$: $\mathcal{B}$ embeds the (secret) challenge key k^* of SKE as key k_ℓ in Game 3 he simulates for adversary $\mathcal{A}$. Note that in this game, due to the key-insulation technique, k_ℓ is only needed in the scope of decryption and encryption operations (and not for generating tokens). Moreover, when encrypting with this key, also the used encryption randomness is required. However, all this can be realized using the decryption and encryption oracle of the S-INT-CTXT game which $\mathcal{B}$ is playing. Thus, Game 3 can be perfectly simulated by $\mathcal{B}$. The forgery $\mathcal{A}$ finally outputs will be a ciphertext $c^*_{e_{\mathrm{cur}}}$ under a (unknown) key $k_{e_{\mathrm{cur}}}$ in the epoch region $\{\ell, \ldots, r\}$ (assuming our guess was correct). This ciphertext is then updated by $\mathcal{B}$ to a ciphertext c^*_ℓ under key k_ℓ using the corresponding reverse tokens. Assuming $(e_{\mathrm{cur}}, c^*_{e_{\mathrm{cur}}}) \notin \mathbf{Q}$ then we also have that $(\ell, c^*_\ell) \notin \mathbf{Q}$ because re-encryption (of valid ciphertext) us a bijection (due to tidiness and randomness-preserving re-encryption, c.f. Cor. 2). If $\mathcal{A}$'s forgery is valid then also $\mathcal{B}$'s forgery, as the former already requires that $c^*_{e_{\text{cur}}}$ updated to k_ℓ decrypts correctly. Since message-randomness pairs and ciphertexts are in bijection, the forgery remains a fresh ciphertext after rotation to k_ℓ, see Cor. 2.

B.2 UP-IND-CCA Security.

Using the key insulation technique repeatedly, we prove the following result for UP-IND-CCA security.

Theorem 7. *Let* UE *be an updatable encryption scheme with deterministic re-encryption. Suppose that* UE *has* randomness-preserving *re-encryption and* simulatable *token generation and the underlying encryption scheme* SKE *is tidy. Then* UE *is UP-IND-CCA-secure if* SKE *is S-IND-CCA-secure.*

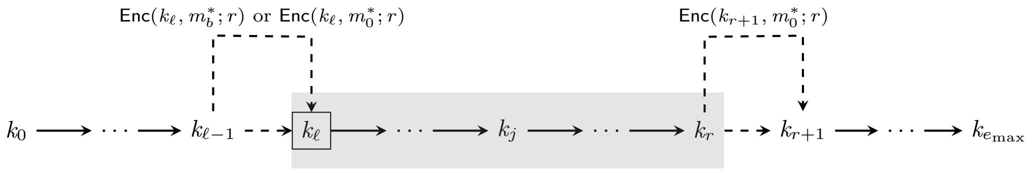
$\mathsf{Enc}(k_\ell, m^*_b; r)$ or $\mathsf{Enc}(k_\ell, m^*_0; r)$
$\mathsf{Enc}(k_{r+1}, m^*_0; r)$
k_0
$k_{\ell-1}$
k_ℓ
k_j
k_r
k_{r+1}
$k_{e_{\max}}$

Fig. 9. A sketch of the indistinguishability argument for hybrid games $\mathsf{H}_{\ell-1}$ and H_ℓ: The S-IND-CCA challenge is embedded in epoch ℓ. Reencryptions of ciphertexts in epoch r with $\mathsf{isChallenge} = 1$ are replaced by encryptions of m^*_0. If a challenge reencryption in epoch $\ell - 1$ is replaced by an encryption of m^*_0 then we are in Game $\mathsf{H}_{\ell-1}$. Otherwise, the game is identical to Game H_ℓ.

The techniques behind the proof are quite similar to the UP-INT-CTXT proof. The proof proceeds in two steps. In the first step, all reencryptions of the challenge ciphertext are replaced by encryptions of the fixed challenge message m^*_0 (instead of m^*_b). This is done using a sequence of hybrid games H_ℓ for $\ell = e_{\max}, \dots, 0$, where in H_ℓ we essentially want to replace challenge reencryptions to epoch $j > \ell$. However, due to *deterministic* re-encryption, doing this in a straightforward way does not work: Assume the adversary corrupted the token Δ_ℓ, then it can trivially spot the inconsistency between the games $\mathsf{H}_{\ell-1}$ and H_ℓ by comparing its own re-encryption result $\mathsf{ReEnc}(\Delta_\ell, c_{\ell-1})$ with the output of the re-encryption oracle call $\mathsf{ReEnc}(c_{\ell-1}, \ell - 1)$ in epoch ℓ. Therefore, in H_ℓ we guess the first token Δ_{r+1} where $r + 1 > \ell$, which $\mathcal{A}$ does not corrupt, and replace challenge reencryptions from epoch r to $r + 1$ with encryptions of m^*_0. Note that this modification is sufficient to replace challenge reencryptions from an arbitrary epoch $i \leq r$ to an arbitrary epoch $j > r$ as we can use all other tokens Δ_t, $t \neq r + 1$, as usual for this purpose. All indistinguishability reductions make use of key insulation and a direct embedding of an S-IND-CCA challenge. Fig. 9 illustrates the indistinguishability argument for hybrids $\mathsf{H}_{\ell-1}$ and H_ℓ. In the actual proof, we have to take care of some special cases, e.g., the hybrids $\mathsf{H}_{\ell-1}$ and H_{ℓ} can be equal sometimes.

In a second step, the challenge encryption is finally replaced by an encryption of m_0^*. Thus no information about the challenge bit b remains. This can again be done using our key insulation technique.

In more detail, the full proof is as follows:

Proof. We proceed in Games. We assume that adversary $\mathcal{A}$ against UP-IND-CCA requests at most $e_{\max}$ keys (via Next).

Game 0 is the standard UP-IND-CCA game for deterministic re-encryption with the queried restriction for ciphertexts. Let $b_{\mathcal{B}} \in \{0,1\}$ be the challenge bit.

Game 1. Let $\mathcal{B}$ be the challenger. $\mathcal{B}$ replaces almost all reencryptions of challenge ciphertexts, by (re-)encryptions of m_0^* (with consistent randomness). More precisely, $\mathcal{B}$ picks $r \leftarrow_{\mathrm{R}} \{0, \dots, e_{\max}\}$ and hopes that r has the property:

– $\mathcal{A}$ corrupts all tokens Δ_j for $j = e^* + 1, \dots r$.
– $\mathcal{A}$ does not corrupt Δ_{r+1} (or it does not exist, in case of $\Delta_{e_{\max}+1}$).

where e^* is the challenge epoch and $c_{e^*}^* \leftarrow \mathsf{Enc}(k_{e^*}, m_{b_{\mathcal{B}}}^*; r^*)$ the challenge ciphertext. For reencryptions, $\mathcal{B}$ acts honestly, i.e. uses $c_j^* \leftarrow \mathsf{ReEnc}(\Delta_j, c_{j-1}^*)$ except for $j = r + 1$. In that case, $\mathcal{B}$ sets $c_{r+1}^* = \mathsf{Enc}(k_{r+1}, m_0^*; r^*)$. Since Δ_{r+1} is not corrupted, we can redefine $\mathsf{isChallenge}$ in a "consistently inconsistent" manner too. Concretely, we split $\mathsf{isChallenge}$ in two parts. For epochs $0, \dots, r$, $\mathsf{isChallenge}$ is computed with $c_{e^*}^*$ (and $\mathbf{K}_{\leq r}$, $\mathbf{T}_{\leq r}$). For epochs $r + 1, \dots, e_{\max}$, $\mathsf{isChallenge}$ is computed with (the inconsistent) c_{r+1}^* (and $\mathbf{K}_{\geq r+1}$, $\mathbf{T}_{\geq r+1}$).

Indistinguishability of Game 0 and Game 1 is seen via a series of hybrid games H_{ℓ} defined as follows (where $\mathcal{B}$ plays challenger): $\mathcal{B}$ first picks $r \leftarrow_{\mathrm{R}} \{\ell, \dots, e_{\max}\}$ and hopes that r has the property that $\mathcal{A}$ corrupts all tokens Δ_j for $j = i + 1, \dots, e_{\max}$, but does not corrupt Δ_{r+1} (or it does not exist). If $\mathcal{B}$'s guess turns out wrong, it aborts. Furthermore, $\mathcal{B}$ changes how (internally) reencryption steps are handled:

– If $j \neq r$ or $\mathsf{isChallenge}(c, j) = 0$ nothing changes, i.e. $\mathsf{ReEnc}(\Delta_j, c)$ is used.
– For $j = r$ and $\mathsf{isChallenge}(c_r, r) = 1$, we reencrypt via $(m^*, r^*) \leftarrow \mathsf{RDec}(k_r, c)$, and $c_{r+1}^* = \mathsf{Enc}(k_{r+1}, m; r^*)$. More precisely, we use bookkeeping instead of RDec, as in Thm. 6, Game 3, and look (m^*, r^*) up in $\mathbf{L}$.

In other words, reencryptions of challenge ciphertexts from r to $r + 1$ are always replaced by encryptions of m_0^* (with consistent randomness). Note again that $\mathsf{isChallenge}$ is adapted accordingly.

Note that Game $\mathsf{H}_{e_{\max}+1}$ is Game 0. Game H_{-1} (already H_{e^*}) is Game 1. The indistinguishability of $\mathsf{H}_{\ell-1}$ and H_{ℓ} uses key-insulation to construct an S-IND-CCA adversary from $\mathcal{A}$. It is detailed in Lemma 6.

Game 2 replaces the challenge encryption by an encryption of m_0^*. Indistinguishability from Game 1 is again shown via a reduction to an S-IND-CCA adversary using key-insulation, very similar to the hybrids. Thus, prove security by reduction to S-IND-CCA with a loss of $\frac{1}{(e_{\max}+1)^2}$. One factor of $\frac{1}{e_{\max}+1}$ is incurred by carrying out the hybrid argument, and another one by the hybrid indistinguishability itsself.

Lemma 6. *Games $\mathsf{H}_{\ell-1}$ and H_{ℓ} in Thm. 7 are indistinguishable. Concretely, if $\mathcal{A}$ has advantage ε for breaking UP-IND-CCA, we obtain an adversary $\mathcal{B}$ with advantage $\frac{\varepsilon}{e_{\max}+1}$ against S-IND-CCA.*

Proof. The proof is a straightforward application of key-insulation (Game 3 in Thm. 6). Suppose $\mathcal{A}$ distinguishes hybrid games Game $\mathsf{H}_{\ell-1}$ and Game H_{ℓ} (for some $0 \leq \ell \leq e_{\max}+1$). We construct an adversary $\mathcal{B}$ against S-IND-CCA as follows. $\mathcal{B}$ guesses $r \leftarrow_{\mathrm{R}} \{\ell, \dots, e_{\max}\}$ with the property that $\mathcal{A}$ corrupts tokens $\Delta_{\ell+1}, \dots, \Delta_{r}$, but does *not* corrupt Δ_{r+1}. If the guess is wrong, $\mathcal{B}$ aborts. In the following, we assume correct guesses. We distinguish several cases.

Case 1. $\mathcal{A}$ corrupts Δ_{ℓ}. Then Game $\mathsf{H}_{\ell-1}$ and Game H_{ℓ} are identical.
Case 2. $\mathcal{A}$ corrupts a key in the range $\{\ell, \dots, r\}$. Then $\mathcal{A}$ must not request a reencryption of a challenge ciphertext into epochs $\{\ell, \dots, r\}$. If it does, the game is aborted (and $\mathcal{A}$ loses). Again, Game $\mathsf{H}_{\ell-1}$ and H_{ℓ} behave identical.
Case 3. $\mathcal{A}$ corrupts neither Δ_{ℓ} nor any key in the range $\{\ell, \dots, r\}$.

Cases 1 and 2 are trivial in the sense that $\mathcal{A}$ cannot win, so $\mathcal{B}$ just returns $b_{\mathcal{B}} \leftarrow_{\mathrm{R}} \{0,1\}$. Thus we concentrate on case 3. In case 3, Game $\mathsf{H}_{\ell-1}$ replaces reencryptions of challenge ciphertexts by encryptions of m_0 for any epoch $e \geq \ell$. Thus, the only difference w.r.t. Game H_{ℓ} are reencryptions of challenge ciphertexts into the region $\{\ell, \dots, r\}$, which are honest in Game H_{ℓ}, but not in Game $\mathsf{H}_{\ell-1}$.

