Fast and Secure Updatable Encryption

Colin Boyd[1], Gareth T. Davies[2][0000−0002−5935−5725], Kristian Gjøsteen[1], and Yao Jiang[1]

[1] Norwegian University of Science and Technology, NTNU
[2] Bergische Universität Wuppertal

Abstract. Updatable encryption allows a client to outsource ciphertexts to some untrusted server and periodically rotate the encryption key. The server can update ciphertexts from an old key to a new key with the help of an update token, received from the client, which should not reveal anything about plaintexts to an adversary.
We provide a new and highly efficient suite of updatable encryption schemes that we collectively call SHINE. In the variant designed for short messages, ciphertext generation consists of applying one permutation and one exponentiation (per message block), while updating ciphertexts requires just one exponentiation. Variants for longer messages provide much stronger security guarantees than prior work that has comparable efficiency. We present a new confidentiality notion for updatable encryption schemes that implies prior notions. We prove that SHINE is secure under our new confidentiality definition while also providing ciphertext integrity.

1 Introduction

The past decades have demonstrated clearly that key compromise is a real threat for deployed systems. The standard technique for mitigating key compromise is to regularly *rotate* the encryption keys – generate new ones and switch the ciphertexts to encryption under the new keys. Key rotation is a well-established technique in applications such as payment cards [9] and cloud storage [16].

For a local drive or server, key rotation is feasible by decrypting and re-encrypting with a new key, since symmetric encryption operations are fast and parallelizable and bandwidth is often plentiful. When ciphertext storage has been outsourced to some (untrusted) cloud storage provider, bandwidth is often considerably more expensive than computation, and even for small volumes of data it may be prohibitively expensive to download, re-encrypt and upload the entire database even once. This means that key rotation by downloading, decrypting, re-encrypting and reuploading is practically infeasible.

An alternative approach to solving this problem is to use *updatable encryption* (UE), first defined by Boneh et al. [3] (henceforth BLMR). The user computes a *token* and sends it to the storage server. The token allows the server to update the ciphertexts so that they become encryptions under some new key. Although the token clearly depends on both the old and new encryption keys, knowledge of the token alone should not allow the server to obtain either key. In a typical usage of UE, the cloud storage provider will receive a new token on a periodic basis, and the provider then updates every stored ciphertext. The time period for which a given key is valid for is called an *epoch*.

In the past few years there has been considerable interest in extending the understanding of UE. A series of prominent papers [3,12,21,17] have provided both new (typically stronger) security definitions and concrete or generic constructions to meet their definitions. (We make a detailed comparison of related work in Section 1.1 next.) An important distinction between earlier schemes is whether or not the token (and in particular its size) depends on the ciphertexts to be updated (and in particular the number of ciphertexts). Schemes for which a token is assigned to each ciphertext are *ciphertext-dependent* and were studied by Everspaugh et al. [12] (henceforth EPRS). If the token is independent of the ciphertexts to be updated, such as in BLMR [4], we have a *ciphertext-independent*[3] scheme. A clear and important goal is to limit the bandwidth required and so, in general, one should prefer ciphertext-independent schemes. Thus, as with the most recent work [21,17], we focus on such schemes in this paper. The ciphertext update procedure, performed by the server, may be *deterministic* or *randomized* – note that in the latter case the server is burdened with producing (good) randomness and using it correctly.

Despite the considerable advances of the past few years, there remain some important open questions regarding basic properties of UE. In terms of security, various features have been added to protect against stronger adversaries. Yet it is not obvious what are the realistic and optimal security goals of UE and whether they have been achieved. In terms of efficiency, we only have a few concrete schemes to compare. As may be expected, schemes with stronger security are generally more expensive but it remains unclear whether this cost is necessary. In this paper we make contributions to both of these fundamental questions by defining **new and stronger security properties** and showing that these can be achieved with **more efficient concrete UE schemes**.

Security. The main security properties that one would expect from updatable encryption are by now well studied; however the breadth of information that is possible to protect in this context is more subtle than at first glance. Consider, for example, a journalist who stores a contact list with a cloud storage provider. At some point, the storage is compromised and an adversary recovers the ciphertexts. At this point, it may be important that the cryptography does not reveal which of the contacts are recent, and which are old. That is, it must be hard to decide if some ciphertext was recently created, or if it has been updated from a ciphertext stored in an earlier epoch.

So how do we define realistic adversaries in this environment? A natural first step for security in updatable encryption is *confidentiality* of ciphertexts –

[3] Note that Boneh et al. [[4], § Definition 7.6] use ciphertext-independence to mean that the updated ciphertext should have the same distribution as a fresh ciphertext (i.e. independent of the ciphertext in the previous epoch) – we follow the nomenclature of Lehmann and Tackmann [21].

given a single ciphertext, the adversarial server should not be able to determine anything about the underlying plaintext. The security model here must take into account that this adversary could be in possession of a number of prior keys or update tokens, and snapshot access to the storage database in different epochs. The next step is to consider *unlinkability* between different epochs arising from the ciphertext update procedure: given a ciphertext for the current epoch, the adversary should not be able to tell which ciphertext (that existed in the previous epoch) a current ciphertext was updated from. Both of these properties can be naturally extended to chosen-ciphertext (CCA) security via provision of a decryption oracle.

These steps have been taken by prior work, but unfortunately even a combination of these properties is not enough to defend against our motivating example. Previous security definitions cannot guarantee that the adversary is unable to distinguish between a ciphertext new in the current epoch and an updated ciphertext from an earlier epoch. We give a single new security property that captures this requirement and implies the notions given in prior work. Therefore we believe that this definition is *the natural confidentiality property* that is required for updatable encryption.

An additional factor to consider is integrity: the user should be confident that their ciphertexts have not been modified by the adversarial server. While prior work has shown how to define and achieve integrity in the context of updatable encryption, a composition result of the style given by Bellare and Namprempre for symmetric encryption [2] – the combination of CPA security and integrity of ciphertexts gives CCA security – has been missing. We close this gap.

Efficiency and Functionality. Although UE is by definition a form of symmetric key cryptography, techniques from asymmetric cryptography appear to be needed to achieve the required functionality in a sensible fashion. All of the previous known schemes with security proofs use exponentiation in both the encryption and update functions, even for those with limited security properties. Since a modern database may contain large numbers of files, efficiency is critical both for clients who will have to encrypt plaintexts initially and for servers who will have to update ciphertexts for all of their users.

To bridge the gap between the academic literature and deployments of encrypted outsourced storage, *it is crucial to design fast schemes.* We present three novel UE schemes that not only satisfy our strong security definitions (CCA and ciphertext integrity), but in the vast majority of application scenarios are also at least twice as fast (in terms of computation each message block) as any previous scheme with comparable security level.

The *ciphertext expansion* of a scheme says how much the size of a ciphertext grows compared to the size of the message. For a cloud server that stores vast numbers of files, it is naturally crucial to minimize the ciphertext expansion rate. It is also desirable to construct UE schemes that can encrypt *arbitrarily large files*, since a client might want to upload media files such as images or videos. Prior schemes that have achieved these two properites have only been secure in comparatively weak models. Our construction suitable for long messages –

enabling encryption of arbitrarily large files with almost no ciphertext expansion – is secure in our strong sense and is thus the first to bridge this gap.

1.1 Related Work

Security Models for UE. We regard the sequential, epoch-based corruption model of Lehmann and Tackmann [21] (LT18) as the most suitable execution environment to capture the threats in updatable encryption. In this model, the adversary advances to the next epoch via an oracle query. It can choose to submit its (single) challenge when it pleases, and it can later update the challenge ciphertext to the 'current' epoch. Further, the adversary is allowed to adaptively corrupt epoch (i.e. file encryption) keys and update tokens at any point in the game: only at the end of the adversary's execution does the challenger determine whether a trivial win has been made possible by some combination of the corruption queries and the challenge.

LT18 introduced two notions: IND-ENC asks the adversary to submit two plaintexts and distinguish the resulting ciphertext, while possibly having corrupted tokens (but of course not keys) linking this challenge ciphertext to prior or later epochs. Further, they introduced IND-UPD: the adversary provides two ciphertexts that it received via regular encryption-oracle queries in the previous epoch, and has to work out which one has been updated. They observed that plaintext information can be leaked not only through the encryption procedure, but also via updates. For schemes with deterministic updates, the adversary would trivially win if it could acquire the update token that takes the adversarially-provided ciphertexts into the challenge epoch, hence the definition for this setting, named detIND-UPD, is different from that for the randomized setting, named randIND-UPD.

LT18's IND-UPD definition was not the first approach to formalizing the desirable property of *unlinkability* of ciphertexts, which attempts to specify that given two already-updated ciphertexts, the adversary cannot tell if the plaintext is the same. Indeed EPRS (UP-REENC) and later KLR19 (UP-REENC-CCA) also considered this problem, in the ciphertext-dependent update and CCA-secure setting respectively. KLR19 [[17], § Appendix A] stated that "an even stronger notion [than IND-UPD] might be desirable: namely that fresh and re-encrypted ciphertexts are indistinguishable... which is not guaranteed by UP-REENC" – we will answer this open question later on in our paper.

In the full version of their work [4], BLMR introduced a security definition for UE denoted update – an extension of a model of symmetric proxy re-encryption. This non-sequential definition is considerably less adaptive than the later work of LT18, since the adversary's key/token corruption queries and ciphertext update queries are very limited. Further, they only considered schemes with deterministic update algorithms.

EPRS [12] provided (non-sequential) definitions for updatable authenticated encryption, in the ciphertext-dependent setting. Their work (inherently) covered CCA security and ciphertext integrity (CTXT). These definitions were ambiguous regarding adaptivity: these issues have since been fixed in the full version [13].

KLR19 attempted to provide stronger security guarantees for ciphertext-independent UE than LT18, concentrating on chosen-ciphertext security (and the weaker replayable CCA) in addition to integrity of plaintexts and ciphertexts. We revisit these definitions later on, and show how a small modification to their INT-CTXT game gives rise to natural composition results.

In practice, LT18's randIND-UPD definition insists that the ciphertext update procedure Upd requires the server to generate randomness for updating each ciphertext. Further, a scheme meeting both IND-ENC and IND-UPD can still leak the epoch in which the file was uploaded (the 'age' of the ciphertext). While it is arguable that metadata is inherent in outsourced storage, the use of updatable encryption is for high-security applications, and it would not be infeasible to design a system that does not reveal meta-data, which is clearly impossible if the underlying cryptosystem reveals the meta-data.

Recent work by Jarecki et al. [15] considers the key wrapping entity as a separate entity from the data owner or the storage server. While this approach seems promising, their security model is considerably weaker than those considered in our work or the papers already mentioned in this section: the adversary must choose whether to corrupt the key management server (and get the epoch key) or the storage server (and get the update token) for each epoch, and thus it cannot dynamically corrupt earlier keys or tokens at a later stage.

Constructions of Ciphertext-Independent UE. The initial description of updatable encryption by Boneh et al. [3] was motivated by providing a symmetric-key version of proxy re-encryption (see below). BLMR imagined doing this in a symmetric manner, where each epoch is simply one period in which re-encryption (rotation) has occurred. Their resulting scheme, denoted BLMR, deploys a key-homomorphic PRF, yet the nonce attached to a ciphertext ensures that IND-UPD cannot be met (the scheme pre-dates the IND-UPD notion).

The symmetric-Elgamal-based scheme of LT18, named RISE, uses a randomized update algorithm and is proven to meet IND-ENC and randIND-UPD under DDH. These proofs entail a seemingly unavoidable loss – a cubic term in the total number of epochs – our results also have this factor. LT18 also presented an extended version of the scheme by BLMR, denoted BLMR+, where the nonce is encrypted: they showed that this scheme meets a weak version of IND-UPD.

The aim of KLR19 was to achieve stronger security than BLMR, EPRS and LT18 in the ciphertext-independent setting: in particular CCA security and integrity protection. They observed that the structure of RISE ensures that ciphertext integrity cannot be achieved: access to just one update token allows the storage provider to construct ciphertexts of messages of its choice. Their generic constructions, based on encrypt-and-MAC and the Naor-Yung paradigm, are strictly less efficient than RISE. We show how to achieve CCA security and integrity protections with novel schemes that are comparably efficient with RISE.

Related Primitives. *Proxy re-encryption* (PRE) allows a ciphertext that is decryptable by some secret key to be re-encrypted such that it can be decrypted by some other key. Security models for PRE are closer to those for encryption than the strictly sequential outsourced-storage-centric models for UE, and as observed by Lehmann and Tackmann [21] the concepts of allowable corruptions and trivial wins for UE need considerable care when translating to the (more general) PRE setting. Unlinkability is not necessarily desired in PRE – updating the entire ciphertext may not be essential for a PRE scheme to be deemed secure – thus even after conversion to the symmetric setting, prior schemes [1,7] cannot meet the indistinguishability requirements that we ask of UE schemes. Recent works by Lee [20] and Davidson et al. [10] have highlighted the links between the work of BLMR and EPRS and PRE, and in particular the second work gives a public-key variant of the (sequential) $\mathsf{IND\text{-}UPD}$ definition of LT18. Myers and Shull [22] presented security models for hybrid proxy re-encryption, and gave a single-challenge version of the $\mathsf{UP\text{-}IND}$ notion of EPRS. While the models are subtly different, the techniques for achieving secure UE and PRE are often similar: in particular rotating keys via exponentiation to some simple function of old and new key. Further, the symmetric-key PRE scheme of Sakurai et al. [25] is at a high level similar to SHINE (their all-or-nothing-transform as an inner layer essentailly serves the same purpose as the ideal cipher in SHINE), but in a security model that does not allow dynamic corruptions.

Tokenization schemes aim to protect short secrets, such as credit card numbers, using deterministic encryption and deterministic updates: this line of work reflects the PCI DSS standard [9] for the payment card industry. Provable security of such schemes was initially explored by Diaz-Santiago et al. [11] and extended to the updatable setting by Cachin et al. [6]. While much of the formalism in the model of Cachin et al. has been used in recent works on UE (in particular the epoch-based corruption model), the requirements on ciphertext indistinguishability are stronger in the UE setting, where we expect probabilistic encryption of (potentially large) files.

