Improving Speed and Security in Updatable Encryption Schemes

Dan Boneh[*] Saba Eskandarian[†] Sam Kim[‡] Maurice Shih[§]

Abstract

An updatable encryption scheme is a symmetric-key encryption scheme that supports key-rotation on ciphertexts. A server that hosts a user's encrypted data can use a user-provided *update token* to rotate the key under which the data is encrypted while not learning any information about the underlying data. Recent work on updatable encryption led to several security definitions and proposed constructions. However, our understanding of this notion still remains quite limited, and the existing constructions are not yet efficient enough for practical adoption.

This work's contributions are threefold. First, we introduce new definitions for updatable encryption (in the *ciphertext-dependent* setting) that capture desirable security properties not covered in prior work. Next, 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 supports a (nearly) unbounded number of updates and relies on *almost* key-homomorphic PRFs. We construct an efficient almost key-homomorphic PRF from the Ring Learning with Errors (RLWE) assumption to concretely instantiate our second construction. Finally, we implement both constructions and compare their performance to prior work. Our RLWE-based construction outperforms an existing updatable encryption scheme based on the hardness of DDH in elliptic-curve groups by over 200× in speed. Our construction based only 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, at which point the RLWE construction dominates.

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 [Bar16], the Payment Card Industry Data Security Standard (PCI DSS) [PCI18], and Google's cloud security recommendations [Goo].

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 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 [BLMR13, EPRS17, LT18, KLR19, BDGJ19] 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

[*]Stanford University. Email: `dabo@cs.stanford.edu`.

[†]Stanford University. Email: `saba@cs.stanford.edu`.

[‡]Stanford University. Email: `skim13@cs.stanford.edu`. Part of this work was done while the author was visiting the Simons Institute for the Theory of Computing.

[§]Cisco Systems. Email: `maushih@cisco.com`

as $\mathsf{ReKeyGen}(\mathsf{k}, \mathsf{k}') \to \Delta$, 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 $\mathsf{ReEncrypt}(\Delta, \mathsf{ct}) \to \mathsf{ct}'$, 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 can then run 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. [BLMR13] against passive adversaries, and was improved and extended to provide security against active adversaries by Everspaugh et al. [EPRS17].

However, we show in Section 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. This issue was recently independently pointed out in [BDGJ19].

To address this leakage, 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 (Sections 3.2 and 3.3). We also strengthen the integrity definition of [EPRS17] to cover additional tampering attacks, as discussed in Section 3.4.

Constructing updatable encryption Next, we look for efficient constructions that satisfy our definitions. We give two constructions: one based on nested authenticated encryption and another based on the Ring Learning With Errors (RLWE) problem [Reg05, LPR13].

Our first construction, presented in Section 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. The main 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 relatively small. Our implementation and experiments, discussed below, make this precise.

Our second construction, presented in Section 5, makes use of an almost key-homomorphic PRF (KH-PRF) built from the RLWE problem. Recall that a key-homomorphic PRF (KH-PRF) [NPR99, BLMR13] is a secure PRF $F : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$, 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 2.5). 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. This basic idea, also used in prior work, must be enhanced with several additional components to satisfy the definitions of confidentiality and integrity.

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 [FS86, BR93], there is a simple KH-PRF based on the Decision Diffie-Hellman (DDH) assumption [NPR99], and a simple almost KH-PRF based on the Learning With Errors (LWE) problem [BLMR13]. There are also several KH-PRFs whose security does not depend on random oracles, as discussed in the related work section.

Everspaugh et al. [EPRS17] constructed an updatable encryption scheme by combining a key-homomorphic PRF with authenticated encryption and a collision-resistant hash function. They evaluated their construction using the KH-PRF derived from DDH, in the random oracle model, instantiated in the 256-bit elliptic curve Curve25519 [Ber06].

We extend the scheme of Everspaugh et al. [EPRS17] to also work using an *almost* key-homomorphic PRF. This leads to some ciphertext expansion to handle the noise that arises from the imperfection of the KH-PRF. We then construct an efficient almost key-homomorphic PRF from the Ring Learning with Errors (RLWE) assumption [Reg05, LPR13], also in the random oracle model, to instantiate our updatable encryption scheme. Our KH-PRF is significantly faster than the DDH-based scheme.

Implementation and experiments In Section 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. [EPRS17], 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 Section 7. Implementations of both constructions are open source and available at [imp].

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 constructions that satisfy the stronger definitions. Finally, we experiment with both constructions and report on their real world performance and ciphertext expansion. Our second construction, based on a key-homomorphic PRF from RLWE, is considerably faster than the previous construction of Everspaugh et al. [EPRS17], 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 [BLMR13, EPRS17] and *ciphertext-independent* schemes [LT18, KLR19, BDGJ19]. 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 Appendix B. Further discussion of the two models can be found in [LT18].

Key-homomorphic PRFs. The concept of key-homomorphic PRFs was introduced by Naor, Pinkas, and Reingold [NPR99], and was first formalized as a cryptographic primitive by Boneh et al. [BLMR13], 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 $G : \mathcal{S} \to \mathcal{S}^2$ gives a key-homomorphic PRF. More constructions for key-homomorphic PRFs from LWE include [BP14, BV15, Kim20].

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 $x \xleftarrow{\text{R}} S$ 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.

2.1 Basic Cryptographic Primitives

In this section, we review a number of basic cryptographic primitives that we use in this work. To analyze the exact security of our constructions in Sections 4 and 5, we parameterize the security of these notions with respect to *advantage functions* $\varepsilon : \mathbb{N} \to \mathbb{R}$ that bounds the probability of an efficient adversary in breaking the security of the primitive.

Definition 2.1 (Collision-Resistant Hash Functions)**.** Let $\mathcal{H}_\lambda = \{H : \mathcal{X} \to \mathcal{Y}_\lambda\}$ be a family of hash functions. We say that $\mathcal{H}$ is $\varepsilon_{\mathsf{cr}}$-secure as a *collision resistant hash function* if for all efficient adversaries $\mathcal{A}$, we have

$$\Pr\left[\mathcal{A}(1^\lambda, H) \to (x_0, x_1) \wedge H(x_0) = H(x_1) : H \xleftarrow{\text{R}} \mathcal{H}_\lambda\right] = \varepsilon_{\mathsf{cr}}(\lambda).$$

We say that $\mathcal{H}$ is secure as a collision resistant hash function if $\varepsilon_{\mathsf{cr}}(\lambda) = \mathsf{negl}(\lambda)$.