We use key-insulation on the region $\{\ell, \dots, r\}$ and modify $\mathcal{B}$ accordingly. Then we embed the strong IND-CCA challenger $\mathcal{C}$ into epoch ℓ in the insulated region. Note that by key-insulation, as in Thm. 6, we run all encryption queries for $e_{\mathrm{cur}} < \ell$ through $\mathcal{C}$ to obtain suitable encryption randomness. Suppose $\mathcal{A}$ sends (m_0^*, m_1^*) as challenge messages. Then $\mathcal{B}$ passes $(m_{b_{\mathcal{B}}}^*, m_0^*)$ as his challenge message to strong IND-CCA challenger $\mathcal{C}$. Let (c_{ℓ}^*, r^*) be $\mathcal{C}$'s answer. The challenge ciphertext for $\mathcal{A}$ is computed as $c^* \leftarrow \mathsf{Enc}(k_{e_{\mathrm{cur}}}, m_{b_{\mathcal{B}}}^*; r^*)$. In the rest of the game, $\mathcal{B}$ proceeds as before except for the following:

- Treat c_{ℓ}^* as the challenge reencryption in epoch ℓ. That is, reencryption from epoch $\ell-1$ to ℓ for c with $\mathsf{isChallenge}(c, \ell-1) = 1$ outputs c_{ℓ}^*. This embeds the challenge from $\mathcal{C}$.
- For challenge reencryption from r to $r+1$, $c_{r+1}^* \leftarrow \mathsf{Enc}(k_{r+1}, m_0^*; r^*)$ is used.
- Split $\mathsf{isChallenge}(c, j)$ into three parts for $j < i$, $i \leq j \leq r$, $r < j$.

Note that due to *deterministic randomness-preserving* reencryption, this all fits together. Close inspection shows the following: If $b_{\mathcal{C}} = 0$, i.e. $\mathcal{C}$ encrypts $m_{b_{\mathcal{B}}}^*$, then the game is identical to Game H_{ℓ}. If $b_{\mathcal{C}} = 1$, i.e. $\mathcal{C}$ encrypts m_0^*, then the game is identical to Game $\mathsf{H}_{\ell-1}$. Consequently, $\mathcal{B}$ has advantage $\frac{1}{e_{\max}+1}\varepsilon$ against S-IND-CCA.

C Generalising Simulatable Token Generation

Here we sketch how token simulation for a large class of schemes can be defined. We consider *updatable keyed schemes* UKS, i.e. schemes which have a secret key k. Moreover, we consider (key-)leakage $\mathsf{leak}\colon \mathcal{K} \to \mathcal{L}$ for UKS. Lastly, for a definition of simulatable token generation in our setting, we need the existence of reverse tokens. First, we give the definition of *reverse* tokens in general.

Definition 19. *Let* UKS *be an updatable keyed scheme. We call a token* Δ' *a* ***reverse token*** *of a token* Δ *if for any pair of keys* $k^{\mathrm{old}}, k^{\mathrm{new}} \in \mathcal{K}$ *with* $\Delta \in \mathrm{supp}(\mathsf{GenTok}(k^{\mathrm{old}}, k^{\mathrm{new}}))$ *we have* $\Delta' \in \mathrm{supp}(\mathsf{GenTok}(k^{\mathrm{new}}, k^{\mathrm{old}}))$.

Caution. Def. 19 for reverse token needs perfect correctness of UKS to be sensible. It can be relaxed in the obvious way, by using "correctness of re-operation" for Δ' instead of $\Delta' \in \mathrm{supp}(\mathsf{GenTok}(k^{\mathrm{new}}, k^{\mathrm{old}}))$.

With this, we can state token simulation in general. We add *(key-)leakage* since it is necessary to treat schemes such as RISE, NYUE and NYUAE, which "leak" information related to the secret key. Namely, Δ explicitly contains $\mathsf{leak}(k^{\mathrm{new}}) = pk^{\mathrm{new}}$ as information. Token simulation must ensure that this "key-leakage" is correctly simulated. This is impossible without knowledge of $\mathsf{leak}(k^{\mathrm{old}})$.

Definition 20. *Let* UKS *be some updatable keyed scheme, i.e. a cryptographic scheme which provides* GenKey *and* GenTok *functionality. For example updatable encryption* UE*, updatable PRF* PRF *or updatable signatures* USIG*. We say that* UKS *has* ***simulatable*** *token generation under (key-)leakage* $\mathsf{leak}\colon \mathcal{K} \to \mathcal{L}$ *if it has following property:*
There is an algorithm $\mathsf{SimTok}(sp, \mathsf{leak}(k^{\mathrm{old}}))$ *which samples a triple* (Δ, Δ', lk) *of token, reverse token and key leak. Furthermore, for arbitrary (fixed)* $k^{\mathrm{old}} \leftarrow \mathsf{GenKey}(sp)$ *following distributions of* Δ *are identical: The distribution of* $(\Delta, lk^{\mathrm{new}})$

- *induced by* $(\Delta, \_, lk^{\mathrm{new}}) \leftarrow_R \mathsf{SimTok}(sp, \mathsf{leak}(k^{\mathrm{old}}))$.
- *induced by* $\Delta \leftarrow_R \mathsf{GenTok}(k^{\mathrm{old}}, k^{\mathrm{new}})$ *and* $lk^{\mathrm{new}} = \mathsf{leak}(k^{\mathrm{new}})$*, where* $k^{\mathrm{new}} \leftarrow_R \mathsf{GenKey}(sp)$.

In other words, honest token generation and token simulation are perfectly indistinguishable, and key leakage can also be simulated.

D Groth–Sahai proofs: Brief overview

In this section, we give a brief overview of Groth–Sahai proofs and OR-compilation. While our constructions do not necessarily rely on Groth–Sahai proofs, they need randomisability and (linear) malleability of proofs. And Groth–Sahai proofs are a prime example of such a proof system. Our presentation of Groth–Sahai proofs (and their linear malleability), follows the presentation in [Hof16]. It is informal and meant to highlight key ideas and techniques. For formal definitions and justifications, we refer to [GS12; EG14; Bel+09; Fuc11; Cha+12] in the relevant places.

Let $(e, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ be a pairing group. Groth–Sahai proofs are typed *commit-and-prove* systems [EG14] for quadratic equations. That is, one commits to a witness, i.e. an assignment of variables, and then proves that the committed witness is solution for the given (system of) quadratic equations.

Definition 21 (GS proofs[GS12; EG14]). *The Groth–Sahai proof system consists of the following algorithms:*

Common reference string: *$crs \leftarrow_R$ $\mathsf{GS.SetupH}(\mathrm{gp})$ (resp. $crs \leftarrow_R$ $\mathsf{GS.SetupB}(\mathrm{gp})$) generates a suitable* hiding *(resp.* binding*) CRS. The CRS crs is an implicit input for all following algorithms.*

Commitments. *$c \leftarrow \mathsf{Com}_{\mathtt{t}}(x; r)$ commits to a variable of type $\mathtt{t}$. The type $\mathtt{t}$ decides for example, if x is in $\mathbb{G}_1$ or $\mathbb{G}_2$ (or a scalar in $\mathbb{F}_p$), c.f. [EG14]. If crs is hiding (resp. binding) the commitments are perfectly hiding (resp. perfectly binding).*

Proofs: *Let $\mathcal{E}$ be an equation which may involve variables and constants over $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{F}_p$. Let $(w_i)_i$ be a solution to $\mathcal{E}$, i.e. a suitable variable assignment with variable w_i of type $\mathtt{t}_i$. Let $(r_i)_i$ be commitment randomness for w_i, and let $(C_i)_i = (\mathsf{Com}_{\mathtt{t}_i}(w_i; r_i))_i$ be the set of commitments. Then $\pi \leftarrow_R$ $\mathsf{Prove}(\mathcal{E}, (w_i)_i, (r_i)_i)$ produces a* proof*.*

Verification: *Given an equation $\mathcal{E}$, commitments $(C_i)_i$, and a proof π, $\mathsf{Verify}(\mathcal{E}, (C_i)_i, \pi)$ either accepts or rejects.*

Systems of equations: *To prove conjunctions, i.e. systems of equations $(\mathcal{E}_j)_j$, it suffices to prove/verify each $\mathcal{E}_j$ for the* same committed solution *$(C_i)_i$.*

Note that π is usually called "the proof" and the commitments are implicit.

Groth–Sahai proofs prove satisfiability of sets of (linear and) quadratic (pairing) equations, e.g. equations of the form[12]

$$\sum_{i,j} \Gamma_{i,j} e([x_i]_1, [y_j]_2) + \sum_i e([x_i]_1, [b_i]_2) + \sum_j e([a_j]_1, [y_j]_2) = [t]_T, \qquad (5)$$

where $\Gamma_{i,j} \in \mathbb{F}_p$, $[x_i]_1, [a_j]_1 \in \mathbb{G}_1$, $[b_i]_2, [y_j]_2 \in \mathbb{G}_2$ and $[t]_T \in \mathbb{G}_T$. The statement (i.e. the equation) is given by $\Gamma_{i,j}$, $[a_j]_1$ and $[b_i]_2$. The witness (i.e. solution) is given by $[x_i]_1$, $[y_j]_2$. More generally, the pairing e in Eq. (5) can be replaced by other bilinear maps between $\mathbb{F}_p$, $\mathbb{G}_1$, $\mathbb{G}_2$, e.g. scalar multiplication in $\mathbb{G}_1$, that is $\mathbb{G}_1 \times \mathbb{F}_p \to \mathbb{G}_1$, $([x]_1, y) \mapsto [x]_1 y = [xy]_1$. This yields quadratic equations of different types. For more details, see App. F and [GS12; EG14].

Theorem 8 (Properties of GS proofs[GS12]). *The Groth–Sahai proof system has following properties.*

Perfect completeness: *For any crs, any solution $(w_i)_i$ of equation $\mathcal{E}$, and any commitment randomness $(r_i)_i$,* Prove *yields an accepting proof.*

[12] Considering only the dlog's in Eq. (5), we get $\sum_{i,j} \Gamma_{i,j} x_i y_j + \sum_i x_i b_i + \sum_j a_j, y_j = t$, which is evidently a general quadratic equation. Hence the name.

Perfect soundness. *Given a* binding *CRS, GS proofs are perfectly sound. (I.e. if the contents of $(c_i)_i$ do not satisfy $\mathcal{E}$, there is no accepting proof.)*

Extraction. *A* binding *CRS can be generated together with trapdoor information* td*, so that commitments to group elements are perfectly extractable.*

Perfect simulation. *A* hiding *CRS crs can be generated together with trapdoor information* td*, which allows to* perfectly *simulate certain equations, c.f. App. F and [EG14] for details. Moreover, $((\mathsf{Com}(w_i, r_i))_i, \mathsf{Prove}(\mathcal{E}, (w_i)_i, (r_i)_i))$ and $((\mathsf{Com}(0, r_i))_i, \mathsf{SimProve}(\mathrm{td}, \mathcal{E}, 0, (r_i)_i))$ are identically distributed.*

Homomorphic rerandomisable commitments. *The commitment schemes are (additively) homomorphic, i.e. $\mathsf{Com}(a; r) + \mathsf{Com}(b; s) = \mathsf{Com}(a + b; r + s)$. Moreover, they are perfectly rerandomisable.*

Rerandomisation. *Groth–Sahai proofs are perfectly rerandomisable. More concretely, given commitments $(C_i)_i$ and proofs π_j, it is possible to perfectly rerandomise commitments and proofs, i.e. the rerandomised commitments and proofs are distributed exactly as a new proof (with the same witness).*

Linear malleability. *Given proofs for a system of equations, one can compute proofs for linear combinations of these equations. More concretely: Adding two equations together as well as multiplying an equation by a scalar and computing an adapted proof is always possible. Combining this with the homomorphism of commitments, one can manipulate statements, witnesses and proofs to obtain proofs for (linearly) related statements (even without knowing a witness).*

The extractability of GS proofs is so useful and wide-spread, that it spawned the property of so-called *structure-preserving* cryptographic schemes. Rerandomisability of GS proofs is another widely used feature, typically applied to obtain unlinkability, e.g. in electronic voting. Rerandomisable proofs were formalised in [Bel+09]. Linear malleability of GS proofs is also well-known. We refer to [Fuc11, Section 5] especially [Fuc11, Lemma 3] for a quick reference, and [Cha+12, Appendix A] for a detailed, but technical, reference.

D.1 OR-compilation for quadratic equations

We first describe the folklore OR-compilation (e.g. [Gro06]) without reference to GS proofs. The OR-compilation technique for sets of equations $\mathcal{E}_0$, $\mathcal{E}_1$ is simple: Enumerate all *constants and variables*, call them $x_1, \dots, x_n$. Let $x_1, \dots, x_k$ be the constants. Add a bit b to the variables, say x_{n+1}. Duplicate all variables x_i as $x_{i,0}$ and $x_{i,1}$ and define the set of equations $\mathcal{E}'_{\mathsf{OR}} = \{bx_{i,1} = bx_i\} \cup \{(1-b)x_{i,0} = (1-b)x_i\} \cup \{b(1-b) = 0\}$. Let $\mathcal{E}'_b$ be $\mathcal{E}_b$, but with x_i replaced by $x_{i,b}$. Define the set of equations $\mathsf{OR}(\mathcal{E}_0, \mathcal{E}_1)$ as the union $\mathcal{E}'_0 \cup \mathcal{E}'_1 \cup \mathcal{E}'_{\mathsf{OR}}$.