1.2 Contributions

Our first major contribution is defining the $\mathsf{xxIND\text{-}UE\text{-}atk}$ security notion, for $(\mathsf{xx}, \mathsf{atk}) \in \{(\mathsf{det}, \mathsf{CPA}), (\mathsf{rand}, \mathsf{CPA}), (\mathsf{det}, \mathsf{CCA})\}$, and comprehensively analyzing its relation to other, existing[4] security notions ($\mathsf{xxIND\text{-}ENC\text{-}atk}$, $\mathsf{xxIND\text{-}UPD\text{-}atk}$). Our single definition requires that ciphertexts output by the encryption algorithm are indistinguishable from ciphertexts output by the update algorithm. We show that our new notion is strictly stronger even than combinations of prior notions, both in the randomized- and deterministic-update settings under chosen-plaintext attack and chosen-ciphertext attack. This not only gives us the unlinkability desired by prior works, but also answers the open question posed

[4] The notions $\mathsf{IND\text{-}ENC}$, $\mathsf{randIND\text{-}UPD}$ and $\mathsf{detIND\text{-}UPD}$ (which we denote as $\mathsf{IND\text{-}ENC\text{-}CPA}$, $\mathsf{randIND\text{-}UPD\text{-}CPA}$ and $\mathsf{detIND\text{-}UPD\text{-}CPA}$, resp.) are from LT18. The notions $\mathsf{UP\text{-}IND\text{-}CCA}$ and $\mathsf{UP\text{-}REENC\text{-}CCA}$ ($\mathsf{detIND\text{-}ENC\text{-}CCA}$ and $\mathsf{detIND\text{-}UPD\text{-}CCA}$, resp.) are from KLR19. LT18 and KLR19 both build upon the definitions given by EPRS.

	IND	INT	$\lvert M\rvert$	$\lvert C\rvert$	Enc (Upd)
BLMR [3]	(det, ENC, CPA)	✘	$l\lvert\mathbb{G}\rvert$	$(l{+}1)\lvert\mathbb{G}\rvert$	$l\mathbf{E}$
BLMR+ [3,21]	(weak, ENC, CPA) (weak, UPD, CPA)	✘	$l\lvert\mathbb{G}\rvert$	$(l{+}1)\lvert\mathbb{G}\rvert$	$l\mathbf{E}$
RISE [21]	(rand, UE, CPA)	✘	$1\lvert\mathbb{G}\rvert$	$2\lvert\mathbb{G}\rvert$	$2\mathbf{E}$
SHINE0[CPA] § 5.1	(det, UE, CPA)	✘	$(1\text{-}\gamma)\lvert\mathbb{G}\rvert$	$1\lvert\mathbb{G}\rvert$	$1\mathbf{E}$
NYUE [17]	(rand, ENC, RCCA) (rand, UPD, RCCA)	✘	$1\lvert\mathbb{G}_1\rvert$	$(34\lvert\mathbb{G}_1\rvert, 34\lvert\mathbb{G}_2\rvert)$	$(60\mathbf{E}, 70\mathbf{E})$
NYUAE [17]	(rand, ENC, RCCA) (rand, UPD, RCCA)	PTXT	$1\lvert\mathbb{G}_1\rvert$	$(58\lvert\mathbb{G}_1\rvert, 44\lvert\mathbb{G}_2\rvert)$	$(110\mathbf{E}, 90\mathbf{E})$
E&M [17]	(det, ENC, CCA) (det, UPD, CCA)	CTXT	$1\lvert\mathbb{G}\rvert$	$3\lvert\mathbb{G}\rvert$	$3\mathbf{E}$
SHINE0 § 5.1	(det, UE, CCA)	CTXT	$(1\text{-}2\gamma)\lvert\mathbb{G}\rvert$	$1\lvert\mathbb{G}\rvert$	$1\mathbf{E}$
MirrorSHINE [5]	(det, UE, CCA)	CTXT	$(1\text{-}\gamma)\lvert\mathbb{G}\rvert$	$2\lvert\mathbb{G}\rvert$	$2\mathbf{E}$
OCBSHINE § 5.1	(det, UE, CCA)	CTXT	$l\lvert\mathbb{G}\rvert$	$(l{+}2)\lvert\mathbb{G}\rvert$	$(l{+}2)\mathbf{E}$

Fig. 1. Comparison of security, ciphertext expansion and efficiency for updatable encryption schemes. (xx, yy, atk) represents the best possible xxIND-yy-atk notion that each scheme can achieve. $\mathbf{E}$ represents the cost of an exponentiation, for encryption Enc and ciphertext update Upd. γ represents the bit-size of the used nonce as a proportion of the group element bit-size. For NYUE and NYUAE, size/cost is in pairing groups $\mathbb{G}_1, \mathbb{G}_2$. SHINE0[CPA] is SHINE0 with a zero-length integrity tag. BLMR, BLMR+ and OCBSHINE support encryption of arbitrary size messages (of l blocks), with $\lvert M\rvert \approx l\lvert\mathbb{G}\rvert$.

by KLR19 mentioned on page 4. Fig. 13 describes the relationship between our new notion xxIND-UE-atk and prior notions.

We slightly tweak KLR19's definition of CTXT and CCA for UE and prove that detIND-yy-CPA + INT-CTXT $\Rightarrow$ detIND-yy-CCA for yy $\in$ {UE, ENC, UPD}. Combining this result with the relations from detIND-UE-atk above, we thus show that the combination of detIND-UE-CPA and INT-CTXT yields detIND-yy-CCA *for all* yy $\in$ {UE, ENC, UPD}.

Our second major contribution is in designing a new, fast updatable encryption scheme SHINE. Our scheme is based on a random-looking permutation combined with the exponentiation map in a cyclic group, and comes in a number of variants: SHINE0, MirrorSHINE (in our full version [5]) and OCBSHINE, for small messages, medium-sized messages and abitrarily large messages respectively. In Fig. 1, we provide a comparison of security, ciphertext expansion and efficiency between our new schemes and those from prior literature. We also further the understanding of schemes with deterministic update mechanisms. In particular, we identify the properties that are necessary of such schemes to meet a generalized version of our detIND-UE-atk notion. Another important contribution is that we further improve on the existing epoch insulation techniques that have been used to create proofs of security in the strong corruption environment we pursue. These have been shown to be very useful for studying updatable encryption schemes, and we expect our new techniques to be useful in the future.

1.3 Further Discussion

We have had to make a number of practical design decisions for our new UE scheme SHINE. The main idea is to permute the (combination of nonce and) message and then exponentiate the resulting value, with different mechanisms for enforcing ciphertext integrity depending on the flavor that is being used (which is in turn defined by the desired message length). In this subsection we give some motivation for why we believe that these choices are reasonable.

Deterministic updates. Since we will require indistinguishability of ciphertexts, we know that the UE encryption algorithm should be randomized. The update algorithm may or may not be randomized, however. All known schemes indicate that randomized updates are more expensive than deterministic updates, but there is a small, well-understood security loss in moving to deterministic updates: an adversary with an update token in an appropriate epoch can trivially distinguish between an update of a known ciphertext and other ciphertexts in the next epoch. As a result, in the detIND-UE-CPA case the adversary is only forbidden from obtaining one token compared to randIND-UE-CPA. Furthermore, UE schemes with randomized updates cannot achieve CTXT and CCA security, which is possible for the deterministic-update setting. We believe that the minor CPA security loss is a small price to pay for stronger security (CTXT and CCA) and efficiency gain, in particular to reduce computations in the UE encryption and update algorithms and also improve ciphertext expansion.

Limited number of epochs. In many applications that we would like to consider, the user of the storage service will control when updates occur (perhaps when an employee with access to key material leaves the organisation, or if an employee loses a key-holding device): this indicates that the total number of key rotations in the lifetime of a storage system might be numbered in the thousands, and in particular could be considerably smaller than the number of outsourced files.

2 Preliminaries

Pseudocode **return** $b' \overset{?}{=} b$ is used as shorthand for **if** $b' = b$ **then return** 1 // **else return** 0, with an output of 1 indicating adversarial success. We use the concrete security framework, defining adversarial advantage as probability of success in the security game, and avoid statements of security with respect to security notions. In the cases where we wish to indicate that notion A implies notion B (for some fixed primitive), i.e. an adversary's advantage against B carries over to an advantage against A, we show this by bounding these probabilities.

We follow the syntax of prior work [17], defining an Updatable Encryption (UE) scheme as a tuple of algorithms $\{\mathsf{UE.KG}, \mathsf{UE.TG}, \mathsf{UE.Enc}, \mathsf{UE.Dec}, \mathsf{UE.Upd}\}$ that operate in epochs, these algorithms are described in Fig. 2. A scheme is defined over some plaintext space $\mathcal{MS}$, ciphertext space $\mathcal{CS}$, key space $\mathcal{KS}$ and token space $\mathcal{TS}$. We specify integer $n + 1$ as the (total) number of epochs over

Algorithm		Rand/Det	Input	Output	Syntax
$\mathsf{UE.KG}$	Key Gen	Rand	λ	k_{e}	$k_{\mathsf{e}} \xleftarrow{\$} \mathsf{UE.KG}(\lambda)$
$\mathsf{UE.TG}$	Token Gen	Det	$k_{\mathsf{e}}, k_{\mathsf{e}+1}$	$\Delta_{\mathsf{e}+1}$	$\Delta_{\mathsf{e}+1} \leftarrow \mathsf{UE.TG}(k_{\mathsf{e}}, k_{\mathsf{e}+1})$
$\mathsf{UE.Enc}$	Encryption	Rand	M, k_{e}	C_{e}	$C_{\mathsf{e}} \xleftarrow{\$} \mathsf{UE.Enc}(k_{\mathsf{e}}, M)$
$\mathsf{UE.Dec}$	Decryption	Det	$C_{\mathsf{e}}, k_{\mathsf{e}}$	M' or $\perp$	$\{M'/\perp\} \leftarrow \mathsf{UE.Dec}(k_{\mathsf{e}}, C_{\mathsf{e}})$
$\mathsf{UE.Upd}$	Update Ctxt	Rand/det	$C_{\mathsf{e}}, \Delta_{\mathsf{e}+1}$	$C_{\mathsf{e}+1}$	$C_{\mathsf{e}+1} \xleftarrow{\$} \mathsf{UE.Upd}(\Delta_{\mathsf{e}+1}, C_{\mathsf{e}})$

Fig. 2. Syntax of algorithms defining an Updatable Encryption scheme UE.

which a UE scheme can operate, though this is only for proof purposes. Correctness [17] is defined as expected: fresh encryptions and updated ciphertexts should decrypt to the correct message under the appropriate epoch key.

In addition to enabling ciphertext updates, in many schemes the token allows ciphertexts to be 'downgraded': performing some analog of the $\mathsf{UE.Upd}$ operation on a ciphertext C created in (or updated to) epoch e yields a valid ciphertext in epoch e-1. Such a scheme is said to have *bi-directional ciphertext updates*. Furthermore, for many constructions, the token additionally enables key derivation, given one adjacent key. If this can be done in both directions – i.e. knowledge of k_{e} and $\Delta_{\mathsf{e}+1}$ allows derivation of $k_{\mathsf{e}+1}$ AND knowledge of $k_{\mathsf{e}+1}$ and $\Delta_{\mathsf{e}+1}$ allows derivation of k_{e} – then such schemes are referred to by LT18 as having *bi-directional key updates*. If such derivation is only possible in one 'direction' then the scheme is said to have *uni-directional key updates*. Much of the prior literature on updatable encryption has distinguished these notions: we stress that <u>all schemes and definitions</u> of security considered in this paper have <u>bi-directional key updates</u> and <u>bi-directional ciphertext updates</u>.

3 Security Models for Updatable Encryption

We consider a number of indistinguishability-based confidentiality games and integrity games for assessing security of updatable encryption schemes. The environment provided by the challenger attempts to give as much power as possible to adversary $\mathcal{A}$. The adversary may call for a number of oracles, and after $\mathcal{A}$ has finished running the challenger computes whether or not any of the actions enabled a trivial win. The available oracles are described in Fig. 3. An overview of the oracles $\mathcal{A}$ has access to in each security game is provided in Fig. 4.

Confidentiality. A generic representation of all confidentiality games described in this paper is detailed in Fig. 5. The current epoch is advanced by an adversarial call to $\mathcal{O}.\mathsf{Next}$ – simulating $\mathsf{UE.KG}$ and $\mathsf{UE.TG}$ – and keys and tokens (for the current or any prior epoch) can be corrupted via $\mathcal{O}.\mathsf{Corr}$. The adversary can encrypt arbitrary messages via $\mathcal{O}.\mathsf{Enc}$, and update these 'non-challenge' ciphertexts via $\mathcal{O}.\mathsf{Upd}$. In CCA games, the adversary can additionally call decryption oracle $\mathcal{O}.\mathsf{Dec}$ (with some natural restrictions to prevent trivial wins). At some point $\mathcal{A}$ makes its challenge by providing two inputs, and receives the challenge ciphertext – and in later epochs can receive an updated version by calling $\mathcal{O}.\mathsf{Upd}\tilde{\mathrm{C}}$

```
$\underline{\mathbf{Setup}(\lambda)}$
  $k_0 \leftarrow \mathsf{UE.KG}(\lambda)$
  $\Delta_0 \leftarrow \perp$;  $\mathsf{e}, \mathsf{c} \leftarrow 0$;  $\mathsf{phase}, \mathsf{twf} \leftarrow 0$
  $\mathcal{L}, \tilde{\mathcal{L}}, \mathcal{C}, \mathcal{K}, \mathcal{T} \leftarrow \emptyset$

$\underline{\mathcal{O}.\mathsf{Enc}(M):}$
  $C \leftarrow \mathsf{UE.Enc}(k_\mathsf{e}, M)$
  $\mathsf{c} \leftarrow \mathsf{c} + 1$;  $\mathcal{L} \leftarrow \mathcal{L} \cup \{(\mathsf{c}, C, \mathsf{e})\}$
  return $C$

$\underline{\mathcal{O}.\mathsf{Dec}(C):}$
  if $\mathsf{phase} = 1$ and $C \in \tilde{\mathcal{L}}^*$ then
    $\mathsf{twf} \leftarrow 1$
  $M'$ or $\perp \leftarrow \mathsf{UE.Dec}(k_\mathsf{e}, C)$
  return $M'$ or $\perp$

$\underline{\mathcal{O}.\mathsf{Next}():}$
  $\mathsf{e} \leftarrow \mathsf{e} + 1$
  $k_\mathsf{e} \xleftarrow{\$} \mathsf{UE.KG}(\lambda)$;  $\Delta_\mathsf{e} \xleftarrow{\$} \mathsf{UE.TG}(k_{\mathsf{e}-1}, k_\mathsf{e})$
  if $\mathsf{phase} = 1$ then
    $\tilde{C}_\mathsf{e} \leftarrow \mathsf{UE.Upd}(\Delta_\mathsf{e}, \tilde{C}_{\mathsf{e}-1})$

$\underline{\mathcal{O}.\mathsf{Upd}(C_{\mathsf{e}-1}):}$
  if $(j, C_{\mathsf{e}-1}, \mathsf{e} - 1) \notin \mathcal{L}$ then
    return $\perp$
  $C_\mathsf{e} \leftarrow \mathsf{UE.Upd}(\Delta_\mathsf{e}, C_{\mathsf{e}-1})$
  $\mathcal{L} \leftarrow \mathcal{L} \cup \{(j, C_\mathsf{e}, \mathsf{e})\}$
  return $C_\mathsf{e}$

$\underline{\mathcal{O}.\mathsf{Corr}(\mathsf{inp}, \hat{\mathsf{e}}):}$
  if $\hat{\mathsf{e}} > \mathsf{e}$ then
    return $\perp$
  if $\mathsf{inp} = \mathsf{key}$ then
    $\mathcal{K} \leftarrow \mathcal{K} \cup \{\hat{\mathsf{e}}\}$
    return $k_{\hat{\mathsf{e}}}$
  if $\mathsf{inp} = \mathsf{token}$ then
    $\mathcal{T} \leftarrow \mathcal{T} \cup \{\hat{\mathsf{e}}\}$
    return $\Delta_{\hat{\mathsf{e}}}$

$\underline{\mathcal{O}.\mathsf{Upd}\tilde{C}:}$
  $\mathcal{C} \leftarrow \mathcal{C} \cup \{\mathsf{e}\}$
  $\tilde{\mathcal{L}} \leftarrow \tilde{\mathcal{L}} \cup \{(\tilde{C}_\mathsf{e}, \mathsf{e})\}$
  return $\tilde{C}_\mathsf{e}$

$\underline{\mathcal{O}.\mathsf{Try}(\tilde{C}):}$
  if $\mathsf{phase} = 1$ then
    return $\perp$
  $\mathsf{phase} \leftarrow 1$
  $\underline{\mathsf{twf} \leftarrow 1 \text{ if}}$ :
    $\mathsf{e} \in \mathcal{K}^*$ or $\tilde{C} \in \mathcal{L}^*$
  $M'$ or $\perp \leftarrow \mathsf{UE.Dec}(k_\mathsf{e}, \tilde{C})$
  if $M' \neq \perp$ then
    $\mathsf{win} \leftarrow 1$
```

Fig. 3. Oracles in security games for updatable encryption. The shaded statement in $\mathcal{O}.\mathsf{Try}$ only applies to $\mathsf{INT\text{-}CTXT^s}$: in this game the adversary is allowed to query the $\mathcal{O}.\mathsf{Try}$ oracle only once. Computing $\tilde{\mathcal{L}}^*$ is discussed in Section 3.2.

