Improving Speed and Security in Updatable Encryption Schemes

Dan Boneh[1], Saba Eskandarian[1(✉)], Sam Kim[1,2], and Maurice Shih[3]

[1] Stanford University, Stanford, CA, USA
saba@cs.stanford.edu
[2] Simons Institute for the Theory of Computing, Berkeley, CA, USA
[3] Cisco Systems, San Jose, CA, USA

Abstract. Periodic key rotation is a common practice designed to limit the long-term power of cryptographic keys. Key rotation refers to the process of re-encrypting encrypted content under a fresh key, and overwriting the old ciphertext with the new one. When encrypted data is stored in the cloud, key rotation can be very costly: it may require downloading the entire encrypted content from the cloud, re-encrypting it on the client's machine, and uploading the new ciphertext back to the cloud.

An *updatable encryption scheme* is a symmetric-key encryption scheme designed to support efficient key rotation in the cloud. The data owner sends a short *update token* to the cloud. This update token lets the cloud rotate the ciphertext from the old key to the new key, without learning any information about the plaintext. Recent work on updatable encryption has led to several security definitions and proposed constructions. However, existing constructions are not yet efficient enough for practical adoption, and the existing security definitions can be strengthened.

In this work we make three contributions. First, we introduce stronger security definitions for updatable encryption (in the *ciphertext-dependent* setting) that capture desirable security properties not covered in prior work. Second, we construct two new updatable encryption schemes. The first construction relies only on symmetric cryptographic primitives, but only supports a bounded number of key rotations. The second construction supports a (nearly) unbounded number of updates, and is built from the Ring Learning with Errors (RLWE) assumption. Due to complexities of using RLWE, this scheme achieves a slightly weaker notion of integrity compared to the first. Finally, we implement both constructions and compare their performance to prior work. Our RLWE-based construction is 200× faster than a prior proposal for an updatable encryption scheme based on the hardness of elliptic curve DDH. Our first construction, based entirely on symmetric primitives, has the highest encryption throughput, approaching the performance of AES, and the highest decryption throughput on ciphertexts that were re-encrypted fewer than fifty times. For ciphertexts re-encrypted over fifty times, the RLWE construction dominates it in decryption speed.

The full version of this paper is available at https://eprint.iacr.org/2020/222.pdf.

S. Moriai and H. Wang (Eds.): ASIACRYPT 2020, LNCS 12493, pp. 559–589, 2020.
https://doi.org/10.1007/978-3-030-64840-4_19

1 Introduction

Consider a ciphertext ct that is a symmetric encryption of some data using key k. Key rotation is the process of decrypting ct using k, and re-encrypting the result using a fresh key k' to obtain a new ciphertext ct'. One then stores ct' and discards ct. Periodic key rotation is recommended, and even required, in several security standards and documents, including NIST publication 800-57 [7], the Payment Card Industry Data Security Standard (PCI DSS) [25], and Google's cloud security recommendations [17].

Key rotation can be expensive when the ciphertext is stored in the cloud, and the cloud does not have access to the keys. Key rotation requires the client to retrieve all the encrypted data from the cloud, re-encrypt it by decrypting with the old key and re-encrypting with the new key, and then upload the resulting ciphertext back to the cloud. The traffic to and from the cloud can incur significant networking costs when large amounts of data are involved. Alternatively, the client can send the old and the new key to the cloud, and have the cloud re-encrypt in place, but this gives the cloud full access to the data in the clear. We note that either way, the cloud must be trusted to discard the old ciphertext.

Updatable encryption [11,12,15,20,21] is a much better approach to key rotation for encrypted data stored in the cloud. Updatable encryption is a symmetric encryption scheme that supports the standard key-generation, encryption, and decryption algorithms, along with two additional algorithms called $\mathsf{ReKeyGen}$ and $\mathsf{ReEncrypt}$ used for key rotation. The re-key generation algorithm is invoked as $\mathsf{ReKeyGen}(\mathsf{k}, \mathsf{k}') \rightarrow \Delta$, taking as input a pair of keys, k and k', and outputting a short "update token" Δ, also called a re-encryption key. The re-encryption algorithm is invoked as $\mathsf{ReEncrypt}(\Delta, \mathsf{ct}) \rightarrow \mathsf{ct}'$, taking as input a short Δ and a ciphertext ct encrypted under k, and outputting an updated ciphertext ct' that is the encryption of the same data as in ct, but encrypted under k'.

If the client's data is encrypted using an updatable encryption scheme, then the client can use the re-key generation algorithm $\mathsf{ReKeyGen}$ to generate a short update token Δ to send the cloud. The cloud then runs the re-encryption algorithm $\mathsf{ReEncrypt}$ to update all the client's ciphertexts. As before, the cloud must be trusted to discard the old ciphertexts.

Defining Security. Intuitively, the update token Δ must not reveal any "useful" information to the cloud. This was formalized by Boneh et al. [11] against passive adversaries, and was improved and extended to provide security against active adversaries by Everspaugh et al. [15].

However, we show in Sect. 3 that these existing elegant definitions can be insufficient, and may not prevent some undesirable information leakage. In particular, we give a simple construction that satisfies the existing definitions, and yet an observer can easily learn the age of a ciphertext, namely the number of times that the ciphertext was re-encrypted since it was initially created. Ideally, this information should not leak to an observer who only sees the ciphertext. This issue was recently independently pointed out in [12].

The age of a ciphertext (i.e., the number of times that the ciphertext was re-encrypted) can leak sensitive private information about the plaintext in many real-world situations. We give two illustrative examples assuming an annual key rotation policy is in use:

- Consider a national database managed in the cloud where information about each individual is stored in a single fixed-size encrypted record. Suppose a newborn is recorded in the database at birth. If an annual key rotation policy is used, and records are encrypted using a scheme that leaks the number of key rotations, then an adversary (or a cloud administrator), who examines the stored ciphertexts will learn every person's age, even though age is regarded as personal identifiable information (PII) and must be protected.
- Consider a dating app, like Tinder or Match.com, that maintains customer information in an encrypted cloud storage. The number of key-updates on a person's file can indicate how long the person has been a customer, which is sensitive information that should be protected.

To address this definitional shortcoming, we define a stronger confidentiality property that requires that a re-encrypted ciphertext is always computationally indistinguishable from a freshly generated ciphertext, no matter how many times it was re-encrypted (Sects. 3.2 and 3.3). This ensures that an observer who sees the encrypted content at a particular point in time, cannot tell the ciphertext age. We also strengthen the integrity definition of [15] to cover additional tampering attacks, as discussed in Sect. 3.4.

Constructing Updatable Encryption. Next, we look for efficient constructions that satisfy our definitions. We give two new constructions: one based on nested authenticated encryption and another based on the Ring Learning With Errors (RLWE) problem [23,26].

Our first construction, presented in Sect. 4, makes use of carefully designed nested encryption, and can be built from any authenticated encryption cipher. It satisfies our strong confidentiality and integrity requirements, so that an adversary cannot learn the age of a ciphertext. However, the scheme only supports a bounded number of re-encryptions, where the bound is set when the initial ciphertext is created. Another limitation of this scheme is that decryption time grows linearly with the age of the ciphertext. Hence, the scheme is practical as long as the maximum number of re-encryptions is not too large. Our implementation and experiments, discussed below, make this precise.

Our second construction, presented in Sect. 5, makes use of an almost key-homomorphic PRF (KH-PRF) built from the RLWE problem. Recall that a key-homomorphic PRF (KH-PRF) [11,24] is a secure PRF $F : \mathcal{K} \times \mathcal{X} \rightarrow \mathcal{Y}$, where $(\mathcal{K}, +)$ and $(\mathcal{Y}, +)$ are finite groups, and the PRF is homomorphic with respect to its key, namely $F(k_1, x)+F(k_2, x) = F(k_1+k_2, x)$ for all $k_1, k_2 \in \mathcal{K}$ and $x \in \mathcal{X}$. We say that the PRF is an *almost* KH-PRF if the equality above holds up to a small additive error (see Definition 2.1). To see why a KH-PRF is useful for updatable encryption, consider a single message block $m_i \in \mathcal{Y}$ that is encrypted using counter mode as $\mathsf{ct}_i \leftarrow \mathsf{m}_i + F(\mathsf{k}, i)$, for some $i \in \mathcal{X}$ and $\mathsf{k} \in \mathcal{K}$. To rotate the key, the client chooses a new key $\mathsf{k}' \leftarrow \mathcal{K}$ and sends $\Delta = \mathsf{k}' - \mathsf{k} \in \mathcal{K}$ to the cloud. The cloud computes $\mathsf{ct}'_i = \mathsf{ct}_i + F(\Delta, i)$, which by the key-homomorphic property satisfies $\mathsf{ct}'_i = \mathsf{m}_i + F(\mathsf{k}', i)$, as required.

It remains an open challenge to construct a secure KH-PRF whose performance is comparable to AES. However, there are several known algebraic constructions. In the random oracle model [8,16], there is a simple KH-PRF based on the Decision Diffie-Hellman (DDH) assumption [24], and a simple almost KH-PRF based on the Learning With Rounding (LWR) problem [11]. There are also several KH-PRFs whose security does not depend on random oracles, as discussed in the related work section.

Everspaugh et al. [15] construct an updatable encryption scheme that supports unbounded key updates by combining a key-homomorphic PRF with authenticated encryption and a collision-resistant hash function. They evaluate their construction using the KH-PRF derived from DDH, in the random oracle model, instantiated in the 256-bit elliptic curve Curve25519 [9]. We show that the Everspaugh et al. [15] construction satisfies our new confidentiality security definitions for updatable encryption. However, compared to our first nested encryption construction that relies only on generic authenticated encryption, the implementation of the Everspaugh et al. construction is much slower as it uses expensive group operations.

In our second updatable encryption scheme, we significantly improve on the performance of the Everspaugh et al. [15] construction by extending it to work with an *almost* key-homomorphic PRF. Our construction supports nearly unbounded key-updates, and outperforms the Everspaugh et al. construction by 200× in speed. The high performance of the scheme is, in part, due to a new almost KH-PRF construction from the RLWE assumption. Almost KH-PRFs can already be constructed from the (Ring-) Learning with Rounding (RLWR) assumption [6,11]. However, we observe that for the specific setting of updatable encryption, the parameters of the PRF can be further optimized by modifying the existing PRF constructions to base security directly on the standard RLWE assumption. We provide the details of our construction in Sect. 6.

The use of an *almost* key-homomorphic PRF leads to some complications. First, there is a small ciphertext expansion to handle the noise that arises from the imperfection of the KH-PRF key-homomorphism. More importantly, due to the noisy nature of the ciphertext, we show that an adversary may gain information about the age of the corresponding plaintext using a chosen ciphertext attack, which violates our new security definition. Therefore, while this construction is attractive due to its performance, it can only be used in settings where revealing the age of a ciphertext is acceptable. In Sect. 5.3 we capture this security property using a relaxed notion of ciphertext integrity, and show that the scheme is secure in this model.

Implementation and Experiments. In Sect. 7, we experiment with our two updatable encryption schemes and measure their performance. For our first construction based on authenticated encryption, we measure the trade-off between its efficiency and the number of key rotations it can support. Based on our evaluation, our first construction performs better than the other schemes in both speed and ciphertext size, as long as any given ciphertext is to be re-encrypted at most twenty times over the course of its lifetime. It outperforms the other schemes in speed (but not in ciphertext size) as long as ciphertexts are re-encrypted at most fifty times.

For our second construction, which uses an almost key-homomorphic PRF based on RLWE, we compare its performance with that of Everspaugh et al. [15], which uses a key-homomorphic PRF over Curve25519. Since we use an almost key-homomorphic PRF that is inherently noisy, any message to be encrypted must be padded on the right to counteract the noise. Therefore, compared to the elliptic-curve based construction of Everspaugh et al., our construction produces larger ciphertexts (32% larger than those of Everspaugh et al.). However, in terms of speed, our implementation shows that our construction outperforms that of Everspaugh et al. by over 200×. We provide a more detailed analysis in Sect. 7. Implementations of both our constructions are open source and available at [1].

Summary of Our Contributions. Our contributions are threefold. First, we strengthen the definition of updatable encryption to provide stronger confidentiality and integrity guarantees. Second, we propose two new constructions. Finally, we experiment with both constructions and report on their real world performance and ciphertext expansion. Encryption throughput of our first construction, while allowing only a bounded number of key rotations, is close to the performance of AES. Our second construction, based on a key-homomorphic PRF from RLWE, is considerably faster than the previous construction of Everspaugh et al. [15], which is based on elliptic curves.

1.1 Related Work

Two Flavors of Updatable Encryption. There are two flavors of updatable encryption: *ciphertext-dependent* schemes [11,15] and *ciphertext-independent* schemes [12,20,21]. In a ciphertext-dependent updatable encryption scheme, the client can re-download a tiny fraction of the ciphertext that is stored by the server before generating the update tokens. In a ciphertext-independent updatable encryption scheme, the client generates its update token without needing to download any components of its ciphertext. In this work, we focus on the ciphertext-dependent setting, where constructions are considerably more efficient. We provide a detailed comparison of the two settings in the full version [10]. Additional discussion of the two models can be found in [21].

Key-Homomorphic PRFs. The concept of key-homomorphic PRFs was introduced by Naor, Pinkas, and Reingold [24], and was first formalized as a cryptographic primitive by Boneh et al. [11], who construct two KH-PRFs secure without random oracles: one from LWE, and another from multilinear maps. They also observe that any seed homomorphic PRG $G : \mathcal{S} \rightarrow \mathcal{S}^2$ gives a key-homomorphic PRF. More constructions for key-homomorphic PRFs from LWE include [5,13,19].

2 Preliminaries

Basic Notation. For an integer $n \geq 1$, we write $[n]$ to denote the set of integers $\{1, \ldots, n\}$. For a distribution $\mathcal{D}$, we write $x \leftarrow \mathcal{D}$ to denote that x is sampled from $\mathcal{D}$; for a finite set S, we write $x \xleftarrow{\mathrm{R}} S$ to denote that x is sampled uniformly from S. We say that a family of distributions $\mathcal{D} = \{\mathcal{D}_\lambda\}_{\lambda \in \mathbb{N}}$ is *B-bounded* if the support of $\mathcal{D}$ is $\{-B, \ldots, B-1, B\}$ with probability 1.