Given a solution $(x_i)_i$ to $\mathcal{E}_b$, one does the following: (Set $x_{n+1} = b$.) Set $x_{i,b} = x_i$ and $x_{i,1-b} = 0$. This is a witness for $\mathsf{OR}(\mathcal{E}_1, \mathcal{E}_2)$,

The idea is simple: Depending on b, we copy the statement (i.e. constants) and witness x_i for $\mathcal{E}_b$ into $x_{i,b}$, and set the unsatisfied statement (and witness) to 0 (hence the equalities trivially hold). It is easy to reconstruct a witness $(x_i)_i$ from a witness for $\mathsf{OR}(\mathcal{E}_0, \mathcal{E}_1)$.

Application to Groth–Sahai proofs. We use the typed commit-and-prove point of view from [EG14]. Our constant/variable types and equation types are as in [EG14] with the notational difference that we use $\mathbb{G}_1$ and $\mathbb{G}_2$ instead of $\widehat{\mathbb{G}}$ and $\check{\mathbb{H}}$, e.g. we write $\mathtt{com}_1$ instead of $\mathtt{com}_{\widehat{\mathbb{G}}}$.

To apply the above technique to Groth–Sahai proofs with asymmetric pairings, one has to distinguish the left and right operands of pairings, exponentiations and multiplications, *even for scalars*, since they use different commitment schemes. The necessary translations for scalars are straigtforward. E.g. to prove that b is a bit, i.e. $b(1-b) = 0$, one needs to: First decide on a type, say b has type $\mathtt{sca}_1$ (i.e. a "left operand"). Now $b \cdot b$ is invalid, since b is not allowed to be a right operand of a multiplication. So define b' of type $\mathtt{sca}_1$ and prove that $b \cdot 1' - 1 \cdot b' = 0$ and $b \cdot b' - 1 \cdot b' = 0$. Here, 1 (resp. $1'$) has type $\mathtt{unit}_1$ (resp. $\mathtt{unit}_2$).

Optimisations and malleability There are some trivial optimisations to the OR-compilation.

- Logical equivalence: Using logic to simplify the OR-statement prior to (or after) compilation. Malleability of the equivalent sets of equations may differ.
- No "over-allocation": Unused variables (or constants) need not be copied. This preserves malleability.
- Zero-knowledge simulation: Equations where Groth–Sahai proofs are zero-knowledge have a special structure, see for example [EG14]. Roughly, when replacing constants of type $\mathtt{unit}_1$, $\mathtt{unit}_2$, $\mathtt{base}_1$ and $\mathtt{base}_2$ and all (equivocable) variables, i.e. types $\mathtt{com}_1$, $\mathtt{com}_2$, $\mathtt{sca}_1$, $\mathtt{sca}_2$, by 0, the quadratic equation must be satisfied. This means that replacing $\mathtt{unit}_1$, $\mathtt{unit}_2$ and $\mathtt{base}_1$, $\mathtt{base}_2$ (i.e. 1, $1'$, $[1]_1$, $[1']_2$) by b resp. $1-b$ (in the correct type) ensures that one of the two equations can be simulated. *This affects malleability,* because it replaces constants by variables.

Also see [Cha+12, Appendix B] for an OR-compilation with a slightly different (better) technique.

Let us briefly sketch why the naive OR-compilation preserves malleability: Mauling a copied variable (by addition/scalar multiplication) is straightforward. Maul the original and both copies. For example $x_1 b' - x_{1,1} b' = 0$ or equivalently $(x_1 - x_{1,1})b' = 0$ can trivially be mauled to $(\alpha x_1 - \alpha x_{1,1})b' = 0$ by using the linearity of GS commitments. Similarly, one can use linearity to add z to x_1 in the OR-statement: $((x_1 + z) - (x_{1,1} + z))b' = (x_1 - x_{1,1})b' = 0$, hence replacing the commitment to x_1 by a mauled commitment to $x_1 + z$ works (if the commitment to $x_{1,1}$ is also replaced by $(x_{1,1} + z)$). Obviously, the same has to be done for $x_{1,0}$.

A bit more concretely, given commitments $c_{b'} = \mathsf{Com}_{\mathtt{sca}_1}(b')$, $c_x = \mathsf{Com}_{\mathtt{com}_1}(x)$, $c_{x_0} = \mathsf{Com}_{\mathtt{com}_1}(x_0)$, $c_z = \mathsf{Com}_{\mathtt{com}_1}(z)$, we let $c_{x+z} := c_x + c_z = \mathsf{Com}(x + z)$ and $c_{x_0+z} := c_{x_0} + c_z$ (with suitable, but unknown randomness). Then

$$\widetilde{e}(c_x, c_{b'}) - \widetilde{e}(c_{x_0}, c_{b'}) = \widetilde{e}(c_x + c_z, c_{b'}) - \widetilde{e}(c_{x_0} + c_z, c_{b'}) = \widetilde{e}(c_{x+z}, c_{b'}) - \widetilde{e}(c_{x_0+z}, c_{b'})$$

and thus the same GS proof works for this mauled "copy-equation". Here $\widetilde{e}$ denotes the pairing defined on the commitments, not the "base pairing" $e\colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$.

This is a sketchy, high-level demonstration that OR-compilation preserves malleability. A detailed, formal explanation can be found in [Cha+12].

E Key-rotatable structure preserving signatures

Here, we define properties such as structure preserving and key-rotation for signatures. Then, we present concrete (one-time and zero-time) structure preserving signatures, which allow key-rotation and have simulatable token generation.

E.1 Key-rotatable structure preserving signatures

In this section, we define signatures and EUF-CMA security as usual. Except perhaps, that we add system parameters, as for all of our schemes.

Definition 22. *A **signature scheme** SIG consists of following algorithms:*

$\mathsf{SIG.GenSP}(pp)$**:** *returns system parameter sp which are implicit input for following algorithms.*
$\mathsf{SIG.GenKey}(sp)$**:** *returns a secret key sk and verification key vk.*
$\mathsf{SIG.Sign}(sk, m)$**:** *returns a signature σ of message m with secret key sk.*
$\mathsf{SIG.Verify}(vk, m, \sigma)$**:** *either accepts or rejects signature σ for message m under vk.*

Definition 23. *A signature scheme SIG is unbounded resp. m-time EUF-CMA secure if any PPT adversary has negligible advantage in the EUF-CMA game in Fig. 10 (given unbounded resp. m signing oracle queries).*

To put this into perspective with "leakage", one may say that SIG is secure under leakage $\mathsf{leak}(k) = vk$.

```
Experiment $\mathsf{Exp}^{\mathsf{euf\text{-}cma}}_{\mathsf{SKE},\mathcal{A}}(\kappa)$
  $\mathbf{M} = \emptyset$; $sp \leftarrow \mathsf{GenSP}(pp)$; $(sk, vk) \leftarrow \mathsf{GenKey}(sp)$;
  $\sigma \leftarrow \mathcal{A}^{\mathsf{Sign}}(pp, sp, vk)$;
  return $m \notin \mathbf{M}$ and $\mathsf{Verify}(vk, m, \sigma) = 1$
```

Fig. 10. The EUF-CMA game. $\mathsf{Sign}(m)$ returns $\mathsf{Sign}(sk, m)$ and adds m to $\mathbf{M}$.

Structure-preserving signatures. Signature schemes which are compatible with Groth–Sahai proofs in the sense that a valid signature-message pair can be extracted from a Groth–Sahai proof of signature verification are called structure preserving.

Definition 24 (See [Abe+10]). *A signature scheme is* ***structure-preserving*** *(or short a SPS) if messages, the verification key and the signature are group elements in $\mathbb{G}_1$ or $\mathbb{G}_2$, and if the verification equation is a pairing product equation. (See App. D for pairing product equations.)*

In our case, a slightly weaker definition of "structure preserving", namely only message and signatures have to be group elements, would suffice. The verification key and equation types are irrelevant, since we only need to extract a valid signature under a known verification key from a valid proof.

Key-rotatable signatures. The definition of key-rotatable signatures is straightforward. Given an update token, one can rotate a signature from one verification key to another. In line with other security definitions under leakage, we may consider the "leakage" of a secret key to be respective (public) verification key. Thus, simulatable token generation is defined as usual for key-rotatable signatures.

Definition 25. *A (key-rotatable or)* ***updatable signature scheme*** USIG *consists of* $(\mathsf{GenSP}, \mathsf{GenKey}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{GenTok}, \mathsf{ReSig})$ *such that*

- $(\mathsf{GenSP}, \mathsf{GenKey}, \mathsf{Sign}, \mathsf{Verify})$ *is the* underlying *signature scheme.*
- $\mathsf{GenTok}(sp, k^{\mathrm{old}}, k^{\mathrm{new}})$ *returns an update token Δ.*
- $\mathsf{ReSig}(\Delta, \sigma^{\mathrm{old}})$ *returns the updated signature σ^{new}.*

All algorithms (except GenSP*) receive sp and pp as* implicit *inputs.*

For correctness, we require the correctness of the underlying signature scheme as well as: For all $sp \leftarrow \mathsf{GenSP}(pp)$, $k^{\mathrm{old}} = (sk^{\mathrm{old}}, vk^{\mathrm{old}}) \leftarrow \mathsf{GenKey}(sp)$, $k^{\mathrm{new}} = (sk^{\mathrm{new}}, vk^{\mathrm{new}}) \leftarrow \mathsf{GenKey}(sp)$, $\Delta \leftarrow \mathsf{GenTok}(sk^{\mathrm{old}}, sk^{\mathrm{new}})$ and for all purported signatures σ, we have

$$\mathsf{Verify}(vk^{\mathrm{new}}, \mathsf{ReSig}(\Delta, \sigma)) = \mathsf{Verify}(vk^{\mathrm{old}}, \sigma)$$

We call USIG *(m-time) EUF-CMA secure, if its underlying signature is (m-time) EUF-CMA secure.*

Just like for key-rotatable PRFs, we do not define security games for key-rotatable signatures. Simulatable token generation for the signature allows to reduce directly to the security of the underlying signature scheme. The definition of simulatable token generation (for signatures) is the general one, i.e. the one given in App. C.

E.2 A key-rotatable one-time signature

In this section, we recall the one-time structure preserving signature of interest from [KPW15]. Our definition differs slightly from [KPW15] since we need to fix a part of the verification key as a system parameter in order to have simulatable token generation. The security proof of [KPW15, Theorem 1] still works without change.

Definition 26 ([KPW15, Fig. 2 with $k = 1$]). *Define the one-time SPS* OTS *for messages in* $\mathbb{G}_1^n$ *as follows:*

– $\mathsf{OTS.GenSP}(pp)$*: returns* $sp = [\mathbf{A}]_2$*, where* $\mathbf{A} = \begin{pmatrix} 1 \\ r \end{pmatrix}$ *for* $r \leftarrow \mathbb{F}_p$.
– $\mathsf{OTS.GenKey}(sp)$ *computes* (sk, vk) *as follows:*
  - $sk = \mathbf{K} \leftarrow \mathbb{F}_p^{(n+1)\times 1}$,
  - $vk = [\mathbf{C}]_2$ *where* $[\mathbf{C}]_2 = [\mathbf{KA}]_2 = \mathbf{K}[\mathbf{A}]_2$.
– $\mathsf{OTS.Sign}(sk, [m]_1) = [(1, m)\mathbf{K}]_1 \in \mathbb{G}_1^{1\times 2}$ *where* $m \in \mathbb{F}^n$.
– $\mathsf{OTS.Verify}(vk, [m]_1, \sigma)$ *checks if* $e(\sigma, [\mathbf{A}]_2) - e([1, m]_1, [\mathbf{C}]_2) = 0$.

Note that signing is deterministic*. For key-rotation, we define*

– $\mathsf{OTS.GenTok}(k^{\text{old}}, k^{\text{new}})$*: for* $sk^{\text{old}} = \mathbf{K}^{\text{old}}$*,* $sk^{\text{new}} = \mathbf{K}^{\text{new}}$ *return* $\Delta = \mathbf{K}^{\text{new}} - \mathbf{K}^{\text{old}}$
– $\mathsf{OTS.ReSig}(\Delta, \sigma^{\text{old}})$*: returns* σ^{new} *as* $\sigma^{\text{new}} = \sigma^{\text{old}} + [(1, m)\Delta]_1 \in \mathbb{G}_1^{1\times 2}$.

For use as the signature scheme SIG in NYUE, one needs $n = 2$. For OTS in NYUAE $n = 1$ is sufficient. Key-rotation is only necessary for OTS.