(computing this value is actually done by $\mathcal{O}.\mathsf{Next}$, a call to $\mathcal{O}.\mathsf{Upd}\tilde{C}$ returns it). $\mathcal{A}$ can then interact with its other oracles again, and eventually outputs its guess bit. The flag phase tracks whether or not $\mathcal{A}$ has made its challenge, and we always give the epoch in which the challenge happens a special identifier $\tilde{\mathsf{e}}$. If $\mathcal{A}$ makes any action that would lead to a trivial win, the flag twf is set as 1 and $\mathcal{A}$'s output is discarded and replaced by a random bit. We follow the bookkeeping techniques of LT18 and KLR19, using the following sets to track ciphertexts and their updates that can be known to the adversary.

- $\mathcal{L}$: List of non-challenge ciphertexts (from $\mathcal{O}.\mathsf{Enc}$ or $\mathcal{O}.\mathsf{Upd}$) with entries of form $(\mathsf{c}, C, \mathsf{e})$, where query identifier c is a counter incremented with each new $\mathcal{O}.\mathsf{Enc}$ query.
- $\tilde{\mathcal{L}}$: List of updated versions of challenge ciphertext (created via $\mathcal{O}.\mathsf{Next}$, received by adversary via $\mathcal{O}.\mathsf{Upd}\tilde{C}$), with entries of form $(\tilde{C}, \mathsf{e})$.

Further, we use the following lists that track epochs only.

- $\mathcal{C}$: List of epochs in which adversary learned updated version of challenge ciphertext (via CHALL or $\mathcal{O}.\mathsf{Upd}\tilde{\mathrm{C}}$).
- $\mathcal{K}$: List of epochs in which the adversary corrupted the encryption key.
- $\mathcal{T}$: List of epochs in which the adversary corrupted the update token.

All experiments necessarily maintain some state, but we omit this for readability reasons. The challenger's state is $\mathbf{S} \leftarrow \{\mathcal{L}, \tilde{\mathcal{L}}, \mathcal{C}, \mathcal{K}, \mathcal{T}\}$, and the system state in the current epoch is given by $\mathsf{st} \leftarrow (\mathrm{k}_\mathsf{e}, \Delta_\mathsf{e}, \mathbf{S}, \mathsf{e})$.

An at-a-glance overview of CHALL for various security definitions is given in Fig. 7. For security games such as LT18's $\mathsf{IND\text{-}UPD}$ notion, where the adversary must submit as its challenge two ciphertexts (that it received from $\mathcal{O}.\mathsf{Enc}$) and one is updated, the game must also track in which epochs the adversary has updates of these ciphertexts. We will later specify a version of our new $\mathsf{xxIND\text{-}UE\text{-}atk}$ notion that allows the adversary to submit a ciphertext that existed in any epoch prior to the challenge epoch, not just the one immediately before: this introduces some additional bookkeeping (discussed further in Section 3.2).

A note on nomenclature: the adversary can make its challenge query to receive *the challenge ciphertext*, and then acquire *updates of the challenge ciphertext* via calls to $\mathcal{O}.\mathsf{Upd}\tilde{\mathrm{C}}$, and additionally it can calculate *challenge-equal ciphertexts* via applying tokens it gets via $\mathcal{O}.\mathsf{Corr}$ queries.

When appropriate, we will restrict our experiments to provide definitions of security that are more suitable for assessing schemes with deterministic update mechanisms. For such schemes, access to the update token for the challenge epoch ($\Delta_{\tilde{\mathsf{e}}}$) allows the adversary to trivially win $\mathsf{detIND\text{-}UPD\text{-}atk}$ and $\mathsf{detIND\text{-}UE\text{-}atk}$ for $\mathsf{atk} \in \{\mathsf{CPA}, \mathsf{CCA}\}$. Note however that the definitions are not restricted to schemes with deterministic updates: such schemes are simply insecure in terms of $\mathsf{randIND\text{-}UPD\text{-}CPA}$ and $\mathsf{randIND\text{-}UE\text{-}CPA}$.

Ciphertext Integrity. In ciphertext integrity (CTXT) game, the adversary is allowed to make calls to oracles $\mathcal{O}.\mathsf{Enc}$, $\mathcal{O}.\mathsf{Next}$, $\mathcal{O}.\mathsf{Upd}$ and $\mathcal{O}.\mathsf{Corr}$. At some point $\mathcal{A}$ attempts to provide a forgery via $\mathcal{O}.\mathsf{Try}$; as part of this query the challenger will assess if it is valid. We distinguish between the $\underline{\text{s}}$ingle-$\mathcal{O}.\mathsf{Try}$ case ($\mathsf{INT\text{-}CTXT}^\mathsf{s}$) and the multi-$\mathcal{O}.\mathsf{Try}$ case ($\mathsf{INT\text{-}CTXT}$). Here, "valid" means decryption outputs a message (i.e. not $\perp$). In the single-$\mathcal{O}.\mathsf{Try}$ case, $\mathcal{A}$ can continue making oracle queries after its $\mathcal{O}.\mathsf{Try}$ query, however this is of no benefit since it has already

Notion	$\mathcal{O}.\mathsf{Enc}$	$\mathcal{O}.\mathsf{Dec}$	$\mathcal{O}.\mathsf{Next}$	$\mathcal{O}.\mathsf{Upd}$	$\mathcal{O}.\mathsf{Corr}$	$\mathcal{O}.\mathsf{Upd}\tilde{\mathrm{C}}$	$\mathcal{O}.\mathsf{Try}$
detIND-yy-CPA	✓	×	✓	✓	✓	✓	×
randIND-yy-CPA	✓	×	✓	✓	✓	✓	×
detIND-yy-CCA	✓	✓	✓	✓	✓	✓	×
INT-CTXT	✓	×	✓	✓	✓	×	✓

Fig. 4. Oracles the adversary is allowed to query in different security games, where $\mathsf{yy} \in \{\mathsf{ENC}, \mathsf{UPD}, \mathsf{UE}\}$. ✓ indicates the adversary has access to the corresponding oracle.

```
$\underline{\mathbf{Exp}^{\mathsf{xxIND\text{-}yy\text{-}atk\text{-}b}}_{\mathsf{UE},\ \mathcal{A}}}$
  do Setup
  $\mathsf{CHALL} \leftarrow \mathcal{A}^{\mathcal{O}.\mathsf{Enc},(\mathcal{O}.\mathsf{Dec}),\mathcal{O}.\mathsf{Next},\mathcal{O}.\mathsf{Upd},\mathcal{O}.\mathsf{Corr}}(\lambda)$
  $\mathsf{phase} \leftarrow 1; \tilde{\mathsf{e}} \leftarrow \mathsf{e}$
  Create $\tilde{\mathrm{C}}$ with $\mathsf{CHALL}$; $\tilde{\mathcal{L}} \leftarrow \tilde{\mathcal{L}} \cup \{(\tilde{\mathrm{C}}_{\mathsf{e}}, \mathsf{e})\}$
  $\mathrm{b}' \leftarrow \mathcal{A}^{\mathcal{O}.\mathsf{Enc},(\mathcal{O}.\mathsf{Dec}),\mathcal{O}.\mathsf{Next},\mathcal{O}.\mathsf{Upd},\mathcal{O}.\mathsf{Corr},\mathcal{O}.\mathsf{Upd}\tilde{\mathrm{C}}}(\tilde{\mathrm{C}})$
  $\underline{\mathsf{twf} \leftarrow 1 \ \mathbf{if}}$ :
    $\mathcal{K}^* \cap \mathcal{C}^* \neq \emptyset$  or
    $\mathsf{xx} = \mathsf{det}$  and  $\mathcal{I}^* \cap \mathcal{C}^* \neq \emptyset$
  if $\mathsf{twf} = 1$ then
    $\mathrm{b}' \xleftarrow{\$} \{0, 1\}$
  return $\mathrm{b}'$
```

Fig. 5. Generic description of confidentiality experiment $\mathbf{Exp}^{\mathsf{xxIND\text{-}yy\text{-}atk\text{-}b}}_{\mathsf{UE},\ \mathcal{A}}$ for scheme UE, for $\mathsf{xx} \in \{\mathsf{det}, \mathsf{rand}\}$, $\mathsf{yy} \in \{\mathsf{ENC}, \mathsf{UPD}, \mathsf{UE}\}$ and $\mathsf{atk} \in \{\mathsf{CPA}, \mathsf{CCA}\}$. We do not consider (and thus do not formally define) randIND-yy-CCA; only in detIND-yy-CCA games does $\mathcal{A}$ have access to $\mathcal{O}$.Dec.

```
$\underline{\mathbf{Exp}^{\mathsf{INT\text{-}CTXT}}_{\mathsf{UE},\ \mathcal{A}}}$
  do Setup; $\mathsf{win} \leftarrow 0$
  $\mathcal{A}^{\mathcal{O}.\mathsf{Enc},\mathcal{O}.\mathsf{Next},\mathcal{O}.\mathsf{Upd},\mathcal{O}.\mathsf{Corr},\mathcal{O}.\mathsf{Try}}(\lambda)$
  if $\mathsf{twf} = 1$ then
    $\mathsf{win} \leftarrow 0$
  return win
```

Fig. 6. INT-CTXT security notion for updatable encryption scheme UE.Deciding twf and computing $\mathcal{L}^*$ are discussed in Section 3.2.

won or lost. In the multi-$\mathcal{O}$.Try case, $\mathcal{A}$ can make any number of $\mathcal{O}$.Try queries: as long as it wins once, it wins the ciphertext integrity game. Formally, the definition of ciphertext integrity is given in Definition 1.

Definition 1. *Let* $\mathsf{UE} = \{\mathsf{UE.KG}, \mathsf{UE.TG}, \mathsf{UE.Enc}, \mathsf{UE.Dec}, \mathsf{UE.Upd}\}$ *be an updatable encryption scheme. Then the* INT-CTXT *advantage of an adversary* $\mathcal{A}$ *against* UE *is defined as*

$$\mathbf{Adv}^{\mathsf{INT\text{-}CTXT}}_{\mathsf{UE},\ \mathcal{A}}(\lambda) = \mathbf{Pr}[\mathbf{Exp}^{\mathsf{INT\text{-}CTXT}}_{\mathsf{UE},\ \mathcal{A}} = 1]$$

where the experiment $\mathbf{Exp}^{\mathsf{INT\text{-}CTXT}}_{\mathsf{UE},\ \mathcal{A}}$ *is given in Fig. 3 and Fig. 6. Particularly, if* $\mathcal{A}$ *is allowed to ask only one* $\mathcal{O}$.Try *query, denote such notion as* $\mathsf{INT\text{-}CTXT}^{\mathsf{s}}$.

Note that INT-CTXT trivially implies $\mathsf{INT\text{-}CTXT}^{\mathsf{s}}$. In the full version [5] we prove that $\mathsf{INT\text{-}CTXT}^{\mathsf{s}}$ implies INT-CTXT too, with loss upper-bounded by the number of $\mathcal{O}$.Try queries. KLR19 define ciphertext integrity with one $\mathcal{O}$.Try

	CHALL	Output of " Create $\tilde{\mathrm{C}}$ with CHALL"(in $\tilde{\mathsf{e}}$)
xxIND-ENC-atk	$\bar{\mathrm{M}}_0, \bar{\mathrm{M}}_1$	$\mathsf{UE.Enc}_{\mathrm{k}_{\tilde{\mathsf{e}}}}(\bar{\mathrm{M}}_0)$ **or** $\mathsf{UE.Enc}_{\mathrm{k}_{\tilde{\mathsf{e}}}}(\bar{\mathrm{M}}_1)$
xxIND-UPD-atk	$\bar{\mathrm{C}}_0, \bar{\mathrm{C}}_1$	$\mathsf{UE.Upd}_{\Delta_{\tilde{\mathsf{e}}}}(\bar{\mathrm{C}}_0)$ **or** $\mathsf{UE.Upd}_{\Delta_{\tilde{\mathsf{e}}}}(\bar{\mathrm{C}}_1)$
xxIND-UE-atk	$\bar{\mathrm{M}}, \bar{\mathrm{C}}$	$\mathsf{UE.Enc}_{\mathrm{k}_{\tilde{\mathsf{e}}}}(\bar{\mathrm{M}})$ **or** $\mathsf{UE.Upd}_{\Delta_{\tilde{\mathsf{e}}}}(\bar{\mathrm{C}})$

Fig. 7. Intuitive description of challenge inputs and outputs in confidentiality games for updatable encryption schemes, for $(\mathsf{xx}, \mathsf{atk}) \in \{(\mathsf{det}, \mathsf{CPA}), (\mathsf{rand}, \mathsf{CPA}), (\mathsf{det}, \mathsf{CCA})\}$.

query plus access to $\mathcal{O}.\mathsf{Dec}$, and the game ends when the $\mathcal{O}.\mathsf{Try}$ query happens. It is hard to prove the generic relation among CPA, CTXT and CCA using this formulation. Notice that decryption oracles give the adversary power to win the CTXT game even it only has one $\mathcal{O}.\mathsf{Try}$ query. The adversary can send its forgery to the decryption oracle to test if it is valid (if $\mathcal{O}.\mathsf{Dec}$ outputs a message and not $\perp$) – thus $\mathcal{A}$ can continue to send forgeries to $\mathcal{O}.\mathsf{Dec}$ until a valid one is found, and then send this as a $\mathcal{O}.\mathsf{Try}$ query (and win the game). So intuitively, a decryption oracle is equivalent to multiple $\mathcal{O}.\mathsf{Try}$ queries. Proving that all these variants of CTXT definitions are equivalent to each other is straightforward, with the loss upper-bounded by the sum of $\mathcal{O}.\mathsf{Try}$ queries and decryption queries.