Definition 2.2 (Pseudorandom Generators)**.** Let $G : \mathcal{X}_\lambda \to \mathcal{Y}_\lambda$ be a keyed function. We say that G is $\varepsilon_{\mathsf{prg}}$-secure as a *pseudorandom generator* (PRG) if for all efficient adversaries $\mathcal{A}$, we have

$$\left|\Pr\left[\mathcal{A}(1^\lambda, y_0) = 1 : s \xleftarrow{\text{R}} \mathcal{X}_\lambda, y_0 \leftarrow G(s)\right] - \Pr\left[\mathcal{A}(1^\lambda, y_1) = 1 : y_1 \xleftarrow{\text{R}} \mathcal{Y}_\lambda\right]\right| = \varepsilon_{\mathsf{prg}}(\lambda).$$

We say that G is a secure pseudorandom generator if $|\mathcal{X}_\lambda| < |\mathcal{Y}_\lambda|$, and $\varepsilon_{\mathsf{prg}}(\lambda) = \mathsf{negl}(\lambda)$.

2.2 Pseudorandom Functions

In this section, we review the definition of a pseudorandom function.

Definition 2.3 (Pseudorandom Functions [GGM84])**.** Let $F : \mathcal{K}_\lambda \times \mathcal{X} \to \mathcal{Y}$ be a keyed function. We say that F is $\varepsilon_{\mathsf{prf}}$-secure as a *pseudorandom function* (PRF) if for all efficient adversaries $\mathcal{A}$, we have

$$\left|\Pr\left[\mathcal{A}^{F(\mathsf{k},\cdot)}(1^\lambda) = 1 : \mathsf{k} \xleftarrow{\text{R}} \mathcal{K}_\lambda\right] - \Pr\left[\mathcal{A}^{f(\cdot)}(1^\lambda) = 1 : f \xleftarrow{\text{R}} \mathsf{Funs}[\mathcal{X}_\lambda, \mathcal{Y}_\lambda]\right]\right| = \varepsilon_{\mathsf{prf}}(\lambda),$$

where $\mathsf{Funs}[\mathcal{X}, \mathcal{Y}]$ denotes the set of all functions with domain $\mathcal{X}$ and range $\mathcal{Y}$. We say that F is a secure pseudorandom function if $\varepsilon_{\mathsf{prf}}(\lambda) = \mathsf{negl}(\lambda)$.

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 its evaluation on any fixed input $x \in \mathcal{X}$ is homomorphic with respect to these group structures. Formally, we define a key-homomorphic PRF as follows.

Definition 2.4 (Key-Homomorphic PRFs [NPR99, BLMR13])**.** Let $(\mathcal{K}, \oplus)$, $(\mathcal{Y}, \otimes)$ be groups. Then, a keyed function $F : \mathcal{K}_\lambda \times \mathcal{X}_\lambda \to \mathcal{Y}_\lambda$ is a *key-homomorphic* PRF (KH-PRF) if

- F is a secure PRF as specified in Definition 2.3.
- For every key $\mathsf{k}_1, \mathsf{k}_2 \in \mathcal{K}$ and every input $x \in \mathcal{X}$, we have

$$F(\mathsf{k}_1, x) \otimes F(\mathsf{k}_2, x) = F(\mathsf{k}_1 \oplus \mathsf{k}_2, x).$$

In this work, 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 $F(k_1, x) \otimes F(k_2, x) \approx F(k_1 \oplus k_2, x)$. Precisely, we define an almost key-homomorphic PRF as follows.

Definition 2.5 (Almost Key-Homomorphic PRFs [BLMR13]). 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 (Definition 2.3).
- 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

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

2.3 Authenticated Encryption

We recall the notion of an authenticated encryption scheme [BN00].

Definition 2.6 (Authenticated Encryption [BN00]). An *authenticated encryption* scheme for a message space $\mathcal{M}$ is a tuple of efficient algorithms $\Pi_{\mathsf{AE}} = (\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ that have the following syntax:

- $\mathsf{KeyGen}(1^\lambda) \to \mathsf{k}$: On input a security parameter λ, the key-generation algorithm returns a key k.
- $\mathsf{Encrypt}(\mathsf{k}, \mathsf{m}) \to \mathsf{ct}$: On input a key k and a message $\mathsf{m} \in \mathcal{M}$, the encryption algorithm returns a ciphertext ct.
- $\mathsf{Decrypt}(\mathsf{k}, \mathsf{ct}) \to \mathsf{m}/\bot$: On input a key k and a ciphertext ct, the decryption algorithm returns a message m or $\bot$.

We define the correctness, confidentiality, and integrity properties for an authenticated encryption scheme in the standard way.

Definition 2.7 (Correctness). We say that an authenticated encryption scheme $\Pi_{\mathsf{AE}} = (\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ is *correct* if for all $\lambda \in \mathbb{N}$ and $\mathsf{m} \in \mathcal{M}$, we have

$$\Pr\left[\mathsf{Decrypt}\big(\mathsf{k}, \mathsf{Encrypt}(\mathsf{k}, \mathsf{m})\big) = \mathsf{m}\right] = 1,$$

where $\mathsf{k} \leftarrow \mathsf{KeyGen}(1^\lambda)$.