Unless specified otherwise, we use λ to denote the security parameter. We say a function $f(\lambda)$ is negligible in λ, denoted by $\mathsf{negl}(\lambda)$, if $f(\lambda) = o(1/\lambda^c)$ for all $c \in \mathbb{N}$. We say an algorithm is efficient if it runs in probabilistic polynomial time in the length of its input. We use $\mathsf{poly}(\lambda)$ to denote a quantity whose value is bounded by a fixed polynomial in λ.

To analyze the exact security of our constructions in Sects. 4 and 5, we parameterize the security of these notions with respect to *advantage functions* $\varepsilon : \mathbb{N} \to \mathbb{R}$ that bound the probability of an efficient adversary breaking the security of the primitive.

Basic Cryptographic Primitives. We use a number of standard cryptographic tools throughout the paper, including collision-resistant hash functions, PRGs, PRFs, and authenticated encryption, definitions of which we provide in the full version of this work [10].

Key-Homomorphic PRFs. In this work, we use a special family of pseudorandom functions called *key-homomorphic PRFs* (KH-PRFs) that satisfy additional algebraic properties. Specifically, the key space $\mathcal{K}$ and the range $\mathcal{Y}$ of the PRF exhibit certain group structures such that evaluation of the PRF on any fixed input $x \in \mathcal{X}$ is homomorphic with respect to these group structures. We formally define a key-homomorphic PRF in the full version [10].

We also work with a slight relaxation of the notion of key-homomorphic PRFs. Namely, instead of requiring that the PRF outputs are perfectly homomorphic with respect to the PRF keys, we require that they are "almost" homomorphic in that $F(k_1, x) \otimes F(k_2, x) \approx F(k_1 \oplus k_2, x)$. Formally, we define an almost key-homomorphic PRF as follows.

Definition 2.1 (Almost Key-Homomorphic PRFs [11]). *Let $(\mathcal{K}, \oplus)$ be a group and let m and q be positive integers. Then, an efficiently computable deterministic function $F : \mathcal{K} \times \mathcal{X} \to \mathbb{Z}_q^m$ is a γ-almost key-homomorphic PRF if*

- *F is a secure PRF [10].*
- *For every key $k_1, k_2 \in \mathcal{K}$ and every $x \in \mathcal{X}$, there exists a vector $\mathbf{e} \in [0, \gamma]^m$ such that*

$$F(k_1, x) + F(k_2, x) = F(k_1 \oplus k_2, x) + \mathbf{e} \pmod{q}.$$

Authenticated Encryption. For our updatable encryption scheme in Sect. 4, we make use of authenticated encryption schemes that satisfy a stronger confidentiality requirement than the standard security requirement. Namely, we rely on authenticated encryption schemes that satisfy *ciphertext pseudorandomness*, which requires that an encryption of any message is computationally indistinguishable from a random string of suitable length. We provide the formal definitions in the full version [10]. Authenticated encryption schemes that satisfy ciphertext pseudorandomness can be constructed from pseudorandom functions or blockciphers in a standard way. Widely-used modes for authenticated encryption such as AES-GCM also satisfy ciphertext pseudorandomness.

3 New Definitions for Updatable Encryption

In this section, we present new security definitions for updatable encryption in the ciphertext dependent setting. Our definitions build upon and strengthen the confidentiality and integrity definitions for an updatable authenticated encryption scheme from Everspaugh et al. [15]. We start by defining the syntax for an updatable encryption scheme and its compactness and correctness conditions in Sect. 3.1. We then present security definitions for confidentiality and integrity, comparing each to prior definitions as we present them.

3.1 Updatable Encryption Syntax

For ciphertext-dependent updatable encryption schemes, it is useful to denote ciphertexts as consisting of two parts: a short ciphertext header $\hat{\mathsf{ct}}$, which the client can download to generate its update token, and a ciphertext body ct that encrypts the actual plaintext.

Formally, we define the syntax for an updatable encryption scheme as follows. To emphasize the ciphertext integrity properties of our constructions in Sect. 4 and Sect. 5, we refer to an updatable encryption scheme as an *updatable authenticated encryption* scheme in our definitions.

Definition 3.1 (Updatable Authenticated Encryption). *An* updatable authenticated encryption *(UAE) scheme for a message space $\mathcal{M} = (\mathcal{M}_\lambda)_{\lambda \in \mathbb{N}}$ is a tuple of efficient algorithms $\Pi_{\mathsf{UAE}} = (\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{ReKeyGen}, \mathsf{ReEncrypt}, \mathsf{Decrypt})$ that have the following syntax:*

- $\mathsf{KeyGen}(1^\lambda) \to \mathsf{k}$*: On input a security parameter λ, the key generation algorithm returns a secret key* k.
- $\mathsf{Encrypt}(\mathsf{k}, \mathsf{m}) \to (\hat{\mathsf{ct}}, \mathsf{ct})$*: On input a key* k *and a message* $\mathsf{m} \in \mathcal{M}_\lambda$*, the encryption algorithm returns a ciphertext header* $\hat{\mathsf{ct}}$ *and a ciphertext body* ct.
- $\mathsf{ReKeyGen}(\mathsf{k}_1, \mathsf{k}_2, \hat{\mathsf{ct}}) \to \Delta_{1,2,\hat{\mathsf{ct}}}/\perp$*: On input two keys* $\mathsf{k}_1, \mathsf{k}_2$*, and a ciphertext header* $\hat{\mathsf{ct}}$*, the re-encryption key generation algorithm returns an update token* $\Delta_{1,2,\hat{\mathsf{ct}}}$ *or* $\perp$.
- $\mathsf{ReEncrypt}(\Delta, (\hat{\mathsf{ct}}, \mathsf{ct})) \to (\hat{\mathsf{ct}}', \mathsf{ct}')/\perp$*: On input an update token* Δ*, and a ciphertext* $(\hat{\mathsf{ct}}, \mathsf{ct})$*, the re-encryption algorithm returns a new ciphertext* $(\hat{\mathsf{ct}}', \mathsf{ct}')$ *or* $\perp$.
- $\mathsf{Decrypt}(\mathsf{k}, (\hat{\mathsf{ct}}, \mathsf{ct})) \to \mathsf{m}/\perp$*: On input a key* k*, and a ciphertext* $(\hat{\mathsf{ct}}, \mathsf{ct})$*, the decryption algorithm returns a message* m *or* $\perp$.

A trivial way of achieving an updatable authenticated encryption scheme is to allow a client to re-download the entire ciphertext, re-encrypt it, and send it back to the server. Therefore, for a UAE scheme to be useful and meaningful, we require that communication between the client and server be bounded and independent of the size of the message encrypted in the ciphertext to be updated. This is captured by the compactness property, which requires that any ciphertext header and update token have lengths that depend only on the security parameter.

Definition 3.2 (Compactness). *We say that an updatable authenticated encryption scheme* $\Pi_{\mathsf{UAE}} = (\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{ReKeyGen}, \mathsf{ReEncrypt}, \mathsf{Decrypt})$ *for a message space* $\mathcal{M} = (\mathcal{M}_\lambda)_{\lambda \in \mathbb{N}}$ *is* compact *if there exist polynomials* $f_1(\cdot)$, $f_2(\cdot)$ *such that for any* $\lambda \in \mathbb{N}$ *and message* $\mathsf{m} \in \mathcal{M}_\lambda$*, we have (with probability 1)*

$$|\hat{\mathsf{ct}}| \le f_1(\lambda), \qquad |\Delta_{1,2,\hat{\mathsf{ct}}}| \le f_2(\lambda),$$

where $\mathsf{k}_1, \mathsf{k}_2 \leftarrow \mathsf{KeyGen}(1^\lambda)$*,* $(\hat{\mathsf{ct}}, \mathsf{ct}) \leftarrow \mathsf{Encrypt}(\mathsf{k}_1, \mathsf{m})$*, and* $\Delta_{1,2,\hat{\mathsf{ct}}} \leftarrow \mathsf{ReKeyGen}(\mathsf{k}_1, \mathsf{k}_2, \hat{\mathsf{ct}})$*. That is, the lengths of the ciphertext header and update token are independent of the message length.*

The correctness condition for an updatable encryption scheme is defined in a natural way.

Definition 3.3 (Correctness). *We say that an updatable authenticated encryption scheme* $\Pi_{\mathsf{UAE}} = (\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{ReKeyGen}, \mathsf{ReEncrypt}, \mathsf{Decrypt})$ *for a message space* $\mathcal{M} = (\mathcal{M}_\lambda)_{\lambda \in \mathbb{N}}$ *is* correct *if for any* $\lambda \in \mathbb{N}$*,* $N \in \mathbb{N}$ *and* $\mathsf{m} \in \mathcal{M}_\lambda$*, we have*

$$\Pr\left[\mathsf{Decrypt}(\mathsf{k}_N, (\hat{\mathsf{ct}}_N, \mathsf{ct}_N)) = \mathsf{m}\right] = 1,$$

where $\mathsf{k}_1, \ldots, \mathsf{k}_N \leftarrow \mathsf{KeyGen}(1^\lambda)$*,* $(\hat{\mathsf{ct}}_1, \mathsf{ct}_1) \leftarrow \mathsf{Encrypt}(\mathsf{k}_1, \mathsf{m})$*, and*

$$(\hat{\mathsf{ct}}_{i+1}, \mathsf{ct}_{i+1}) \leftarrow \mathsf{ReEncrypt}\big(\mathsf{ReKeyGen}(\mathsf{k}_i, \mathsf{k}_{i+1}, \hat{\mathsf{ct}}_i), (\hat{\mathsf{ct}}_i, \mathsf{ct}_i)\big),$$

for $i = 1, \ldots, N-1$*.*

We note that the definition above requires that the correctness of decryption to hold even after *unbounded* number of key updates. In Definition 4.1, we define a relaxation of this definition that requires correctness of decryption for a bounded number of updates.

3.2 Prior Notions of Confidentiality

Standard semantic security for a symmetric encryption scheme requires that an encryption of a message does not reveal any information about the message. In a regular symmetric encryption scheme, there exists only one way to produce a ciphertext: via the encryption algorithm. In an updatable authenticated encryption scheme, there exist two ways of producing a ciphertext: the encryption algorithm $\mathsf{Encrypt}$ that generates *fresh* ciphertexts and the re-encryption algorithm $\mathsf{ReEncrypt}$ that generates *re-encrypted* ciphertexts. Previous formulations of updatable encryption security capture the security of these algorithms in two separate security experiments. The security of the regular encryption algorithm $\mathsf{Encrypt}$ is captured by the notion of *message confidentiality* [11,15] while the security of the re-encryption algorithm $\mathsf{ReEncrypt}$ is captured by the notion of *re-encryption indistinguishability* [15].

Both security experiments are divided into three phases, and are parameterized by h, the number of *honest* keys, and d, the number of *dishonest* keys. During the *setup phase* of the security experiment, the challenger generates h keys $\mathsf{k}_1, \ldots, \mathsf{k}_h \leftarrow \mathsf{KeyGen}(1^\lambda)$ that are the game kept private from the adversary, and d keys $\mathsf{k}_{h+1}, \ldots, \mathsf{k}_{h+d}$ that are provided to the adversary. During the *query phase* of the experiment, the adversary is given access to a set of oracles that evaluate the algorithms $\mathsf{Encrypt}$, $\mathsf{ReKeyGen}$, and $\mathsf{ReEncrypt}$, allowing the adversary to obtain ciphertexts under honest keys and rekey them.

The only distinction between the message-confidentiality and re-encryption indistinguishability experiments is in the way we define the final *challenge* oracle. In the message confidentiality experiment, the adversary is given access to a challenge oracle where it can submit a pair of messages $(\mathsf{m}_0, \mathsf{m}_1)$. As in a standard semantic security definition, the challenge oracle provides the adversary with an encryption of either m_0 or m_1 under a specified honest key, and the adversary's goal is to guess which of the messages was encrypted. In the re-encryption indistinguishability experiment, on the other hand, the adversary submits a pair of *ciphertexts* $\big((\hat{\mathsf{ct}}_0, \mathsf{ct}_0), (\hat{\mathsf{ct}}_1, \mathsf{ct}_1)\big)$ of the same length to the challenge oracle and receives a *re-encryption* of one of the ciphertexts. The adversary's goal in the re-encryption indistinguishability experiment is to guess which of the two ciphertexts was *re-encrypted.*

During the query phase of the experiment, the adversary can make queries to all four oracles as long as their evaluations do not allow the adversary to "trivially" learn which messages are encrypted by the challenge oracle. In particular, this means that no oracle will be allowed to rekey a challenge ciphertext from an honest key to a dishonest key. To this end, the challenger in each experiment keeps a table of challenge ciphertexts generated under each honest key and their re-encryptions. Much of the apparent complexity of formalizing the definition arises from enforcing this straightforward check. We provide the full definitions of Everspaugh et al. [15] in the full version [10].

3.3 Improving Confidentiality

One property that is not captured by the combination of message confidentiality and re-encryption indistinguishability is the indistinguishability of fresh ciphertexts from re-encrypted ciphertexts. In particular, an encryption scheme in which fresh ciphertexts have a completely different structure than those of re-encrypted ciphertexts can still separately satisfy message confidentiality for fresh encryptions and re-encryption indistinguishability for re-encryptions. In many situations, an adversary that learns whether a ciphertext is a fresh encryption or a re-encryption can deduce information about the underlying plaintext of a message.

Furthermore, in the re-encryption indistinguishability experiment, an adversary is required to submit two ciphertexts $\mathsf{ct}_0, \mathsf{ct}_1$ that have the *same* size $|\mathsf{ct}_0| = |\mathsf{ct}_1|$. If we consider the re-encryption algorithm $\mathsf{ReEncrypt}$ to be another form of fresh encryption, this admissibility condition on the adversary is quite intuitive. However, equal length plaintexts do not necessarily result in equal-length ciphertexts after different numbers of re-encryptions. This means existing definitions permit schemes that have a different structure for every possible number of re-encryptions.

Thus, the existing confidentiality definitions for an authenticated updatable encryption scheme fail to enforce the following properties:

- **Property 1**: Freshly generated ciphertexts are indistinguishable from ciphertexts that are generated via re-encryption.
- **Property 2**: Ciphertexts do not reveal how many times a re-encryption algorithm was performed on a given ciphertext.

We state the two properties separately because ciphertexts in our experiment comparing freshly-generated and re-encrypted ciphertexts must be of the same length to prevent trivial wins, which does not rule out the possibility of ciphertext length leaking information about age.