Remark 1. The one-time signature OTS from Def. 26 satisfies simulatable token generation under "leakage" of the verification key $\mathsf{leak}(sk) = vk = [\mathbf{C}]_2$. This is straightforward to see: Namely the relation between vk^{old} and vk^{new} is $[\mathbf{C}^{\text{new}}] = \Delta[\mathbf{A}] - [\mathbf{C}^{\text{old}}]$. Hence, given Δ and one verification key the *unique* other verification key can be efficiently computed. Also, $\mathsf{invert}(\Delta) = -\Delta$ yields a reverse token. Moreover, given any two entries of (sk_1, Δ, sk_2), there is an (efficiently computable) *unique* third. With this, it's easily checked that token generation is simulatable under "leakage" of $\mathsf{leak}(sk) = vk$.

Lemma 7. *The one-time SPS* OTS *in Def. 26 is EUF-CMA secure under SXDH in* $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$.

Proof. Theorem 1 of [KPW15] easily extends to the setting where $[\mathbf{A}]_2$ is used as a system parameter. Indeed, the proof embeds a "$\mathcal{D}_k$-KerMDH" challenge in $[\mathbf{A}]_2$ (and a purported signature), and works essentially without change. The "$\mathcal{D}_k$-KerMDH" assumption is implied by SXDH, c.f. [MRV16].

Key-rotation. We now take a closer look at the ReSig algorithm and verification equation. The former is affine, the latter is linear. All in all, we can use linear malleability of GS proofs to adapt the proof of knowledge of a signature.

In the proof π (c.f. Def. 16) for statement $S_{\text{NY+I}}$, we have commitments to $[\widehat{m}]_1$ and $[\widehat{\sigma}^{\text{old}}]_1 \in \mathbb{G}_1^2$. We need a rotated committed signature $[\widehat{\sigma}^{\text{new}}]_1 \in \mathbb{G}_1^2$.

Let $[\widehat{M}] := \begin{bmatrix} 1 \\ \widehat{m} \end{bmatrix}$ and note that it is unaffected by key-rotation, which must preserve the signed message. The relevant equations (in dlogs) are

$$\widehat{\sigma}^{\text{old}}\mathbf{A} - \widehat{M}\mathbf{C}^{\text{old}} = 0 \tag{6}$$

$$\widehat{\sigma}^{\text{new}} = \widehat{\sigma}^{\text{old}} + \widehat{M}\Delta \tag{7}$$

The update token Δ is by definition $\mathbf{K}_2 - \mathbf{K}_1$. Note that in Eq. (7), a commitment to $\widehat{\sigma}^{\text{new}}$ can be computed from the commitments to $\widehat{\sigma}^{\text{old}}$, $\widehat{M}$, since Groth–Sahai commitments are linearly homomorphic. Consequently, given a proof π for Eq. (6), and the update token Δ, we find a proof for Eq. (7). Using $\Delta\mathbf{A} = \mathbf{C}^{\text{new}} - \mathbf{C}^{\text{old}}$ in Eq. (7), yields

$$\begin{aligned} 0 &= \widehat{\sigma}^{\text{old}}\mathbf{A} - \widehat{M}\mathbf{C}^{\text{old}} \\ &= (\widehat{\sigma}^{\text{old}} + \widehat{M}\Delta)\mathbf{A} - \widehat{M}(\mathbf{C}^{\text{old}} + \Delta\mathbf{A}) \\ &= \widehat{\sigma}^{\text{new}}\mathbf{A} - \widehat{M}\mathbf{C}^{\text{new}} \end{aligned}$$

Note that we merely substituted the definition of $\widehat{\sigma}^{\text{new}}$. Computing the commitment to $[\widehat{\sigma}^{\text{new}}]_1 = [\widehat{\sigma}^{\text{old}}] + [\widehat{M}]_1\Delta$ homomorphically from the commitments to $[\widehat{\sigma}^{\text{old}}]_1$ and $[\widehat{M}]_1$ is possible (and by construction, satisfies the verfication equation under the vk^{new}). Thus, the GS proof for the original statement also proves the adapted equation. This shows that OTS supports key-rotation under Groth–Sahai proofs.

E.3 Key-rotatable zero-time signature

In our security proof for plaintext integrity of NYUAE, we encounter the interesting situation that we do not rely on the *one*-time security, but the *zero*-time security, of OTS. That is, we obtain an adversary which breaks OTS without requesting a signature at all. Here, we give another scheme, which is only zero-time secure.

Zero-time EUF-CMA security is "strange". A zero-time signature can be independent of the message, i.e. Verify can ignore the message. Indeed, a hard relation already yields a zero-time signature. Thus, a zero-time signature can simply return its secret key as the signature. Without seeing a signature, finding sk such that $\mathsf{Verify}(pk, m, sk)$ is hard. In other words, $\mathsf{Verify}(pk, m, \_)$ is a hard relation for all messages. After seeing a single signature, there are no security guarantees anymore. So having message-independent "signature" $\sigma = sk$ is not a problem.

These weak security guarantees allow quite simple and efficient instantiations.

Definition 27. *We define a zero-time key-rotatable structure preserving signature* ZTS *as follows.*

- $\mathsf{ZTS.GenSP}(pp)$ *does nothing, i.e.* $sp = pp$.
- $\mathsf{ZTS.GenKey}(sp)$ *picks* $x \leftarrow_R \mathbb{F}_p$ *and returns* $(sk, vk) = (x, [x]_2) \in \mathbb{F}_p \times \mathbb{G}_2$.
- $\mathsf{ZTS.Sign}(sk, m)$ *returns* $\sigma = [sk]_1 = [x]_1$.
- $\mathsf{ZTS.Verify}(vk, m, \sigma)$ *accepts if* $e(\sigma, [1]_2) = e([1]_1, vk)$.
- $\mathsf{ZTS.GenTok}(sk^{\text{old}}, sk^{\text{new}})$ *returns* $\Delta = sk_2 - sk_1$.
- $\mathsf{ZTS.ReSig}(\Delta, \sigma^{\text{old}})$ *returns* $\sigma^{\text{new}} = \sigma^{\text{old}} + [\Delta]_1$.

It is easy to see that ZTS as above is a zero-time signature if the SXDH assumption holds in $(e, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$. Furthermore, the verification equations and key-rotation are linear, hence GS proofs of ZTS verification can be adapted to the new key. Lastly, ZTS has simulatable token generation under "leakage" of vk.

F Detailed definition of NYUE and NYUAE

In this section, we present more details regarding the definitions of NYUE and NYUAE. We start with an intuitive explanation of the malleability, ignoring the OR-compilation. Then we explain how OR-compilation works and give a bit of intuition why OR-compilation preserves malleability. Finally, we give the exact statements (i.e. equations) whose proof (of satisfiability) shows $\mathsf{OR}(\mathcal{S}_{\mathrm{NY}}, \mathcal{S}_{\mathsf{SIG}})$.

F.1 RISE

In this section, we recall RISE, an updatable encryption scheme defined in [LT18]. RISE is essentially ElGamal encryption. It is proven UP-IND-CPA secure for queried reencryptions in [LT18]. The definition of updatable encryption schemes in [LT18] is slightly more general than ours, so we redefine RISE in our setting.

Definition 28. *The updatable encryption scheme* RISE *is defined as follows:*

$\mathsf{RISE.GenSP}(pp)$**:** *does nothing. We have* $\mathcal{K} = \mathbb{F}_p^\times$, $\mathcal{D} = \mathbb{F}_p^\times \times \mathbb{G}^\times$, $\mathcal{M} = \mathbb{G}$.
$\mathsf{RISE.GenKey}(sp)$**:** *Pick* $sk \leftarrow_R \mathcal{K}$. *Return* $k = (sk, pk)$ *where* $pk = [sk]$.
$\mathsf{RISE.Enc}(k, m; r)$**:** *For* $r \in \mathbb{F}_p$ *(which is drawn uniformly at random if not specified), parse* $k = (sk, pk)$ *and return* $[c] = [c_1, c_2] := [0, m] + r[pk, 1] \in \mathbb{G}^2$.
$\mathsf{RISE.Dec}(k, c)$**:** *parse* $k = (sk, pk)$ *and return* $[m] = [c_2] - \frac{1}{sk}[c_1]$.
$\mathsf{RISE.GenTok}(k^{\mathrm{old}}, k^{\mathrm{new}})$**:** *Parse* $k^{\mathrm{old}} = (sk^{\mathrm{old}}, pk^{\mathrm{old}})$, $k^{\mathrm{new}} = (sk^{\mathrm{new}}, pk^{\mathrm{new}})$ *and let* $\Delta = (\frac{sk^{\mathrm{new}}}{sk^{\mathrm{old}}}, pk^{\mathrm{new}}) \in \mathbb{F}_p^\times \times \mathbb{G}^\times$.
$\mathsf{RISE.ReEnc}(\Delta, c)$**:** *parse* $\Delta = (\rho, pk^{\mathrm{new}})$, *pick* $r \leftarrow_R \mathbb{F}_p$ *and return*

$$c' = \begin{bmatrix} \rho c_1 \\ c_2 \end{bmatrix} + \begin{bmatrix} pk^{\mathrm{new}} \\ 1 \end{bmatrix} r, \quad \text{where } c = [c_1, c_2]^\top \in \mathbb{G}^2. \tag{8}$$

We consider security (of the underlying encryption) under leakage $\mathsf{leak}(k) = pk$, *as* pk *is necessary for reencryption (as part of* Δ*).*

It is easy to see that RISE satisfies correctness. Under the DDH assumption on $\mathbb{G}$, the underlying encryption scheme of RISE is IND-CPA secure for key-leakage $\mathsf{leak}(k) = pk$. Indeed, IND-CPA security reduces directly DDH.

We do not (explicitly) need any further security guarantees of RISE — its UP-IND-CPA security is not directly needed anywhere. This is because we essentially reprove the security in our setting.

Remark 2. RISE has perfect re-encryption. Indeed, reencryption of RISE should be seen as a two-step process. In a first step, the first component c_1 of the ciphertext c is multiplied by ρ (where $\Delta = (\rho, pk^{\mathrm{new}})$). This changes the encryption from k^{old} to k^{new}. In a second step, the key-rotated ciphertext is rerandomised under pk^{new}. Since randomisation is perfect (i.e. looks like a fresh encryption), so is re-encryption.

Lemma 8. RISE *has simulatable token generation under leakage* $\mathsf{leak}(k) = pk$.

Proof. Pick $\Delta = (\rho, pk^{\mathrm{new}})$ via $\rho \leftarrow \mathbb{F}_p^\times$, $pk^{\mathrm{new}} \leftarrow \rho pk^{\mathrm{old}}$ where $pk^{\mathrm{old}} = \mathsf{leak}(k)$. Let $\Delta' = \mathsf{invert}((\rho, pk^{\mathrm{new}})) = (\rho^{-1}, \rho^{-1} pk^{\mathrm{new}})$. Now Δ' is the reverse token for Δ and sampling Δ as described satisfies the properties for SimTok.

F.2 Reencryption

For convenience, we repeat the 4 steps for reencryption of NYUE (and NYUAE). Every step adapts the GS proofs, if necessary.

(1) Verify ciphertext. Verify the consistency proof. Return $\perp$ on failure.
(2) Key rotation. Use key rotation of RISE on the ciphertexts parts c_1 and c_2 of $c = (c_1, c_2, \pi)$, but *without* the implicit *re-randomisation*. For NYUAE, also the committed signature is rotated.
(3) Re-randomise c_1, c_2. The ciphertexts c_1, c_2 are re-randomised, thus completing the computation of $\mathsf{RISE.ReEnc}(\Delta_i, c_i)$ for $i = 1, 2$.
(4) Re-randomise π. The proof π is rerandomised. (More precisely, all GS commitments and all GS proofs are rerandomised.)

We now describe the individual steps (2)-(3) for key-rotation and re-randomisation of (c_1, c_2) in more detail. Steps (1) and (4) are trivial and therefore omitted.

For constructing the OR-proof in NYUE we can use folklore compilation techniques (App. D.1). Fortunately, this compilation preserves linear malleability of the original set of equations [Cha+12]. Therefore, we only explain how to maul proofs for S_{NY} and S_{SIG} separately.

(2) Key rotation. First, we apply RISE.ReEnc without rerandomisation, i.e. Eq. (8) with $r = 0$. The equations proven in S_{NY} are

$$\forall i \in \{1,2\}: \quad \begin{bmatrix} 0 \\ \widehat{m} \end{bmatrix}_1 1 + \begin{bmatrix} pk_i^{\mathrm{old}} \\ 1 \end{bmatrix}_1 \widehat{r}_1 - \begin{bmatrix} c_{i,1}^{\mathrm{old}} \\ c_{i,2}^{\mathrm{old}} \end{bmatrix}_1 1 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}_1 . \tag{9}$$

There are two corresponding equations in S_{SIG} (with $\widehat{m_i}$ instead of $\widehat{m}$) which are dealt with completely analogously. Key-rotation for RISE with $r = 0$ simply multiplies the top row in Eq. (9) by ρ_i, yielding $(c_{i,1}^{\mathrm{new}} c_{i,2}^{\mathrm{new}}) = (\rho_i c_{i,1}^{\mathrm{old}} c_{i,2}^{\mathrm{old}})$. Note that $pk_i^{\mathrm{new}} = \rho_i pk_i^{\mathrm{old}}$, so we end up with the right key.