Definition 2.8 (Confidentiality). Let $\Pi_{\mathsf{AE}} = (\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ be an authenticated encryption scheme for a message space $\mathcal{M}$. We say that Π_{AE} satisfies $\varepsilon_{\mathsf{ae}}^{\mathsf{conf}}$-*confidentiality* if for all efficient adversaries $\mathcal{A}$, we have

$$\left| \Pr\left[\mathcal{A}^{\mathcal{O}_{\mathsf{k},0}(\cdot,\cdot)}(1^\lambda) = 1\right] - \Pr\left[\mathcal{A}^{\mathcal{O}_{\mathsf{k},1}(\cdot,\cdot)}(1^\lambda) = 1\right] \right| = \varepsilon_{\mathsf{ae}}^{\mathsf{conf}}(\lambda),$$

where $\mathsf{k} \leftarrow \mathsf{KeyGen}(1^\lambda)$, and the oracle $\mathcal{O}_{\mathsf{k},b}$ for $b \in \{0,1\}$ is defined as follows:

- $\mathcal{O}_{\mathsf{k},b}(\mathsf{m}^{(0)}, \mathsf{m}^{(1)})$: On input two messages $\mathsf{m}^{(0)}, \mathsf{m}^{(1)} \in \mathcal{M}$, the oracle computes $\mathsf{ct} \leftarrow \mathsf{Encrypt}(\mathsf{k}, \mathsf{m}^{(b)})$ and returns ct.

We say that Π_{AE} satisfies confidentiality if $\varepsilon_{\mathsf{ae}}^{\mathsf{conf}}(\lambda) = \mathsf{negl}(\lambda)$.

Definition 2.9 (Integrity). Let $\Pi_{\mathsf{AE}} = (\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ be an authenticated encryption scheme for a message space $\mathcal{M}$. We say that Π_{AE} satisfies $\varepsilon_{\mathsf{ae}}^{\mathsf{int}}$-*integrity* if for all efficient adversaries $\mathcal{A}$, we have

$$\Pr\left[\mathcal{A}^{\mathcal{O}_{\mathsf{k}}(\cdot)}(1^\lambda) = \mathsf{ct} \wedge \mathsf{Decrypt}(\mathsf{k}, \mathsf{ct}) \neq \bot \wedge \mathsf{ct} \notin \mathsf{Table}\right] = \varepsilon_{\mathsf{ae}}^{\mathsf{int}}(\lambda),$$

where $\mathsf{k} \xleftarrow{\mathrm{R}} \mathsf{KeyGen}(1^\lambda)$, and the oracle $\mathcal{O}_{\mathsf{k}}$ and table Table are defined as follows:

- $\mathcal{O}_{\mathsf{k}}(\mathsf{m})$: On input a message $\mathsf{m} \in \mathcal{M}$, the oracle computes $\mathsf{ct} \leftarrow \mathsf{Encrypt}(\mathsf{k}, \mathsf{m})$, adds $\mathsf{Table} = \mathsf{Table} \cup \{\mathsf{ct}\}$, and returns ct.

We say that Π_{AE} satisfies integrity if $\varepsilon_{\mathsf{ae}}^{\mathsf{int}}(\lambda) = \mathsf{negl}(\lambda)$.

For our updatable encryption scheme in Section 4, we make use of authenticated encryption schemes that satisfy a stronger confidentiality requirement than Definition 2.8. 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. 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.

Definition 2.10 (Ciphertext Pseudorandomness)**.** Let $\Pi_{\mathsf{AE}} = (\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ be an authenticated encryption scheme for a message space $\mathcal{M}$. We say that Π_{AE} satisfies $\varepsilon_{\mathsf{ae}}^{\mathsf{rand}}$-*ciphertext pseudorandomness* if for all efficient adversaries $\mathcal{A}$, we have

$$\left| \Pr\left[\mathcal{A}^{\mathcal{O}_{\mathsf{k},0}(\cdot)}(1^\lambda) = 1\right] - \Pr\left[\mathcal{A}^{\mathcal{O}_{\mathsf{k},1}(\cdot)}(1^\lambda) = 1\right] \right| = \varepsilon_{\mathsf{ae}}^{\mathsf{rand}}(\lambda),$$

where $\mathsf{k} \leftarrow \mathsf{KeyGen}(1^\lambda)$, and the oracles $\mathcal{O}_{\mathsf{k},b}$ for $b \in \{0,1\}$ are defined as follows:

- $\mathcal{O}_{\mathsf{k},0}(\mathsf{m})$: On input a message $\mathsf{m} \in \mathcal{M}$, the oracle computes $\mathsf{ct} \leftarrow \mathsf{Encrypt}(\mathsf{k}, \mathsf{m})$ and returns ct.
- $\mathcal{O}_{\mathsf{k},1}(\mathsf{m})$: On input a message $\mathsf{m} \in \mathcal{M}$, the oracle computes $\mathsf{ct} \leftarrow \mathsf{Encrypt}(\mathsf{k}, \mathsf{m})$, samples $\mathsf{ct}' \xleftarrow{\mathrm{R}} \{0,1\}^{|\mathsf{ct}|}$, and returns ct'.

We say that Π_{AE} satisfies ciphertext pseudorandomness if $\varepsilon_{\mathsf{ae}}^{\mathsf{rand}}(\lambda) = \mathsf{negl}(\lambda)$.

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. [EPRS17]. We start by defining the syntax for an updatable encryption scheme and its compactness and correctness conditions in Section 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 Section 4 and Section 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.
- $\mathsf{ReKeyGen}(\mathsf{k}_1, \mathsf{k}_2, \hat{\mathsf{ct}}) \to \Delta_{1,2,\hat{\mathsf{ct}}}/\bot$: 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 $\bot$.
- $\mathsf{ReEncrypt}(\Delta, (\hat{\mathsf{ct}}, \mathsf{ct})) \to (\hat{\mathsf{ct}}', \mathsf{ct}')/\bot$: 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 $\bot$.

- $\mathsf{Decrypt}(\mathsf{k}, (\hat{\mathsf{ct}}, \mathsf{ct})) \to \mathsf{m}/\perp$: 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 functions $\mathsf{poly}_1(\cdot)$, $\mathsf{poly}_2(\cdot)$ such that for any $\lambda \in \mathbb{N}$ and message $\mathsf{m} \in \mathcal{M}_\lambda$, we have

$$|\hat{\mathsf{ct}}| \leq \mathsf{poly}_1(\lambda), \qquad |\Delta_{1,2,\hat{\mathsf{ct}}}| \leq \mathsf{poly}_2(\lambda),$$

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 $\Delta_{1,2,\hat{\mathsf{ct}}} \leftarrow \mathsf{ReKeyGen}(\mathsf{k}_1, \mathsf{k}_2, \hat{\mathsf{ct}})$.