We now augment the confidentiality security definitions of Everspaugh et al. [15] to enforce these two properties.

Enforcing Property 1. A natural way to enforce that fresh ciphertexts are indistinguishable from re-encrypted ciphertexts is to define a security experiment analogous to the definitions of message confidentiality and re-encryption indistinguishability, but with respect to a challenge oracle that takes in either a message m or a ciphertext $(\hat{\mathsf{ct}}, \mathsf{ct})$ and either *encrypts* m or *re-encrypts* $(\hat{\mathsf{ct}}, \mathsf{ct})$.

We present the full definition of confidentiality below. The various checks included in the description of the oracles only serve to ensure that an adversary cannot take a challenge ciphertext under an honest key and obtain its re-encryption under a dishonest key, as this would result in a trivial win.

Definition 3.4 (Confidentiality). *Let* $\Pi_{\mathsf{UAE}} = (\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{ReKeyGen}, \mathsf{ReEncrypt}, \mathsf{Decrypt})$ *be an updatable authenticated encryption scheme for a message space* $\mathcal{M} = (\mathcal{M}_\lambda)_{\lambda \in \mathbb{N}}$. *Then, for a security parameter* λ, *positive integers* $h, d \in \mathbb{N}$, *an adversary* $\mathcal{A}$, *and a binary bit* $b \in \{0, 1\}$, *we define the confidentiality experiment* $\mathsf{Expt}^{\mathsf{conf}}_{\Pi_{\mathsf{UAE}}}(\lambda, h, d, \mathcal{A}, b)$ *and oracles* $\mathcal{O} = (\mathcal{O}_{\mathsf{Encrypt}}, \mathcal{O}_{\mathsf{ReKeyGen}}, \mathcal{O}_{\mathsf{ReEncrypt}}, \mathcal{O}_{\mathsf{Challenge}})$ *in Fig. 1. The experiment maintains a look-up table* T, *accessible by all the oracles, that maps* key index *and* ciphertext header *pairs to* ciphertext bodies.

```
$\mathsf{Expt}^{\mathsf{conf}}_{\Pi_{\mathsf{UAE}}}(\lambda, h, d, \mathcal{A}, b)$:
$\mathsf{k}_1, \ldots, \mathsf{k}_{h+d} \leftarrow \mathsf{KeyGen}(1^\lambda)$
$b' \leftarrow \mathcal{A}^{\mathcal{O}}(k_{h+1}, ..., k_{h+d})$
Output $b' = b$

$\mathcal{O}_{\mathsf{Encrypt}}(i, \mathsf{m})$:
Output $\mathsf{Encrypt}(\mathsf{k}_i, \mathsf{m})$

$\mathcal{O}_{\mathsf{Challenge}}\big(i, j, \mathsf{m}, (\hat{\mathsf{ct}}, \mathsf{ct})\big)$:
if $j > h$:
    Output $\perp$
$(\hat{\mathsf{ct}}_0', \mathsf{ct}_0') \leftarrow \mathsf{Encrypt}(\mathsf{k}_j, \mathsf{m})$
$\Delta_{i,j,\hat{\mathsf{ct}}} \leftarrow \mathsf{ReKeyGen}(\mathsf{k}_i, \mathsf{k}_j, \hat{\mathsf{ct}})$
$(\hat{\mathsf{ct}}_1', \mathsf{ct}_1') \leftarrow \mathsf{ReEncrypt}(\Delta_{i,j,\hat{\mathsf{ct}}}, (\hat{\mathsf{ct}}, \mathsf{ct}))$
if $(\hat{\mathsf{ct}}_0', \mathsf{ct}_0') = \perp$ or $(\hat{\mathsf{ct}}_1', \mathsf{ct}_1') = \perp$:
    Output $\perp$
if $|\hat{\mathsf{ct}}_0'| \neq |\hat{\mathsf{ct}}_1'|$ or $|\mathsf{ct}_0'| \neq |\mathsf{ct}_1'|$:
    Output $\perp$
$\mathsf{T}[j, \hat{\mathsf{ct}}_b'] \leftarrow \mathsf{ct}_b'$
Output $(\hat{\mathsf{ct}}_b', \mathsf{ct}_b')$

$\mathcal{O}_{\mathsf{ReKeyGen}}(i, j, \hat{\mathsf{ct}})$:
if $j > h$ and $T[i, \hat{\mathsf{ct}}] \neq \perp$:
    Output $\perp$
$\Delta_{i,j,\hat{\mathsf{ct}}} \leftarrow \mathsf{ReKeyGen}(\mathsf{k}_i, \mathsf{k}_j, \hat{\mathsf{ct}})$
if $T[i, \hat{\mathsf{ct}}] \neq \perp$:
    $(\hat{\mathsf{ct}}', \mathsf{ct}') \leftarrow \mathsf{ReEncrypt}\big(\Delta_{i,j,\hat{\mathsf{ct}}}, (\hat{\mathsf{ct}}, \mathsf{T}[i, \hat{\mathsf{ct}}])\big)$
    $\mathsf{T}[j, \hat{\mathsf{ct}}'] \leftarrow \mathsf{ct}'$
Output $\Delta_{i,j,\hat{\mathsf{ct}}}$

$\mathcal{O}_{\mathsf{ReEncrypt}}\big(i, j, (\hat{\mathsf{ct}}, \mathsf{ct})\big)$:
$\Delta_{i,j,\hat{\mathsf{ct}}} \leftarrow \mathsf{ReKeyGen}(\mathsf{k}_i, \mathsf{k}_j, \hat{\mathsf{ct}})$
$(\hat{\mathsf{ct}}', \mathsf{ct}') \leftarrow \mathsf{ReEncrypt}\big(\Delta_{i,j,\hat{\mathsf{ct}}}, (\hat{\mathsf{ct}}, \mathsf{ct})\big)$
if $j > h$ and $\mathsf{T}[i, \hat{\mathsf{ct}}] \neq \perp$:
    Output $\perp$
if $j \leq h$ and $\mathsf{T}[i, \hat{\mathsf{ct}}] \neq \perp$:
    $\mathsf{T}[j, \hat{\mathsf{ct}}'] \leftarrow \mathsf{ct}'$
Output $(\hat{\mathsf{ct}}', \mathsf{ct}')$
```

Fig. 1. Security experiment for confidentiality (Definition 3.4) and update independence (Definition 3.6)

We say that an updatable authenticated encryption scheme Π_{UAE} satisfies confidentiality *if there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all $h, d \leq$ $\mathsf{poly}(\lambda)$ and efficient adversaries $\mathcal{A}$, we have*

$$\left| \Pr\left[\mathsf{Expt}^{\mathsf{conf}}_{\Pi_{\mathsf{UAE}}}(\lambda, h, d, \mathcal{A}, 0) = 1\right] - \Pr\left[\mathsf{Expt}^{\mathsf{conf}}_{\Pi_{\mathsf{UAE}}}(\lambda, h, d, \mathcal{A}, 1) = 1\right] \right| \leq \mathsf{negl}(\lambda).$$

Although our original goal in defining the confidentiality experiment above is to enforce the condition that fresh ciphertexts are indistinguishable from re-encrypted ciphertexts, the experiment captures a much wider class of confidentiality properties for an updatable authenticated encryption scheme. In fact, it is straightforward to show that a UAE scheme that satisfies the single confidentiality definition above automatically satisfies both message confidentiality and re-encryption indistinguishability. Specifically, since the confidentiality definition above implies that an encryption of a message is indistinguishable from a re-encryption of a ciphertext (given that the resulting ciphertexts are of the same length), this implies that for any two messages $\mathsf{m}_0, \mathsf{m}_1$ such that $|\mathsf{m}_0| = |\mathsf{m}_1|$, we have

$$\mathsf{Encrypt}(\mathsf{k}, \mathsf{m}_0) \approx_c (\hat{\mathsf{ct}}', \mathsf{ct}') \approx_c \mathsf{Encrypt}(\mathsf{k}, \mathsf{m}_1),$$

for any key k that is hidden from an adversary and any re-encrypted ciphertext $(\hat{\mathsf{ct}}', \mathsf{ct}')$ of appropriate length. Similarly, the confidentiality definition above implies that for two ciphertexts $(\hat{\mathsf{ct}}_0, \mathsf{ct}_0)$ and $(\hat{\mathsf{ct}}_1, \mathsf{ct}_1)$ of the same length,

$$\begin{aligned}&\mathsf{ReEncrypt}\big(\mathsf{ReKeyGen}(\mathsf{k}, \mathsf{k}', \hat{\mathsf{ct}}_0), (\hat{\mathsf{ct}}_0, \mathsf{ct}_0)\big)\\ &\qquad\qquad\qquad\qquad \approx_c (\hat{\mathsf{ct}}', \mathsf{ct}') \approx_c \\ &\qquad\qquad\qquad\qquad\qquad\qquad \mathsf{ReEncrypt}\big(\mathsf{ReKeyGen}(\mathsf{k}, \mathsf{k}', \hat{\mathsf{ct}}_1), (\hat{\mathsf{ct}}_1, \mathsf{ct}_1)\big),\end{aligned}$$

for an appropriate key k' that is hidden from an adversary and any fresh ciphertext $(\hat{\mathsf{ct}}', \mathsf{ct}')$ of appropriate length.

In combination with our new strong compactness requirement (which we introduce in Definition 3.5), the security experiment in Definition 3.4 captures all the confidentiality properties we expect from an updatable encryption scheme. This is why we refer to the experiment in Definition 3.4 simply as the "confidentiality" experiment.

Enforcing Property 2. Enforcing that an updatable encryption ciphertext hides the number of key updates is less straightforward. Perhaps the most natural and general way to enforce this property is to modify the challenge oracle in Definition 3.4 as follows:

- $\mathcal{O}_{\mathsf{Challenge}}\big(\mathcal{I}, (\hat{\mathsf{ct}}_{0,0}, \mathsf{ct}_{0,0}), \mathcal{J}, (\hat{\mathsf{ct}}_{1,0}, \mathsf{ct}_{1,0})\big)$: A query consists of two sequences of indices $\mathcal{I} = (i_1, \ldots, i_\tau)$, $\mathcal{J} = (j_1, \ldots, j_{\tau'})$ for $\tau, \tau' \in \mathbb{N}$ such that $i_\tau = j_{\tau'}$ are honest keys, and $|\mathsf{ct}_{0,0}| = |\mathsf{ct}_{1,0}|$. The challenger computes two sequences of ciphertexts

  $$\begin{aligned}&\Delta_{i_{\gamma-1}, i_\gamma} \leftarrow \mathsf{ReKeyGen}(\mathsf{k}_{i_{\gamma-1}}, \mathsf{k}_{i_\gamma}, \hat{\mathsf{ct}}_{0,i_\gamma})\\ &(\hat{\mathsf{ct}}_{0,i_\gamma}, \mathsf{ct}_{0,i_\gamma}) \leftarrow \mathsf{ReEncrypt}(\Delta_{i_{\gamma-1}, i_\gamma}, \hat{\mathsf{ct}}_{0,i_{\gamma-1}}, \mathsf{ct}_{0,i_{\gamma-1}}) \quad \forall \gamma \in [\tau],\end{aligned}$$

  and

  $$\begin{aligned}&\Delta'_{j_{\gamma-1}, j_\gamma} \leftarrow \mathsf{ReKeyGen}(\mathsf{k}_{j_{\gamma-1}}, \mathsf{k}_{j_\gamma}, \hat{\mathsf{ct}}_{1,j_\gamma})\\ &(\hat{\mathsf{ct}}_{1,j_\gamma}, \mathsf{ct}_{1,j_\gamma}) \leftarrow \mathsf{ReEncrypt}(\Delta'_{j_{\gamma-1}, j_\gamma}, \hat{\mathsf{ct}}_{1,j_{\gamma-1}}, \mathsf{ct}_{1,j_{\gamma-1}}) \quad \forall \gamma \in [\tau'].\end{aligned}$$

  It returns either $(\hat{\mathsf{ct}}_{0,j_\tau}, \mathsf{ct}_{0,j_\tau})$ or $(\hat{\mathsf{ct}}_{1,j_{\tau'}}, \mathsf{ct}_{1,j_{\tau'}})$.

The challenge oracle above takes in two sequences of indices $\mathcal{I}$, $\mathcal{J}$, and re-encrypts either the ciphertext $(\hat{\mathsf{ct}}_{0,0}, \mathsf{ct}_{0,0})$ according to the sequence of keys specified by $\mathcal{I}$ or the ciphertext $(\hat{\mathsf{ct}}_{1,0}, \mathsf{ct}_{1,0})$ according to $\mathcal{J}$. Since the two sequences $\mathcal{I}$ and $\mathcal{J}$ can have differing lengths, an updatable encryption scheme that satisfies a security experiment with respect to such a challenge oracle must hide the number of times the re-encryption algorithm was applied to a ciphertext.

However, a security experiment that is defined with respect to the challenge oracle above is generally difficult to work with and requires notationally complicated proofs. Hence, instead of using the challenge oracle as defined above, we define a stronger *compactness* requirement on the ciphertexts of an updatable encryption scheme. Specifically, in addition to the compactness requirement as specified in Definition 3.2, we require that the size of a ciphertext always remains fixed no matter how many times the re-encryption algorithm is performed on a ciphertext.