We then use the GS malleability to adapt the consistency proof π to this new statement. Note that we know the token $\Delta = (\Delta_1, \Delta_2)$, which includes ρ_i and pk_i^{new} (and hence implicitly pk_i^{old}). By linear malleability, an adapted proof π^{new} after multiplication by ρ_i can be computed. This leaves us with a proof for

$$\forall i \in \{1,2\}: \quad \begin{bmatrix} 0 \\ \widehat{m} \end{bmatrix}_1 1 + \begin{bmatrix} pk_i^{\mathrm{new}} \\ 1 \end{bmatrix}_1 \widehat{r}_1 - \begin{bmatrix} c_{i,1}^{\mathrm{new}} \\ c_{i,2}^{\mathrm{new}} \end{bmatrix}_1 1 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}_1 . \tag{10}$$

Thus, we now have an updated ciphertext under k^{new}. Therefore, we drop the “old” and “new” superscripts in the following.

(3) Re-randomise (c_1, c_2). Here, we exploit that we can add two equations with “attached” GS proofs and produce an adapted GS proof for the sum. Again, we merely consider how S_{NY}, i.e. Eq. (9), is affected by a re-randomisation. Moreover we only look at c_1, i.e. we consider re-randomisation of a RISE ciphertext under GS proofs. Dealing with c_2 and S_{SIG} is completely analogous.

First, we pick random $\widehat{r}' \leftarrow \mathbb{F}_p$ and compute the rerandomised ciphertext $[c_1']_1 = [c_{1,1}', c_{1,2}']_1$, with rerandomisation randomness $\widehat{r}'$. Then we construct a GS proof π' for

$$\begin{bmatrix} pk_1 \\ 1 \end{bmatrix}_1 \widehat{r}' + \begin{bmatrix} c_{1,1} \\ c_{1,2} \end{bmatrix}_1 - \begin{bmatrix} c_{1,1}' \\ c_{1,2}' \end{bmatrix}_1 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}_1,$$

which states $c_1' = (c_{1,1}', c_{1,2}')$ is a rerandomisation of c_1, c.f. Rem. 2. Now, we use the linearity of GS proofs to find that, given proofs for

$$\begin{bmatrix} 0 \\ \widehat{m} \end{bmatrix}_1 1 + \begin{bmatrix} pk_1 \\ 1 \end{bmatrix}_1 \widehat{r}_1 - \begin{bmatrix} c_{1,1} \\ c_{1,2} \end{bmatrix}_1 1 = 0 \quad \text{and} \quad \begin{bmatrix} pk_1 \\ 1 \end{bmatrix}_1 \widehat{r}' + \begin{bmatrix} c_{1,1} \\ c_{1,2} \end{bmatrix}_1 1 - \begin{bmatrix} c_{1,1}' \\ c_{1,2}' \end{bmatrix}_1 1 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}_1,$$

we can compute a proof for the sum:

$$\begin{bmatrix} 0 \\ \widehat{m} \end{bmatrix}_1 1 + \begin{bmatrix} pk_1 \\ 1 \end{bmatrix}_1 (\widehat{r}_1 + \widehat{r}') - \begin{bmatrix} c_{1,1}' \\ c_{1,2}' \end{bmatrix}_1 1 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}_1.$$

Thus, we successfully rerandomised c_1 (to c_1' with new encryption randomness $\widehat{r}_1 + \widehat{r}'$) and can adapt the proof.

F.3 The consistency proof for NYUE

We use the notation of Def. 15. For convenience, we repeat the proven statement $\mathsf{OR}(S_{\mathrm{NY}}, S_{\mathsf{SIG}})$:

- S_{NY}: $\exists \widehat{m}, \widehat{r}_1, \widehat{r}_2$: $\mathsf{RISE.Enc}(pk_1, \widehat{m}; \widehat{r}_1) = c_1, \wedge \mathsf{RISE.Enc}(pk_2, \widehat{m}; \widehat{r}_2) = c_2$
- S_{SIG}: $\exists \widehat{m_1}, \widehat{m_2}, \widehat{r}_1, \widehat{r}_2, \widehat{\sigma},$: $\mathsf{RISE.Enc}(pk_1, \widehat{m_1}; \widehat{r}_2) = c_1, \wedge \mathsf{RISE.Enc}(pk_2, \widehat{m_2}; \widehat{r}_2) = c_2 \wedge \mathsf{SIG.Verify}(vk_{\mathsf{SIG}}, (\widehat{m_1}, \widehat{m_2}), \widehat{\sigma}) = 1$

Here pk_i is the key-leakage, essentially an ElGamal public key $pk_i = [sk_i]_1$.

From App. D.1, it should be evident how to compile the equations naively. We present a simpler (and more efficient) OR-compilation, essentially applying some of the stated optimisations. We prove that (for S_{NY}) we have $[\widehat{m}_1]_1 = [\widehat{m}_1]_2 = [\widehat{m}]_1$. But in S_{SIG} the variables $[\widehat{m}_1]_1$ and $[\widehat{m}_1]_2$ are unconstrained. In both cases, the RISE.Enc parts must be satisfied.

Compilation for NYUE. We have following typed equations:

$$b \cdot 1' - 1 \cdot b' = 0 \quad \wedge \quad b \cdot b' - 1 \cdot b' = 0 \tag{11}$$

where b is of type $\mathtt{sca}_1$, b' of type $\mathtt{sca}_2$. (1 has type $\mathtt{unit}_1$ and $1'$ has type $\mathtt{unit}_2$.) The equation type is $\mathtt{QE}$. If $b = 0$, we want to prove S_{NY}. If $b = 1$, we want to prove S_{SIG}.

The OR-compiled "message consistency" for S_{NY} is given by

$$([\widehat{m}_1]_1 - [\widehat{m}]_1) \cdot (1' - b') = 0 \quad \wedge \quad ([\widehat{m}_2]_1 - [\widehat{m}]_1) \cdot (1' - b') = 0$$

where $[\widehat{m}_i]_1$ and $[m]$ have type $\mathtt{com}_1$. The equation types are $\mathtt{ME}_1$.

Proving the equations

$$\begin{bmatrix} 0 \\ \widehat{m}_1 \end{bmatrix}_1 1' + \begin{bmatrix} pk_1 \\ 1 \end{bmatrix}_1 \widehat{r}_1 - \begin{bmatrix} c_{1,1} \\ c_{1,2} \end{bmatrix}_1 1' = \begin{bmatrix} 0 \\ 0 \end{bmatrix}_T$$
$$\begin{bmatrix} 0 \\ \widehat{m}_2 \end{bmatrix}_1 1' + \begin{bmatrix} pk_2 \\ 1 \end{bmatrix}_1 \widehat{r}_2 - \begin{bmatrix} c_{2,1} \\ c_{2,2} \end{bmatrix}_1 1' = \begin{bmatrix} 0 \\ 0 \end{bmatrix}_T$$

already ensures S_{NY} (if $b = 0$). Here, pk_1 has type $\mathtt{pub}_1$ and $\widehat{r}$ has type $\mathtt{sca}_2$. The equations have type $\mathtt{ME}_1$ in the first and $\mathtt{MConst}_1$ in the second component.

It should be evident, that this equation is still linearly malleable. Furthermore, this part of the equation is the only part which must be malleable. To finish the set of equations, we need to choose some signature SIG and finish the OR-statement, which is: If $b = 0$, then there is a committed valid signature $[\widehat{\boldsymbol{\sigma}}]_1$ on $[\widehat{m}_1, \widehat{m}_2]_1$.

For example, when using the one-time signature OTS from Def. 26, we could introduce an additional "bit" $[d]_1 \in \mathbb{G}_1$, committed with $\mathtt{com}_{\mathbb{G}_1}$ and prove

$$[d]_1 1' - [1]_1(1' - b') = [0]_1,$$

which has type $\mathtt{ME}_1$. The verification equation of the signature becomes

$$e([\widehat{\boldsymbol{\sigma}}]_1, [\mathbf{A}]_2) - e([d, \widehat{m}_1, \widehat{m}_2]_1, [\mathbf{C}]_2) = [0]_T,$$

where $[\mathbf{A}]_2$ and $[\mathbf{C}]_2$ are part of the signature verification key. This equation has type $\mathtt{PPE}$. Clearly, if $b = 1$, then $[d]_1 = [1]_1$ and this enforces a valid signature. If $b = 0$, then $[d]_1 = [0]_1$ and $\widehat{\boldsymbol{\sigma}} = 0$ is a trivial solution to the equation. Note that the signature verification is *not* malleable anymore.

Compilation for NYUAE. For NYUAE with OTS, we *cannot* just use the above compilation, (with equation $[d]_1(1 - b') = [0]_1$ replaced by $[d]_1 b' = [0]_1$). Because now we need *malleability of the sigature*, which is destroyed in the above OR-compilation. For efficiency and space reasons, we therefore instantiate NYUAE with the zero-time signature ZTS from Def. 27. We use the naive OR-compilation, where we copy the constants (only vk in this case). Thus, we add following equations:

$$b \cdot ([\widehat{v}]_1 - [vk]_2) = 0$$
$$e([\widehat{\boldsymbol{\sigma}}]_1, [1]_2) - e([1]_1, [\widehat{v}]_2) = 0$$

The equations have type $\mathtt{ME}_2$ and $\mathtt{PPE}$. If $b = 0$, then $[\widehat{\boldsymbol{\sigma}}]_1] = [0]_1$ and $[\widehat{v}]_2 = [0]_2$ allow simulation. Signature rotation of ZTS is preserved due to linear malleability of the (compiled) equation.

This completes $S_{\mathrm{NY+I}}$. For S_{SIG}, we need some (unbounded) EUF-CMA signature scheme. Using [KPW15, Fig. 3] with SXDH ($k = 1$), we have signature consisting of $[\widehat{\boldsymbol{\sigma}_i}]_1 \in \mathbb{G}_1^{1\times 2}$ for $i = 1, 2, 3$ and $[\widehat{\sigma}_4]_2 \in \mathbb{G}_2$. Moreover, the verification equation consists of $k + 1$ $\mathtt{PPE}$'s, which can be OR-compiled like for OTS before. (Remember that we do not need malleability for S_{SIG}. Simulation works by zeroing all $\widehat{\boldsymbol{\sigma}}_i$ and $\widehat{m_i}$ and using $[d] = [0]$ as before.)

π	$\mathtt{sca}_1$	$\mathtt{sca}_2$	$\mathtt{com}_1$	$\mathtt{com}_2$	QE	$\mathtt{ME}_1$	$\mathtt{MConst}_1$	$\mathtt{ME}_2$	PPE	$\mathtt{PConst}_2$	$\mathbb{G}_1$	$\mathbb{G}_2$
NYUE	1	4	5	0	2	5	2	0	0	1	28	34
NYUAE	1	4	$5+6$	$1+1$	2	4	2	1	$0+3$	0	54	44

Table 1. Size of consistency proofs. (Without public constants, e.g. the ciphertexts c_1, c_2 and the signature verification keys.) We use the signatures described in the detailed instantiation of NYUE and NYUAE. The costs introdcued by SIG (instantiated with [KPW15, Fig. 3]) are written additively in NYUAE.

Efficiency estimate. In the following, we give rough efficiency outlines. For NYUE we count

We could use the encryption type from [EG14] and replace group commitments with encryptions. This would decrease the size of the proofs since $\mathtt{MEnc}_1$ has size $(2, 2)$ opposed to $(2, 4)$. Another option is to may be [Ràf15], which could yield more efficient OR-proofs. However, the number of commitments to necessary values alone is so much, that the scheme is still very far practical interest. Therefore, we did not pursue such optimisations further.

We approximate the number of exponentiations by twice for proof generation by the number of group elements (and rerandomisation and reencryption). (This is exact for commitments, but for proofs, it is a rough approximation.) This give us $\approx (60, 70)$ exponentiations (in $\mathbb{G}_1$ resp. $\mathbb{G}_2$) for NYUE and $\approx (110, 90)$ for NYUAE. We do not know whether the prover-chosen commitment technique of [EG14] preserves rerandomisability of proofs. It would reduce the necessary exponentiations roughly by half. Still, proofs and rerandomisation remain expensive.

For decryption, we need to check the equations. Using the probabilistic verification of [Her+17], we can reduce the number of pairings needed to compute to rougly 22 for NYUE and 29 for NYUE. We ignore the additional overhead, which is one (small) exponentiation per commitment and several per proof.

G Security proofs for NYUE and NYUAE

In this section, we give formal proofs for of security of NYUE and NYUAE. We prove their underlying encryption schemes secure under (key-)leakage $\mathsf{leak}(k) = pk$. Then we prove their security as updatable encryption schemes. The proofs have a structure which is very similar to the proof of UP-IND-CCA security in Thm. 1. Important differences between schemes allowed in Thm. 1 and NYUE (resp. NYUAE) are:

- Ciphertext consistency is *publicly verifiable.*
- Reencryption is *perfect.*
- (Key-)Leakage is non-trivial and simulatable token generation *under leakage* is necessary, see App. C.