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 $\lambda \in \mathbb{N}$, $N = \mathsf{poly}(\lambda)$, 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* [BLMR13, EPRS17] while the security of the re-encryption algorithm $\mathsf{ReEncrypt}$ is captured by the notion of *re-encryption indistinguishability* [EPRS17].

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 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. [EPRS17] for reference in Appendix A. [1]

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 real-world scenarios, 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, unlike the encryption algorithm $\mathsf{Encrypt}$, the re-encryption algorithm $\mathsf{ReEncrypt}$ is designed to be applied multiple times on a single ciphertext and therefore, it must take into account the varying lengths of ciphertexts that are handled by the re-encryption algorithm.

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 now augment the confidentiality security definitions of Everspaugh et al. [EPRS17] 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,

[1]The definitions that we present in Appendix A are actually simpler variants of the Everspaugh et al. [EPRS17] definitions. Our improvements to their definitions, presented in this section, are orthogonal to any simplifications that we make. See the appendix for a full discussion.

positive integers $h, d \in \mathbb{N}$, an adversary $\mathcal{A}$, and a binary bit $b \in \{0,1\}$, we define the confidentiality experiment $\mathsf{Expt}^{\mathsf{conf}}_{\Pi_{\mathsf{UAE}}}(\lambda, h, d, \mathcal{A}, b)$ as follows:

$\mathsf{Expt}^{\mathsf{conf}}_{\Pi_{\mathsf{UAE}}}(\lambda, h, d, \mathcal{A}, b)$:

- **Setup phase**: At the start of the experiment, the challenger generates h uncorrupted keys $\mathsf{k}_1, \ldots, \mathsf{k}_h \leftarrow \mathsf{KeyGen}(1^\lambda)$ and d corrupted keys $\mathsf{k}_{h+1}, \ldots, \mathsf{k}_{h+d} \leftarrow \mathsf{KeyGen}(1^\lambda)$. It provides the corrupted keys $\mathsf{k}_{h+1}, \ldots, \mathsf{k}_{h+d}$ to the adversary $\mathcal{A}$.
- **Query phase**: Throughout the query phase of the experiment, the challenger maintains a look-up table T that maps *key index* and *ciphertext header* pairs to *ciphertext bodies*. This table holds the body of the encryption of a challenge ciphertext, and its re-encryptions. The adversary is allowed to make the following queries to the challenger:
  - $\mathcal{O}_{\mathsf{Encrypt}}(i, \mathsf{m})$: A query consists of an index $i \in [h]$ and a message $\mathsf{m} \in \mathcal{M}_\lambda$. The challenger returns $\mathsf{Encrypt}(\mathsf{k}_i, \mathsf{m})$ to $\mathcal{A}$.
  - $\mathcal{O}_{\mathsf{ReKeyGen}}(i, j, \hat{\mathsf{ct}})$: A query consists of indices $i, j \in [h+d]$ and a ciphertext header $\hat{\mathsf{ct}}$. If $j > h$ and $\mathsf{T}[i, \hat{\mathsf{ct}}] \neq \perp$, the challenger returns $\perp$. Otherwise, it computes $\Delta_{i,j,\hat{\mathsf{ct}}} \leftarrow \mathsf{ReKeyGen}(\mathsf{k}_i, \mathsf{k}_j, \hat{\mathsf{ct}})$ and returns $\Delta_{i,j,\hat{\mathsf{ct}}}$ to $\mathcal{A}$. If $\mathsf{T}[i, \hat{\mathsf{ct}}] \neq \perp$, then the challenger computes $\mathsf{ReEncrypt}\big(\Delta_{i,j,\hat{\mathsf{ct}}}, (\hat{\mathsf{ct}}, \mathsf{T}[i, \hat{\mathsf{ct}}])\big)$ and sets $\mathsf{T}[j, \hat{\mathsf{ct}}'] \leftarrow \mathsf{ct}'$.
  - $\mathcal{O}_{\mathsf{ReEncrypt}}\big(i, j, (\hat{\mathsf{ct}}, \mathsf{ct})\big)$: A query consists of indices $i, j \in [h+d]$ and a ciphertext $(\hat{\mathsf{ct}}, \mathsf{ct})$. The challenger computes an update token $\Delta_{i,j,\hat{\mathsf{ct}}} \leftarrow \mathsf{ReKeyGen}(\mathsf{k}_i, \mathsf{k}_j, \hat{\mathsf{ct}})$ and updated ciphertext $(\hat{\mathsf{ct}}', \mathsf{ct}') \leftarrow \mathsf{ReEncrypt}\big(\Delta_{i,j,\hat{\mathsf{ct}}}, (\hat{\mathsf{ct}}, \mathsf{ct})\big)$. If $j > h$ and $\mathsf{T}[i, \hat{\mathsf{ct}}] \neq \perp$, the challenger returns $\perp$. Otherwise, it returns $(\hat{\mathsf{ct}}', \mathsf{ct}')$ to $\mathcal{A}$. Finally, if $j \leq h$ and $\mathsf{T}[i, \hat{\mathsf{ct}}] \neq \perp$, the challenger sets $\mathsf{T}[j, \hat{\mathsf{ct}}'] \leftarrow \mathsf{ct}'$.
  - $\mathcal{O}_{\mathsf{Challenge}}\big(i, j, \mathsf{m}, (\hat{\mathsf{ct}}, \mathsf{ct})\big)$: A query consists of indices $i \in [h+d]$, $j \in [h]$, a message $\mathsf{m} \in \mathcal{M}_\lambda$, and a ciphertext $(\hat{\mathsf{ct}}, \mathsf{ct})$. The challenger computes the following values:

$$\begin{aligned}
&(\hat{\mathsf{ct}}'_0, \mathsf{ct}'_0) \leftarrow \mathsf{Encrypt}(\mathsf{k}_j, \mathsf{m}) \\
&\Delta_{i,j,\hat{\mathsf{ct}}} \leftarrow \mathsf{ReKeyGen}(\mathsf{k}_i, \mathsf{k}_j, \hat{\mathsf{ct}}) \\
&(\hat{\mathsf{ct}}'_1, \mathsf{ct}'_1) \leftarrow \mathsf{ReEncrypt}(\Delta_{i,j,\hat{\mathsf{ct}}}, (\hat{\mathsf{ct}}, \mathsf{ct}))
\end{aligned}$$