Definition 3.5 (Strong Compactness). *We say that an updatable authenticated encryption scheme* $\Pi_{\mathsf{UAE}} = (\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{ReKeyGen}, \mathsf{ReEncrypt}, \mathsf{Decrypt})$ *for a message space* $\mathcal{M} = (\mathcal{M}_\lambda)_{\lambda \in \mathbb{N}}$ *is* strongly compact *if for any* $\lambda \in \mathbb{N}$ *and any message* $\mathsf{m} \in \mathcal{M}_\lambda$*, it satisfies the* header compactness *and* body compactness *(with probability 1) after the following operations.*

- $\mathsf{k}_0, \mathsf{k}_1, \ldots, \mathsf{k}_N \leftarrow \mathsf{KeyGen}(1^\lambda)$
- $(\hat{\mathsf{ct}}_0, \mathsf{ct}_0) \leftarrow \mathsf{Encrypt}(\mathsf{k}_0, \mathsf{m})$
- *for* $i \in [N]$*:*
- $\Delta_{i,i-1,\hat{\mathsf{ct}}_{i-1}} \leftarrow \mathsf{ReKeyGen}(\mathsf{k}_{i-1}, \mathsf{k}_i, \hat{\mathsf{ct}}_{i-1})$
- $(\hat{\mathsf{ct}}_i, \mathsf{ct}_i) \leftarrow \mathsf{ReEncrypt}\big(\Delta_{i,i-1,\hat{\mathsf{ct}}_{i-1}}, (\hat{\mathsf{ct}}_{i-1}, \mathsf{ct}_{i-1})\big)$

- Header compactness*: There exist polynomials* $f_1(\cdot)$*,* $f_2(\cdot)$ *such that* $|\hat{\mathsf{ct}}_i| \leq f_1(\lambda)$ *and* $|\Delta_{i,i-1,\hat{\mathsf{ct}}_{i-1}}| \leq f_2(\lambda)$ *for all* $i \in [N]$*, i.e., header and update token lengths do not depend on the message length or the number of re-encryptions.*
- Body compactness*: We have* $|\mathsf{ct}_i| = |\mathsf{ct}_j|$ *for all* $0 \leq i, j \leq N$*.*

In combination with Definition 3.4, the strong compactness property implies that ciphertexts do not reveal how many times a re-encryption algorithm was performed on a given ciphertext. The confidentiality property of Definition 3.4 implies that the re-encryption of any two ciphertexts of the *same size* must be indistinguishable to an adversary. The strong compactness property requires that no matter how many re-encryption operations are performed on a given ciphertext, its length always *remains* the same size, thereby complementing Definition 3.4.

Update independence. In Construction 4.2, we present a UAE scheme that satisfies the strong compactness property of Definition 3.5 as well as message confidentiality and re-encryption indistinguishability, but does not fully satisfy the stronger notion of confidentiality as defined in Definition 3.4. Therefore, we define a slight relaxation of the confidentiality requirement as formulated in Definition 3.4 that we call *update independence* and show that Construction 4.2 satisfies this security definition. An update independence security experiment is defined identically to the confidentiality security experiment but without the re-encryption key generation oracle $\mathcal{O}_{\mathsf{ReKeyGen}}$. Since this oracle is removed, update independence does not suffice to imply message confidentiality and re-encryption indistinguishability. However, it still suffices to guarantee that fresh ciphertexts are indistinguishable from re-encrypted ciphertexts as long as update tokens are hidden from an adversary.

Definition 3.6 (Update Independence). *Let* $\Pi_{\mathsf{UAE}} = (\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{ReKeyGen}, \mathsf{ReEncrypt}, \mathsf{Decrypt})$ *be an updatable authenticated encryption scheme for a message space* $\mathcal{M} = (\mathcal{M}_\lambda)_{\lambda \in \mathbb{N}}$*. Then, for a security parameter* λ*, positive integers* $h, d \in \mathbb{N}$*, an adversary* $\mathcal{A}$*, and a binary bit* $b \in \{0, 1\}$*, we* *define the update independence experiment* $\mathsf{Expt}^{\mathsf{upd\text{-}ind}}_{\Pi_{\mathsf{UAE}}}(\lambda, h, d, \mathcal{A}, b)$ *and oracles* $\mathcal{O} = (\mathcal{O}_{\mathsf{Encrypt}}, \mathcal{O}_{\mathsf{ReEncrypt}}, \mathcal{O}_{\mathsf{Challenge}})$ *as in Fig. 1 with the* $\mathcal{O}_{\mathsf{ReKeyGen}}$ *oracle omitted. The experiment maintains a look-up table* T*, accessible by all the oracles, that maps* key index *and* ciphertext header *pairs to* ciphertext bodies*.*

We say that an updatable authenticated encryption scheme Π_{UAE} *satisfies* update independence *if there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $h, d \leq \mathsf{poly}(\lambda)$ *and efficient adversaries* $\mathcal{A}$*, we have*

$$\left| \Pr\left[\mathsf{Expt}^{\mathsf{upd\text{-}ind}}_{\Pi_{\mathsf{UAE}}}(\lambda, h, d, \mathcal{A}, 0) = 1\right] - \Pr\left[\mathsf{Expt}^{\mathsf{upd\text{-}ind}}_{\Pi_{\mathsf{UAE}}}(\lambda, h, d, \mathcal{A}, 1) = 1\right] \right| \leq \mathsf{negl}(\lambda).$$

In combination with the message confidentiality and re-encryption indistinguishability properties, this relaxed requirement of update independence suffices for many practical scenarios. Since update tokens are generally sent over secure channels (e.g. TLS connection) from a client to a server, no malicious eavesdropper can gain access to them. For malicious servers that have access to update tokens, on the other hand, hiding how many times a re-encryption operation was previously applied on a ciphertext is less useful since the storage metadata of the ciphertexts already reveal this information to the server. In essence, update independence, when combined with message confidentiality and re-encryption indistinguishability, seems to satisfy the two properties we wanted from our new confidentiality definition without the convenient benefit of a single unified definition.

3.4 Integrity

The final security property that an updatable authenticated encryption scheme must provide is *ciphertext integrity.* The ciphertext integrity experiment for UAE is analogous to the standard ciphertext integrity experiment of an authenticated encryption scheme. As in the confidentiality experiment, the challenger starts the experiment by generating a set of honest keys, which are kept private from the adversary, and dishonest keys, which are provided to the adversary. Then, given oracle access to $\mathcal{O}_{\mathsf{Encrypt}}$, $\mathcal{O}_{\mathsf{ReEncrypt}}$, and $\mathcal{O}_{\mathsf{ReKeyGen}}$, the adversary's goal is to generate a new valid ciphertext that was not (1) previously output by $\mathcal{O}_{\mathsf{Encrypt}}$ or $\mathcal{O}_{\mathsf{ReEncrypt}}$, and (2) cannot be trivially derived via update tokens output by $\mathcal{O}_{\mathsf{ReKeyGen}}$.

Our integrity definition is similar to that of Everspaugh et al. [15], except the previous definition does not include the re-encryption oracle $\mathcal{O}_{\mathsf{ReEncrypt}}$, which we add. Giving the adversary access to a re-encryption oracle captures scenarios that are not covered by the previous definition. For instance, security with respect to our stronger integrity experiment guarantees that an adversary who compromises the key for a ciphertext cannot tamper with the data after the key has been rotated and the data re-encrypted.

Definition 3.7 (Integrity). *Let* $\Pi_{\mathsf{UAE}} = (\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{ReKeyGen}, \mathsf{ReEncrypt}, \mathsf{Decrypt})$ *be an updatable authenticated encryption scheme for a message space* $\mathcal{M} = (\mathcal{M}_\lambda)_{\lambda \in \mathbb{N}}$*. Then, for a security parameter* λ*, positive integers*

```
$\underline{\mathsf{Expt}^{\mathsf{int}}_{\Pi_{\mathsf{UAE}}}(\lambda, h, d, \mathcal{A})}$:
$\mathsf{k}_1, \ldots, \mathsf{k}_{h+d} \leftarrow \mathsf{KeyGen}(1^\lambda)$
$(i, (\hat{\mathsf{ct}}, \mathsf{ct})) \leftarrow \mathcal{A}^{\mathcal{O}}(k_{h+1}, ..., k_{h+d})$
if $i > h$:
    Output 0
$\mathsf{m} \leftarrow \mathsf{Decrypt}(\mathsf{k}_i, (\hat{\mathsf{ct}}, \mathsf{ct}))$
if $\mathsf{m} = \perp$ or $\mathsf{T}[i, \hat{\mathsf{ct}}] = \mathsf{ct}$:
    Output 0
else:
    Output 1

$\underline{\mathcal{O}_{\mathsf{Encrypt}}(i, \mathsf{m})}$:
$(\hat{\mathsf{ct}}, \mathsf{ct}) \leftarrow \mathsf{Encrypt}(\mathsf{k}_i, \mathsf{m})$
$\mathsf{T}[i, \hat{\mathsf{ct}}] \leftarrow \mathsf{ct}$
Output $(\hat{\mathsf{ct}}, \mathsf{ct})$
```

```
$\underline{\mathcal{O}_{\mathsf{ReEncrypt}}(i, j, (\hat{\mathsf{ct}}, \mathsf{ct}))}$:
$\Delta_{i,j,\hat{\mathsf{ct}}} \leftarrow \mathsf{ReKeyGen}(\mathsf{k}_i, \mathsf{k}_j, \hat{\mathsf{ct}})$
$(\hat{\mathsf{ct}}', \mathsf{ct}') \leftarrow \mathsf{ReEncrypt}(\Delta_{i,j,\hat{\mathsf{ct}}}, (\hat{\mathsf{ct}}, \mathsf{ct}))$
if $j \leq h$:
    $\mathsf{T}[j, \hat{\mathsf{ct}}'] \leftarrow \mathsf{ct}'$
Output $(\hat{\mathsf{ct}}', \mathsf{ct}')$

$\underline{\mathcal{O}_{\mathsf{ReKeyGen}}(i, j, \hat{\mathsf{ct}})}$:
if $i > h$ and $j \leq h$:
    Output $\perp$
$\Delta_{i,j,\hat{\mathsf{ct}}} \leftarrow \mathsf{ReKeyGen}(\mathsf{k}_i, \mathsf{k}_j, \hat{\mathsf{ct}})$
if $\mathsf{T}[i, \hat{\mathsf{ct}}] \neq \perp$:
    $(\hat{\mathsf{ct}}', \mathsf{ct}') \leftarrow \mathsf{ReEncrypt}(\Delta_{i,j,\hat{\mathsf{ct}}}, (\hat{\mathsf{ct}}, \mathsf{T}[i, \hat{\mathsf{ct}}]))$
    $\mathsf{T}[j, \hat{\mathsf{ct}}'] \leftarrow \mathsf{ct}'$
Output $\Delta_{i,j,\hat{\mathsf{ct}}}$
```

Fig. 2. Security experient for integrity (Definition 3.7)

$h, d \in \mathbb{N}$, and an adversary $\mathcal{A}$, we define the re-encryption integrity experiment $\mathsf{Expt}^{\mathsf{int}}_{\Pi_{\mathsf{UAE}}}(\lambda, h, d, \mathcal{A})$ and oracles $\mathcal{O} = (\mathcal{O}_{\mathsf{Encrypt}}, \mathcal{O}_{\mathsf{ReKeyGen}}, \mathcal{O}_{\mathsf{ReEncrypt}})$ in Fig. 2. The experiment maintains a look-up table T, accessible by all the oracles, that maps key index *and* ciphertext header *pairs to* ciphertext bodies*.*

We say that an updatable authenticated encryption scheme Π_{UAE} satisfies re-encryption integrity *if there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $h, d \leq \mathsf{poly}(\lambda)$ and any efficient adversary $\mathcal{A}$, we have*

$$\Pr\left[\mathsf{Expt}^{\mathsf{int}}_{\Pi_{\mathsf{UAE}}}(\lambda, h, d, \mathcal{A}) = 1\right] \leq \mathsf{negl}(\lambda).$$

Although our UAE construction in Sect. 4 can be shown to satisfy the strong notion of integrity formulated above, the construction in Sect. 5 that relies on almost key-homomorphic PRFs is not sufficient to satisfy the stronger notion. In Sect. 5, we formulate a relaxation of the notion of integrity that we call *relaxed integrity* and show that Construction 5.2 satisfies this weaker variant.

4 UAE with Bounded Updates

We begin this section by presenting an *insecure* UAE scheme that demonstrates the importance of the new definitions presented in Sect. 3. This scheme leaks the age of ciphertexts but nonetheless satisfies all security definitions for ciphertext-dependent UAE from prior work.

Next, we extend the insecure scheme to hide the age of ciphertexts, thereby satisfying the definition of update independence (Sect. 3.3, Definition 3.6). This upgrade comes at the cost of relaxing the correctness requirement of an updatable encryption scheme: the correctness of decryption is guaranteed only for an a priori bounded number of key updates.

4.1 A Simple Nested Construction

In this section, we provide a simple updatable authenticated encryption scheme using any authenticated encryption scheme. Our simple construction inherently leaks information about the message; namely, the construction leaks how many re-encryption operations were previously performed on a given ciphertext, thereby leaking information about the age of the encrypted message. Despite this information leakage, the construction satisfies all the UAE security definitions of Everspaugh et al. [15]. Hence, this construction demonstrates that prior security definitions did not yet capture all the necessary security properties that an updatable encryption scheme must provide.

The construction uses an authenticated encryption (AE) scheme. A key for this UAE scheme is a standard AE key $\hat{\mathsf{k}}$, which we call the *header key.* The UAE encryption algorithm implements standard chained encryption. To encrypt m using $\hat{\mathsf{k}}$, first generate a fresh *body key* k_{ae} and then encrypt the plaintext $\mathsf{ct} \leftarrow \mathsf{AE.Encrypt}(\mathsf{k}_{\mathsf{ae}}, \mathsf{m})$. Next, the body key k_{ae} is encrypted under the header key $\hat{\mathsf{ct}} \leftarrow \mathsf{AE.Encrypt}(\hat{\mathsf{k}}, \mathsf{k}_{\mathsf{ae}})$ to form the ciphertext header. Finally, output the UAE ciphertext $(\hat{\mathsf{ct}}, \mathsf{ct})$.

To update a ciphertext, the client and server proceed as follows:

- *Client*: The client downloads the ciphertext header $\hat{\mathsf{ct}}$ to recover the body key k_{ae}. It then generates fresh header and body keys $\hat{\mathsf{k}}'$ and $\mathsf{k}'_{\mathsf{ae}}$, and sends a new ciphertext header $\hat{\mathsf{ct}}' \leftarrow \mathsf{AE.Encrypt}\big(\hat{\mathsf{k}}', (\mathsf{k}'_{\mathsf{ae}}, \mathsf{k}_{\mathsf{ae}})\big)$ along with $\mathsf{k}'_{\mathsf{ae}}$ to the server.
- *Server*: The server replaces the old ciphertext header $\hat{\mathsf{ct}}$ with the new header $\hat{\mathsf{ct}}'$. It also generates a new ciphertext body by encrypting the original ciphertext as $\mathsf{ct}' \leftarrow \mathsf{AE.Encrypt}\big(\mathsf{k}'_{\mathsf{ae}}, (\hat{\mathsf{ct}}, \mathsf{ct})\big)$.

