(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. [1] 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 [21], and several cloud storage providers, such as Google and Amazon, offer data-at-rest encryption with rotatable keys [8].

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. [2] 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. [8] 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. [1] and recently Lehmann and Tackmann [17] 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 [17].

	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 [17] 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 [8] 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	$\|c\|$	(Re)Enc	Dec
BLMR [2]	DDH (+ ROM)	✗	✗	(?)	✗	(?)	$2\mathbb{G}^*$	$2\mathbb{G}$	$2\mathbb{G}$
EPRS [8]	DDH + ROM	✗	(✗)	CPA	CTXT	✓	$2\mathbb{G}^*$	$2\mathbb{G}$	$2\mathbb{G}$
RISE [17]	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 [8], 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 [8], 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 [20] 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 [17] and GS proofs [12, 5] 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 [3]. 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 [8, 17]. 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 [8, 17]. 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. [14] 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. [10] 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 [10] 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. [16]. This would improve upon the superpolynomial loss in [10].

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 [8, 17] we refer the formal treatment of this notion to [16].

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 [6]. 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 [18].

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*

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

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

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

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 [3] 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 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

[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 $\mathcal{A}^{\mathsf{Enc},\mathsf{Dec}}(state, c^*) \overset{?}{=} b$
```

```
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 $\mathcal{A}^{\mathsf{Enc},\mathsf{Dec}}(state, c^*) \overset{?}{=} b$
```

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.

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* $\mathsf{leak}\colon \mathcal{K} \to \mathcal{L}$ *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 [17], 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 [16] 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).

	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 $0 \leq i \leq e$			

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.

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 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 $\mathsf{isChallenge}$, 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 $0 \leq i < e$ for re-encryption queries, and $0 \leq i \leq e$ 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`.

$\mathsf{Next}()$: 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 $e \leftarrow e + 1$.

$\mathsf{Enc}(m)$: Returns $c \leftarrow_{\mathrm{R}} \mathsf{UE.Enc}(k_e, m)$ and sets $\mathbf{Q} \leftarrow \mathbf{Q} \cup \{(e, c)\}$.

$\mathsf{Dec}(c)$: If $\mathsf{isChallenge}(k_e, c) = \mathtt{false}$, it returns $m \leftarrow \mathsf{UE.Dec}(k_e, c)$.

$\mathsf{ReEnc}(i, c)$: 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 $c^* \leftarrow_R \mathsf{UE.Enc}(k_e, m_b)$ 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 $\mathsf{Exp}^{\mathsf{up\text{-}ind\text{-}cca}}_{\mathsf{UE},\mathcal{A}}(pp, b)$
    $(sp, k_0, \Delta_0, \mathbf{Q}, \mathbf{K}, \mathbf{T}, \mathbf{C}^*) \leftarrow \mathsf{Init}(pp)$
    $(m_0, m_1, state) \leftarrow_R \mathcal{A}^{\mathsf{Enc},\mathsf{Dec},\mathsf{Next},\mathsf{ReEnc},\mathsf{Corrupt}}(sp)$
    proceed only if $|m_0| = |m_1|$ and $m_0, m_1 \in \mathcal{M}_{sp}$
    $c^* \leftarrow_R \mathsf{UE.Enc}(k_e, m_b)$, $\mathbf{C}^* \leftarrow \{e\}$, $e^* \leftarrow e$
    $b' \leftarrow_R \mathcal{A}^{\mathsf{Enc},\mathsf{Dec},\mathsf{Next},\mathsf{ReEnc},\mathsf{Corrupt}}(c^*, state)$
    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. [8].

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

We use the notation from [17] 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

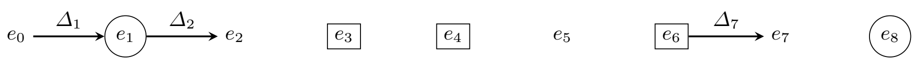