The first two properties actually *simplify* the proof. The last one hardly affects the proof. It does however affect the security requirements on the underlying encryption schemes.

Reencryption and token generation. For NYUE and NYUAE we showed perfect reencryption when defining how reencryption works. For token simulation, it is easy to see that it suffices if the building blocks, i.e. RISE and OTS have simulatable token generation.

Corollary 3. NYUE *and* NYUAE *have simulatable token generation under leakage* $\mathsf{leak}(k) = pk$ *as in Definitions 15 and 16. Moreover they have perfect reencryption.*

G.1 IND-RCCA Security of NYUE

Proposition 2. *Suppose* SIG *is (one-time) EUF-CMA secure and SXDH holds in* $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$*. Then the* underlying *encryption scheme of* NYUE *is IND-RCCA-secure under leakage* $\mathsf{leak}(k) = pk$ *as in Def. 15.*

The security proof is a straightforward use of double encryption [NY90]. First, we possibly make the second component of the challenge ciphertext, c_2, inconsistent, by always encrypting m_0^* there. Then, we switch the decryption from first to the second component. To distinguish, the adversary has to break the (one-time) EUF-CMA security of SIG, which is used to provide (a weak form of) simulation soundness. After this, we also encrypt m_0^* in c_1, thus perfectly hiding the challenge bit b from $\mathcal{A}$.

Proof. The assumption, that DDH holds in $\mathbb{G}_1$ and $\mathbb{G}_2$, guarantees that GS proofs are zero-knowledge and the underlying encryption of RISE is secure under the given leakge. Requiring that SIG is (one-time) EUF-CMA secure ensures that the adversary cannot simulate a consistency proof for a fresh message, even in the presence of simulated proofs. More precisely, if $\mathrm{crs}_{\mathrm{GS}}$ is a binding CRS, then proving S_{SIG} is a proof of *knowledge* of a (fresh) signature on $(\widehat{m_1}, \widehat{m_2})$. On the other hand S_{NY}, guarantees that both ciphertexts encrypt the same plaintext. Since GS proofs are perfectly sound for binding CRS, any *simulated* proof contains a valid signature.

A sketch of the security reduction for IND-RCCA is in Table 2 in the form of game hops. The comments in Game i state the properties used to prove indistinguishability of Game $(i-1)$ and Game i.

From Game 3 to Game 4, we embed an IND-CPA challenger (against RISE) as follows: We generate $\mathrm{crs}_{\mathrm{GS}}$ and save the simulation trapdoor $\mathrm{td}_{\mathrm{GS}}$. All proofs are simulated, so we do not need a satisfying witness. In particular, we can generate c_2 by simply using the public key of the IND-CPA challenger. When the distinguisher $\mathcal{D}$ sends his challenge (m_0^*, m_1^*), we pick $m^* := m_b^*$. We send (m^*, m_0^*) as our IND-CPA challenge, and obtain c_2^*. We generate $c_1 \leftarrow_{\mathrm{R}} \mathsf{RISE.Enc}(pk_1, m_b^*)$ and simulate π^* (as always). Finally, at the end of the experiment, $\mathcal{D}$ returns b', and we also return b'. Depending on the IND-CPA challenger's bit $b_{\mathcal{C}}$, we played Game 3 ($b_{\mathcal{C}} = 0$) or Game 4 ($b_{\mathcal{C}} = 1$). If $\mathcal{D}$ distinguishes Game 3 and Game 4 with advantage ε, then we break IND-CPA with advantage ε.

The crucial step is the transition from Game 7 to Game 8. Here, we switch decryption from c_1 to c_2, by playing a (one-time) EUF-CMA game. That is, we obtain as the verification key vk_{SIG} from the EUF-CMA challenger. We make exactly one signature request, namely for (m_b^*, m_0^*).

We modify *internal* decryption of (c_1, c_2, π) as follows: Check the consistency proof π, return $\perp$ if false. Otherwise, return $\mathsf{Dec}(sk_2, c_2)$. The decryption *oracle* Dec adds the usual check for RCCA, i.e. it returns `invalid` if the interal decryption procedure yields $m \in \{m_0^*, m_1^*\}$.

For $\mathcal{A}$, there is at most a difference in Game 7 and Game 8, if $\mathcal{A}$ queries Dec with a ciphertext (c_1, c_2, π) where $\widehat{m}_1 \neq \widehat{m}_2$ and $(\widehat{m}_1, \widehat{m}_2) \neq (m^*, m_0^*)$. Since GS proofs are in binding mode, they are perfectly sound and we have an extraction trapdoor. Because S_{SIG} is structure-preserving, we can extract a witness for S_{SIG} from any such inconsistent *fresh* ciphertext. Hence, we obtain a *valid* signature on $(\widehat{m}_1, \widehat{m}_2)$, which is a signature on a *fresh* message, breaking (one-time) EUF-CMA.

Game	Dec	*stmt*	$\widehat{m}_1$	$\widehat{m}_2$	π^*	$\mathrm{td}_{\mathrm{GS}}$	$\mathrm{crs}_{\mathrm{GS}}$	
1	c_1	hon	m^*	m^*	Prove	$\perp$	hide	Honest.
2	c_1	hon	m^*	m^*	SimProve	sim	hide	Simulate proofs. (Perfect ZK)
3	c_1	sig	m^*	m^*	SimProve	sim	hide	(Perfect) ZK of GS.
4	c_1	sig	m^*	m_0^*	SimProve	sim	hide	IND-CPA
5	c_1	sig	m^*	m_0^*	Prove	sim	hide	(Perfect) ZK of GS.
6	c_1	sig	m^*	m_0^*	Prove	$\perp$	bind	Switch $\mathrm{crs}_{\mathrm{GS}}$. (SXDH)
7	c_1	sig	m^*	m_0^*	Prove	ext	bind	Stat. indist.
8	c_2	sig	m^*	m_0^*	Prove	ext	bind	EUF-CMA (*)
9	c_2	sig	m^*	m_0^*	Prove	$\perp$	bind	Stat. indist.
10	c_2	sig	m^*	m_0^*	Prove	$\perp$	hide	Switch $\mathrm{crs}_{\mathrm{GS}}$. (SXDH)
11	c_2	sig	m^*	m_0^*	SimProve	sim	hide	Simulate proofs. (Perfect ZK)
12	c_2	sig	m_0^*	m_0^*	SimProve	sim	hide	IND-CPA

Table 2. Proof sketch: The columns $\widehat{m}_1$ and $\widehat{m}_2$ denote the messages in the ciphertexts (c_1^*, c_2^*, π^*) (which can be extracted under a binding GS CRS with the extraction trapdoor). For the honest branch S_{NY}, the proof ensures $\widehat{m}_1 = \widehat{m}_2 = m$. For the signature branch S_{SIG}, $\widehat{m}_1 \neq \widehat{m}_2$ is possible. The reduction eventually replaces the challenge message $m^* = m_b^*$ by m_0^*. Hence, Game 12 is independent of the bit b, and $\mathcal{A}$ has advantage 0.

G.2 INT-PTXT Security of NYUAE

We only give a proof sketch for plaintext integrity of NYUAE in Table 3, as IND-RCCA of NYUAE follows as in Prop. 2. In fact, the proof for plaintext integrity is also similar.

Proposition 3. *Suppose* SIG *is unbounded EUF-CMA secure, and SXDH holds in* $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$. *Then the underlying encryption scheme of* NYUAE *in Def. 16 is IND-RCCA and INT-PTXT secure under (key-)leakage* $\mathsf{leak}(k) = pk$. *Moreover, it has perfect reencryption and simulatable token generation.*

Perfect reencryption and token simulation were already established, see Cor. 3. SXDH ensures that NYUE is IND-RCCA secure and OTS (from App. E) one-time EUF-CMA secure.

Proof (Sketch). The proof for IND-RCCA is identical to the IND-RCCA proof for NYUE, Prop. 2. For plaintext integrity, the same overall strategy as for IND-RCCA works. The major difference is that we switch *all* consistency proofs to simulated proofs via S_{SIG}. This reduces to zero-knowledge of GS proofs. Then we switch GS proofs into binding mode. Hence, from a valid consistency proof, a signature for either OTS or SIG can be extracted. Thus, a forgery for a fresh message m either yields a fresh SIG-signature, or a fresh OTS-signature. Thus, breaking the (one-time) EUF-CMA security of (OTS resp.) SIG. The strategy is detailed in Table 3. For completeness, we explain Game 6. In Game 6, GS proofs are in binding mode and *extractable*, hence perfectly sound. In particular, we can extract (messages and) signatures. If $\mathcal{A}$ wins, there are two cases:

– $\mathcal{A}$ proves the simulation branch S_{SIG}. Since the encrypted and committed plaintext pair $[\widehat{m}_1, \widehat{m}_2]_1$ is fresh, $\mathcal{A}$ breaks EUF-CMA security of SIG.
– $\mathcal{A}$ proves the honest branch $S_{\mathrm{NY+I}}$. The signature σ_{OTS} on $[\widehat{m}_1]_1 = [\widehat{m}_2]_1$ must be valid and fresh, hence $\mathcal{A}$ breaks the *zero*-time EUF-CMA security of OTS. (Zero-time, because we never constructed/requested a OTS-signature. Thus, a one-time signature is more than necessary.)

Consequently, we see that $\mathcal{A}$ can be converted into an adversary against either SIG or OTS (or both).

Note that a *zero-time* structure-preserving signature (essentially a hard relation) is sufficient for the security of NYUAE. Inuitively the reason is the following: The scheme is already IND-RCCA secure, even without the signature. To prevent an adversary from generating its own ciphertext it is therefore sufficient to add some secret to the message and fail decryption if the secret is wrong. To preserve public verifiability, we need a verifiable secret which is compatible with Groth–Sahai proofs.

Remark 3. At first glance, having OTS may seem unnecessary, because one can reduce to breaking a SIG-signature even without OTS. But these are two conceptually different things: Either breaking *simulation soundness* of the proof system, or breaking *zero-time security* of OTS (i.e. integrity).

G.3 Security of NYUE (UP-IND-RCCA and UP-INT-PTXT)

The security of NYUE as an updatable encryption scheme can be shown with a proof resembling the proof for deterministic updatable encryption of Thm. 1. Roughly, we replace all (re)encryptions of the challenge by 0. This is done via hybrid games, starting from the last reencryption. Notably, due to *perfect* reencryption, these hybrids are simpler than the ones used in Thm. 1 for deterministic

Game	σ_{OTS}	*stmt*	π	$\mathrm{td}_{\mathrm{GS}}$	$\mathrm{crs}_{\mathrm{GS}}$	
1	hon	hon	Prove	$\perp$	hide	Honest.
2	hon	hon	SimProve	*sim*	hide	Perfect ZK.
3	hon	sig	SimProve	*sim*	hide	Perfect ZK.
4	0	sig	SimProve	*sim*	hide	Perfect ZK.
5	0	sig	Prove	$\perp$	bind	ZK (of GS proofs).
6	0	sig	Prove	*ext*	bind	Extraction.

Table 3. A proof sketch for plaintext integrity of NYUAE.

reencryption. Compared to randomness-preserving reencryption, perfect reencryption spares us from ensuring "consistent reencryption randomness". Proving indistinguishability of hybrids is very similar to Thm. 1, i.e. we guess a region $\{\ell, \dots, r\}$ in which we embed an RCCA challenger.

Supporting arbitrary reencryption (instead of queried) works by construction. Remember that the strategy of replacing reencryptions by decrypt-then-encrypt was the main reason for restricting to queried reencryption in the security proofs, e.g. in Thm. 1. The problem was that one cannot ensure that replacing reencryption by decrypt-then-encrypt works for *invalid* ciphertexts, i.e. for ciphertexts c with $\mathsf{Dec}(c) = \perp$.[13] However, the public verifiability of the consistency proof ensures that $\mathsf{ReEnc}(c)$ can safely return $\perp$ if the proof is invalid. If it is valid, decryption succeeds! Thus, replacing reencryption by decrypt-then-encrypt works for NYUE and NYUAE. This is enough to support arbitrary reencryption.

We state the results in more general terms.

Theorem 9. *Let* UE *be an encryption scheme and consider (key-)leakage* $\mathsf{leak}(k) = pk$. *Suppose* UE *has perfect reencryption, i.e. decrypt-then-encrypt and reencrypt are indistinguishable even for invalid ciphertexts. Suppose moreover that* UE *has simulatable token generation under (key-)leakage* leak. *Then* UE *is*

- *UP-IND-RCCA-secure if* UE *is (multi-)IND-RCCA-secure under leakage* leak.
- *UP-INT-PTXT-secure if* UE *is INT-PTXT-secure under leakage* leak.

under arbitrary reencryption.