Now, even with many such key updates, the client can still recover the original ciphertext. Specifically, the client can first use its current header key $\hat{\mathsf{k}}$ to decrypt the ciphertext header and recover a body key k_{ae} *and* the old header key $\hat{\mathsf{k}}'$. It uses k_{ae} to remove the outer layer of encryption and recover the old ciphertext $(\hat{\mathsf{ct}}', \mathsf{ct}')$. The client repeats the same procedure with the old header key $\hat{\mathsf{k}}'$ and the old ciphertext $(\hat{\mathsf{ct}}', \mathsf{ct}')$. Note that decryption time grows linearly in the number of re-encryption operations.

To prove security, we must introduce an additional step during a ciphertext update. Namely, instead of setting the new ciphertext body as the encryption of the old ciphertext header and body $\mathsf{ct}' \leftarrow \mathsf{AE.Encrypt}\big(\mathsf{k}'_{\mathsf{ae}}, (\hat{\mathsf{ct}}, \mathsf{ct})\big)$, the server replaces $\hat{\mathsf{ct}}$ with a new ciphertext header $\hat{\mathsf{ct}}_{\mathsf{history}}$ that the client provides to the server encrypted under a new key $\hat{\mathsf{k}}_{\mathsf{history}}$. The main intuition of the construction, however, remains unchanged from the description above. Since the construction is a simpler form of the one formalized in Construction 4.2, we defer the formal statement of the construction and its associated security theorems for compactness, correctness, update independence, message confidentiality, re-encryption indistinguishability, and ciphertext integrity to the full version [10].

4.2 Bounded Correctness

We now define a variation of correctness that we call *bounded correctness*. The bounded correctness condition is defined in a natural way and analogously to Definition 3.3 (correctness). However, we do modify the syntax of the key generation algorithm KeyGen to additionally take in a parameter $t \in \mathbb{N}$ that specifies an upper bound on the number of key updates that a scheme can support. This allows the key generator to flexibly set this parameter according to its needs.

Definition 4.1 (Bounded Correctness). *We say that an updatable authenticated encryption scheme* $\Pi_{\mathsf{UAE}} = (\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{ReKeyGen}, \mathsf{ReEncrypt}, \mathsf{Decrypt})$ *for a message space* $\mathcal{M} = (\mathcal{M}_\lambda)_{\lambda \in \mathbb{N}}$ *satisfies* bounded correctness *if for any* $\lambda, t \in \mathbb{N}$*, and* $\mathsf{m} \in \mathcal{M}_\lambda$*, we have (with probability 1)*

$$\Pr\left[\mathsf{Decrypt}(\mathsf{k}_t, (\hat{\mathsf{ct}}_t, \mathsf{ct}_t)) = \mathsf{m}\right] \geq 1 - \mathsf{negl}(\lambda),$$

where $\mathsf{k}_1, \ldots, \mathsf{k}_t \leftarrow \mathsf{KeyGen}(1^\lambda, 1^t)$*,* $(\hat{\mathsf{ct}}_1, \mathsf{ct}_1) \leftarrow \mathsf{Encrypt}(\mathsf{k}_1, \mathsf{m})$*, and*

$$(\hat{\mathsf{ct}}_{i+1}, \mathsf{ct}_{i+1}) \leftarrow \mathsf{ReEncrypt}\big(\mathsf{ReKeyGen}(\mathsf{k}_i, \mathsf{k}_{i+1}, \hat{\mathsf{ct}}_i), (\hat{\mathsf{ct}}_i, \mathsf{ct}_i)\big),$$

for $i = 1, \ldots, t-1$.

4.3 Nested Construction with Padding

Our modification of the nested construction is straightforward: we pad the ciphertexts such that as long as the number of key updates is bounded, their lengths are independent of the number of key updates that are performed on the ciphertexts. However, executing this simple idea requires some care. First, padding the (original) ciphertexts with structured strings reveals information about how many updates were previously performed on the ciphertexts. Therefore, we modify the encryption algorithm such that it pads the ciphertexts with random strings. If the underlying authenticated encryption scheme satisfies ciphertext pseudorandomness [10], an adversary cannot determine which component of a ciphertext corresponds to the original ciphertext and which component corresponds to a pad.[1]

However, simply padding the (original) ciphertexts with random strings also makes them highly malleable and easy to forge. To achieve integrity, we modify the encryption and re-encryption algorithms to additionally sample a pseudorandom generator (PRG) seed and include it as part of the UAE ciphertext header.

[1] As discussed in Sect. 2, authenticated encryption schemes that satisfy pseudorandomness can be constructed from pseudorandom functions or blockciphers in a standard way. Widely-used modes for authenticated encryption such as AES-GCM also satisfy pseudorandomness.

The encryption and re-encryption algorithms then generate the ciphertext pads from an evaluation of the PRG. By PRG security, the original ciphertext components and the pads are still computationally indistinguishable to an adversary, but now the adversary cannot easily forge ciphertexts as the decryption algorithm can verify the validity of a pad using the PRG seed.

The only remaining issue is correctness. Since the ciphertexts of our UAE scheme are pseudorandom, the re-encryption algorithm also does not have information about where the original ciphertext ends and padding begins. Therefore, we include this information as part of the re-encryption key (update token). This is the reason why this scheme satisfies update independence instead of our full confidentiality definition – even though ciphertexts fully hide their age, update tokens reveal information about the age of the ciphertext they are updating. The re-encryptor can now apply the re-encryption on the original ciphertext and adjust the padding length accordingly. We formalize the construction below.

Construction 4.2 (Nested Authenticated Encryption). *Our construction uses the following building blocks:*

- *An authenticated encryption scheme* $\Pi_{\mathsf{AE}} = (\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ *with message space* $\mathcal{M} = (\mathcal{M}_\lambda)_{\lambda\in\mathbb{N}}$*. We additionally assume that* $\mathsf{AE.Encrypt}$ *satisfies* $\varepsilon_{\mathsf{ae}}^{\mathsf{rand}}$*-ciphertext pseudorandomness, i.e., that encryptions under* AE *are indistinguishable from random strings.*
  For the construction description below, we let $\rho = \rho_\lambda$ *denote the maximum size of an authenticated encryption key and we let* $\nu = \mathsf{poly}(\lambda)$ *be an additive overhead incurred by the encryption algorithm. For any key* $\mathsf{k}_{\mathsf{ae}} \leftarrow \mathsf{AE.KeyGen}(1^\lambda)$ *and any message* $\mathsf{m} \in \mathcal{M}_\lambda$*, we have* $|\mathsf{k}_{\mathsf{ae}}| = \rho$ *and* $|\mathsf{ct}| \leq |\mathsf{m}| + \nu$*, where* $\mathsf{ct} \leftarrow \mathsf{AE.Encrypt}(\mathsf{k}_{\mathsf{ae}}, \mathsf{m})$*.*
- *A pseudorandom generator* $G : \{0,1\}^\lambda \to \{0,1\}^*$*. To simplify the presentation of the construction, we assume that* G *has unbounded output that is truncated to the required length on each invocation.*

We construct an updatable authenticated encryption scheme $\Pi_{\mathsf{UAE}} = (\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{ReKeyGen}, \mathsf{ReEncrypt}, \mathsf{Decrypt})$ *for message space* $\mathcal{M} = (\mathcal{M}_\lambda)_{\lambda\in\mathbb{N}}$ *in Fig. 3.*

We formally state the compactness, correctness, and security properties of Construction 4.2 in the following theorem. We provide the formal proof in the full version [10].

Theorem 4.3. *Suppose the authenticated encryption scheme* Π_{AE} *satisfies correctness,* $\varepsilon_{\mathsf{ae}}^{\mathsf{conf}}$*-confidentiality,* $\varepsilon_{\mathsf{ae}}^{\mathsf{int}}$*-integrity, and* $\varepsilon_{\mathsf{ae}}^{\mathsf{rand}}$*-ciphertext pseudorandomness, and* G *satisfies* $\varepsilon_{\mathsf{prg}}$ *PRG security. Then the updatable authenticated encryption scheme* Π_{UAE} *in Construction 4.2 satisfies strong compactness, correctness, update independence, message confidentiality, and re-encryption indistinguishability.*

```
$\underline{\mathsf{KeyGen}(1^\lambda, 1^t):}$
$\hat{\mathsf{k}} \leftarrow \mathsf{AE.KeyGen}(1^\lambda)$
$\mathsf{k} \leftarrow (\hat{\mathsf{k}}, t)$
Output $\mathsf{k}$

$\underline{\mathsf{Encrypt}(\mathsf{k}, \mathsf{m})}$
$(\hat{\mathsf{k}}, t) \leftarrow \mathsf{k}$
$\mathsf{k}_{\mathsf{ae}} \leftarrow \mathsf{AE.KeyGen}(1^\lambda)$
$s \xleftarrow{\text{R}} \{0,1\}^\lambda$
$\mathsf{ct}_{\mathsf{payload}} \leftarrow \mathsf{AE.Encrypt}\big(\mathsf{k}_{\mathsf{ae}}, \mathsf{m}\big)$
$\mathsf{ct}_{\mathsf{pad}} \leftarrow G(s)$ such that $\mathsf{ct}_{\mathsf{pad}} \in \{0,1\}^{t \cdot (2\rho + \nu)}$
$\hat{\mathsf{ct}} \leftarrow \mathsf{AE.Encrypt}\big(\hat{\mathsf{k}}, (s, |\mathsf{ct}_{\mathsf{payload}}|, \mathsf{k}_{\mathsf{ae}}, \perp)\big)$
$\mathsf{ct} \leftarrow (\mathsf{ct}_{\mathsf{payload}}, \mathsf{ct}_{\mathsf{pad}})$
Output $(\hat{\mathsf{ct}}, \mathsf{ct})$

$\underline{\mathsf{ReKeyGen}(\mathsf{k}_1, \mathsf{k}_2, \hat{\mathsf{ct}}):}$
$(\hat{\mathsf{k}}_1, t) \leftarrow \mathsf{k}_1$
$(\hat{\mathsf{k}}_2, t) \leftarrow \mathsf{k}_2$
$(s, \ell, \mathsf{k}_{\mathsf{ae}}, \hat{\mathsf{k}}_{\mathsf{history}}) \leftarrow \mathsf{AE.Decrypt}(\hat{\mathsf{k}}_1, \hat{\mathsf{ct}})$
if $(s, \ell, \mathsf{k}_{\mathsf{ae}}, \hat{\mathsf{k}}_{\mathsf{history}}) = \perp$, output $\perp$
$\hat{\mathsf{k}}'_{\mathsf{history}} \leftarrow \mathsf{AE.KeyGen}(1^\lambda)$
$\hat{\mathsf{ct}}_{\mathsf{history}} \leftarrow \mathsf{AE.Encrypt}(\hat{\mathsf{k}}'_{\mathsf{history}}, (\mathsf{k}_{\mathsf{ae}}, \hat{\mathsf{k}}_{\mathsf{history}}))$
$\mathsf{k}'_{\mathsf{ae}} \leftarrow \mathsf{AE.KeyGen}(1^\lambda)$
$s' \xleftarrow{\text{R}} \{0,1\}^\lambda$
$\ell' \leftarrow \ell + |\hat{\mathsf{ct}}_{\mathsf{history}}|$
$\hat{\mathsf{ct}}' \leftarrow \mathsf{AE.Encrypt}\big(\hat{\mathsf{k}}_2, (s', \ell', \mathsf{k}'_{\mathsf{ae}}, \hat{\mathsf{k}}'_{\mathsf{history}})\big)$
$\Delta_{1,2,\hat{\mathsf{ct}}} \leftarrow (\hat{\mathsf{ct}}', \hat{\mathsf{ct}}_{\mathsf{history}}, \ell, \mathsf{k}'_{\mathsf{ae}}, s')$
Output $\Delta_{1,2,\hat{\mathsf{ct}}}$
```

```
$\underline{\mathsf{ReEncrypt}\big(\Delta_{1,2,\hat{\mathsf{ct}}}, (\hat{\mathsf{ct}}, \mathsf{ct})\big):}$
$(\hat{\mathsf{ct}}', \hat{\mathsf{ct}}_{\mathsf{history}}, \ell, \mathsf{k}'_{\mathsf{ae}}, s') \leftarrow \Delta_{1,2,\hat{\mathsf{ct}}}$
$(\mathsf{ct}_{\mathsf{payload}}, \mathsf{ct}_{\mathsf{pad}}) \leftarrow \mathsf{ct} \in \{0,1\}^\ell \times \{0,1\}^{|\mathsf{ct}| - \ell}$
if $|\mathsf{ct}| < \ell$, output $\perp$
$\mathsf{ct}'_{\mathsf{payload}} \leftarrow \mathsf{AE.Encrypt}\big(\mathsf{k}'_{\mathsf{ae}}, (\mathsf{ct}_{\mathsf{payload}}, \hat{\mathsf{ct}}_{\mathsf{history}})\big)$
if $|\mathsf{ct}'_{\mathsf{payload}}| > |\mathsf{ct}|$, output $\perp$
$\mathsf{ct}'_{\mathsf{pad}} \leftarrow G(s')[1, \ldots, |\mathsf{ct}| - |\mathsf{ct}'_{\mathsf{payload}}|]$
$\mathsf{ct}' \leftarrow (\mathsf{ct}'_{\mathsf{payload}}, \mathsf{ct}'_{\mathsf{pad}}) \in \{0,1\}^{|\mathsf{ct}|}$
Output $(\hat{\mathsf{ct}}', \mathsf{ct}')$

$\underline{\mathsf{Decrypt}\big(\mathsf{k}, (\hat{\mathsf{ct}}, \mathsf{ct})\big):}$
$(\hat{\mathsf{k}}, t) \leftarrow \mathsf{k}$
$(s, \ell, \mathsf{k}'_{\mathsf{ae}}, \hat{\mathsf{k}}'_{\mathsf{history}}) \leftarrow \mathsf{AE.Decrypt}(\hat{\mathsf{k}}, \hat{\mathsf{ct}})$
if $(s, \ell, \mathsf{k}'_{\mathsf{ae}}, \hat{\mathsf{k}}'_{\mathsf{history}}) = \perp$, output $\perp$
if $|\mathsf{ct}| < \ell$, output $\perp$
$(\mathsf{ct}_{\mathsf{payload}}, \mathsf{ct}_{\mathsf{pad}}) \leftarrow \mathsf{ct} \in \{0,1\}^\ell \times \{0,1\}^{|\mathsf{ct}| - \ell}$
$\mathsf{ct}'_{\mathsf{pad}} \leftarrow G(s)$ such that $|\mathsf{ct}'_{\mathsf{pad}}| = |\mathsf{ct}_{\mathsf{pad}}|$
if $\mathsf{ct}'_{\mathsf{pad}} \neq \mathsf{ct}_{\mathsf{pad}}$, output $\perp$
$(\mathsf{ct}', \hat{\mathsf{ct}}'_{\mathsf{history}}) \leftarrow \mathsf{AE.Decrypt}(\mathsf{k}'_{\mathsf{ae}}, \mathsf{ct}_{\mathsf{payload}})$
if $(\mathsf{ct}', \hat{\mathsf{ct}}'_{\mathsf{history}}) = \perp$, output $\perp$
while $\hat{\mathsf{k}}'_{\mathsf{history}} \neq \perp$:
    $\mathsf{k}_{\mathsf{ae}} \leftarrow \mathsf{k}'_{\mathsf{ae}}$
    $\hat{\mathsf{k}}_{\mathsf{history}} \leftarrow \hat{\mathsf{k}}'_{\mathsf{history}}$
    $\mathsf{ct} \leftarrow \mathsf{ct}'$
    $\hat{\mathsf{ct}}_{\mathsf{history}} \leftarrow \hat{\mathsf{ct}}'_{\mathsf{history}}$
    $(\mathsf{k}'_{\mathsf{ae}}, \hat{\mathsf{k}}'_{\mathsf{history}}) \leftarrow \mathsf{AE.Decrypt}(\hat{\mathsf{k}}_{\mathsf{history}}, \hat{\mathsf{ct}}_{\mathsf{history}})$
    if $(\mathsf{k}'_{\mathsf{ae}}, \hat{\mathsf{k}}'_{\mathsf{history}}) = \perp$, output $\perp$
    $(\mathsf{ct}', \hat{\mathsf{ct}}'_{\mathsf{history}}) \leftarrow \mathsf{AE.Decrypt}(\mathsf{k}_{\mathsf{ae}}, \mathsf{ct})$
    if $(\mathsf{ct}', \hat{\mathsf{ct}}'_{\mathsf{history}}) = \perp$, output $\perp$
$\mathsf{m} \leftarrow \mathsf{AE.Decrypt}(\mathsf{k}'_{\mathsf{ae}}, \mathsf{ct}')$
Output $\mathsf{m}$
```