Fig. 6. Example of corrupted tokens, keys (boxed) and challenge-equal epochs (circled) in a UP-IND-CCA game. Corrupting Δ_3 and Δ_8 is forbidden, as they would allow to re-encrypt the challenge ciphertext into an epoch where $\mathcal{A}$ knows the secret key.

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: [17] 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 challenge-equal:

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

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.

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

[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.

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 [8] 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. ***Strong*** *IND-X, IND\$-X and INT-Y notions are defined as sketched above. (See also [16].)*

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 $k^{\text{old}}, k^{\text{new}} \in \mathcal{K}$ with $\Delta \in \operatorname{supp}(\mathsf{UE}.\mathsf{GenTok}(k^{\text{old}}, k^{\text{new}}))$ we have $\Delta' \in \operatorname{supp}(\mathsf{UE}.\mathsf{GenTok}(k^{\text{new}}, k^{\text{old}}))$.*

[5] The CTXT definition in the proceedings version of their paper did not have such a restriction, however the revised ePrint version [7] later showed that the original notion is not achievable and a weaker CTXT definition is introduced instead.

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)* $k^{\text{old}} \leftarrow \mathsf{UE.GenKey}(sp)$ *following distributions of* Δ *are identical: The distribution of* Δ

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

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 $c = \mathsf{UE.Enc}(k^{\text{old}}, m; r)$ 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* $k^{\text{old}}, k^{\text{new}} \leftarrow_R \mathsf{UE.GenKey}(sp)$*, tokens* $\Delta \leftarrow_R \mathsf{UE.GenTok}(k^{\text{old}}, k^{\text{new}})$*, and all valid ciphertexts* c *under* k^{old}*, we have*

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

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

In [16], we argue that this randomness-preserving property additionally guarantees unlinkability of re-encrypted ciphertexts (UP-REENC security) as considered by prior work [17, 8].

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 [16].

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 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

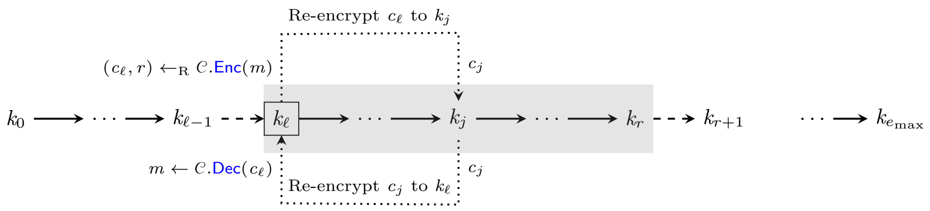
Re-encrypt c_ℓ to k_j
$(c_\ell, r) \leftarrow_R \mathcal{C}.\mathsf{Enc}(m)$
c_j
k_0
$k_{\ell-1}$
k_ℓ
k_j
k_r
k_{r+1}
$k_{e_{\max}}$
$m \leftarrow \mathcal{C}.\mathsf{Dec}(c_\ell)$
c_j
Re-encrypt c_j to k_ℓ

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.

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 $\{\ell, \dots, r\}$ (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.

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 $\Delta_{\ell+1}, \dots, \Delta_{r+1}$ 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

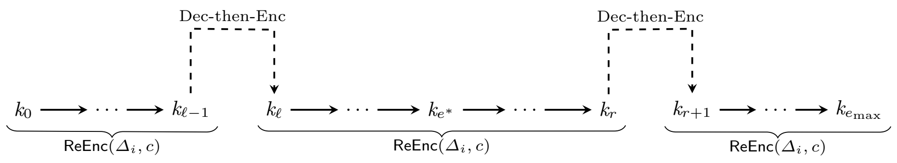
Dec-then-Enc
Dec-then-Enc
$k_0 \longrightarrow \cdots \longrightarrow k_{\ell-1}$
$k_\ell \longrightarrow \cdots \longrightarrow k_{e^*} \longrightarrow \cdots \longrightarrow k_r$
$k_{r+1} \longrightarrow \cdots \longrightarrow k_{e_{\max}}$
$\mathsf{ReEnc}(\Delta_i, c)$
$\mathsf{ReEnc}(\Delta_i, c)$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

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.

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 [18] 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 ([16]) and randomness-preserving reencryption (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 [16]) 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 [16] (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. [16], 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. $\mathsf{leak}(k) = \bot$.

$\mathsf{PRF}_{\mathsf{DDH}}$. 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 [19, 2]. 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 [19], 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:* $\mathsf{Adv}^{\textit{up-ind-rcca}}_{\mathsf{UE},\mathcal{A}}(pp) := \left|\Pr[\mathsf{Exp}^{\textit{up-ind-rcca}}_{\mathsf{UE},\mathcal{A}}(pp, 0) = 1] - \Pr[\mathsf{Exp}^{\textit{up-ind-rcca}}_{\mathsf{UE},\mathcal{A}}(pp, 1) = 1]\right|$.

```
Experiment $\mathsf{Exp}^{\text{up-ind-rcca}}_{\mathsf{UE},\mathcal{A}}(pp, b)$
    $(sp, k_0, \Delta_0, \mathbf{Q}, \mathbf{K}, \mathbf{T}, \mathbf{C}^*) \leftarrow \mathsf{Init}(pp)$
    $(m_0, m_1, state) \leftarrow_{\mathrm{R}} \mathcal{A}^{\mathsf{Enc},\mathsf{Dec},\mathsf{Next},\mathsf{ReEnc},\mathsf{Corrupt}}(sp)$
    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 $\mathtt{true}$. Else, return $\mathtt{false}$.