Remark 1. The definition of $\mathsf{INT\text{-}CTXT}$ is more natural for defining ciphertext integrity, however, it is easier to use $\mathsf{INT\text{-}CTXT^s}$ notion to prove ciphertext integrity for specific UE schemes. As $\mathsf{INT\text{-}CTXT} \iff \mathsf{INT\text{-}CTXT^s}$, we use both definitions in this paper.

3.1 Existing Definitions of Confidentiality

Here we describe existing confidentiality notions given by LT18 and KLR19, including formal definitions for their $\mathsf{IND\text{-}yy\text{-}CPA}$ and $\mathsf{IND\text{-}yy\text{-}CCA}$ notions, respectively. (Note that KLR19 used UP-REENC to refer to the the unlinkability notion that we and LT18 call $\mathsf{IND\text{-}UPD}$). We will define our new security notion in Section 4.1 and compare the relationship between all notions in Section 4.2.

Definition 2. *Let* $\mathsf{UE} = \{\mathsf{UE.KG}, \mathsf{UE.TG}, \mathsf{UE.Enc}, \mathsf{UE.Dec}, \mathsf{UE.Upd}\}$ *be an updatable encryption scheme. Then the* $\mathsf{xxIND\text{-}ENC\text{-}atk}$ *advantage, for* $(\mathsf{xx}, \mathsf{atk}) \in \{(\mathsf{det}, \mathsf{CPA}), (\mathsf{rand}, \mathsf{CPA}), (\mathsf{det}, \mathsf{CCA})\}$*, of an adversary* $\mathcal{A}$ *against* UE *is defined as*

$$\mathbf{Adv}^{\mathsf{xxIND\text{-}ENC\text{-}atk}}_{\mathsf{UE},\ \mathcal{A}}(\lambda) = \left|\mathbf{Pr}[\mathbf{Exp}^{\mathsf{xxIND\text{-}ENC\text{-}atk\text{-}1}}_{\mathsf{UE},\ \mathcal{A}} = 1] - \mathbf{Pr}[\mathbf{Exp}^{\mathsf{xxIND\text{-}ENC\text{-}atk\text{-}0}}_{\mathsf{UE},\ \mathcal{A}} = 1]\right|,$$

where the experiment $\mathbf{Exp}^{\mathsf{xxIND\text{-}ENC\text{-}atk\text{-}b}}_{\mathsf{UE},\ \mathcal{A}}$ *is given in Fig. 3, Fig. 5 and Fig. 8.*

Definition 3. *Let* $\mathsf{UE} = \{\mathsf{UE.KG}, \mathsf{UE.TG}, \mathsf{UE.Enc}, \mathsf{UE.Dec}, \mathsf{UE.Upd}\}$ *be an updatable encryption scheme. Then the* $\mathsf{xxIND\text{-}UPD\text{-}atk}$ *advantage, for* $(\mathsf{xx}, \mathsf{atk}) \in \{(\mathsf{det}, \mathsf{CPA}), (\mathsf{rand}, \mathsf{CPA}), (\mathsf{det}, \mathsf{CCA})\}$*, of an adversary* $\mathcal{A}$ *against* UE *is defined as*

$$\mathbf{Adv}^{\mathsf{xxIND\text{-}UPD\text{-}atk}}_{\mathsf{UE},\ \mathcal{A}}(\lambda) = \left|\mathbf{Pr}[\mathbf{Exp}^{\mathsf{xxIND\text{-}UPD\text{-}atk\text{-}1}}_{\mathsf{UE},\ \mathcal{A}} = 1] - \mathbf{Pr}[\mathbf{Exp}^{\mathsf{xxIND\text{-}UPD\text{-}atk\text{-}0}}_{\mathsf{UE},\ \mathcal{A}} = 1]\right|,$$

where the experiments $\mathbf{Exp}^{\mathsf{xxIND\text{-}UPD\text{-}atk\text{-}b}}_{\mathsf{UE},\ \mathcal{A}}$ *are given in Fig. 3, Fig. 5 and Fig. 9.*

We do not define $\mathsf{randIND\text{-}ENC\text{-}CCA}$ or $\mathsf{randIND\text{-}UPD\text{-}CCA}$ – these notions were formalized by KLR19. Note that trivial win via direct update (see Section 3.2) is never triggered in the $\mathsf{detIND\text{-}ENC\text{-}CPA}$ game. Thus, $\mathsf{randIND\text{-}ENC\text{-}CPA}$ is equivalent to $\mathsf{detIND\text{-}ENC\text{-}CPA}$. For simplicity, we will often denote the notion $\mathsf{xxIND\text{-}ENC\text{-}CPA}$ as $\mathsf{IND\text{-}ENC\text{-}CPA}$.

```
$\mathbf{Exp}^{\mathsf{xxIND\text{-}ENC\text{-}atk\text{-}b}}_{\mathsf{UE},\ \mathcal{A}}(\lambda)$ :
  $(\bar{\mathrm{M}}_0, \bar{\mathrm{M}}_1) \leftarrow \mathcal{A}$
  Create $\tilde{\mathrm{C}}$ with $(\bar{\mathrm{M}}_0, \bar{\mathrm{M}}_1)$ :
    if $|\bar{\mathrm{M}}_0| \neq |\bar{\mathrm{M}}_1|$ then
      return $\perp$
    $\tilde{\mathrm{C}} \xleftarrow{\$} \mathsf{UE.Enc}(\mathrm{k}_{\tilde{\mathsf{e}}}, \bar{\mathrm{M}}_{\mathrm{b}})$
    return $\tilde{\mathrm{C}}$
```

Fig. 8. Challenge call definition for xxIND-ENC-atk security experiment; the full experiment is given in combination with Fig. 3 and Fig. 5.

```
$\mathbf{Exp}^{\mathsf{xxIND\text{-}UPD\text{-}atk\text{-}b}}_{\mathsf{UE},\ \mathcal{A}}(\lambda)$ :
  $(\bar{\mathrm{C}}_0, \bar{\mathrm{C}}_1) \leftarrow \mathcal{A}$
  Create $\tilde{\mathrm{C}}$ with $(\bar{\mathrm{C}}_0, \bar{\mathrm{C}}_1)$ :
    if $|\bar{\mathrm{C}}_0| \neq |\bar{\mathrm{C}}_1|$ or $(\bar{\mathrm{C}}_0, \tilde{\mathsf{e}}\text{-}1) \notin \mathcal{L}$
      or $(\bar{\mathrm{C}}_1, \tilde{\mathsf{e}}\text{-}1) \notin \mathcal{L}$ then
      return $\perp$
    $\tilde{\mathrm{C}} \xleftarrow{\$} \mathsf{UE.Upd}(\Delta_{\tilde{\mathsf{e}}}, \bar{\mathrm{C}}_{\mathrm{b}})$
    return $\tilde{\mathrm{C}}$
```

Fig. 9. Challenge call definition for xxIND-UPD-atk security experiment; the full experiment is given in combination with Fig. 3 and Fig. 5.

Remark 2. LT18 defined weakIND-ENC-CPA and weakIND-UPD-CPA for analyzing BLMR+, a modification of BLMR's scheme where the nonce is encrypted using symmetric encryption. In this notion, the adversary trivially loses if it obtains an update token linking the challenge epoch to the epoch before or after.

3.2 Trivial Win Conditions

Trivial Win Conditions in Confidentiality Games

Trivial wins via keys and ciphertexts. The following is for analyzing all confidentiality games. We again follow LT18 in defining the epoch identification sets $\mathcal{C}^*$, $\mathcal{K}^*$ and $\mathcal{T}^*$ as the extended sets of $\mathcal{C}$, $\mathcal{K}$ and $\mathcal{T}$ in which the adversary has learned or inferred information via its acquired tokens. These extended sets are used to exclude cases in which the adversary trivially wins, i.e. if $\mathcal{C}^* \cap \mathcal{K}^* \neq \emptyset$, then there exists an epoch in which the adversary knows the epoch key and a valid update of the challenge ciphertext. Note that the challenger computes these sets once the adversary has finished running. We employ the following algorithms of LT18 (for bi-directional updates):

$$\mathcal{K}^* \leftarrow \{\mathsf{e} \in \{0, \ldots, n\} | \mathsf{CorrK}(\mathsf{e}) = \mathsf{true}\}$$
$$\mathsf{true} \leftarrow \mathsf{CorrK}(\mathsf{e}) \iff (\mathsf{e} \in \mathcal{K}) \vee (\mathsf{CorrK}(\mathsf{e}\text{-}1) \wedge \mathsf{e} \in \mathcal{T}) \vee (\mathsf{CorrK}(\mathsf{e}{+}1) \wedge \mathsf{e}{+}1 \in \mathcal{T})$$
$$\mathcal{T}^* \leftarrow \{\mathsf{e} \in \{0, \ldots, n\} | (\mathsf{e} \in \mathcal{T}) \vee (\mathsf{e} \in \mathcal{K}^* \wedge \mathsf{e}\text{-}1 \in \mathcal{K}^*)\}$$
$$\mathcal{C}^* \leftarrow \{\mathsf{e} \in \{0, \ldots, n\} | \mathsf{ChallEq}(\mathsf{e}) = \mathsf{true}\}$$
$$\mathsf{true} \leftarrow \mathsf{ChallEq}(\mathsf{e}) \iff$$
$$(\mathsf{e} = \tilde{\mathsf{e}}) \vee (\mathsf{e} \in \mathcal{C}) \vee (\mathsf{ChallEq}(\mathsf{e}\text{-}1) \wedge \mathsf{e} \in \mathcal{T}^*) \vee (\mathsf{ChallEq}(\mathsf{e}{+}1) \wedge \mathsf{e}{+}1 \in \mathcal{T}^*)$$

Trivial wins via direct updates. The following is for analyzing detIND-yy-atk security notions, for yy $\in \{\mathsf{UE}, \mathsf{UPD}\}$ and atk $\in \{\mathsf{CPA}, \mathsf{CCA}\}$, where the adversary provides as its challenge one or two ciphertexts that it received from $\mathcal{O}.\mathsf{Enc}$. The challenger needs to use $\mathcal{L}$ to track the information the adversary has about these challenge input values.

Define a new list $\mathcal{I}$ as the list of epochs in which the adversary learned an updated version of the ciphertext(s) given as a challenge input. Furthermore, define $\mathcal{I}^*$ to be the extended set in which the adversary has learned or inferred information via token corruption. We will use this set to exclude cases which the adversary trivially wins, i.e. if $\mathcal{I}^* \cap \mathcal{C}^* \neq \emptyset$, then there exists an epoch in which the adversary knows the updated ciphertext of $\bar{\mathrm{C}}$ and a valid challenge-equal ciphertext. For deterministic updates, the adversary can simply compare these ciphertexts to win the game. In particular, if $\bar{\mathrm{C}}$ is restricted to come from $\tilde{\mathsf{e}} - 1$ (recall the challenge epoch is $\tilde{\mathsf{e}}$), then the condition $\mathcal{I}^* \cap \mathcal{C}^* \neq \emptyset$ is equivalent to the win condition that LT18 used for $\mathsf{IND\text{-}UPD}$: $\Delta_{\tilde{\mathsf{e}}} \in \mathcal{T}^*$ or $\mathcal{A}$ did $\mathcal{O}.\mathsf{Upd}(\bar{\mathrm{C}})$ in $\tilde{\mathsf{e}}$. Our generalization is necessary for a variant of $\mathsf{xxIND\text{-}UE\text{-}atk}$ that we define later in which the challenge ciphertext input can come from any prior epoch, and not just the epoch immediately before the one in which the challenge is made.

To compute $\mathcal{I}$, find an entry in $\mathcal{L}$ that contains challenge input $\bar{\mathrm{C}}$. Then for that entry, note the query identifier c, scan $\mathcal{L}$ for other entries with this identifier, and add into list $\mathcal{I}$ all found indices: $\mathcal{I} \leftarrow \{\mathsf{e} \in \{0, ..., n\} | (\mathsf{c}, \cdot, \mathsf{e}) \in \mathcal{L}\}$. Then compute $\mathcal{I}^*$ as follows:

$$
\begin{aligned}
&\mathcal{I}^* \leftarrow \{\mathsf{e} \in \{0, ..., n\} | \mathsf{ChallinputEq}(\mathsf{e}) = \mathsf{true}\} \\
&\quad \mathsf{true} \leftarrow \mathsf{ChallinputEq}(\mathsf{e}) \iff \\
&\qquad (\mathsf{e} \in \mathcal{I}) \vee (\mathsf{ChallinputEq}(\mathsf{e}\text{-}1) \wedge \mathsf{e} \in \mathcal{T}^*) \vee (\mathsf{ChallinputEq}(\mathsf{e}+1) \wedge \mathsf{e}+1 \in \mathcal{T}^*)
\end{aligned}
$$

Additionally, if the adversary submits two ciphertexts $\bar{\mathrm{C}}_0, \bar{\mathrm{C}}_1$ as challenge (as in $\mathsf{xxIND\text{-}UPD\text{-}atk}$), we compute $\mathcal{I}_i, \mathcal{I}_i^*, i \in \{0, 1\}$ first and then use $\mathcal{I}^* = \mathcal{I}_0^* \cup \mathcal{I}_1^*$ to check the trivial win condition. An example of trivial win conditions $\mathcal{K}^* \cap \mathcal{C}^* \neq \emptyset$ and $\mathcal{I}^* \cap \mathcal{C}^* \neq \emptyset$ is provided in the full version [5].

We do not consider this trivial win condition for the ENC notion, as there is no ciphertext in the challenge input value, i.e. $\mathcal{I}^* = \emptyset$. Thus, the assessment $\mathcal{I}^* \cap \mathcal{C}^* \neq \emptyset$ in experiment $\mathbf{Exp}^{\mathsf{detIND\text{-}ENC\text{-}atk\text{-}b}}_{\mathsf{UE},\ \mathcal{A}}$ (see Fig. 5) will never be true.