Fig. 3. Our nested scheme.

For confidentiality, we have the following concrete security bounds for all $h, d = \mathsf{poly}(\lambda)$ *and efficient adversaries* $\mathcal{A}$ *that make at most* Q *oracle queries:*

$$\Big| \Pr\big[\mathsf{Expt}^{\mathsf{upd\text{-}ind}}_{\Pi_{\mathsf{UAE}}}(\lambda, h, d, \mathcal{A}, 0) = 1\big] - \Pr\big[\mathsf{Expt}^{\mathsf{upd\text{-}ind}}_{\Pi_{\mathsf{UAE}}}(\lambda, h, d, \mathcal{A}, 1) = 1\big] \Big| \\ \leq 2h \cdot \varepsilon^{\mathsf{conf}}_{\mathsf{ae}}(\lambda) + 2h \cdot \varepsilon^{\mathsf{int}}_{\mathsf{ae}}(\lambda) + 2Q \cdot \varepsilon_{\mathsf{prg}}(\lambda) + 4Q \cdot \varepsilon^{\mathsf{rand}}_{\mathsf{ae}}(\lambda)$$

$$\left| \Pr\left[\mathsf{Expt}^{\mathsf{msg\text{-}conf}}_{\Pi_{\mathsf{UAE}}}(\lambda, h, d, \mathcal{A}, 0) = 1\right] - \Pr\left[\mathsf{Expt}^{\mathsf{msg\text{-}conf}}_{\Pi_{\mathsf{UAE}}}(\lambda, h, d, \mathcal{A}, 1) = 1\right] \right| \\ \leq (2h + 4Q) \cdot \varepsilon^{\mathsf{conf}}_{\mathsf{ae}}(\lambda) + 2h \cdot \varepsilon^{\mathsf{int}}_{\mathsf{ae}}(\lambda)$$

$$\left| \Pr\left[\mathsf{Expt}^{\mathsf{re\text{-}enc\text{-}ind}}_{\Pi_{\mathsf{UAE}}}(\lambda, h, d, \mathcal{A}, 0) = 1\right] - \Pr\left[\mathsf{Expt}^{\mathsf{re\text{-}enc\text{-}ind}}_{\Pi_{\mathsf{UAE}}}(\lambda, h, d, \mathcal{A}, 1) = 1\right] \right| \\ \leq (2h + 4Q) \cdot \varepsilon^{\mathsf{conf}}_{\mathsf{ae}}(\lambda) + 2h \cdot \varepsilon^{\mathsf{int}}_{\mathsf{ae}}(\lambda)$$

For integrity, we have the following bound for all $h, d = \mathsf{poly}(\lambda)$ and efficient adversaries $\mathcal{A}$ that make at most Q challenge, ReKeyGen*, or* ReEncrypt *queries:*

$$\Pr\left[\mathsf{Expt}^{\mathsf{int}}_{\Pi_{\mathsf{UAE}}}(\lambda, h, d, \mathcal{A}) = 1\right] \leq (h + Q) \cdot \varepsilon^{\mathsf{int}}_{\mathsf{ae}}(\lambda) + (h + Q) \cdot \varepsilon^{\mathsf{conf}}_{\mathsf{ae}}(\lambda) + Q/2^{\lambda}$$

5 UAE from Key-Homomorphic PRFs

In this section, we generalize the updatable authenticated encryption construction of Everspaugh et al. [15] that is built from a perfectly key-homomorphic PRF, to also work using an almost key-homomorphic PRF. We do this by incorporating a plaintext encoding scheme into the construction such that encrypted messages can still be decrypted correctly after noisy key rotations. We show that this generalized UAE construction satisfies our notion of confidentiality (Definition 3.4), but only satisfies a relaxed integrity property. We first describe the construction in Sect. 5.2, and then analyze and prove its security in Sect. 5.3.

5.1 Encoding Scheme

Our construction of an updatable authenticated encryption scheme relies on an *almost* key-homomorphic PRF for which key-homomorphism holds under small noise. To cope with the noise in our updatable encryption scheme in Sect. 5.2, we must encode messages prior to encrypting them such that they can be fully recovered during decryption. A simple way of encoding the messages is to pad them with additional least-significant bits. However, more sophisticated ways of encoding the messages are possible with general error-correcting codes. In our construction description in Sect. 5.2, we use the syntax of a general encoding scheme that is described in Fact 5.1 below. In Sect. 7, we test the performance of our construction in Sect. 5.2 with simple padding.

Fact 5.1. *Let n, q, γ be positive integers such that $\gamma < q/4$, $\mu = \mu(\lambda)$ be a polynomial in λ, and $\mathcal{M} = \left(\{0,1\}^{\mu(\lambda)}\right)_{\lambda \in \mathbb{N}}$ be a message space. Then there exists a set of algorithms* ($\mathsf{Encode}, \mathsf{Decode}$) *with the following syntax:*

- $\mathsf{Encode}(\mathsf{m}) \rightarrow (\mathsf{m}_1, \ldots, \mathsf{m}_\ell)$*: On input a message $\mathsf{m} \in \mathcal{M}_\lambda$, the encoding algorithm returns a set of vectors $\mathsf{m}_1, \ldots, \mathsf{m}_\ell \in \mathbb{Z}_q^n$ for some $\ell \in \mathbb{N}$.*
- $\mathsf{Decode}(\mathsf{m}_1, \ldots, \mathsf{m}_\ell) \rightarrow \mathsf{m}$*: On input a set of vectors $\mathsf{m}_1, \ldots, \mathsf{m}_\ell \in \mathbb{Z}_q^n$, the decoding algorithm returns a message $\mathsf{m} \in \mathcal{M}_\lambda$.*

The algorithms $(\mathsf{Encode}, \mathsf{Decode})$ *satisfy the following property: for all strings* $\mathsf{m} \in \mathcal{M}_\lambda$ *and any error vectors* $\mathbf{e} = \mathbf{e}_1, \ldots, \mathbf{e}_\ell \in [\gamma]^n$*, if we set* $(\mathsf{m}_1, \ldots, \mathsf{m}_\ell) \leftarrow \mathsf{Encode}(\mathsf{m})$*, we have*

$$\mathsf{Decode}(\mathsf{m}_1 + \mathbf{e}_1, \ldots, \mathsf{m}_\ell + \mathbf{e}_\ell) = \mathsf{m}.$$

Due to the use of an encoding scheme, our construction can be viewed as supporting only a bounded number of updates – the encoding can only support so much noise before decoding fails. However, for our almost key-homomorphic PRF construction in Sect. 5.2, a simple padding scheme can be used as the encoding scheme. In this case, the bound on the number of updates grows exponentially in the size of the parameters of the scheme and therefore, the construction can be interpreted as permitting unbounded updates.

5.2 Construction

We next present our UAE scheme from an almost key-homomorphic PRF. We analyze its security in the next two subsections.

```
$\underline{\mathsf{KeyGen}(1^\lambda, 1^t)}$:
$\mathsf{k} \leftarrow \mathsf{AE.KeyGen}(1^\lambda)$
Output $\mathsf{k}$

$\underline{\mathsf{ReKeyGen}(\mathsf{k}_1, \mathsf{k}_2, \hat{\mathsf{ct}})}$:
$\mu \leftarrow \mathsf{AE.Decrypt}(\mathsf{k}_1, \hat{\mathsf{ct}})$
if $\mu = \perp$, output $\perp$
$(\mathsf{k}_{\mathsf{prf}}, \mathsf{h}) \leftarrow \mu$
$\mathsf{k}'_{\mathsf{prf}} \xleftarrow{\mathrm{R}} \mathcal{K}_{\mathsf{PRF}}$
$\mathsf{k}^{\mathsf{up}}_{\mathsf{prf}} \leftarrow \mathsf{k}'_{\mathsf{prf}} - \mathsf{k}_{\mathsf{prf}}$
$\hat{\mathsf{ct}}' \leftarrow \mathsf{AE.Encrypt}\big(\mathsf{k}_2, (\mathsf{k}'_{\mathsf{prf}}, \mathsf{h})\big)$
$\Delta_{1,2,\hat{\mathsf{ct}}} \leftarrow (\hat{\mathsf{ct}}', \mathsf{k}^{\mathsf{up}}_{\mathsf{prf}})$

$\underline{\mathsf{ReEncrypt}\big(\Delta_{1,2,\hat{\mathsf{ct}}}, (\hat{\mathsf{ct}}, \mathsf{ct})\big)}$:
$(\hat{\mathsf{ct}}', \mathsf{k}^{\mathsf{up}}_{\mathsf{prf}}) \leftarrow \Delta_{1,2,\hat{\mathsf{ct}}}$
$(\mathsf{ct}_1, \ldots, \mathsf{ct}_\ell) \leftarrow \mathsf{ct}$
for $i \in [\ell]$:
    $\mathsf{ct}'_i \leftarrow \mathsf{ct}_i + F(\mathsf{k}^{\mathsf{up}}_{\mathsf{prf}}, i)$
$\mathsf{ct}' \leftarrow (\mathsf{ct}'_1, \ldots, \mathsf{ct}'_\ell)$
Output $(\hat{\mathsf{ct}}', \mathsf{ct}')$
```

```
$\underline{\mathsf{Encrypt}(\mathsf{k}, \mathsf{m})}$
$(\mathsf{m}_1, \ldots, \mathsf{m}_\ell) \leftarrow \mathsf{Encode}(\mathsf{m})$
$\mathsf{k}_{\mathsf{prf}} \xleftarrow{\mathrm{R}} \mathcal{K}_{\mathsf{PRF}}$
$\mathsf{h} \leftarrow H(\mathsf{m})$
$\hat{\mathsf{ct}} \leftarrow \mathsf{AE.Encrypt}\big(\mathsf{k}_{\mathsf{ae}}, (\mathsf{k}_{\mathsf{prf}}, \mathsf{h})\big)$
for $i \in [\ell]$:
    $\mathsf{ct}_i \leftarrow \mathsf{m}_i + F(\mathsf{k}_{\mathsf{prf}}, i)$
$\mathsf{ct} = (\mathsf{ct}_1, \ldots, \mathsf{ct}_\ell)$
Output $(\hat{\mathsf{ct}}, \mathsf{ct})$

$\underline{\mathsf{Decrypt}\big(\mathsf{k}, (\hat{\mathsf{ct}}, \mathsf{ct})\big)}$:
$\mu \leftarrow \mathsf{AE.Decrypt}(\mathsf{k}, \hat{\mathsf{ct}})$
if $\mu = \perp$, output $\perp$
$(\mathsf{k}_{\mathsf{prf}}, \mathsf{h}) \leftarrow \mu$
$(\mathsf{ct}_1, \ldots, \mathsf{ct}_\ell) \leftarrow \mathsf{ct}$
for $i \in [\ell]$:
    $\mathsf{m}_i \leftarrow \mathsf{ct}_i - F(\mathsf{k}_{\mathsf{prf}}, i)$
$\mathsf{m}' \leftarrow \mathsf{Decode}(\mathsf{m}_1, \ldots, \mathsf{m}_\ell)$
if $H(\mathsf{m}') = \mathsf{h}$, output $\mathsf{m}'$
else, output $\perp$
```

Fig. 4. Our UAE from almost Key-Homomorphic PRFs.

Construction 5.2 (UAE from almost Key-Homomorphic PRFs). *Let n, q, γ, and β be positive integers. Our construction uses the following:*

- *A standard authenticated encryption scheme* $\Pi_{\mathsf{AE}} = (\mathsf{AE.KeyGen}, \mathsf{AE.Encrypt}, \mathsf{AE.Decrypt})$ *with message space* $\mathcal{M} = (\mathcal{M}_\lambda)_{\lambda \in \mathbb{N}}$.
- *A β-almost key-homomorphic PRF* $F : \mathcal{K}_{\mathsf{PRF}} \times \{0,1\}^* \to \mathbb{Z}_q^n$ *where* $(\mathcal{K}_{\mathsf{PRF}}, +)$ *and* $(\mathbb{Z}_q^n, +)$ *form groups.*
- *A collision resistant hash family* $\mathcal{H} = \left\{H : \mathcal{M}_\lambda \to \{0,1\}^\lambda\right\}$. *To simplify the construction, we assume that a description of a concrete hash function* $H \xleftarrow{\mathrm{R}} \mathcal{H}$ *is included in each algorithm as part of a global set of parameters.*
- *An encoding scheme* $(\mathsf{Encode}, \mathsf{Decode})$ *that encodes messages in* $(\mathcal{M}, \lambda)_{\lambda \in \mathbb{N}}$ *as elements in* $\mathbb{Z}_q^n$. *The* Decode *algorithm decodes any error vectors* $\mathbf{e} \in [\gamma]^n$ *as in Fact 5.1 for any fixed* $\gamma = \beta \cdot \lambda^{\omega(1)}$.