Whereas the decryption oracle will ignore any query where $\mathsf{isChallenge}(k_e, c) = \mathtt{true}$, the re-encryption oracle is more generous: When ReEnc is invoked on (i, c) where $\mathsf{isChallenge}(k_i, c) = \mathtt{true}$, 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 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:* $\mathsf{Adv}^{\textit{up-int-ptxt}}_{\mathsf{UE},\mathcal{A}}(pp) := \Pr[\mathsf{Exp}^{\textit{up-int-ptxt}}_{\mathsf{UE},\mathcal{A}}(pp) = 1]$.

```
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 [20] (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 [17], 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* [12, 5]. 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 [16] 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 [16] or [4, 9] 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 $\mathsf{Enc}(k, \perp) = \perp$ *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[20] 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[12]. 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 [22, 11]) 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 [13]. 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* $\mathsf{NYUE.GenKey}$*. 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. [16].

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 $\mathsf{RISE.ReEnc}$, separated into key-rotation and re-randomisation, c.f. [16].

(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 $\mathsf{NYUE.ReEnc}$ first verify the consistency proof of a ciphertext before starting the update procedure. Thus, re-encryption only works for well-formed, *decryptable* ciphertexts.

[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.

(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 [16].

Lemma 4. *The updatable encryption scheme* NYUE *has perfect re-encryption and simulatable token generation under leakage* $\mathsf{leak}(k) = pk$*, c.f. [16].*

Lemma 4 follows easily from token simulation for RISE, see [16]. 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 [16]. There are three major differences compared to UP-IND-CCA:

First, $\mathsf{NYUE.ReEnc}$ uses the public verifiability of ciphertexts to reject invalid inputs, i.e., it updates only ciphertexts for which $\mathsf{NYUE.Dec}$ 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 ([16]). *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 [15] 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 [15] 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 [16]. 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 [15, Fig. 2] for USIG with simulatable token generation and malleable signature verification. In [16] 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. [16]) 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* $\mathsf{NYUE.ReEnc}$ *(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 [16]. The proof of security as an updatable encryption scheme follows the usual blueprint. As for NYUE, UP-REENC security follows from [16].

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

[1] Dan Boneh, Kevin Lewi, Hart William Montgomery, and Ananth Raghunathan. "Key Homomorphic PRFs and Their Applications". In: *CRYPTO 2013, Part I*. Vol. 8042. Aug. 2013.

[2] 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.

[3] Ran Canetti, Hugo Krawczyk, and Jesper Buus Nielsen. "Relaxing Chosen-Ciphertext Security". In: *CRYPTO 2003*. Vol. 2729. Aug. 2003.

[4] 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.

[5] Alex Escala and Jens Groth. "Fine-Tuning Groth-Sahai Proofs". In: *PKC 2014*. Vol. 8383. Mar. 2014.

[6] Alex Escala, Gottfried Herold, Eike Kiltz, Carla Ràfols, and Jorge Villar. "An Algebraic Framework for Diffie-Hellman Assumptions". In: *CRYPTO 2013, Part II*. Vol. 8043. Aug. 2013.

[7] 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.

[8] Adam Everspaugh, Kenneth G. Paterson, Thomas Ristenpart, and Samuel Scott. “Key Rotation for Authenticated Encryption”. In: *CRYPTO 2017, Part III*. Vol. 10403. Aug. 2017.

[9] Georg Fuchsbauer. “Commuting Signatures and Verifiable Encryption”. In: *EUROCRYPT 2011*. Vol. 6632. May 2011.

[10] 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.

[11] Jens Groth. “Simulation-Sound NIZK Proofs for a Practical Language and Constant Size Group Signatures”. In: *ASIACRYPT 2006*. Vol. 4284. Dec. 2006.

[12] Jens Groth and Amit Sahai. “Efficient Noninteractive Proof Systems for Bilinear Groups”. In: *SIAM J. Comput.* 41.5 (2012).

[13] Dennis Hofheinz and Tibor Jager. “Tightly Secure Signatures and Public-Key Encryption”. In: *CRYPTO 2012*. Vol. 7417. Aug. 2012.

[14] 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.

[15] Eike Kiltz, Jiaxin Pan, and Hoeteck Wee. “Structure-Preserving Signatures from Standard Assumptions, Revisited”. In: *CRYPTO 2015, Part II*. Vol. 9216. Aug. 2015.

[16] Michael Klooß, Anja Lehmann, and Andy Rupp. *(R)CCA Secure Updatable Encryption with Integrity Protection*. IACR ePrint 2019/222. URL: `http://eprint.iacr.org/2019/222`.

[17] Anja Lehmann and Björn Tackmann. “Updatable Encryption with Post-Compromise Security”. In: *EUROCRYPT 2018, Part III*. Vol. 10822. 2018.

[18] Chanathip Namprempre, Phillip Rogaway, and Thomas Shrimpton. “Reconsidering Generic Composition”. In: *EUROCRYPT 2014*. Vol. 8441. May 2014.

[19] Moni Naor, Benny Pinkas, and Omer Reingold. “Distributed Pseudo-random Functions and KDCs”. In: *EUROCRYPT’99*. Vol. 1592. May 1999.

[20] Moni Naor and Moti Yung. “Public-key Cryptosystems Provably Secure against Chosen Ciphertext Attacks”. In: *22nd ACM STOC*. May 1990.

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

[22] Amit Sahai. “Non-Malleable Non-Interactive Zero Knowledge and Adaptive Chosen-Ciphertext Security”. In: *40th FOCS*. Oct. 1999.