Trivial wins via decryptions. The following is for analyzing $\mathsf{detIND\text{-}yy\text{-}CCA}$ notions, for $\mathsf{yy} \in \{\mathsf{UE}, \mathsf{ENC}, \mathsf{UPD}\}$, where the adversary has access to $\mathcal{O}.\mathsf{Dec}$. We follow the trivial win analysis in KLR19: suppose the adversary knows a challenge ciphertext $(\tilde{\mathrm{C}}, \mathsf{e}_0) \in \tilde{\mathcal{L}}$ and tokens from epoch $\mathsf{e}_0 + 1$ to epoch e, then the adversary can update the challenge ciphertext from epoch e_0 to epoch e. If $\mathcal{A}$ sends the updated ciphertext to $\mathcal{O}.\mathsf{Dec}$ this will reveal the underlying message, and $\mathcal{A}$ trivially wins the game: we shall exclude this type of attack.

Define $\tilde{\mathcal{L}}^*$ to be the extended set of $\tilde{\mathcal{L}}$ in which the adversary has learned or inferred information via token corruption. Whenever $\mathcal{O}.\mathsf{Dec}$ receives a ciphertext located in $\tilde{\mathcal{L}}^*$, the challenger will set the trivial win flag twf to be 1. The list $\tilde{\mathcal{L}}^*$ is updated while the security game is running. After the challenge query happens, the challenger updates $\tilde{\mathcal{L}}^*$ whenever an element is added to list $\tilde{\mathcal{L}}$ or a token is corrupted. In Fig. 10 we show how list $\tilde{\mathcal{L}}^*$ is updated.

Trivial Win Conditions in Ciphertext Integrity Games We again follow the trivial win analysis in KLR19. In ciphertext integrity games for updatable encryption, we do not consider the randomized update setting as the adversary can update an old ciphertext via a corrupted token to provide any number of new valid forgeries to the Try query to trivially win this game.

Trivial wins via keys. If an epoch key is corrupted, then the adversary can use this key to forge ciphertexts in this epoch. We exclude this trivial win: if the adversary provides a forgery in an epoch in list $\mathcal{K}^*$, the challenger sets twf to 1.

Trivial wins via ciphertexts. Suppose the adversary knows a ciphertext $(\mathrm{C}, \mathsf{e}_0) \in \mathcal{L}$ and tokens from epoch $\mathsf{e}_0 + 1$ to epoch e, then the adversary can provide a forgery by updating C to epoch e. We shall exclude this type of forgeries.

Define $\mathcal{L}^*$ to be the extended set of $\mathcal{L}$ in which the adversary has learned or inferred information via token corruption. If $\mathcal{O}.\mathsf{Try}$ receives a ciphertext located in $\mathcal{L}^*$, the challenger will set twf to 1. The list $\mathcal{L}^*$ is updated while the security game is running. Ciphertexts output by $\mathcal{O}.\mathsf{Enc}$ and $\mathcal{O}.\mathsf{Upd}$ are known to the adversary. Furthermore, whenever a token is corrupted, the challenger may update list $\mathcal{L}^*$ as well. In Fig. 11 we show how list $\mathcal{L}^*$ is updated.

3.3 Firewall Technique

In order to prove security for updatable encryption in the epoch-based model with strong corruption capabilities, cryptographic separation is required between the epochs in which the adversary knows key material, and those in which it knows challenge-equal ciphertexts (acquired/calculated via queries to $\mathcal{O}.\mathsf{Upd}\tilde{\mathrm{C}}$ and $\mathcal{O}.\mathsf{Corr}(\Delta)$). To ensure this, we follow prior work in explicitly defining the 'safe' or *insulated* regions, as we explain below. These regions insulate epoch keys, tokens and ciphertexts: outside of an insulated region a reduction in a security proof can generate keys and tokens itself, but within these regions it must embed its challenge while still providing the underlying adversary with

```
**if** challenge query **or** $\mathcal{O}.\mathsf{Upd}\tilde{\mathrm{C}}$ happens **then**
  $\tilde{\mathcal{L}}^* \leftarrow \tilde{\mathcal{L}}^* \cup \{(\tilde{\mathrm{C}}, \cdot)\}$
**if** $\mathsf{phase} = 1$ **and** $\mathcal{O}.\mathsf{Corr}(\mathsf{token}, \cdot)$ happens **then**
  **for** $i \in \mathcal{T}^*$ **and** $(\tilde{\mathrm{C}}_{i-1}, i-1) \in \tilde{\mathcal{L}}^*$ **do**
    $\tilde{\mathcal{L}}^* \leftarrow \tilde{\mathcal{L}}^* \cup \{(\tilde{\mathrm{C}}_i, i)\}$
```

Fig. 10. Updating list $\tilde{\mathcal{L}}^*$.

```
**if** $\mathcal{O}.\mathsf{Enc}$ **or** $\mathcal{O}.\mathsf{Upd}$ happens **then**
  $\mathcal{L}^* \leftarrow \mathcal{L}^* \cup \{(\cdot, \mathrm{C}, \cdot)\}$
**if** $\mathcal{O}.\mathsf{Corr}(\mathsf{token}, \cdot)$ happens **then**
  **for** $i \in \mathcal{T}^*$ **do**
    **for** $(j, \mathrm{C}_{i-1}, i-1) \in \mathcal{L}^*$ **do**
      $\mathrm{C}_i \leftarrow \mathsf{UE.Upd}(\Delta_i, \mathrm{C}_{i-1})$
      $\mathcal{L}^* \leftarrow \mathcal{L}^* \cup \{(j, \mathrm{C}_i, i)\}$
```

Fig. 11. Updating list $\mathcal{L}^*$.

access to the appropriate oracles. A thorough discussion of how we leverage these insulated regions in proofs is given in Section 5.3.

To understand the idea of firewalls, consider any security game (for bi-directional schemes) in which the trivial win conditions are *not* triggered. If the adversary $\mathcal{A}$ corrupts all tokens then either it never corrupts any keys or it never asks for a challenge ciphertext. Suppose that $\mathcal{A}$ does ask for a challenge ciphertext in epoch $\tilde{\mathsf{e}}$ [5]. Then there exists an (unique) epoch continuum around $\tilde{\mathsf{e}}$ such that no keys in this epoch continuum, and no tokens in the boundaries of this epoch continuum are corrupted. Moreover, we can assume that all tokens within this epoch continuum are corrupted, because once the adversary has finished corrupting keys, it can corrupt any remaining tokens that do not 'touch' those corrupted keys. This observation is first used in the IND-UPD proof of RISE provided by Lehmann and Tackmann [21], and Klooß et al. [17] provided an extended description of this 'key insulation' technique. We name these epoch ranges *insulated regions* and their boundaries to be *firewalls*.

Definition 4. *An* insulated region *with* firewalls fwl *and* fwr *is a consecutive sequence of epochs* $(\mathsf{fwl}, \ldots, \mathsf{fwr})$ *for which:*

- *no key in the sequence of epochs* $(\mathsf{fwl}, \ldots, \mathsf{fwr})$ *is corrupted;*
- *the tokens* Δ_{fwl} *and* $\Delta_{\mathsf{fwr}+1}$ *are not corrupted (if they exist);*
- *all tokens* $(\Delta_{\mathsf{fwl}+1}, \ldots, \Delta_{\mathsf{fwr}})$ *are corrupted (if any exist).*

We denote the firewalls bordering the special insulated region that contains $\tilde{\mathsf{e}}$ as $\hat{\mathsf{fwl}}$ and $\hat{\mathsf{fwr}}$ – though note that there could be (many, distinct) insulated regions elsewhere in the epoch continuum. Specifically, when the adversary asks for updated versions of the challenge ciphertext, the epoch in which this query occurs must also fall within (what the challenger later calculates as) an insulated region. In the full version [5] we give an algorithm for computing firewall locations. The list $\mathcal{FW}$ tracks, and appends a label to, each insulated region and its firewalls. Observe that if an epoch is a left firewall, then neither the key nor the token for that epoch are corrupted. From the left firewall, since we assume that all tokens are corrupted, track to the right until either a token is not corrupted or a key is.

4 On the Security of Updates

In this section we present a new notion of security for updatable encryption schemes, which we denote xxIND-UE-atk. This notion captures both security of fresh encryptions (i.e. implies xxIND-ENC-atk) and unlinkability (i.e. implies xxIND-UPD-atk). We first explain the new notion and then describe its relation to previous notions. Then, we prove a generic relationship among CPA, CTXT and CCA to complete the picture for security notions for UE schemes.

[5] In the situation that the adversary does not corrupt any keys to the left or the right (or both) of the challenge epoch, the insulated region thus extends to the boundary (or boundaries) of the epoch continuum.

4.1 A New Definition of Confidentiality

In the security game for xxIND-UE-atk, the adversary submits one message and a ciphertext from an earlier epoch that the adversary received via a call to $\mathcal{O}$.Enc. The challenger responds with either an encryption of that message or an update of that earlier ciphertext, in the challenge (current) epoch $\tilde{\mathsf{e}}$.

Definition 5 **(xxIND-UE-atk).** *Let* $\mathsf{UE} = \{\mathsf{UE.KG}, \mathsf{UE.TG}, \mathsf{UE.Enc}, \mathsf{UE.Dec}, \mathsf{UE.Upd}\}$ *be an updatable encryption scheme. Then the* xxIND-UE-atk *advantage, for* $(\mathsf{xx}, \mathsf{atk}) \in \{(\mathsf{det}, \mathsf{CPA}), (\mathsf{rand}, \mathsf{CPA}), (\mathsf{det}, \mathsf{CCA})\}$*, of an adversary* $\mathcal{A}$ *against* UE *is defined as*

$$\mathbf{Adv}^{\mathsf{xxIND\text{-}UE\text{-}atk}}_{\mathsf{UE},\ \mathcal{A}}(\lambda) = \left|\mathbf{Pr}[\mathbf{Exp}^{\mathsf{xxIND\text{-}UE\text{-}atk\text{-}1}}_{\mathsf{UE},\ \mathcal{A}} = 1] - \mathbf{Pr}[\mathbf{Exp}^{\mathsf{xxIND\text{-}UE\text{-}atk\text{-}0}}_{\mathsf{UE},\ \mathcal{A}} = 1]\right|$$

where the experiment $\mathbf{Exp}^{\mathsf{xxIND\text{-}UE\text{-}atk\text{-}b}}_{\mathsf{UE},\ \mathcal{A}}$ *is given in Fig. 3, Fig. 5 and Fig. 12.*

Note that randIND-UE-CPA is strictly stronger than detIND-UE-CPA, since the adversary has strictly more capabilities. A generalized version of xxIND-UE-atk, denoted xxIND-UE*-atk, is also given in Fig. 12. In this game the input challenge ciphertext can come from (i.e. be known to $\mathcal{A}$ in) any prior epoch, not just the epoch immediately before $\tilde{\mathsf{e}}$. Note that xxIND-UE-atk is a special case of xxIND-UE*-atk. Under some fairly weak requirements (that all schemes discussed in this paper satisfy) we can prove that xxIND-UE-atk implies xxIND-UE*-atk – this analysis is given in the full version [5].

```
$\mathbf{Exp}^{\mathsf{xxIND\text{-}UE\text{-}atk\text{-}b}}_{\mathsf{UE},\ \mathcal{A}}(\lambda)$ :
  $(\bar{\mathrm{M}}, \bar{\mathrm{C}}) \leftarrow \mathcal{A}$
  Create $\tilde{\mathrm{C}}$ with $(\bar{\mathrm{M}}, \bar{\mathrm{C}})$ :
    if $(\bar{\mathrm{C}}, \tilde{\mathsf{e}} - 1) \notin \mathcal{L}$ then
      return $\perp$
    if $\mathrm{b} = 0$ then
      $\tilde{\mathrm{C}} \leftarrow \mathsf{UE.Enc}(\mathrm{k}_{\tilde{\mathsf{e}}}, \bar{\mathrm{M}})$
    else
      $\tilde{\mathrm{C}} \leftarrow \mathsf{UE.Upd}(\Delta_{\tilde{\mathsf{e}}}, \bar{\mathrm{C}})$
    return $\tilde{\mathrm{C}}$
```

```
$\mathbf{Exp}^{\mathsf{xxIND\text{-}UE^*\text{-}atk\text{-}b}}_{\mathsf{UE},\ \mathcal{A}}(\lambda)$ :
  $(\bar{\mathrm{M}}, (\bar{\mathrm{C}}, \mathsf{e}')) \leftarrow \mathcal{A}$
  Create $\tilde{\mathrm{C}}$ with $(\bar{\mathrm{M}}, (\bar{\mathrm{C}}, \mathsf{e}'))$ :
    if $(\bar{\mathrm{C}}, \mathsf{e}') \notin \mathcal{L}$ then
      return $\perp$
    if $\mathrm{b} = 0$ then
      $\tilde{\mathrm{C}}_{\tilde{\mathsf{e}}} \leftarrow \mathsf{UE.Enc}(\mathrm{k}_{\tilde{\mathsf{e}}}, \bar{\mathrm{M}})$
    else
      $\tilde{\mathrm{C}}_{\mathsf{e}'} \leftarrow \bar{\mathrm{C}}$
      for $j \in \{\mathsf{e}'+1, \ldots, \tilde{\mathsf{e}}\}$ do
        $\tilde{\mathrm{C}}_j \leftarrow \mathsf{UE.Upd}(\Delta_j, \tilde{\mathrm{C}}_{j-1})$
    return $\tilde{\mathrm{C}}_{\tilde{\mathsf{e}}}$
```

Fig. 12. Challenge call definition for xxIND-UE-atk and xxIND-UE*-atk security experiments, the full experiment is defined in Fig. 3, Fig. 5.