We construct an updatable authenticated encryption scheme $\Pi_{\mathsf{UAE}} = (\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{ReKeyGen}, \mathsf{ReEncrypt}, \mathsf{Decrypt})$ *for message space* $(\mathcal{M}_\lambda)_{\lambda \in \mathbb{N}}$ *in Fig. 4.*

5.3 Security Under Relaxed Integrity

We will show in the next subsection that neither Construction 5.2 nor the construction of Everspaugh et al. [15] satisfy our integrity definition. To prove security of either scheme we must relax the notion of integrity in Definition 3.7 to obtain what we call *relaxed integrity*. In this section we define relaxed integrity and then prove security of Construction 5.2. In the next subsection we discuss the implications of relaxed integrity to the security of the scheme in practice.

The relaxed integrity experiment modifies Definition 3.7 (integrity) in two ways. First, we require that an adversary's queries to the re-encryption oracle are well-formed ciphertexts that do not decrypt to "$\perp$". Without this restriction, there is an attack on both Construction 5.2 and the Everspaugh et al. [15] scheme, as we will discuss below.

Second, we modify the adversary's winning condition in the integrity game. When we use an *almost* key-homomorphic PRFs to instantiate Construction 5.2, any re-encryption incurs a small error that affects the low-order bits of the ciphertext. Therefore, to achieve correctness, we encrypt an encoding of a message (Fact 5.1) such that the decryption algorithm can still recover the full message even if the low-ordered bits are corrupted. This forces the construction to violate traditional ciphertext integrity as an adversary can forge new ciphertexts by adding noise to the low-order bits of a ciphertext. Our construction still guarantees that an adversary cannot generate new ciphertexts by modifying plaintexts or the high-order bits of ciphertexts. To capture this formally, we require that the ciphertext space $\mathcal{CT}$ associated with the UAE has a corresponding metric function $d : \mathcal{CT} \times \mathcal{CT} \to \mathbb{Z}$ (e.g., Euclidean distance) that gives a distance between any two ciphertexts. Then, in our relaxed integrity definition that is parameterized with a positive integer $\gamma \in \mathbb{N}$, an adversary wins the security experiment only if it produces a valid ciphertext that differs from any of the ciphertexts that it is given by more than γ.

The rest of the definition of relaxed integrity exactly matches Definition 3.7. We present the formal definition of relaxed integrity in the full version [10].

Security. The following theorem states the compactness, correctness, and security properties of Construction 5.2. The proof is presented in the full version [10].

Theorem 5.3. *Let Π_{UAE} be the updatable authenticated encryption scheme in Construction 5.2. If the authenticated encryption scheme Π_{AE} satisfies correctness, $\varepsilon_{\mathsf{ae}}^{\mathsf{conf}}$-confidentiality and $\varepsilon_{\mathsf{ae}}^{\mathsf{int}}$-integrity, $F : \mathcal{K}_{\mathsf{PRF}} \times \{0,1\}^* \rightarrow \mathcal{Y}$ satisfies $\varepsilon_{\mathsf{prf}}$-security, and $H : \mathcal{M}_\lambda \rightarrow \{0,1\}^\lambda$ is a $\varepsilon_{\mathsf{cr}}$-secure collision resistant hash function, then Π_{UAE} satisfies strong compactness, correctness, confidentiality, and γ-relaxed integrity.*

For confidentiality, we have the following concrete security bounds for all $h, d = \mathsf{poly}(\lambda)$ and efficient adversaries $\mathcal{A}$ that make at most Q challenge queries:

$$\left| \Pr\left[\mathsf{Expt}_{\Pi_{\mathsf{UAE}}}^{\mathsf{conf}}(\lambda, h, d, \mathcal{A}, 0) = 1\right] - \Pr\left[\mathsf{Expt}_{\Pi_{\mathsf{UAE}}}^{\mathsf{conf}}(\lambda, h, d, \mathcal{A}, 1) = 1\right] \right|$$
$$\leq 2h \cdot \varepsilon_{\mathsf{ae}}^{\mathsf{conf}}(\lambda) + 2h \cdot \varepsilon_{\mathsf{ae}}^{\mathsf{int}}(\lambda) + 2Q \cdot \varepsilon_{\mathsf{prf}}(\lambda)$$

For integrity, we have the following bound for all $h, d = \mathsf{poly}(\lambda)$ and efficient adversaries $\mathcal{A}$:

$$\Pr\left[\mathsf{Expt}_{\Pi_{\mathsf{UAE}}}^{\mathsf{relaxed\text{-}int}}(\lambda, h, d, \gamma, \mathcal{A}) = 1\right] \leq h \cdot \varepsilon_{\mathsf{ae}}^{\mathsf{int}}(\lambda) + \varepsilon_{\mathsf{cr}}(\lambda)$$

We note that when we instantiate Construction 5.2 with a perfect key-homomorphic PRF, we can use the trivial encoding scheme for $\gamma = 0$. In this case, the relaxed integrity experiment $\mathsf{Expt}_{\Pi_{\mathsf{UAE}}}^{\mathsf{relaxed\text{-}int}}(\lambda, h, d, 0, \mathcal{A})$ is comparable to the ciphertext integrity notion in [15].

5.4 Consequences of Relaxed Integrity

The relaxed integrity definition from Sect. 5.3 places two restrictions on the adversary relative to our full integrity definition (Definition 3.7). We discuss these two restrictions and their implications below.

Weakened Re-encryption Oracle. The first restriction of relaxed integrity is the weakened re-encryption oracle, which only re-encrypts well-formed ciphertexts. This relaxation of the definition is necessary to prove security of Construction 5.2 as there exists a simple adversary that breaks the integrity experiment when it is provided arbitrary access to the re-encryption oracle $\mathcal{O}_{\mathsf{ReEncrypt}}$. This attack applies equally well to the construction of Everspaugh et al. [15].

To carry out the attack, the adversary does the following:

1. Uses encryption oracle $\mathcal{O}_{\mathsf{Encrypt}}$ to receive a ciphertext $(\hat{\mathsf{ct}}, \mathsf{ct}) \leftarrow \mathcal{O}_{\mathsf{Encrypt}}(i, \mathsf{m})$ for a message $\mathsf{m} \in \mathcal{M}_\lambda$ and an honest key index i. For simplicity, suppose that the message m is encoded as a single vector in $\mathbb{Z}_q^n$: $\mathsf{Encode}(\mathsf{m}) \in \mathbb{Z}_q^n$ and therefore, $\mathsf{ct} \in \mathbb{Z}_q^n$.

2. Subtracts an arbitrary vector m' from the ciphertext body $\tilde{\mathsf{ct}} \leftarrow \mathsf{ct} - \mathsf{m}'$.
3. Submits the ciphertext $(\hat{\mathsf{ct}}, \tilde{\mathsf{ct}})$ to the re-encryption oracle $\mathcal{O}_{\mathsf{ReEncrypt}}$ to receive a new ciphertext $(\hat{\mathsf{ct}}', \tilde{\mathsf{ct}}') \leftarrow \mathcal{O}_{\mathsf{ReEncrypt}}\big(i, j, (\hat{\mathsf{ct}}, \tilde{\mathsf{ct}})\big)$ for an honest key index j.
4. Returns $(\hat{\mathsf{ct}}', \tilde{\mathsf{ct}}' + \mathsf{m}')$ as the ciphertext forgery.

Since the re-encryption algorithm is homomorphic, we have

$$\mathcal{O}_{\mathsf{ReEncrypt}}(i, j, \hat{\mathsf{ct}}, \tilde{\mathsf{ct}} - \mathsf{m}') + \mathsf{m}' = \mathcal{O}_{\mathsf{ReEncrypt}}(i, j, \hat{\mathsf{ct}}, \tilde{\mathsf{ct}}).$$

Therefore, the ciphertext $(\hat{\mathsf{ct}}', \tilde{\mathsf{ct}}' + \mathsf{m})$ is a valid forgery. This attack is ruled out in the relaxed integrity experiment, where the re-encryption oracle $\mathcal{O}_{\mathsf{ReEncrypt}}$ outputs a re-encrypted ciphertext only when the input ciphertexts are well-formed.

To carry out the attack above, an adversary must have arbitrary access to a re-encryption oracle. Therefore, Construction 5.2 still provides security against any active adversary that has arbitrary access to the decryption oracle, but only observes key rotations on well-formed ciphertexts. For applications where an adversary (e.g. a corrupted server) gains arbitrary access to the re-encryption oracle, Construction 5.2 provides passive security as opposed to active security. This also applies to [15].

Handling Noise. The second restriction imposed on the adversary is needed due to the noise allowed in Construction 5.2. In particular, the encoding scheme used in the construction allows an adversary to create new ciphertexts by adding small amounts of noise to an existing ciphertext. In combination with the decryption oracle, an adversary can take advantage of this property to gain information about the age of a ciphertext using a chosen ciphertext attack. Namely, an adversary can take a ciphertext and incrementally add noise to it before submitting the ciphertext to the decryption oracle. Based on how much noise an adversary can add to the ciphertext before the decryption oracle returns $\perp$, the adversary can approximate the relative size of the noise in the ciphertext. Since each key rotation in increases the noise associated with a ciphertext by a fixed amount, an adversary can gain information about the age of the ciphertext by learning the size of the noise in the ciphertext. Hence, the age of a ciphertext can be exposed using a chosen ciphertext attack.

For applications where the age of a ciphertext is not sensitive information, Construction 5.2 can be used as an efficient alternative to existing UAE schemes. When combined with confidentiality (Definition 3.4), the relaxed integrity definition provides an "approximate" analogue of the traditional chosen-ciphertext security. To see this, take any CCA-secure encryption scheme Π_{Enc} and modify it into a scheme Π'_{Enc} that is identical to Π_{Enc}, but the encryption algorithm appends a bit 0 to every resulting ciphertext, and the decryption algorithm discards the last bit of the ciphertext before decrypting. The scheme Π'_{Enc} is no longer CCA-secure as an adversary can take any ciphertext and flip its last bit to produce different valid ciphertext. However, the introduction of the last bit does not cause the scheme Π'_{Enc} to be susceptible to any concrete attack that violates security. Similarly, Construction 5.2 does not satisfy full ciphertext integrity due to its noisy nature; however, it still suffices to guarantee CCA security in practice.

These variants of CCA security were previously explored under the name of *Replayable CCA* and *Detectable CCA* [14,18], where it was argued that they are sufficient to provide security against an active attacker in practice.

6 Almost Key-Homomorphic PRFs from Lattices

In this section, we construct an almost key-homomorphic PRF from the Learning with Errors (LWE) assumption [26]. There are a number of standard variants of the LWE assumption in the literature that give rise to efficient PRF constructions. For instance, using the Learning with Rounding (LWR) [6,11] assumption, one can construct an almost key-homomorphic PRF in both the random-oracle and standard models. However, any LWR-based PRF involves a modular rounding step [6] that forces the output space of the PRF to be quite small compared to the key space. Hence, these PRFs are less optimal for the application of updatable encryption as the noise that is incurred by each key updates grows faster in the smaller output space. In this work, we modify the existing LWR-based KH-PRF constructions to work over the ring variant of the LWE problem called the Ring Learning with Errors (RLWE) problem [22]. We provide the precise definition in the full version [10]. The use of RLWE as opposed to LWR (or Ring-LWR) allows us to construct almost KH-PRFs that can support more key updates when applied to Construction 5.2.

We construct an almost key-homomorphic PRF from the hardness of the Ring Learning with Errors problem as follows.

Construction 6.1. *Let n, q, B, r, ℓ be positive integers, $\mathcal{R} = \mathbb{Z}[X]/(\phi)$ a polynomial ring for $\phi \in \mathbb{Z}[X]$, $\mathcal{R}_q = \mathbb{Z}_q[X]/(\phi)$, and χ an error distribution over $\mathcal{E}_B \subseteq \mathcal{R}$. We let $\mathsf{Samp}_\chi : \{0,1\}^r \rightarrow \mathcal{E}_B$ be a sampler for the error distribution χ that takes in a uniformly random string in $\{0,1\}^r$ and produces a ring element in $\mathcal{E}_B$ according to the distribution χ. For our construction, we set $\mathcal{X} = \{0,1\}^\ell$ to be the domain of the PRF and use two hash functions that are modeled as random oracles:*

- $H_0 : \{0,1\}^\ell \rightarrow \mathcal{R}_q$,
- $H_1 : \mathcal{R}_q \times \{0,1\}^\ell \rightarrow \{0,1\}^r$.

We define our pseudorandom function $F : \mathcal{R}_q \times \{0,1\}^\ell \rightarrow \mathcal{R}_q$ as follows:

```
$F(s, x)$:

1. Evaluate $a \leftarrow H_0(x)$, $\rho \leftarrow H_1(s, x)$.
2. Sample $e \leftarrow \mathsf{Samp}_\chi(\rho)$.
3. Output $y \leftarrow a \cdot s + e$.
```

We summarize the security and homomorphic properties of the PRF construction above in the following theorem. We provide its proof in the full version [10].

Theorem 6.2. *Let n, q, B, r, ℓ be positive integers, $\mathcal{R} = \mathbb{Z}[X]/(\phi)$ a polynomial ring for $\phi \in \mathbb{Z}[X]$, $\mathcal{R}_q = \mathbb{Z}_q[X]/(\phi)$, and χ an error distribution over $\mathcal{E}_B \subseteq \mathcal{R}_q$. Then, assuming that $\mathsf{RLWE}_{n,q,\chi}$ [10] is $\varepsilon_{\mathsf{RLWE}}$-secure, the pseudorandom function in Construction 6.1 is a $\varepsilon_{\mathsf{prf}}$-secure $2B$-almost key-homomorphic PRF (Definition 2.1) with key space and range $(\mathcal{R}_q, +)$ such that $\varepsilon_{\mathsf{prf}}(\lambda) = \varepsilon_{\mathsf{RLWE}}(\lambda)$.*

7 Evaluation

In this section we evaluate the performance of our nested and KH-PRF based UAE constructions (Constructions 4.2 and 5.2), comparing their performance to that of the ReCrypt scheme of Everspaugh et al. [15] both in terms of running time and ciphertext size. We find that our constructions dramatically improve on the running time of the Everspaugh et al. [15] UAE at the cost of an increase in ciphertext size (albeit our ciphertext sizes are still considerably smaller than those of ciphertext-independent schemes [12,20,21]).