  If $|\hat{\mathsf{ct}}'_0| \neq |\hat{\mathsf{ct}}'_1|$, $|\mathsf{ct}'_0| \neq |\mathsf{ct}'_1|$, or if either of the algorithms $\mathsf{ReKeyGen}(\mathsf{k}_i, \mathsf{k}_j, \hat{\mathsf{ct}})$, $\mathsf{ReEncrypt}(\Delta_{i,j,\hat{\mathsf{ct}}}, (\hat{\mathsf{ct}}, \mathsf{ct}))$ outputs $\perp$, then the challenger returns $\perp$. Otherwise, it sets $\mathsf{T}[j, \hat{\mathsf{ct}}'_b] \leftarrow \mathsf{ct}'_b$ and returns $(\hat{\mathsf{ct}}'_b, \mathsf{ct}'_b)$ to $\mathcal{A}$.
- **Output phase**: At the end of the experiment, the adversary $\mathcal{A}$ outputs a bit $b \in \{0,1\}$, which is the output of the experiment.

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 $h, d = \mathsf{poly}(\lambda)$ and efficient adversaries $\mathcal{A}$, we have

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

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 given two ciphertexts $(\hat{\mathsf{ct}}_0, \mathsf{ct}_0)$ and $(\hat{\mathsf{ct}}_1, \mathsf{ct}_1)$ of the same length, we have

$$\mathsf{ReEncrypt}\big(\mathsf{ReKeyGen}(\mathsf{k}, \mathsf{k}', \hat{\mathsf{ct}}_0), (\hat{\mathsf{ct}}_0, \mathsf{ct}_0)\big) \approx_c (\hat{\mathsf{ct}}', \mathsf{ct}') \approx_c \mathsf{ReEncrypt}\big(\mathsf{ReKeyGen}(\mathsf{k}, \mathsf{k}', \hat{\mathsf{ct}}_1), (\hat{\mathsf{ct}}_1, \mathsf{ct}_1)\big),$$

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 it satisfies the following two properties:

- *Header compactness*: There exist functions $\mathsf{poly}_1(\cdot)$, $\mathsf{poly}_2(\cdot)$ such that for any $\lambda \in \mathbb{N}$ and message $\mathsf{m} \in \mathcal{M}_\lambda$, we have

  $$|\hat{\mathsf{ct}}| \leq \mathsf{poly}_1(\lambda), \quad |\Delta_{1,2,\hat{\mathsf{ct}}}| \leq \mathsf{poly}_2(\lambda),$$

  where $\mathsf{k}_1, \mathsf{k}_2 \leftarrow \mathsf{KeyGen}(1^\lambda)$.

- *Body compactness*: For any $\mathsf{m} \in \mathcal{M}_\lambda$ and sequence of keys $\mathsf{k}_0, \mathsf{k}_1, \ldots, \mathsf{k}_N \leftarrow \mathsf{KeyGen}(1^\lambda)$, if we set $(\hat{\mathsf{ct}}_0, \mathsf{ct}_0) \leftarrow \mathsf{Encrypt}(\mathsf{k}_0, \mathsf{m})$, and
$$(\hat{\mathsf{ct}}_i, \mathsf{ct}_i) \leftarrow \mathsf{ReEncrypt}\big(\mathsf{ReKeyGen}(\mathsf{k}_{i-1}, \mathsf{k}_i, \hat{\mathsf{ct}}_{i-1}), (\hat{\mathsf{ct}}_{i-1}, \mathsf{ct}_{i-1})\big),$$
for $i \in [N]$, we have $|\mathsf{ct}_i| = |\mathsf{ct}_j|$ for all $0 \le i, j \le N$.

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 oracle $\mathcal{O}_{\mathsf{ReEncrypt}}$. Since the re-encryption 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. For completeness, we state the full definition of update independence in Appendix C.

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. [EPRS17] (rewritten in Appendix A), 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 3.6 (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 $h, d \in \mathbb{N}$, and an adversary $\mathcal{A}$, we define the re-encryption integrity experiment $\mathsf{Expt}^{\mathsf{int}}_{\Pi_{\mathsf{UAE}}}(\lambda, h, d, \mathcal{A})$ as follows:

$\mathsf{Expt}^{\mathsf{int}}_{\Pi_{\mathsf{UAE}}}(\lambda, h, d, \mathcal{A})$:

- **Setup phase**: At the start of the experiment, the challenger generates h uncorrupted keys $\mathsf{k}_1, \ldots, \mathsf{k}_h \leftarrow \mathsf{KeyGen}(1^\lambda)$ and d corrupted keys $\mathsf{k}_{h+1}, \ldots, \mathsf{k}_{h+d} \leftarrow \mathsf{KeyGen}(1^\lambda)$. It provides the corrupted keys $\mathsf{k}_{h+1}, \ldots, \mathsf{k}_{h+d}$ to the adversary $\mathcal{A}$.
- **Query phase**: Throughout the query phase of the experiment, the challenger maintains a look-up table T that maps *key index* and *ciphertext header* pairs to *ciphertext bodies.* The adversary is allowed to make the following queries to the challenger:
  - $\mathcal{O}_{\mathsf{Encrypt}}(i, \mathsf{m})$: A query consists of an index $i \in [h+d]$ and a message $\mathsf{m} \in \mathcal{M}_\lambda$. The challenger computes $(\hat{\mathsf{ct}}, \mathsf{ct}) \leftarrow \mathsf{Encrypt}(\mathsf{k}_i, \mathsf{m})$ and sets $\mathsf{T}[i, \hat{\mathsf{ct}}] \leftarrow \mathsf{ct}$. It returns $(\hat{\mathsf{ct}}, \mathsf{ct})$ to $\mathcal{A}$.
  - $\mathcal{O}_{\mathsf{ReKeyGen}}(i, j, \hat{\mathsf{ct}})$: A query consists of indices $i, j \in [h+d]$ and a ciphertext header $\hat{\mathsf{ct}}$. If $i > h$ and $j \leq h$, the challenger returns $\perp$. Otherwise, it computes $\Delta_{i,j,\hat{\mathsf{ct}}} \leftarrow \mathsf{ReKeyGen}(\mathsf{k}_i, \mathsf{k}_j, \hat{\mathsf{ct}})$ and returns $\Delta_{i,j,\hat{\mathsf{ct}}}$ to $\mathcal{A}$. If $\mathsf{T}[i, \hat{\mathsf{ct}}] \neq \perp$, the challenger computes $(\hat{\mathsf{ct}}', \mathsf{ct}') \leftarrow \mathsf{ReEncrypt}\big(\Delta_{i,j,\hat{\mathsf{ct}}}, (\hat{\mathsf{ct}}, \mathsf{T}[i, \hat{\mathsf{ct}}])\big)$ and sets $\mathsf{T}[j, \hat{\mathsf{ct}}'] \leftarrow \mathsf{ct}'$.
  - $\mathcal{O}_{\mathsf{ReEncrypt}}\big(i, j, (\hat{\mathsf{ct}}, \mathsf{ct})\big)$: A query consists of indices $i, j \in [h+d]$ and a ciphertext $(\hat{\mathsf{ct}}, \mathsf{ct})$. The challenger computes an update token $\Delta_{i,j,\hat{\mathsf{ct}}} \leftarrow \mathsf{ReKeyGen}(\mathsf{k}_i, \mathsf{k}_j, \hat{\mathsf{ct}})$, updated ciphertext $(\hat{\mathsf{ct}}', \mathsf{ct}') \leftarrow \mathsf{ReEncrypt}\big(\Delta_{i,j,\hat{\mathsf{ct}}}, (\hat{\mathsf{ct}}, \mathsf{ct})\big)$, and returns $(\hat{\mathsf{ct}}', \mathsf{ct}')$ to $\mathcal{A}$. If $j \leq h$, it sets $\mathsf{T}[j, \hat{\mathsf{ct}}'] \leftarrow \mathsf{ct}'$.
- **Output phase**: At the end of the experiment, the adversary $\mathcal{A}$ returns an index $i \in [h]$ and a ciphertext $(\hat{\mathsf{ct}}, \mathsf{ct})$. The challenger computes $\mathsf{m} \leftarrow \mathsf{Decrypt}\big(\mathsf{k}_i, (\hat{\mathsf{ct}}, \mathsf{ct})\big)$ and checks the following conditions:
  - $\mathsf{m} = \perp$,
  - $\mathsf{T}[i, \hat{\mathsf{ct}}] = \mathsf{ct}$.

  If either of the conditions above are met, then the challenger returns 0. Otherwise, it returns 1.

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 $h, d \in \mathbb{N}$ and any efficient adversary $\mathcal{A}$, we have

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

Although our UAE construction in Section 4 can be shown to satisfy the strong notion of integrity formulated above, the construction in Section 5 that relies on almost key-homomorphic PRFs is not sufficient to satisfy the stronger notion. In Section 5, we formulate a relaxation of the notion of integrity that we call *relaxed integrity* and show that UAE in 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 Section 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 (Section 3.3, Definition C.1). 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. [EPRS17]. 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 Appendix D.

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

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

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 (Definition 2.10), an adversary cannot determine which component of a ciphertext corresponds to the original ciphertext and which component corresponds to a pad.[2]

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. 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). The re-encryptor can now apply the re-encryption on the original ciphertext and adjust the padding length accordingly. Formally, our nested UAE works as follows.

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 $\mathsf{AE.Encrypt}$ behaves as a PRF, 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 $\mathsf{k_{ae}} \leftarrow \mathsf{AE.KeyGen}(1^\lambda)$ and any message $\mathsf{m} \in \mathcal{M}_\lambda$, we have

  $$|\mathsf{k_{ae}}| = \rho, \qquad |\mathsf{ct}| \leq |\mathsf{m}| + \nu,$$

  where $\mathsf{ct} \leftarrow \mathsf{AE.Encrypt}(\mathsf{k_{ae}}, \mathsf{m})$.

- 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 $\mathcal{M} = (\mathcal{M}_\lambda)_{\lambda \in \mathbb{N}}$ as follows:

- $\mathsf{KeyGen}(1^\lambda, 1^t) \to \mathsf{k}$: On input the security parameter λ, and a bound on the number of re-encryption updates $t \in \mathbb{N}$, the key generation algorithm samples an authenticated encryption key $\hat{\mathsf{k}} \leftarrow \mathsf{AE.KeyGen}(1^\lambda)$ and sets $\mathsf{k} \leftarrow (\hat{\mathsf{k}}, t)$.

- $\mathsf{Encrypt}(\mathsf{k}, \mathsf{m}) \to (\hat{\mathsf{ct}}, \mathsf{ct})$: On input a key $\mathsf{k} = (\hat{\mathsf{k}}, t)$ and a message $\mathsf{m} \in \mathcal{M}_\lambda$, the encryption algorithm first samples a new authenticated encryption key $\mathsf{k_{ae}} \leftarrow \mathsf{AE.KeyGen}(1^\lambda)$ and a PRG seed $s \xleftarrow{\mathrm{R}} \{0,1\}^\lambda$. It then generates the ciphertext components as follows:

  – $\mathsf{ct_{payload}} \leftarrow \mathsf{AE.Encrypt}\big(\mathsf{k_{ae}}, \mathsf{m}\big)$,

[2]As discussed in Section 2.3, 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.

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

  It then sets $\mathsf{ct} \leftarrow (\mathsf{ct}_{\mathsf{payload}}, \mathsf{ct}_{\mathsf{pad}})$ and returns the ciphertext $(\hat{\mathsf{ct}}, \mathsf{ct})$.