Remark 3. The definition of xxIND-UE-atk is more concise and intuitively easier to understand than xxIND-UE*-atk, however in the full version [5] we show that xxIND-UE-atk $\Longleftrightarrow$ xxIND-UE*-atk. This result and our generic proof techniques mean that all results in this paper that hold for xxIND-UE-atk, also hold for xxIND-UE*-atk, and vice versa.

$$\begin{tikzcd}
\begin{array}{c}\mathsf{IND\text{-}ENC\text{-}CPA}\\ +\mathsf{randIND\text{-}UPD\text{-}CPA}\end{array} \arrow[d, Rightarrow, "/"{marking}, "*"] & & & \begin{array}{c}\mathsf{IND\text{-}ENC\text{-}CPA}\\ +\mathsf{detIND\text{-}UPD\text{-}CPA}\end{array} \arrow[d, Rightarrow, "/"{marking}, "*"] & & \begin{array}{c}\mathsf{IND\text{-}ENC\text{-}CCA}\\ +\mathsf{detIND\text{-}UPD\text{-}CCA}\end{array} \arrow[d, Rightarrow, "/"{marking}, "*"] \\
\mathsf{randIND\text{-}UE\text{-}CPA} \arrow[rrr, Rightarrow, shift left, "\text{Def. 5}"] \arrow[ddd, Rightarrow, bend right, "\text{Thm. 1.1}"'] \arrow[dr, Rightarrow, "*"] & & & \mathsf{detIND\text{-}UE\text{-}CPA} \arrow[lll, Rightarrow, shift left, "/"{marking}, "*"] \arrow[rr, Rightarrow, "+\mathsf{CTXT}", "\text{Thm. 1.2}"'] \arrow[ddl, Rightarrow, "*"'] \arrow[d, Rightarrow, "*"] & & \mathsf{detIND\text{-}UE\text{-}CCA} \arrow[d, Rightarrow, "*"'] \arrow[dd, Rightarrow, bend left, "*"] \\
& \mathsf{randIND\text{-}UPD\text{-}CPA} & & \mathsf{detIND\text{-}UPD\text{-}CPA} \arrow[rr, Rightarrow, "+\mathsf{CTXT}", "\text{Thm. 1.2}"'] & & \mathsf{detIND\text{-}UPD\text{-}CCA} \\
\mathsf{IND\text{-}ENC\text{-}CPA} & & \mathsf{IND\text{-}ENC\text{-}CPA} \arrow[rrr, Rightarrow, "+\mathsf{CTXT}", "\text{Thm. 1.2}"'] & & & \mathsf{detIND\text{-}ENC\text{-}CCA}
\end{tikzcd}$$

Fig. 13. Relations among confidentiality notions $\mathsf{xxIND\text{-}yy\text{-}atk}$ for $\mathsf{xx} \in \{\mathsf{det}, \mathsf{rand}\}$, $\mathsf{yy} \in \{\mathsf{UE}, \mathsf{ENC}, \mathsf{UPD}\}$, $\mathsf{atk} \in \{\mathsf{CPA}, \mathsf{CCA}\}$, and ciphertext integrity ($\mathsf{INT\text{-}CTXT}$). Results that are given only in the full version [5] are marked with $*$.

Remark 4. In the full version [5] we show that the RISE scheme presented by LT18 is randIND-UE-CPA secure under DDH. While this result is perhaps unsurprising, the proof techniques we use are novel. We give an Oracle-DDH-like game that inherits the epoch-based nature of the updatable encryption security model, and then use this as a bridge to prove security.

4.2 Relations among Security Notions

In Fig. 13 we show the relationship between the new and existing UE security notions. Note that our new notion is strictly stronger than the xxIND-ENC-atk and xxIND-UPD-atk notions presented in prior work, and is in fact stronger than the combination of the prior notions. Further, we show that the generic relation among CPA, CTXT and CCA, that CPA security coupled with ciphertext integrity implies CCA security, also holds for updatable encryption schemes.

Theorem 1 (Informal Theorem). *The relationship among the security notions* xxIND-UE-atk*,* xxIND-ENC-atk *and* xxIND-UPD-atk *are as in Fig. 13. The relationship is proven via Theorems in the full version [5] and due to space constraints we show Theorems 1.1 and Theorem 1.2 only.*

Theorem 1.1 *Let* $\mathsf{UE} = \{\mathsf{UE.KG}, \mathsf{UE.TG}, \mathsf{UE.Enc}, \mathsf{UE.Dec}, \mathsf{UE.Upd}\}$ *be an updatable encryption scheme. For any* IND-ENC-CPA *adversary* $\mathcal{A}$ *against* UE*, there exists an* randIND-UE-CPA *adversary* $\mathcal{B}_{1.1}$ *against* UE *such that*

$$\mathbf{Adv}^{\mathsf{IND\text{-}ENC\text{-}CPA}}_{\mathsf{UE},\ \mathcal{A}}(\lambda) \leq 2 \cdot \mathbf{Adv}^{\mathsf{randIND\text{-}UE\text{-}CPA}}_{\mathsf{UE},\ \mathcal{B}_{1.1}}(\lambda).$$

Proof. We construct a reduction $\mathcal{B}_{1.1}$ running the randIND-UE-CPA experiment which will simulate the responses of queries made by the IND-ENC-CPA adversary $\mathcal{A}$. To provide a valid non-challenge ciphertext to its own challenger, $\mathcal{B}_{1.1}$ must run $\mathcal{A}$ out of step with its own game, so epoch 0 as far as $\mathcal{A}$ is concerned is actually epoch 1 for $\mathcal{B}_{1.1}$, and so on.

1. $\mathcal{B}_{1.1}$ chooses $\mathrm{b} \xleftarrow{\$} \{0,1\}$.
2. $\mathcal{B}_{1.1}$ receives the setup parameters from its randIND-UE-CPA challenger, chooses $\mathrm{M} \xleftarrow{\$} \mathcal{MS}$ and calls $\mathcal{O}.\mathsf{Enc}(\mathrm{M})$ which returns some C^0. Then $\mathcal{B}_{1.1}$ calls $\mathcal{O}.\mathsf{Next}$ once and sends the setup parameters to $\mathcal{A}$.
3. (a) Whenever $\mathcal{B}_{1.1}$ receives the queries $\mathcal{O}.\mathsf{Enc}, \mathcal{O}.\mathsf{Upd}, \mathcal{O}.\mathsf{Corr}$ from $\mathcal{A}$, $\mathcal{B}_{1.1}$ sends these queries to its randIND-UE-CPA challenger, and forwards the responses to $\mathcal{A}$.
   (b) Whenever $\mathcal{O}.\mathsf{Next}$ is called by $\mathcal{A}$, $\mathcal{B}_{1.1}$ randomly chooses a message $\mathrm{M} \xleftarrow{\$} \mathcal{MS}$ and calls $\mathcal{O}.\mathsf{Enc}(\mathrm{M})$ to receive some C^{e}, and then calls $\mathcal{O}.\mathsf{Next}$.
4. At some point, in epoch $\tilde{\mathsf{e}}$ (for its game), $\mathcal{B}_{1.1}$ receives the challenge query $(\bar{\mathrm{M}}_0, \bar{\mathrm{M}}_1)$ from $\mathcal{A}$. Then $\mathcal{B}_{1.1}$ sends the pair $(\bar{\mathrm{M}}_{\mathrm{b}}, \mathrm{C}^{\tilde{\mathsf{e}}-1})$ as challenge to its own randIND-UE-CPA challenger. After receiving the challenge ciphertext, $\tilde{\mathrm{C}}_{\tilde{\mathsf{e}}}$, from its challenger, $\mathcal{B}_{1.1}$ sends $\tilde{\mathrm{C}}_{\tilde{\mathsf{e}}}$ to $\mathcal{A}$.
5. $\mathcal{B}_{1.1}$ continues to answer $\mathcal{A}$'s queries using its own oracles, now including the challenge ciphertext update oracle $\mathcal{O}.\mathsf{Upd}\tilde{\mathrm{C}}$.
6. Finally $\mathcal{B}_{1.1}$ receives the output bit b' from $\mathcal{A}$. If $\mathrm{b} = \mathrm{b}'$ then $\mathcal{B}_{1.1}$ returns 0. Otherwise $\mathcal{B}_{1.1}$ returns 1.

We now bound the advantage of $\mathcal{B}_{1.1}$. The point is that whenever $\mathcal{B}_{1.1}$ returns a random encryption to $\mathcal{A}$, $\mathcal{B}_{1.1}$'s probability of winning is exactly $1/2$ because the bit b' from $\mathcal{A}$ is independent of its choice of b. This happens with probability $1/2$. However, when $\mathcal{B}_{1.1}$ returns a "correct" value to $\mathcal{A}$ (an encryption of $\bar{\mathrm{M}}_0$ or $\bar{\mathrm{M}}_1$), then $\mathcal{B}_{1.1}$'s probability of winning is the same as the probability that $\mathcal{A}$ wins. First note that, as usual,

$$\mathbf{Adv}^{\mathsf{randIND\text{-}UE\text{-}CPA}}_{\mathsf{UE},\mathcal{B}_{1.1}} = |\mathbf{Pr}[\mathbf{Exp}^{\mathsf{randIND\text{-}UE\text{-}CPA\text{-}1}}_{\mathsf{UE},\ \mathcal{B}_{1.1}} = 1] - \mathbf{Pr}[\mathbf{Exp}^{\mathsf{randIND\text{-}UE\text{-}CPA\text{-}0}}_{\mathsf{UE},\ \mathcal{B}_{1.1}} = 1]|.$$

We claim that $\mathbf{Pr}[\mathbf{Exp}^{\mathsf{randIND\text{-}UE\text{-}CPA\text{-}1}}_{\mathsf{UE},\ \mathcal{B}_{1.1}} = 1] = 1/2$ because in this case $\tilde{\mathrm{C}}_{\tilde{\mathsf{e}}}$ is independent of b and so b' must also be independent of b. Then we have:

$$\begin{aligned}\mathbf{Adv}^{\mathsf{randIND\text{-}UE\text{-}CPA}}_{\mathsf{UE},\mathcal{B}_{1.1}} &= \left|\frac{1}{2} - \mathbf{Pr}[\mathbf{Exp}^{\mathsf{randIND\text{-}UE\text{-}CPA\text{-}0}}_{\mathsf{UE},\ \mathcal{B}_{1.1}} = 1]\right| \\ &= \left|\frac{1}{2} - \left(\frac{1}{2}\cdot\mathbf{Pr}[\mathbf{Exp}^{\mathsf{IND\text{-}ENC\text{-}CPA\text{-}0}}_{\mathsf{UE},\ \mathcal{A}} = 1] + \frac{1}{2}\cdot\mathbf{Pr}[\mathbf{Exp}^{\mathsf{IND\text{-}ENC\text{-}CPA\text{-}1}}_{\mathsf{UE},\ \mathcal{A}} = 0]\right)\right| \\ &= \frac{1}{2}\cdot\mathbf{Adv}^{\mathsf{IND\text{-}ENC\text{-}CPA}}_{\mathsf{UE},\ \mathcal{A}}.\end{aligned}$$

□

The three separation arrows at the top of Fig. 13 are all demonstrated in the same manner. Begin with a scheme UE that meets both of the two notions at the very top of the figure. All algorithms for UE' are the same as for UE, except $\mathsf{UE}'.\mathsf{Enc}$ is defined by modifying $\mathsf{UE.Enc}$ to append the epoch number in which the ciphertext was initially created (and $\mathsf{UE}'.\mathsf{Dec}$ ignores this appended value). This does not affect an adversary's ability to win the $\mathsf{IND\text{-}ENC\text{-}atk}$ or $\mathsf{xxIND\text{-}UPD\text{-}atk}$ games but trivially breaks $\mathsf{xxIND\text{-}UE\text{-}atk}$ security.

Generic Composition. The following theorem tells the relation among CPA, CTXT and CCA security. The full proof is given in the full version [5].

Theorem 1.2 *Let* $\mathsf{UE} = \{\mathsf{UE.KG}, \mathsf{UE.TG}, \mathsf{UE.Enc}, \mathsf{UE.Dec}, \mathsf{UE.Upd}\}$ *be an updatable encryption scheme. For any* $\mathsf{detIND\text{-}yy\text{-}CCA}$ *adversary* $\mathcal{A}$ *against* UE*, there exists an* $\mathsf{INT\text{-}CTXT}$ *adversary* $\mathcal{B}_{1.2a}$ *and an* $\mathsf{detIND\text{-}yy\text{-}CPA}$ *adversary* $\mathcal{B}_{1.2b}$ *against* UE *such that*

$$\mathbf{Adv}^{\mathsf{detIND\text{-}yy\text{-}CCA}}_{\mathsf{UE},\ \mathcal{A}}(\lambda) \leq 2\mathbf{Adv}^{\mathsf{INT\text{-}CTXT}}_{\mathsf{UE},\ \mathcal{B}_{1.2a}}(\lambda) + \mathbf{Adv}^{\mathsf{detIND\text{-}yy\text{-}CPA}}_{\mathsf{UE},\ \mathcal{B}_{1.2b}}(\lambda)$$

where $\mathsf{yy} \in \{\mathsf{UE}, \mathsf{ENC}, \mathsf{UPD}\}$.

Proof sketch. The proof is adapted from the proof of Theorem 3.2 of Bellare and Namprempre [2]. We modify the $\mathsf{detIND\text{-}yy\text{-}CCA}$ game such that in the new game the decryption oracle will answer $\perp$ if the input is a fresh ciphertext.

In more detail, suppose $\mathcal{A}$ is an adversary playing the new game, then we can then construct a reduction playing $\mathsf{detIND\text{-}yy\text{-}CPA}$ game and simulating the responses to $\mathcal{A}$. To answer decryption oracle queries made by $\mathcal{A}$, the reduction performs bookkeeping for ciphertexts and messages to successfully respond to the decryption of already-existing ciphertexts, or simply replying $\perp$ for the decryption of fresh ciphertexts. So the advantage of winning the new game is upper-bounded by the $\mathsf{detIND\text{-}yy\text{-}CPA}$ advantage.

Furthermore, notice that two games are identical until a valid fresh ciphertext is sent to the decryption oracle. Which means the probability of an adversary these games is upper-bounded by the probability that a valid fresh forgery is produced: this successful adversary can win the $\mathsf{INT\text{-}CTXT}$ game. Therefore, the difference between the new game and the original $\mathsf{detIND\text{-}yy\text{-}CCA}$ game can be upper-bounded by the $\mathsf{INT\text{-}CTXT}$ advantage. □

5 The SHINE Schemes

We now describe our new UE scheme SHINE (Secure Homomorphic Ideal-cipher Nonce-based Encryption). The encryption algorithm uses a permutation to obfuscate the input to the exponentiation function. Updating a ciphertext simply requires exponentiation once by the update token, which itself is the quotient of the current epoch key and the previous epoch key. The scheme comes in three flavors: SHINE0 is presented in Fig. 14 and takes in short messages and only uses a single permutation. The second flavor, MirrorSHINE, is provided in the full version [5] and runs two different permutations with the same input. The

```
SHINE0.KG(λ):
  $\mathrm{k} \xleftarrow{\$} \mathbb{Z}_q^*$
  return k

SHINE0.TG($\mathrm{k_e}, \mathrm{k_{e+1}}$):
  $\Delta_{\mathrm{e+1}} \leftarrow \frac{\mathrm{k_{e+1}}}{\mathrm{k_e}}$
  return $\Delta_{\mathrm{e+1}}$

SHINE0.Enc($\mathrm{k_e}, \mathrm{M}$):
  $\mathrm{N} \xleftarrow{\$} \mathcal{N}$
  $\mathrm{C_e} \leftarrow (\pi(\mathrm{N}\|\mathrm{M}\|0^{\mathrm{t}}))^{\mathrm{k_e}}$
  return $\mathrm{C_e}$

SHINE0.Dec($\mathrm{k_e}, \mathrm{C_e}$):
  $a \leftarrow \pi^{-1}(\mathrm{C_e}^{1/\mathrm{k_e}})$
  parse† $a$ as $\mathrm{N}'\|\mathrm{M}'\|\mathrm{Z}$
  if $\mathrm{Z} = 0^{\mathrm{t}}$ then
    return $\mathrm{M}'$
  else
    return $\perp$

SHINE0.Upd($\Delta_{\mathrm{e+1}}, \mathrm{C_e}$):
  $\mathrm{C_{e+1}} \leftarrow (\mathrm{C_e})^{\Delta_{\mathrm{e+1}}}$
  return $\mathrm{C_{e+1}}$
```

Fig. 14. Updatable encryption scheme SHINE0. †: $\|\mathrm{N}'\| = \mathrm{v}, \|\mathrm{M}'\| = \mathrm{m}, \|\mathrm{Z}\| = \mathrm{t}$. Note that there may be an additional embedding step after the permutation π, as discussed in Section 5.6.

third flavor OCBSHINE is given in Fig. 16 and is for applications with arbitrarily long messages, using a family of permutations.