By *multi*-IND-RCCA, we mean multi-*ciphertext* security.

We note that the proof of security also works for UP-IND-CPA under queried reencryption. The reason is that queried reencryption plus CPA gives a trivial way to implement a "decryption" oracle (namely table lookup), which can be used instead of the decryption oracle in the proof — nothing breaks thanks to queried reencryption which ensures that decrypt-then-encrypt works. In a sense, restriction to queried reencryptions in the security notion allows to relax perfect reencryption to "perfect for valid ciphertexts". Thus, our general proof essentially subsumes the security of RISE.

[13] Indeed, there are concrete attacks for certain schemes, see App. H.

The strategy of the proof is identical to the UP-IND-CCA and UP-INT-CTXT proofs, c.f. App. B. Indeed, the definition of hybrids and key-insulation steps is essentially identical. The arguments are somewhat simpler, because encryption randomness does not need to be "tracked" anymore, thanks to perfect reencryption.

Proof. **UP-IND-RCCA:** We proceed in Games. We assume that the adversary $\mathcal{A}$ requests at most $e_{\max}$ epochs (via Next).

Game 0 is the standard UP-IND-RCCA game for probabilistic reencryption with the "arbitrary" restriction for reencryption. The challenge bit is $b_{\mathcal{B}} \in \{0, 1\}$.

In **Game 1**, the challenge reencryptions are replaced by (fresh) encryptions of m_0^*. More concretely, $\mathcal{B}$ picks $r \leftarrow_{\mathrm{R}} \{0, \dots, e_{\max}\}$ and hopes that r has the property:

- $\mathcal{A}$ corrupts all tokens Δ_j for $j = e^* + 1, \dots, r$.
- $\mathcal{A}$ does not corrupt Δ_{r+1} (or it does not exist, in case of $\Delta_{e_{\max}+1}$).

where e^* is the challenge epoch and $c^*_{e^*} \leftarrow \mathsf{Enc}(k_{e^*}, m^*_{b_{\mathcal{B}}}; r^*)$ the challenge ciphertext. Reencryptions $\mathsf{ReEnc}(c, i)$ in case of $\mathsf{isChallenge}(c, i) = 1$ are computed as follows: $\mathsf{ReEnc}(c, i)$ computes c_j for $j = i + 1, \dots, e_{\mathrm{cur}}$ as

- if $j \leq r$, $c_j = \mathsf{ReEnc}(\Delta_j, c_{j-1})$.
- if $j \geq r + 1$, $c_j = \mathsf{Enc}(k_j, m_0^*)$. That is, the reencryption is replaced by a fresh encryption of m_0^*.[14]

If $\mathsf{isChallenge}(c, i) = 0$, there are no changes to reencryption.

We prove indistinguishability of Game 0 and Game 1 by a hybrid argument. Game H_ℓ is defined as Game 1, but the (definition of the) region $\{e^*, \dots, r\}$ is replaced by $\{\ell, \dots, r\}$. (Compare to Thm. 7.) Clearly, H_{e^*} is Game 1, and Game $\mathsf{H}_{e_{\max}+1}$ is Game 0. Thus, we have to prove that distinguishing Game $\mathsf{H}_{\ell-1}$ and Game H_ℓ yields a (multi-)IND-RCCA adversary. We do this by using the same key-insulation technique as in Thm. 1. We delay the proof to Lemma 9.

In **Game 2**, we replace the encryption of the challenge ciphertext by an encryption of m_0^*. Again, we use key-insulation to embed an IND-RCCA challenger, analogous to the IND-RCCA embedding in the hybrid games.

This finishes the proof, as Game 2 contains no information about the challenge bit anymore. Overall, we reduce to the underlying (multi-)IND-RCCA security with a loss of (roughly) $\frac{1}{(e_{\max}+1)^2}$. One factor of $\frac{1}{e_{\max}+1}$ is incurred by carrying out a hybrid argument, and another factor of $\frac{1}{e_{\max}+1}$ is incurred by indistinguishability of the hybrids, c.f. Lemma 9, which requires guessing.

UP-INT-PTXT: To prove UP-INT-PTXT from INT-PTXT, the same strategy works. In fact, the proof of Thm. 1 is easily adapted: The region $\{\ell, \dots, r\}$ we guess has the same properties, i.e. the UP-INT-PTXT adversary will forge a ciphertext and corrupt every token in this region. We use key insulation and

[14] Doing this replacement only for the first time is enough. Repeating the replacement (i.e. freshly encrypting m_0^*) or honestly reencrypting is perfectly indistinguishable (due to perfect reencryption). Depending on the situation, either action is preferable.

embed the INT-PTXT challenger in the insulated region. Note that there are no special cases for Enc, Dec or ReEnc since there is no challenge message and INT-PTXT will never reject a decryption (or encryption) query. Thus we obtain an adversary against UP-INT-PTXT from an adversary against INT-PTXT. We suffer a security loss of (roughly) $\frac{1}{(e_{\max}+1)^2}$.

Lemma 9. *The hybrid games $\mathsf{H}_{\ell-1}$ and H_ℓ are indistinguishable. Concretely,a distinguisher with advantage ε yields a multi-IND-RCCA adversary with advantage at least $\frac{\varepsilon}{e_{\max}+1}$, where $e_{\max}$ is the final epoch generated via* Next.

Proof. Suppose an adversary $\mathcal{A}$ distinguishes hybrid games Game $\mathsf{H}_{\ell-1}$ and Game H_ℓ (for $0 \leq \ell \leq e_{\max}+1$). We construct an adversary $\mathcal{B}$ against IND-RCCA from $\mathcal{A}$ with polynomial loss in advantage (namely $\frac{1}{e_{\max}+1}$). Let $\mathcal{C}$ be the IND-RCCA adversary.

The reduction is as follows:[15] $\mathcal{B}$ guesses $r \leftarrow_{\mathrm{R}} \{\ell, \dots, e_{\max}\}$ with the property that $\mathcal{A}$ corrupts tokens $\Delta_{\ell+1}, \dots, \Delta_r$, but does *not* corrupt Δ_{r+1}. For simplicity, $\mathcal{B}$ aborts, i.e. returns a random $b \leftarrow_{\mathrm{R}} \{0,1\}$ to $\mathcal{C}$, if the guess is wrong. In the following, we assume correct guesses. We distinguish following cases.

Case 1. $\mathcal{A}$ corrupts Δ_ℓ. Then Game $\mathsf{H}_{\ell-1}$ and Game H_ℓ are identical.
Case 2. $\mathcal{A}$ corrupts a key in epochs $\{\ell, \dots, r\}$. Then $\mathcal{A}$ must not request a reencryption of challenge ciphertexts into epochs $\{\ell, \dots, r\}$. If it does, the game is aborted (and $\mathcal{A}$ loses). Again, Game $\mathsf{H}_{\ell-1}$ and H_ℓ behave identical.
Case 3. $\mathcal{A}$ corrupts neither Δ_ℓ nor any key in the range $\{\ell, \dots, r\}$.

Cases 1 and 2 are trivial in the sense that $\mathcal{B}$ simply returns $b_{\mathcal{B}} \leftarrow_{\mathrm{R}} \{0,1\}$ as its guess. ($\mathcal{A}$ is of no use after all.) Thus we concentrate on Case 3. In Case 3, Game $\mathsf{H}_{\ell-1}$ replaces reencryptions of challenge ciphertexts by encryptions of m_0^* for any epoch $e \geq \ell$. Thus, the only difference w.r.t. Game H_ℓ are reencryptions of challenge ciphertexts in the region $\{\ell, \dots, r\}$, which are honest in Game H_ℓ, but not in Game $\mathsf{H}_{\ell-1}$.

We use key-insulation on the region $\{\ell, \dots, r\}$ and modify $\mathcal{B}$ accordingly. Then we embed the IND-RCCA challenger $\mathcal{C}$ into the insulated region as follows.

We recall the changes introduced by key-insulation (Thm. 1 and App. B), now with *leakage* $\mathsf{leak}(k) = pk$. These changes are perfectly indistinguishable due to perfect reencryption and simulatable token generation.

- Adversary $\mathcal{B}$
  - does not generate keys $k_{\ell+1}, \dots, k_r$,
  - does not generate tokens Δ_ℓ and Δ_{r+1},
  - simulates tokens $\Delta_{\ell+1}, \dots, \Delta_r$ along with reverse tokens $\Delta'_{\ell+1}, \dots, \Delta'_r$ using $(\Delta_i, \Delta_{i+1}, pk_{i+1}) \leftarrow_{\mathrm{R}} \mathsf{SimTok}(pk_\ell)$ where $pk_\ell = \mathsf{leak}(k_\ell)$.
- An Enc(m) call in epoch
  - $\ell < e_{\mathrm{cur}} \leq r$ is handled as by first computing $c_\ell \leftarrow_{\mathrm{R}} \mathsf{UE.Enc}(k_\ell, m)$ and then $c_{e_{\mathrm{cur}}}$ via $c_j \leftarrow \mathsf{UE.ReEnc}(\Delta_j, c_{j-1})$ for $\ell + 1 \leq j \leq e_{\mathrm{cur}}$.

[15] Again, this reduction is essentially the same as in Thm. 1, but slightly simpler since we have perfect reencryption.

  - Other epochs e_{cur} are handled without change.[16]
- A $\mathsf{Dec}(c)$ call in epoch $\ell < e_{\text{cur}} \leq r$ is handled by first computing c_ℓ via $c_{j-1} \leftarrow \mathsf{UE.ReEnc}(\Delta'_j, c_j)$ for $e_{\text{cur}} \geq j > \ell$ and then $m \leftarrow \mathsf{UE.Dec}(k_\ell, c_\ell)$. Otherwise, $\mathcal{B}$ computes Dec without change.
- A $\mathsf{ReEnc}(c_i, i)$ call in epoch $e_{\text{cur}} > i$ is handled by computing a series of ciphertexts c_j for $i < j \leq e_{\text{cur}}$, where
  - Except computing c_ℓ and c_{r+1} reencryption uses the tokens as usual.
  - c_ℓ is computed from $c_{\ell-1}$ by decrypting $c_{\ell-1}$ and freshly encrypting under epoch ℓ.
  - c_{r+1} is computed from c_r by decrypting c_r, using the changed decryptiong procedure described above, and freshly encrypting under epoch $r+1$.
- For $\mathsf{isChallenge}(c, i)$, we return 0 if $c \in \mathbf{Q}$. Otherwise, we decrypt c under epoch i to m, perhaps using the modified decryption procedure above, and return 1 iff $m \in \mathrm{M}^*$.

After insulating the region $\{\ell, \dots, r\}$ as above, we can embed the IND-RCCA challenger $\mathcal{C}$ in epoch ℓ. Oracle queries for $\mathsf{ReEnc}(c, i)$ with $\mathsf{isChallenge}(c, i) = 0$, as well as Enc and Dec, in the insulated region (i.e. for $\ell \leq e_{\text{cur}} \leq r$) are handled as described. Note that blackbox-access to encryption and decryption in epoch ℓ, plus leakage pk_ℓ are sufficient for key-insulation. Thus, the IND-RCCA challenger $\mathcal{C}$ gives $\mathcal{B}$ the necessary oracles (plus leakage).

Note that checking $\mathsf{isChallenge}(c, i)$ with $\ell < i \leq r$, boils down to using the decryption oracle $\mathcal{C}.\mathsf{Dec}$ (after rotating the ciphertext to epoch ℓ). By definition of the hybrid games, $\mathcal{C}.\mathsf{Dec}(c)$ responds with `invalid` if and only if c encrypts a message in $\mathrm{M}^* = \{m_0^*, m_1^*\}$. (We note that the additional “generosity” of allowing reencryptions of *queried* ciphertexts, even if they contain a challenge message, is preserved.)

Challenge ciphertext reencryptions are treated differently, depending on the epochs. We describe how a single hop from $j-1$ to j is implemented. So consider a challenge ciphertext c_{j-1}^* under epoch $j-1$ (with $\mathsf{isChallenge}(c_j^*, j-1) = 1$).

- $j \neq \ell$ and $j \neq r+1$: The update token Δ_j is used.
- $j = \ell$: Decrypt $m := \mathsf{Dec}(k_{\ell-1}, c^*) \in \mathcal{M}$. Instead of freshly encrypting with $\mathcal{C}.\mathsf{Enc}$, $\mathcal{B}$ sends a challenge message pair (m, m_0^*) to $\mathcal{C}$ and receives a ciphertext c_ℓ^*.
- $j = r+1$: Encrypt $c_{r+1}^* = \mathsf{Enc}(k_{r+1}, m_0^*)$ in compliance with both hybrid games.[17]

Note that we rely on multi-IND-RCCA. Otherwise, we could only replace one reencryption from $\ell - 1$ to ℓ.