- $\mathsf{ReKeyGen}(\mathsf{k}_1, \mathsf{k}_2, \hat{\mathsf{ct}}) \to \Delta_{1,2,\hat{\mathsf{ct}}}/\perp$: On input two keys $\mathsf{k}_1 = (\hat{\mathsf{k}}_1, t)$, $\mathsf{k}_2 = (\hat{\mathsf{k}}_2, t)$ and a ciphertext header $\hat{\mathsf{ct}}$, the re-encryption key generation algorithm first checks if $\mathsf{AE.Decrypt}(\mathsf{k}_1, \hat{\mathsf{ct}}) = \perp$. If this is the case, then it returns $\perp$. Otherwise, it proceeds as follows:

  1. It sets $(s, \ell, \mathsf{k}_{\mathsf{ae}}, \hat{\mathsf{k}}_{\mathsf{history}}) \leftarrow \mathsf{AE.Decrypt}(\hat{\mathsf{k}}_1, \hat{\mathsf{ct}})$ (output $\perp$ if decryption outputs $\perp$).
  2. It samples a new authenticated encryption key $\hat{\mathsf{k}}'_{\mathsf{history}} \leftarrow \mathsf{AE.KeyGen}(1^\lambda)$ and encrypts $\hat{\mathsf{ct}}_{\mathsf{history}} \leftarrow \mathsf{AE.Encrypt}(\hat{\mathsf{k}}'_{\mathsf{history}}, (\mathsf{k}_{\mathsf{ae}}, \hat{\mathsf{k}}_{\mathsf{history}}))$.
  3. It samples a new authenticated encryption key $\mathsf{k}'_{\mathsf{ae}} \leftarrow \mathsf{AE.KeyGen}(1^\lambda)$, a new PRG seed $s' \xleftarrow{\mathrm{R}} \{0,1\}^\lambda$, sets $\ell' \leftarrow \ell + |\hat{\mathsf{ct}}_{\mathsf{history}}|$, and encrypts $\hat{\mathsf{ct}}' \leftarrow \mathsf{AE.Encrypt}\big(\hat{\mathsf{k}}_2, (s', \ell', \mathsf{k}'_{\mathsf{ae}}, \hat{\mathsf{k}}'_{\mathsf{history}})\big)$.

  It sets the token as $\Delta_{1,2,\hat{\mathsf{ct}}} \leftarrow (\hat{\mathsf{ct}}', \hat{\mathsf{ct}}_{\mathsf{history}}, \ell, \mathsf{k}'_{\mathsf{ae}}, s')$, and returns $\Delta_{1,2,\hat{\mathsf{ct}}}$.

- $\mathsf{ReEncrypt}\big(\Delta_{1,2,\hat{\mathsf{ct}}}, (\hat{\mathsf{ct}}, \mathsf{ct})\big) \to (\hat{\mathsf{ct}}', \mathsf{ct}')/\perp$: On input an update token $\Delta_{1,2,\hat{\mathsf{ct}}}$ and a ciphertext $(\hat{\mathsf{ct}}, \mathsf{ct})$, the re-encryption algorithm first parses the update token and the ciphertext body as

  - $\Delta_{1,2,\hat{\mathsf{ct}}} = (\hat{\mathsf{ct}}', \hat{\mathsf{ct}}_{\mathsf{history}}, \ell, \mathsf{k}'_{\mathsf{ae}}, s')$,
  - $\mathsf{ct} = (\mathsf{ct}_{\mathsf{payload}}, \mathsf{ct}_{\mathsf{pad}}) \in \{0,1\}^\ell \times \{0,1\}^{|\mathsf{ct}|-\ell}$ (if $|\mathsf{ct}| < \ell$, output $\perp$).

  Then, the re-encryption algorithm encrypts $\mathsf{ct}'_{\mathsf{payload}} \leftarrow \mathsf{AE.Encrypt}\big(\mathsf{k}'_{\mathsf{ae}}, (\mathsf{ct}_{\mathsf{payload}}, \hat{\mathsf{ct}}_{\mathsf{history}})\big)$ and evaluates the PRG $\mathsf{ct}'_{\mathsf{pad}} \leftarrow G(s')$ such that $\mathsf{ct}' \leftarrow (\mathsf{ct}'_{\mathsf{payload}}, \mathsf{ct}'_{\mathsf{pad}}) \in \{0,1\}^{|\mathsf{ct}|}$ (if $|\mathsf{ct}'_{\mathsf{payload}}| > |\mathsf{ct}|$, return $\perp$). Finally, it returns $(\hat{\mathsf{ct}}', \mathsf{ct}')$.

- $\mathsf{Decrypt}\big(\mathsf{k}, (\hat{\mathsf{ct}}, \mathsf{ct})\big) \to \mathsf{m}/\perp$: on input a key $\mathsf{k} = (\hat{\mathsf{k}}, t)$ and a ciphertext $(\hat{\mathsf{ct}}, \mathsf{ct})$, the decryption algorithm proceeds as follows:

  1. It sets $(s, \ell, \mathsf{k}'_{\mathsf{ae}}, \hat{\mathsf{k}}'_{\mathsf{history}}) \leftarrow \mathsf{AE.Decrypt}(\hat{\mathsf{k}}, \hat{\mathsf{ct}})$ and outputs $\perp$ if decryption outputs $\perp$.
  2. It parses $\mathsf{ct} = (\mathsf{ct}_{\mathsf{payload}}, \mathsf{ct}_{\mathsf{pad}}) \in \{0,1\}^\ell \times \{0,1\}^{|\mathsf{ct}|-\ell}$ (if $|\mathsf{ct}| < \ell$, output $\perp$)
  3. It evaluates $\mathsf{ct}'_{\mathsf{pad}} \leftarrow G(s)$, $|\mathsf{ct}'_{\mathsf{pad}}| = |\mathsf{ct}_{\mathsf{pad}}|$ and outputs $\perp$ if $\mathsf{ct}'_{\mathsf{pad}} \neq \mathsf{ct}_{\mathsf{pad}}$.
  4. It decrypts $(\mathsf{ct}', \hat{\mathsf{ct}}'_{\mathsf{history}}) \leftarrow \mathsf{AE.Decrypt}(\mathsf{k}'_{\mathsf{ae}}, \mathsf{ct}_{\mathsf{payload}})$ and outputs $\perp$ if decryption outputs $\perp$.
  5. It relabels $\mathsf{k}'_{\mathsf{ae}}$ as k_{ae}, $\hat{\mathsf{k}}'_{\mathsf{history}}$ as $\hat{\mathsf{k}}_{\mathsf{history}}$, ct' as ct, and $\hat{\mathsf{ct}}'_{\mathsf{history}}$ as $\hat{\mathsf{ct}}_{\mathsf{history}}$.
  6. It decrypts $(\mathsf{k}'_{\mathsf{ae}}, \hat{\mathsf{k}}'_{\mathsf{history}}) \leftarrow \mathsf{AE.Decrypt}(\hat{\mathsf{k}}_{\mathsf{history}}, \hat{\mathsf{ct}}_{\mathsf{history}})$, and outputs $\perp$ if decryption outputs $\perp$.
  7. It decrypts $(\mathsf{ct}', \hat{\mathsf{ct}}'_{\mathsf{history}}) \leftarrow \mathsf{AE.Decrypt}(\mathsf{k}_{\mathsf{ae}}, \mathsf{ct})$, and outputs $\perp$ if decryption outputs $\perp$.
  8. If $\hat{\mathsf{k}}'_{\mathsf{history}} \neq \perp$, it returns to Step 5 above. Otherwise it proceeds.
  9. It sets $\mathsf{m} \leftarrow \mathsf{AE.Decrypt}(\mathsf{k}'_{\mathsf{ae}}, \mathsf{ct}')$ and outputs m.