We discuss implementation details of the SHINE schemes in Section 5.6. In particular, for each scheme in the SHINE suite, it is necessary to embed the output of the permutation (a regular block cipher) into an appropriate DDH-hard group.

Our proofs of security, given as Theorem 2 (Theorem 3), bound an adversary's detIND-UE-CPA (INT-CTXT$^{\mathsf{s}}$) advantage by DDH (CDH), and are provided in the ideal cipher model. Furthermore, combining the results of Theorem 1.2, Theorem 2 and Theorem 3, we have that the suite of SHINE schemes (i.e. SHINE0, MirrorSHINE and OCBSHINE) are detIND-UE-CCA secure.

5.1 Construction of SHINE schemes

SHINE via Zero Block: SHINE0. Suppose a message space of $\mathcal{MS} = \{0,1\}^{\mathrm{m}}$ and random nonce space $\mathcal{N} = \{0,1\}^{\mathrm{v}}$. The encryption algorithm feeds as input to the permutation a nonce, the message, and a zero string. The decryption algorithm will return $\perp$ if the decrypted value does not end with 0^{t}. The SHINE0 scheme is defined in Fig. 14. If ciphertext integrity is not required (or file/ciphertext integrity is performed in some other manner), then SHINE0 without the zero block results in a scheme (denoted SHINE0[CPA]) that is still detIND-UE-CPA secure.

SHINE for Long Messages via Checksum: OCBSHINE. The schemes SHINE0 and MirrorSHINE both require that the message space be smaller than the size of an element of the exponentiation group. This ciphertext expansion is undesirable in many practical scenarios, and so we wish to construct a SHINE scheme which gives us (almost) no ciphertext expansion and can be applied to arbitrarily long messages. We build a new SHINE scheme, OCBSHINE, with these properties.

The construction of OCBSHINE is inspired by the authenticated encryption scheme OCB [24]. Different from OCB mode, the nonce is encrypted inside the ciphertext instead of sending it along with the ciphertext. In order to determine the length of the last message block, the encryption algorithm of OCB mode removes some bits of the last ciphertext block to reveal this information. However in our setting, the output of the permutations are (mapped to) the input of the exponentiation function: thus all bits of permutation outputs must be included. Therefore, OCBSHINE includes the length of the last message block in the first ciphertext component. If ciphertext integrity is not required, then OCBSHINE can be improved by removing the last ciphertext block.

OCBSHINE is formally defined in Fig. 16 and the encryption process is pictorially represented in Fig. 15; we give an intuitive description here. Suppose the blocksize is m, and assume the encryption algorithm OCBSHINE.Enc has input message M. By “partition M into $\mathrm{M}_1, \dots, \mathrm{M}_l$” we mean setting $l \leftarrow \max\{\lceil |\mathrm{M}|/m \rceil, 1\}$ and dividing M into l blocks, i.e. $\mathrm{M}_1, \dots, \mathrm{M}_l$, where $|\mathrm{M}_1| = \dots = |\mathrm{M}_{l-1}| = m$. The last message block M_l is padded with zeros to make it length m before computing the permutation output and the checksum, i.e $\mathrm{M}_l \| 0^*$ with $|\mathrm{M}_l \| 0^*| = m$. Let $a = \lceil \log(m) \rceil$, so the length of M_l ($|\mathrm{M}_l| \leq m$) can be written as an a-bit representation.

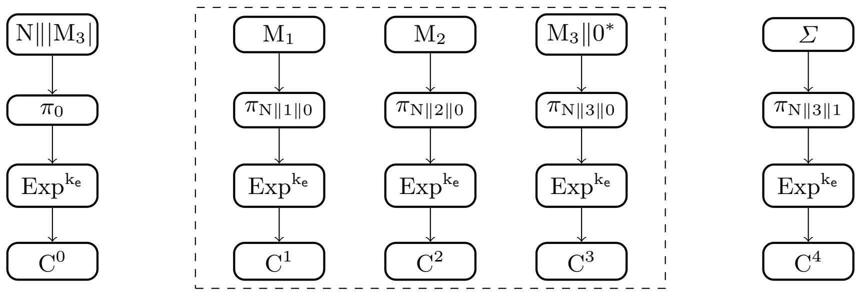

Fig. 15. Diagram describing how the OCBSHINE encryption algorithm works on message $\mathrm{M} = (\mathrm{M}_1, \mathrm{M}_2, \mathrm{M}_3)$. $\Sigma = \mathrm{M}_1 \oplus \mathrm{M}_2 \oplus \mathrm{M}_3 \| 0^*$. There may be an additional embedding step after the permutations, as discussed in Section 5.6.

Let Perm(m) be the set of all permutations on $\{0,1\}^m$. Randomly choose $\pi_0 \xleftarrow{\$} \mathrm{Perm}(m)$, and use this permutation to randomize the concatenation of the nonce N and an a-bit representation of the last message block length. Then, index the (random) permutations used to encrypt message blocks by the nonce and a counter. Let Perm(S, m) be the set of all mappings from S to permutations on $\{0,1\}^m$. Suppose the nonce space is $\mathcal{N} = \{0,1\}^{m-a}$, $S = \mathcal{N} \times \mathbb{N}^* \times \{0,1\}$, for each ($\mathrm{N} \in \mathcal{N}, i \in \mathbb{N}^*, b \in \{0,1\}$), set $\pi_{\mathrm{N}\|i\|b} \xleftarrow{\$} \mathrm{Perm}(\mathcal{N} \times \mathbb{N}^* \times \{0,1\}, m)$, which

```
$\underline{\mathsf{OCBSHINE.KG}(m):}$
  $\mathrm{k} \xleftarrow{\$} \mathbb{Z}_q^*$
  return $\mathrm{k}$

$\underline{\mathsf{OCBSHINE.TG}(\mathrm{k_e}, \mathrm{k_{e+1}}):}$
  $\Delta_{\mathrm{e+1}} \leftarrow \frac{\mathrm{k_{e+1}}}{\mathrm{k_e}}$
  return $\Delta_{\mathrm{e+1}}$

$\underline{\mathsf{OCBSHINE.Enc}(\mathrm{k_e}, \mathrm{M}):}$
  partition M into $\mathrm{M}_1, \ldots, \mathrm{M}_l$
  $\Sigma \leftarrow \oplus_{i=1}^{l-1} \mathrm{M}_i \oplus \mathrm{M}_l \| 0^*$
  $\mathrm{N} \xleftarrow{\$} \mathcal{N}$
  $\mathrm{C}^0 \leftarrow \left(\pi_0(\mathrm{N} \| |\mathrm{M}_l|)\right)^{\mathrm{k_e}}$
  $\mathrm{C}^{l+1} \leftarrow \left(\pi_{\mathrm{N}\|l\|1}(\Sigma)\right)^{\mathrm{k_e}}$
  for $i = 1, \ldots, l\text{-}1$ do
    $\mathrm{C}^i \leftarrow \left(\pi_{\mathrm{N}\|i\|0}(\mathrm{M}_i)\right)^{\mathrm{k_e}}$
  $\mathrm{C}^l \leftarrow \left(\pi_{\mathrm{N}\|l\|0}(\mathrm{M}_l \| 0^*)\right)^{\mathrm{k_e}}$
  $\mathrm{C_e} \leftarrow (\mathrm{C}^0, \ldots, \mathrm{C}^l, \mathrm{C}^{l+1})$
  return $\mathrm{C_e}$

$\underline{\mathsf{OCBSHINE.Dec}(\mathrm{k_e}, \mathrm{C_e}):}$
  parse $\mathrm{C_e} = (\mathrm{C}^0, \ldots, \mathrm{C}^l, \mathrm{C}^{l+1})$
  $\mathrm{N}' \| A' \leftarrow \pi_0^{-1}\left(\left(\mathrm{C}^0\right)^{1/\mathrm{k_e}}\right)$
  $\Sigma' \leftarrow \pi_{\mathrm{N}'\|l\|1}^{-1}\left(\left(\mathrm{C}^{l+1}\right)^{1/\mathrm{k_e}}\right)$
  for $i = 1, \ldots, l$ do
    $\mathrm{M}_i' \leftarrow \pi_{\mathrm{N}'\|i\|0}^{-1}\left(\left(\mathrm{C}^i\right)^{1/\mathrm{k_e}}\right)$
  if $\Sigma' = \oplus_{i=1}^{l} \mathrm{M}_i'$ then
    $\mathrm{M}' \leftarrow (\mathrm{M}_1', \ldots, \mathrm{M}_l'[\text{first } A'\text{-bit}])$
    return $\mathrm{M}'$
  else
    return $\perp$

$\underline{\mathsf{OCBSHINE.Upd}(\Delta_{\mathrm{e+1}}, \mathrm{C_e}):}$
  parse $\mathrm{C_e} = (\mathrm{C}^0, \ldots, \mathrm{C}^l, \mathrm{C}^{l+1})$
  for $i = 0, \ldots, l+1$ do
    $\mathrm{C}_{\mathrm{e+1}}^i \leftarrow (\mathrm{C}_{\mathrm{e}}^i)^{\Delta_{\mathrm{e+1}}}$
  return $\mathrm{C_{e+1}}$
```

Fig. 16. Updatable encryption scheme OCBSHINE. Note that there may be an additional embedding step after the permutations, as discussed in Section 5.6.

form a random permutation family: we use these permutations to randomize message blocks and the checksum.

5.2 Security - SHINE is detIND-UE-CPA, INT-CTXT, detIND-UE-CCA Secure

All three SHINE schemes, i.e. SHINE0, MirrorSHINE and OCBSHINE, have the same security properties, and the proofs are very similar for each flavor. We refer to SHINE to mean the family containing all these three schemes. In Theorem 2, we show that SHINE is detIND-UE-CPA in the ideal cipher model, if DDH holds. In Theorem 3, we show that SHINE is INT-CTXT$^{\mathsf{s}}$, and therefore INT-CTXT (INT-CTXT and INT-CTXT$^{\mathsf{s}}$ are equivalent, recall Section 3), in the ideal cipher model, if CDH holds. The loss incurred by this proof is the normal $(n+1)^3$ (or $(n+1)^2$ for INT-CTXT) and also the number of encryption queries the adversary makes before it makes its challenge: to avoid the issues described in Section 5.3 we not only need to guess the locations of the challenge firewalls but also the ciphertext that the adversary will submit as its challenge.

The ideal cipher model, a version of which was initially given by Shannon [26] and shown to be equivalent to the random oracle model by Coron et al. [8], gives all parties access to a permutation chosen randomly from all possible key-permutation possibilities of appropriate length. The SHINE schemes exponentiate the output of the permutation by the epoch key to encrypt, so our reduction can 'program' the transformation from permutation outputs to group elements.

In the following two Theorems we detail the security properties met by SHINE, i.e. detIND-UE-CPA, INT-CTXT and thus detIND-UE-CCA. Note that this is the strongest known security property for updatable encryption schemes with deterministic updates. In Section 5.3 we discuss the challenges that arise in the proofs of these two theorems, and in Section 5.4 and Section 5.5 we describe the novel techniques and methods used in the proofs. Full proofs are provided in the full version [5].

Theorem 2 **(SHINE is detIND-UE-CPA).** *Let* SHINE $\in$ {SHINE0, MirrorSHINE, OCBSHINE} *be the UE scheme described above. For any ideal cipher model adversary $\mathcal{A}$ (that makes max $\mathrm{Q_E}$ encryption queries before its challenge), there exists an adversary $\mathcal{B}_2$ such that*

$$\mathbf{Adv}^{\mathsf{detIND\text{-}UE\text{-}CPA}}_{\mathsf{SHINE},\ \mathcal{A}}(\lambda) \leq O(1)(n+1)^3 \cdot \mathrm{Q_E} \cdot \mathbf{Adv}^{\mathsf{DDH}}_{\mathbb{G},\ \mathcal{B}_2}(\lambda).$$

Theorem 3 **(SHINE is INT-CTXT$^{\mathsf{s}}$).** *Let* SHINE $\in$ {SHINE0, MirrorSHINE, OCBSHINE} *be the UE scheme described above. For any ideal cipher model adversary $\mathcal{A}$ (that makes max $\mathrm{Q_E}$ encryption queries before calling $\mathcal{O}$.Try), there exists an adversary $\mathcal{B}_3$ such that*

$$\mathbf{Adv}^{\mathsf{INT\text{-}CTXT^s}}_{\mathsf{SHINE},\ \mathcal{A}}(\lambda) \leq O(1)(n+1)^2 \cdot \mathrm{Q_E} \cdot \mathbf{Adv}^{\mathsf{CDH}}_{\mathcal{B}_3} + \textit{negligible terms}$$

Remark 5. Combining the results of Theorem 1.2, Theorem 2 and Theorem 3, we have that SHINE is detIND-UE-CCA.

5.3 Proof Challenges in Schemes with Deterministic Updates

In each variant of SHINE all ciphertext components are raised to the epoch key, so the update mechanism transforms a ciphertext for epoch e to one for $\mathsf{e}+1$ by raising this value to $\frac{\mathrm{k_{e+1}}}{\mathrm{k_e}}$. We now highlight the difficulties in creating security proofs for such 'single-component' updatable encryption schemes. Randomness is used in creation of the initial ciphertext (via N) but updates are completely deterministic, and thus in any reduction it is necessary to provide consistent ciphertexts to the adversary (i.e. the N value must be consistent). The (cryptographic) separation gained by using the firewall technique (see Section 3.3 for discussion and definition) assists with producing (updates of) non-challenge ciphertexts, but embedding any challenge value while also providing answers to the $\mathcal{O}$.Corr queries of the underlying adversary is very challenging.

The regular key insulation technique as introduced by LT18 – where the reduction constructs one hybrid for each epoch – does not work. Specifically, in any reduction to a DDH-like assumption, it is not possible to provide a challenge ciphertext in a left or right sense (to the left of this challenge ciphertext are of some form, and to the right of this challenge ciphertext are of some other form) if the underlying adversary asks for tokens around the challenge epoch:

deterministic updates mean that tokens will make these ciphertexts of the same form and this gap will be easily distinguishable.