RLWE Parameters

	$\|q\| = 28$	$\|q\| = 60$	$\|q\| = 120$	$\|q\| = 128$
n	1024	2048	4096	4096
B	352	498	704	704

Fig. 5. RLWE parameters for each value of $|q|$ used in our evaluation.

We implemented our constructions in C and evaluated their performance on an 8-core Ubuntu virtual machine with 4 GB of RAM running on a Windows 10 computer with 64 GB and a 12-core AMD 1920x processor @3.8 GHz. We use AES-NI instructions to accelerate AES and AVX instructions for applicable choices of lattice parameters. Our implementation is single-threaded and does not take advantage of opportunities for parallelism beyond a single core. We rely on OpenSSL for standard cryptographic primitives and rely on prior implementations of NTT and the SHAKE hash function [4,27]. All numbers reported are averages taken over at least 1,000 trials. Our choice of lattice parameters for each modulus size $|q|$ (the length of q in bits) is based on the best known attacks on RLWE [3], as shown in Fig. 5. We discuss some aspects of our KH-PRF implementation in the full version [10]. Our implementation is open source and available at [1].

Encryption and Re-encryption Costs. Figure 6 shows encryption and re-encryption times for our KH-PRF based UAE construction for various block sizes of the underlying KH-PRF as well as the ReCrypt scheme [15] and our nested construction with padding configured to support up to 128 re-encryptions. Our lattice-based KH-PRF scheme, when run with the best parameters, has from 250× to over 500× higher encryption throughput than ReCrypt as the message size increases from 4 kB to 100 kB. We note that, since KH-PRFs imply

Encrypt and ReEncrypt Throughput (MB/sec)

	KH-PRF UAE					ReCrypt	Nested
	$\lvert q\rvert = 28$	$\lvert q\rvert = 28$ (AVX)	$\lvert q\rvert = 60$	$\lvert q\rvert = 120$	$\lvert q\rvert = 128$	[15]	$t = 128$
			4KB Messages				
Encrypt	24.85	**31.97**	20.32	0.76	0.70	0.12	406.69
ReEncrypt	29.80	**41.03**	32.13	0.82	0.74	0.14	706.37
			32KB Messages				
Encrypt	29.85	39.89	**61.90**	5.94	5.50	0.12	1836.9
ReEncrypt	32.33	44.51	**83.06**	6.43	5.85	0.15	2606.8
			100KB Messages				
Encrypt	31.03	41.63	**65.11**	9.42	9.12	0.12	3029.5
ReEncrypt	33.30	45.77	**79.63**	9.92	8.70	0.14	3766.2

Fig. 6. Comparing the throughput of our KH-PRF, ReCrypt, and our nested construction configured to allow 128 re-encryptions, for messages of length 4 kB, 32 kB, and 100 kB. Higher numbers are better. Our KH-PRF is evaluated with four choices of q. The AVX column refers to an implementation that takes advantage of Intel's AVX vector instructions.

key exchange [2], we should not expect to be able to instantiate the KH-PRF approach with performance any better than that of public key primitives. The nested AES construction, on the other hand, has 13–47× the encryption throughput of our KHPRF-based construction. The nested AES scheme approaches the machine's peak AES throughput of 4.45 GB/s as the message size increases.

We find that for small messages (4 kB), our KH-PRF with 28 bit output space (and accelerated with AVX instructions) performs the best, but as messages grow larger the KH-PRF with 60 bit output space outperforms other categories. Larger block sizes tend to perform worse because the output of the PRF no longer fits into compiler provided primitive types, causing arithmetic operations to become less efficient. Increasing the message size improves performance because the proportion of total time occupied by fixed-cost operations decreases, e.g.,

KeyGen and ReKeyGen Time (μsecs)

	KH-PRF UAE	ReCrypt	Nested
	$\lvert q\rvert = 60$	[15]	$t = 128$
	32KB Messages		
KeyGen	3.0	1.0	2.6
ReKeyGen	72.7	308.8	10.1

Fig. 7. KeyGen and ReKeyGen costs. The main differences in performance are caused by whether the ReKeyGen algorithm needs to sample only AES keys or also KH-PRF keys, the type of KH-PRF used, and the number of ciphertexts contained in the update token.

due to the large blocks in which the KH-PRF output is generated. We run our remaining experiments with $|q| = 60$ because it has the overall best performance.

Key Generation. Key generation is a faster and less time-sensitive operation than encryption, re-encryption, and decryption because it only occurs once for a small ciphertext header before an entire ciphertext is encrypted or re-encrypted. We show the performance of our KH-PRF based UAE as well as ReCrypt and nested encryption on KeyGen and ReKeyGen operations in Fig. 7. Generating a key in all three schemes is very fast because it only requires generating a random 128-bit symmetric key. The cost of rekeying depends on the underlying tool used to re-encrypt. ReKeyGen runs very quickly in the nested construction because it only consists of a couple AES-GCM encryptions of a fixed-size ciphertext header. The other two constructions rely on different types of KH-PRFs and incur most of their costs in generating the update keys for those PRFs.

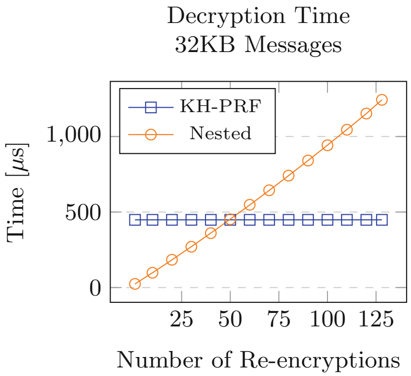

Fig. 8. KH-PRF based UAE ($|q| = 60$) and nested UAE ($t = 128$) decryption times. The KH-PRF construction decrypts faster than nested AES when there are more than 50 re-encryptions. ReCrypt is not depicted as it takes 500× longer than our KH-PRF based UAE to decrypt.

Ciphertext Expansion
32KB Messages

KH-PRF UAE	
$\lvert q\rvert = 28$	133%
$\lvert q\rvert = 60$	36%
$\lvert q\rvert = 120$	20%
$\lvert q\rvert = 128$	19%
Nested UAE	
$t = 20$	3%
$t = 128$	19%
ReCrypt [15]	3%

Fig. 9. Ciphertext body expansion for the KH-PRF based UAE, Nested UAE, and ReCrypt. Our constructions generally have larger ciphertext expansion than ReCrypt, although the Nested UAE matches ReCrypt for some settings, e.g., annually re-keying data for 20 years.

Decryption Costs. Figure 8 shows decryption costs for our two main constructions and the tradeoffs between them. We omit the decryption performance of ReCrypt from this graph because it is 500× slower than our KH-PRF based construction and is strictly dominated by both schemes for the range of parameters we measured. Decryption time for the nested AES construction depends linearly on the number of re-encryptions that have occurred because decryption needs to remove each layer of encryption to reach the plaintext. As such, it begins much faster than the KH-PRF construction, as it only requires standard symmetric primitives for which hardware acceleration is available, but becomes slower after about 50 re-encryptions. The KH-PRF construction could also vary its performance slightly based on the number of expected re-encryptions by varying the amount of padding applied in the message encoding process. However, we chose to evaluate the scheme with a fixed amount of padding that is enough to support about 128 re-encryptions.

Ciphertext Size. The ciphertext size of a ciphertext-dependent UAE scheme consists of two parts: a fixed-size header and the body, whose size depends on the plaintext. Figure 9 compares ciphertext body expansion between our constructions and ReCrypt. Our KH-PRF based scheme and ReCrypt have 80-Byte headers, while our nested construction has a 116-Byte header. Our KH-PRF based construction is implemented with padding on each block depending on the size $|q|$. For example, a 60-bit block contains 44 bits of plaintext and 16 bits of padding. This corresponds to a 36% ciphertext size expansion. The lowest ciphertext expansion for our evaluation of the KH-PRF based scheme occure when $|q| = 128$, with 19% expansion. ReCrypt has lower ciphertext expansion, at 3%. The ciphertext size of our nested construction depends on the expected number of encryptions. It has a constant 32-Byte overhead on top of the plaintext, followed by another 48 Bytes for each re-encryption. For a 32 kB message, a ReCrypt ciphertext takes 33 kB and a ciphertext under our KH-PRF scheme takes 43.6 kB. A ciphertext under our nested construction will match the size of a ReCrypt ciphertext after 19 re-encryptions. This fits well with a ciphertext that is re-encrypted once a year over a 20-year lifetime. Supporting 128 re-encryptions still only requires a 38.3 kB ciphertext, matching the expansion of the KH-PRF based PRF when $|q| = 128$.

Conclusions. Based on the performance of the schemes we evaluated, we can make the following recommendations:

- If the ciphertext is to be re-encrypted only 10 or 20 times over the course of its lifetime, say once a year for twenty years to satisfy NIST recommendations [7] and PCI DSS [25] requirements, then one should use the nested construction, as it will provide the best performance and ciphertext size. This is especially true of ciphertexts that are decrypted infrequently.
- If the ciphertext is to be re-encrypted more frequently and its age is sensitive information, then Recrypt [15] should be used.
- If the ciphertext is to be re-encrypted frequently, but its age is less sensitive, then our almost KH-PRF based scheme can be used for high performance.

Future Work. We have constructed a performant updatable encryption scheme based on RLWE, but it remains an open problem to construct a UAE scheme from RLWE that satisfies our strongest integrity definition with decryption time independent of ciphertext age. We hope that future work will result in such a construction.

Acknowledgments. This work was funded by NSF, DARPA, a grant from ONR, and the Simons Foundation. Opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of DARPA. Part of this work was done while the third author was visiting the Simons Institute for the Theory of Computing as a Ripple Research Fellow.

References

1. Source code repository. https://github.com/moshih/UpdateableEncryption_Code
2. Alamati, N., Montgomery, H., Patranabis, S.: Symmetric primitives with structured secrets. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019. LNCS, vol. 11692, pp. 650–679. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-26948-7_23
3. Albrecht, M.R., Player, R., Scott, S.: On the concrete hardness of learning with errors. J. Math. Cryptol. **9**(3), 169–203 (2015)
4. Alkim, E., Ducas, L., Pöppelmann, T., Schwabe, P.: Post-quantum key exchange - a new hope. In: USENIX Security (2016)
5. Banerjee, A., Peikert, C.: New and improved key-homomorphic pseudorandom functions. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014. LNCS, vol. 8616, pp. 353–370. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44371-2_20
6. Banerjee, A., Peikert, C., Rosen, A.: Pseudorandom functions and lattices. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 719–737. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29011-4_42
7. Barker, E.: NIST special publication 800–57 part 1 revision 4: recommendation for key management (2016)
8. Bellare, M., Rogaway, P.: Random oracles are practical: a paradigm for designing efficient protocols. In: CCS (1993)
9. Bernstein, D.J.: Curve25519: new Diffie-Hellman speed records. In: Yung, M., Dodis, Y., Kiayias, A., Malkin, T. (eds.) PKC 2006. LNCS, vol. 3958, pp. 207–228. Springer, Heidelberg (2006). https://doi.org/10.1007/11745853_14
10. Boneh, D., Eskandarian, S., Kim, S., Shih, M.: Improving speed and security in updatable encryption schemes. Cryptology ePrint Archive, Report 2020/222 (2020). https://eprint.iacr.org/2020/222
11. Boneh, D., Lewi, K., Montgomery, H., Raghunathan, A.: Key homomorphic PRFs and their applications. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013. LNCS, vol. 8042, pp. 410–428. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40041-4_23
12. Boyd, C., Davies, G.T., Gjøsteen, K., Jiang, Y.: Fast and secure updatable encryption. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020. LNCS, vol. 12170, pp. 464–493. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-56784-2_16
13. Brakerski, Z., Vaikuntanathan, V.: Constrained key-homomorphic PRFs from standard lattice assumptions. In: Dodis, Y., Nielsen, J.B. (eds.) TCC 2015. LNCS, vol. 9015, pp. 1–30. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46497-7_1
14. Canetti, R., Krawczyk, H., Nielsen, J.B.: Relaxing chosen-ciphertext security. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 565–582. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45146-4_33

15. Everspaugh, A., Paterson, K., Ristenpart, T., Scott, S.: Key rotation for authenticated encryption. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017. LNCS, vol. 10403, pp. 98–129. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63697-9_4
16. Fiat, A., Shamir, A.: How to prove yourself: practical solutions to identification and signature problems. In: Odlyzko, A.M. (ed.) CRYPTO 1986. LNCS, vol. 263, pp. 186–194. Springer, Heidelberg (1987). https://doi.org/10.1007/3-540-47721-7_12
17. Google. Key rotation. https://cloud.google.com/kms/docs/key-rotation
18. Hohenberger, S., Lewko, A., Waters, B.: Detecting dangerous queries: a new approach for chosen ciphertext security. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 663–681. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29011-4_39
19. Kim, S.: Key-homomorphic pseudorandom functions from LWE with small modulus. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020. LNCS, vol. 12106, pp. 576–607. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-45724-2_20
20. Klooß, M., Lehmann, A., Rupp, A.: (R)CCA secure updatable encryption with integrity protection. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019. LNCS, vol. 11476, pp. 68–99. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17653-2_3
21. Lehmann, A., Tackmann, B.: Updatable encryption with post-compromise security. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018. LNCS, vol. 10822, pp. 685–716. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-78372-7_22
22. Lyubashevsky, V., Peikert, C., Regev, O.: On ideal lattices and learning with errors over rings. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 1–23. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13190-5_1
23. Lyubashevsky, V., Peikert, C., Regev, O.: A toolkit for ring-LWE cryptography. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 35–54. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38348-9_3
24. Naor, M., Pinkas, B., Reingold, O.: Distributed pseudo-random functions and KDCs. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 327–346. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48910-X_23
25. PCI Security Standards Council. Payment card industry data security standard (2018)
26. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. In: STOC (2005)
27. Seiler, G.: Faster AVX2 optimized NTT multiplication for Ring-LWE lattice cryptography. IACR Cryptology ePrint Archive 2018:39 (2018)