By close inspection, one finds that $\mathcal{A}$ plays game Game $\mathsf{H}_{\ell-b}$. That is, if $\mathcal{C}$ picks $b = 0$, then $\mathcal{A}$ plays a game which is identical to Game H_ℓ, otherwise it is identical to Game $\mathsf{H}_{\ell-1}$.

[16] Here *perfect* reencryption spares us the “get randomness from epoch ℓ”, which was necessary for randomness-preserving reencryption.

[17] Both hybrid games replace challenge reencryptions by encryption of m_0^* for epochs after ℓ.

Thus, $\mathcal{B}$ essentially returns $\mathcal{A}$'s guess $b_{\mathcal{A}}$ as its own. The distinguishing advantage $\varepsilon_{\mathcal{A}}$ of $\mathcal{A}$ translates to a multi-IND-RCCA adversary with advantage $\varepsilon_{\mathcal{B}} \geq \frac{1}{e_{\max}+1}\varepsilon_{\mathcal{A}}$.

Remark 4. Another, maybe more direct proof strategy for UP-IND-RCCA, along the lines of the UP-INT-PTXT proof, is also possible. In Game 1, replace all reencryptions by decrypt-then-encrypt. In Game 2, guess ℓ and r with $\ell \leq e^* \leq r$. Use it to embed the IND-RCCA challenger $\mathcal{C}$ into ℓ as usual. And replace all challenge reencryptions by fresh challenge encryptions, i.e. $\mathcal{C}.\mathsf{EncLR}(m_0^*, m_1^*)$ calls. The security loss is essentially the same as for the proof we presented.

H Difficulties of updatable reencryption and open problems

In this section, we explain some difficulties and obstacles where solutions (different from the ones we presented) remain as major open problems. Note that we exclusively consider *bidirectional* schemes.

Arbitrary reencryption. Consider an updatable encryption scheme ReEnc in a *group setting*. Since (generic) groups essentially offer *linear* operations, and since re/de/encryption algorithms typically use group operations in a blackbox manner, let us assume that $m \mapsto \mathsf{Enc}(sk, m; r)$, $c \mapsto \mathsf{Dec}(sk, c)$, $c \mapsto \mathsf{ReEnc}(\Delta, c; r)$ are *essentially linear/affine* maps.[18] By essentially linear, we mean that if the result is not $\perp$, i.e. if all consistency checks pass, then the map is linear. Deterministic linear reencryption satisfies (for arbitrary c_1, c_2, Δ)

$$\mathsf{ReEnc}(\Delta, c_1 + c_2) = \mathsf{ReEnc}(\Delta, c_1) + \mathsf{ReEnc}(\Delta, c_2).$$

After applying $c \mapsto \mathsf{Dec}(k, c)$ to both sides, this equality can also hold for probabilistic schemes, for example RISE. This "unchecked" linearity of reencryption allows a trivial attack if arbitrary reencryptions are allowed: Split c^* randomly into $c^* = c_1 + c_2$. Query reencryptions c_1', c_2' of c_1, c_2 to an uncompromised key. Ask for a decryption of $c_1' + c_2'$. This shows that non-linearity, which gives some kind of "integrity" or "non-malleability", is essential for security under arbitrary re-encryption.

In NYUE we achieve this "non-linearity" by making ciphertext consistency publicly verifiable. Another way to artificially prevent this attack, for example by restricting to reencryptions to (honest previously) queried ciphertexts.

(Probabilistic) RCCA-secure encryption without public verifiability. The solution of NYUE, to enforce public verifiability of consistency of ciphertexts, is not evidently necessary. After all updatable encryption is secret key, not public key, primitive. Thus, hash proof systems and designated verifier NIZKs with *suitable malleability* should also work. It is likely that the decrypt-then-encrypt step in our proofs has to be replaced by some other (perhaps entirely different) argument for such schemes. Finding such schemes is an interesting open problem.

[18] Pairing groups break this assumption.

Simulatable token generation implies randomisation. For probabilistic schemes, simulatable token generation essentially implies randomisability of the underlying encryption scheme. Simply define $\mathsf{ReRand}(c)$ as $\mathsf{ReEnc}(\Delta', \mathsf{ReEnc}(\Delta, c))$ for $(\Delta, \Delta') \leftarrow_{\mathrm{R}} \mathsf{SimTok}(sp)$.

Probabilistic CCA-secure updatable encryption. The implied randomisation from simulatable token generation is an obstruction to CCA-security. Evidently, "bidirectional" reencryption allows rerandomisation via back-and-forth reencryption. Hence, CCA-secure probabilistic reencryption must not allow ciphertext downgrades. It is not clear if CCA-security already enforces unidirectional security. Nevertheless, CCA-secure probabilistic updatable encryption should be "close" to being unidirectional.

Computational indistinguishability. Most of our properties are perfect (or statistical). Finding secure schemes where these properties are only *computational* may offer significant efficiency improvements and insights about the nature of these properties and their relation to the security proofs.

I Conjectured application to adaptive security of unidirectional proxy-reencryption

We conjecture, that our approach to security proofs, namely key-insulation, also works for unidirectional proxy re-encryption, at least in the setting where only queried reencryptions are allowed. (Consequently, we expect it to work for *unidirectional* updatable encryption in the same setting.) Note that the stronger security definition of *honest reencryption attacks* (HRA, c.f. [Coh17; Fuc+18]) for proxy reencryption is essentially the security for (unidirectional) updatable encryption defined in [LT18]. Namely, queried (re)encryptions only and no decryption oracle. Recalling the definitions of proxy reencryptions, let alone the many flavours in [Fuc+18] here is too much and not our goal. We only note we are somewhat confident, that our techniques are applicable in that setting, and improve significantly over those in [Fuc+18] (for a certain class of schemes). Since the topic is delicate and prone to mistakes, we do not claim correctness, as we do not present formal proofs of the intuition below.

I.1 Remarks on proxy-reencryption and [Fuc+18]

Typically, security for proxy reencryption is only shown selectively secure, i.e. the adversary can corrupt in an initial Phase 1. In the second Phase 2, no corruption oracle is available, but instead the adversary now has (re)encryption and challenge oracles. In [Fuc+18], the approach of "pebbling" from [Jaf+17], is used to obtain tighter, non-trivial security loss for *adaptively secure* unidirectional proxy reencryption. Their security loss depends on the allowed (directed) "graph of reencryptions" (called "recoding graph"), i.e. they restrict the available reencryption tokens to a (family of) graphs (with say n nodes). Their proven security loss is superpolynomial, namely at least $n^{O(\log(n))}$, even in very simple cases, such as a "chain" of reencryptions. This "chain graph" essentially is the setting of unidirectional updatable encryption. Moreover, for general graphs, their loss is still exponential.

The schemes considered in [Jaf+17] are *unidirectional* schemes with special properties. Very roughly, these properties are:

IND-CPA: The underlying encryption scheme is IND-CPA secure.
Weak key-privacy for reencryption tokens: A reencryption token can be simulated given only the target public key. (Note that in the *unidirectional* (proxy reencryption) setting, we essentially have $\Delta_{i\to j} \leftarrow_{\mathrm{R}} \mathsf{GenTok}(sk_i, pk_j)$.)
Source-hiding: Reencryptions and fresh encryption are indistinguishable. (This is essentially what we call perfect reencryption.)

These properties are very close to our properties (although our setting is a priori neither public key nor unidirectional). The major difference is weak key-privacy. This difference is mostly caused by *unidirectionality*, which essentially forces a change in the simulation of tokens. Namely, only *outbound*[19] reencryption tokens can be simulated (without secret key), but only those need to be simulated. Due to the unidirectional setting, one can always create tokens $\Delta_{i\to j}$ if sk_i is known (and pk_j), so this poses no problems.

Unidirectionality causes a change in the arguments, but seems to simplify rather than complicate. The basic idea to achieve adaptive security is exactly as for updatable encryption.

– Establish a *key-insulation* technique. In fact, in this setting key-insulation truly insulates a single key, thanks to unidirectionality. (Unsurprisingly, weak key-privacy and source-hiding is used here.)
– Define suitable hybrids, which only differ in places which can be key-insulated.
– Show the hybrids are indistinguishable by using IND-CPA security of the underlying encryption scheme.

Due to unidirectionality, hybrids and key-insulation are greatly simplified.

Hybrids. For a reencryption graph of size n, we define hybrid games $\mathsf{H}_0, \dots \mathsf{H}_{n+1}$ as follows: In Game H_i

– all challenge encryptions[20] under keys pk_j for $j \geq i$ encrypt m_0 (independent of b).
– all *inbound* challenge re-encryptions to pk_j for $j \geq i$ are replaced by (fresh) encryptions of m_0.

[19] By *outbound*, we mean token from (challenge) keys to other keys. That is $\Delta_{i\to j}$ is outbound for key i and inbound for key j.
[20] There is only a single challenge reencryption in the standard security games. But for multi-ciphertext games, this is how it must be defined.

Hybrid indistinguishability. Evidently, hybrid H_i and H_{i+1} differ only in their actions on pk_i. Thus, it is sufficient to consider this key. There are two cases:

- pk_i is not challenge-related.
- pk_i is challenge-related (i.e. a challenge (re)encryption is made there).

In the first case, Game H_i and Game H_{i+1} are identical. Thus, we focus on the second case. In this case, we want to embed a IND-CPA challenger (of the underlying encryption scheme). However, since we claimed to replace *all* (inbound re-)encryption of the challenge at once, we actually assume multi-IND-CPA (to spare us another hybrid argument). Clearly, to embed an IND-CPA challenger, we need to key-insulate first (because we cannot generate outbound tokens without knowledge of sk_i). As in the blueprint for updatable encryption, we separate this insulation into an argument of its own.

Key-insulation of key sk_i. Key insulation in this setting is very straightforward. A sketch is provided in Fig. 11. First, one uses source-hiding (i.e. perfect) reencryption, to compute all outbound reencryption requests (i.e. from i to j) by decrypt-then-encrypt.[21] Thus, we do not need any tokens $\Delta_{i\to j}$ anymore, and essentially only Enc calls are used for pk_i.

Second, one uses weak key-privacy to simulate all outbound tokens $\Delta_{i\to j}$ via $\Delta_{i\to j} \leftarrow_{\mathrm{R}} \mathsf{SimTok}(pk_i, sk_j)$. W.l.o.g. $\mathcal{A}$ never queries sk_i. (Because we assumed that pk_i is challenge-related, querying sk_i implies that $\mathcal{A}$ loses the game). Thus, embedding weak key-privacy is straightforward.

Now, we do not need sk_i anymore and only use encryption for pk_i (so no reencryption either).[22] (Because (fresh) encryption only needs pk_i, and tokens are simulated.)

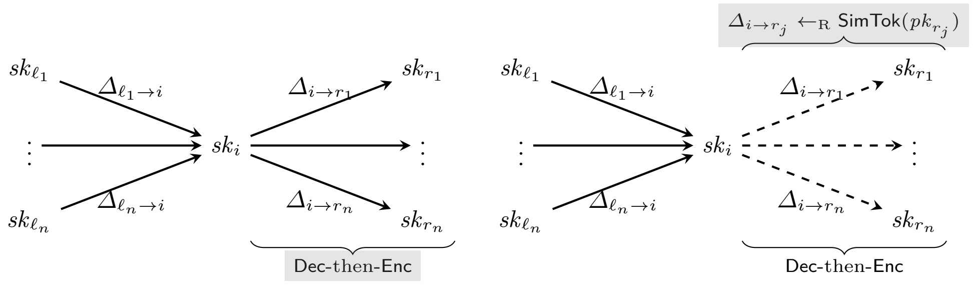

Fig. 11. A sketch of key-insulation. Changes are highlighted in grey.

[21] "Decrypt" is simply a table lookup since only queried reencryption is allowed.
[22] By forwarding decryption queries to the challenger, RCCA and CCA security may also be provably secure using these techniques.

Estimated security loss. We used a polynomial number of key-privacy, source-hiding and multi-IND-CPA embeddings. Namely, at most n^2 for an n-vertex graph (i.e. at most one for each edge), distributed over all hybrid games. Hence we estimate the security loss to be roughly $O(n^2(\varepsilon_{\text{key-anon}} + \varepsilon_{\text{source-hiding}}) + nQ\varepsilon_{\text{IND-CPA}})$, where Q is a bound on the number of challenge (re)encryption queries. This would improve upon [Fuc+18] as the bound holds for an *arbitrary* graph of allowed reencryptions, and is polynomial (instead of superpolynomial.)

Note again that this is not a formal proof, and the devil is in the details. We do not claim that this sketch works without change (or at all).

Finally, we expect that using (rerandomisable) zero-knowledge succinct non-interactive proofs of knowledge (zk-SNARK), the same Naor-Yung technique used in NYUE should yield RCCA secure unidirectional PRE (and updatable encryption as well). (Unfortunately, all known SNARKs rely on non-falsifiable assumptions, see also [GW11]. But SNARKs are not a necessity. Weaker primitives may already allow malleable proofs, as seen with Groth–Sahai proofs in the bidirectional setting.)