We counteract this problem by constructing a *hybrid argument across insulated regions*. This means that in each hybrid, we can embed at one firewall of the insulated region, and simulate all tokens within that insulated region to enable answering queries to both $\mathcal{O}.\mathsf{Upd}$ and $\mathcal{O}.\mathsf{Upd}\tilde{\mathrm{C}}$. The reduction's distinguishing task is thus ensured to be at the boundaries of the insulated regions, the firewalls, so any (non-trivial) win for the underlying adversary is ensured to carry through directly to the reduction.

5.4 Proof Method for Confidentiality: Constructing a Hybrid argument across Insulated Regions

The confidentiality proof of $\mathsf{SHINE0}$ is extendable to $\mathsf{MirrorSHINE}$ and $\mathsf{OCBSHINE}$, so we only show proof method of $\mathsf{SHINE0}$. We now explain how we bound the advantage of any adversary playing the $\mathsf{detIND\text{-}UE\text{-}CPA}$ game for $\mathsf{SHINE0}$ by the advantage of a reduction playing DDH.

We apply the firewall technique to set up hybrid games such that in hybrid i, we embed within the i-th insulated region: this means that to the left of the i-th insulated region the game responds with the $\mathrm{b} = 1$ case of the $\mathsf{detIND\text{-}UE\text{-}CPA}$ experiment, and to the right of the i-th insulated region it gives an encryption of the challenge input message as output, i.e. $\mathrm{b} = 0$. This means we have one hybrid for each insulated region, moving left-to-right across the epoch space.

We construct a reduction $\mathcal{B}$ playing the DDH experiment in hybrid i. Initially, $\mathcal{B}$ guesses the location of the i-th insulated region. If the underlying adversary has performed a corrupt query within this insulated region that would lead to the reduction failing, the reduction aborts the game. In the full version [5] we detail a dedicated algorithm for checking this event.

In particular, within the insulated region, the reduction can simulate challenge ciphertexts and non-challenge ciphertexts using its DDH tuple. Furthermore, ciphertexts can be moved around within the insulated region by tokens.

Remark 6. We note that the problem of challenge insulation in schemes with deterministic updates was also observed independently by Klooß et al. [[18], § B.2]. Their solution (though in the different context of CCA security of UE with certain properties) is to form a hybrid argument with a hybrid for each epoch, and essentially guess an epoch r which is the first token 'after' the hybrid index that the adversary has not corrupted, and use the inherent 'gap' in the adversary's knowledge continuum to replace challenge updates across this gap with encryptions of just one of the challenge messages. It is not clear if this approach would work for showing $\mathsf{detIND\text{-}UE\text{-}CPA}$ (or $\mathsf{IND\text{-}ENC\text{-}CPA}$) of $\mathsf{SHINE0}$. We conjecture that even if it were possible to construct a reduction in this vein, our approach enables a more direct proof: in particular we do not need to assume specific additional properties of the UE scheme in question for it to work.

5.5 Proof Method for Integrity

The integrity proof of SHINE0 is extendable to MirrorSHINE and OCBSHINE, so we only show proof method of SHINE0. In the INT-CTXT$^{\mathsf{s}}$ game, the challenger will keep a list of consistent values for ciphertexts (i.e. the underlying permutation output $\pi(\mathrm{N}\|\mathrm{M}\|0^l)$). Suppose $\tilde{\mathrm{C}}$ is a forgery attempt sent to the $\mathcal{O}.\mathsf{Try}$ query in epoch $\tilde{\mathsf{e}}$. Let $\tilde{c} = (\tilde{\mathrm{C}})^{1/\mathrm{k}_{\tilde{\mathsf{e}}}}$ be the underlying permutation output. We claim that if $\tilde{c}$ is a new value, then the adversary wins the game with negligible probability; if $\tilde{c}$ is a value that existed before, then the probability that the adversary wins the game can be bounded by the CDH advantage.

If $\tilde{c}$ is a new value, since π is a random permutation, then the INT-CTXT$^{\mathsf{s}}$ challenger simulates the preimage of $\tilde{c}$ under π to be a random string. So the probability that this random string ends with a (fixed length) zero block is negligible, and this carries over to the probability that the adversary wins the INT-CTXT$^{\mathsf{s}}$ game. If $\tilde{c}$ is an already-existing value, and suppose this event happens with probability p. We construct a reduction playing the CDH game such that it wins CDH game with probability $p \cdot \frac{1}{\mathrm{Q_E}(n+1)^2}$. Similar to the proof method of confidentiality, we construct a reduction playing the CDH experiment by guessing the location of the firewalls around the challenge epoch. The reduction embeds the CDH value and simulates the INT-CTXT$^{\mathsf{s}}$ game, using any successfully-forged ciphertext to compute the CDH output to its CDH challenger.

5.6 Implementing the SHINE schemes

In the proofs of Theorem 2 and Theorem 3, we require that π is a random (unkeyed) permutation which must be followed by a mapping to an appropriate group for exponentiation by the epoch key. For the permutation we do not need any specific and strong properties that are provided by modern constructions of block ciphers and sponges. As far as the proof goes, and in practice, the property that we want from this permutation is that given a ciphertext and the inverse of the epoch key k_{e}, the only way to extract useful information about the message is to apply the inverse permutation π^{-1}. The random permutation model (or ideal cipher model) is thus the tool we need here to create a simple interface for this aspect of our proof.

The different members of the SHINE family are suited to different application scenarios. The variants SHINE0 and MirrorSHINE are best suited to cases where messages are of small, fixed size, such as customer credentials (or phone contact details, to return to the motivating example in the Introduction). For applications with longer messages (i.e. larger than the size of the exponentiation group), OCBSHINE is considerably faster and we will assume that these choices are made in our implementation suggestions. This removes any need for larger groups in order to encrypt longer messages. Using larger groups would not only carry a significant performance penalty, but also force us to construct custom large blocklength block ciphers. Although this can be done (and has been for RSA groups [14], where our approach would not work), the analysis is tricky.

Instantiating the ideal permutation. The message block in SHINE0, MirrorSHINE and the final message block in OCBSHINE must be appropriately padded to allow application of the permutation. The permutation could be deployed using a variable-output-length sponge construction, a block cipher or an authenticated encryption scheme with a fixed key and suitably large nonce space. In practice, we suggest to instantiate the random permutation with a block cipher of a suitable block length. AES has only 128-bit blocks which does not match the minimum required size of the group, so we instead suggest block ciphers such as Threefish, or original Rijndael, allowing block lengths of 256 or 512 bits.

Mapping to elliptic curve group. We would like to instantiate our groups using elliptic curves. Using modern techniques it is always possible to find a suitable curve over a field with a size matching the block length of the ideal permutation, but using standard curves like NIST P-256 or P-521 seems desirable. A standard approach [19] is to embed bit strings in the X-coordinate of a point as follows. Note that close to half the field elements are X-coordinates of points. Given a field of size q, we consider a t-bit block as an integer x_0 and find a small integer u such that $u2^t + x_0$ is the X-coordinate of a curve point. If $\log q - t$ is between 8 and 9, this will fail to terminate with probability around 2^{-256} under reasonable assumptions.

With this approach we could use Threefish with 512-bit blocks together with NIST P-521 curve. If we want to use 256-bit blocks from Threefish, or original Rijndael, together with NIST P-256 curve, we can use a standard block cipher iteration trick [23] to reduce the block length from 256 bits, so that embedding in the X-coordinate still works, as follows. With block length $t + \tau$, concatenate a t-bit block with τ leading zeros and apply the block cipher until the τ leading bits of the result are all zeros. Discard these zeros to get a t-bit block. This is fairly cheap as for our purposes 8 or 9 bits will do.

Note that we have constructed an injective embedding of a block into an elliptic curve, not a bijection as assumed in our proofs. When we sample group elements in our proof, we must take care to sample points in the image of our embedding, but this can be done cheaply.

Acknowledgements. This research was funded by the Research Council of Norway under Project No. 248166. The second author is supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme, grant agreement 802823. We would like to thank Frederik Armknecht, Håvard Raddum and Mohsen Toorani for fruitful discussions in the initial stages of this project, and anonymous reviewers for a number of useful suggestions for improvement.

References

1. Ateniese, G., Fu, K., Green, M., Hohenberger, S.: Improved proxy re-encryption schemes with applications to secure distributed storage. In: Proceedings of NDSS 2005. The Internet Society (2005). https://doi.org/10.1145/1127345.1127346

2. Bellare, M., Namprempre, C.: Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. J. Cryptology **21**(4), 469–491 (2008). https://doi.org/10.1007/s00145-008-9026-x
3. Boneh, D., Lewi, K., Montgomery, H.W., Raghunathan, A.: Key homomorphic PRFs and their applications. In: Canetti, R., Garay, J.A. (eds.) Proceedings of CRYPTO 2013 I. LNCS, vol. 8042, pp. 410–428. Springer (2013). https://doi.org/10.1007/978-3-642-40041-4_23
4. Boneh, D., Lewi, K., Montgomery, H.W., Raghunathan, A.: Key homomorphic PRFs and their applications. IACR Cryptology ePrint Archive, Report 2015/220 (2015), http://eprint.iacr.org/2015/220
5. Boyd, C., Davies, G.T., Gjøsteen, K., Jiang, Y.: Fast and secure updatable encryption. IACR Cryptology ePrint Archive, Report 2019/1457 (2019), https://eprint.iacr.org/2019/1457
6. Cachin, C., Camenisch, J., Freire-Stögbuchner, E., Lehmann, A.: Updatable tokenization: Formal definitions and provably secure constructions. In: Kiayias, A. (ed.) Proceedings of Financial Cryptography and Data Security 2017. LNCS, vol. 10322, pp. 59–75. Springer (2017). https://doi.org/10.1007/978-3-319-70972-7_4
7. Canetti, R., Hohenberger, S.: Chosen-ciphertext secure proxy re-encryption. In: Ning, P., di Vimercati, S.D.C., Syverson, P.F. (eds.) Proceedings of ACM CCS 2007. pp. 185–194. ACM (2007). https://doi.org/10.1145/1315245.1315269
8. Coron, J., Patarin, J., Seurin, Y.: The random oracle model and the ideal cipher model are equivalent. In: Wagner, D.A. (ed.) Proceedings of CRYPTO 2008. LNCS, vol. 5157, pp. 1–20. Springer (2008). https://doi.org/10.1007/978-3-540-85174-5_1
9. Council, P.S.S.: Data security standard (PCI DSS v3.2.1) (2018), https://www.pcisecuritystandards.org/
10. Davidson, A., Deo, A., Lee, E., Martin, K.: Strong post-compromise secure proxy re-encryption. In: Jang-Jaccard, J., Guo, F. (eds.) Proceedings of ACISP 2019. LNCS, vol. 11547, pp. 58–77. Springer (2019). https://doi.org/10.1007/978-3-030-21548-4_4
11. Diaz-Santiago, S., Rodríguez-Henríquez, L.M., Chakraborty, D.: A cryptographic study of tokenization systems. In: Obaidat, M.S., Holzinger, A., Samarati, P. (eds.) Proceedings of SECRYPT 2014. pp. 393–398. SciTePress (2014). https://doi.org/10.5220/0005062803930398
12. Everspaugh, A., Paterson, K.G., Ristenpart, T., Scott, S.: Key rotation for authenticated encryption. In: Katz, J., Shacham, H. (eds.) Proceedings of CRYPTO 2017 (III). LNCS, vol. 10403, pp. 98–129. Springer (2017). https://doi.org/10.1007/978-3-319-63697-9_4
13. Everspaugh, A., Paterson, K.G., Ristenpart, T., Scott, S.: Key rotation for authenticated encryption. IACR Cryptology ePrint Archive, Report 2017/527 (2017), http://eprint.iacr.org/2017/527
14. Gentry, C., O'Neill, A., Reyzin, L.: A unified framework for trapdoor-permutation-based sequential aggregate signatures. In: Abdalla, M., Dahab, R. (eds.) Public-Key Cryptography – PKC 2018. pp. 34–57. Springer International Publishing, Cham (2018)
15. Jarecki, S., Krawczyk, H., Resch, J.K.: Updatable oblivious key management for storage systems. In: Cavallaro, L., Kinder, J., Wang, X., Katz, J. (eds.) Proceedings of ACM CCS 2019. pp. 379–393. ACM (2019). https://doi.org/10.1145/3319535.3363196
16. Kallahalla, M., Riedel, E., Swaminathan, R., Wang, Q., Fu, K.: Plutus: Scalable secure file sharing on untrusted storage. In: Chase, J. (ed.) Proceedings of FAST 2003. USENIX (2003). https://doi.org/10.5555/1090694.1090698

17. Klooß, M., Lehmann, A., Rupp, A.: (R)CCA secure updatable encryption with integrity protection. In: Ishai, Y., Rijmen, V. (eds.) Proceedings of EUROCRYPT 2019 (I). LNCS, vol. 11476, pp. 68–99. Springer (2019). https://doi.org/10.1007/978-3-030-17653-2_3
18. Klooß, M., Lehmann, A., Rupp, A.: (R)CCA secure updatable encryption with integrity protection. IACR Cryptology ePrint Archive, Report 2019/222 (2019), https://eprint.iacr.org/2019/222
19. Koblitz, N.: Elliptic curve cryptosystems. Mathematics of Computation **48**(177), 203–209 (1987)
20. Lee, E.: Improved security notions for proxy re-encryption to enforce access control. In: Lange, T., Dunkelman, O. (eds.) Proceedings of LATINCRYPT 2017. LNCS, vol. 11368, pp. 66–85. Springer (2017). https://doi.org/10.1007/978-3-030-25283-0_4
21. Lehmann, A., Tackmann, B.: Updatable encryption with post-compromise security. In: Nielsen, J.B., Rijmen, V. (eds.) Proceedings of EUROCRYPT 2018 III. LNCS, vol. 10822, pp. 685–716. Springer (2018). https://doi.org/10.1007/978-3-319-78372-7_22
22. Myers, S., Shull, A.: Practical revocation and key rotation. In: Smart, N.P. (ed.) Proceedings of CT-RSA 2018. LNCS, vol. 10808, pp. 157–178. Springer (2018). https://doi.org/10.1007/978-3-319-76953-0_9
23. Rivest, R.L., Shamir, A., Adleman, L.M.: A method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM **21**, 120–126 (1978)
24. Rogaway, P., Bellare, M., Black, J., Krovetz, T.: OCB: a block-cipher mode of operation for efficient authenticated encryption. In: Reiter, M.K., Samarati, P. (eds.) Proceedings of ACM CCS 2001. pp. 196–205. ACM (2001). https://doi.org/10.1145/501983.502011
25. Sakurai, K., Nishide, T., Syalim, A.: Improved proxy re-encryption scheme for symmetric key cryptography. In: Proceedings of IWBIS 2017. pp. 105–111. IEEE (2017). https://doi.org/10.1109/IWBIS.2017.8275110
26. Shannon, C.E.: Communication theory of secrecy systems. Bell system technical journal **28**(4), 656–715 (1949)