We formally state the compactness, correctness, and security properties of Construction 4.2 in the following theorem. We provide the formal proof in Appendix E.

Theorem 4.3. *Suppose 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.*

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:

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

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

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

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 construct an updatable authenticated encryption scheme from almost key-homomorphic PRFs (Definition 2.5). The construction is heavily based on the construction of Everspaugh et al. [EPRS17] that works over perfectly key-homomorphic PRFs. The main difference between their construction and our construction is the use of a suitable encoding scheme for the messages. In our construction, we use an encoding scheme to encode the messages before applying the encryption to correct any small errors that are incurred by the use of almost key-homomorphic PRFs. We provide the syntax for an encoding scheme that we use in Section 5.1. Due to the use of an encoding scheme, our construction can be viewed as supporting only a bounded number of updates. However, 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. Moreover, due to the use of an encoding scheme, our construction cannot satisfy the full integrity condition of Definition 3.6. In Section 5.2, we present an integrity notion that we call *relaxed integrity*. Finally, in Section 5.3, we present our UAE scheme based on almost key-homomorphic PRFs.

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 Section 5.3, 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 Section 5.3, we use the syntax of a general encoding scheme that is described in Fact 5.1 below. In Section 7, we test the performance of our construction in Section 5.3 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:

- $\mathsf{Encode}(\mathsf{m}) \to (\mathsf{m}_1, \ldots, \mathsf{m}_\ell)$: 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 $\ell \in \mathbb{N}$.
- $\mathsf{Decode}(\mathsf{m}_1, \ldots, \mathsf{m}_\ell) \to \mathsf{m}$: 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 $\mathbf{e}_1, \ldots, \mathbf{e}_\ell \in [\gamma]^n$, if we set $(\mathsf{m}_1, \ldots, \mathsf{m}_\ell) \leftarrow \mathsf{Encode}(\mathsf{m})$, we have

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

5.2 Relaxing Integrity

In this section, we formulate a relaxation of the notion of integrity in Definition 3.6 that we call *relaxed integrity.* The relaxed integrity security experiment relaxes Definition 3.6 (integrity) in two aspects. First, in the relaxed integrity security experiment, an adversary is provided access to a re-encryption oracle $\mathcal{O}_{\mathsf{ReEncrypt}}$ as in Definition 3.6, but we require that any of an adversary's queries to the oracle are well-formed ciphertexts that do not decrypt to "$\perp$". As we discuss in Remark 5.4, this relaxation of Definition 3.6 is necessary as there are attacks on Construction 5.2 that violate Definition 3.6.

The second relaxation that we make on Definition 3.6 is the adversary's winning condition in the experiment. Since we use *almost* key-homomorphic PRFs, any re-encryption of the ciphertexts evidently incurs a small error that affects the low-ordered bits of the ciphertext. Therefore, to achieve perfect 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. However, this also forces the construction to violate traditional ciphertext integrity as an adversary can forge new ciphertexts by adding noise to the low-ordered bits of the ciphertext. Therefore, we define the notion of *relaxed integrity* that is parameterized by a positive integer $\gamma \in \mathbb{N}$. An adversary wins in the relaxed integrity experiment only if it produces a valid ciphertext that differs from any of the ciphertexts that it is given on the high ordered bits of the ciphertext or decrypts to a plaintext that differs in any way from a plaintext it has given to the challenger. Variants of this relaxation of ciphertext integrity have been defined in a number of previous works (e.g. [Kra01, CKN03]) and are sufficient for most practical applications. Since the definiton of relaxed integrity is otherwise very similar to Definition 3.6, we present the formal statement of the definition in Appendix C.

5.3 Construction

Our construction (based on that of Everspaugh et al. [EPRS17]) encrypts messages using almost key-homomorphic PRFs in counter mode. Since almost KH-PRFs work over groups (i.e. vectors in $\mathbb{Z}_q^n$), we use an encoding scheme to map messages to group elements. The almost KH-PRF key that is used to encrypt the message via counter mode is encrypted inside the ciphertext header such that a user can recover the key from the header. The ciphertext header also stores the hash of the message for the decryption algorithm to check ciphertext integrity.

Construction 5.2 (UAE from Key-Homomorphic PRFs)**.** Let n, q, γ, and β be positive integers such that $\gamma = \lambda^{\omega(1)}$, $q > \gamma/4$, and $n, \beta = \mathsf{poly}(\lambda)$. Our construction uses the following building blocks:

- 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 B-almost key-homomorphic PRF $F : \mathcal{K}_{\mathsf{PRF}} \times \{0,1\}^* \rightarrow \mathbb{Z}_q^n$ where $(\mathcal{K}_{\mathsf{PRF}}, +)$ and $(\mathbb{Z}_q^n, +)$ form groups.
- A collision resistant hash family $\mathcal{H} = \{H : \mathcal{M}_\lambda \rightarrow \{0,1\}^\lambda\}$. To simplify the description of the construction, we assume that a description of a concrete hash function $H \xleftarrow{\text{R}} \mathcal{H}$ is included in each algorithms below 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.

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

- $\mathsf{KeyGen}(1^\lambda) \to \mathsf{k}$: On input the security parameter λ, the key generation algorithm samples an authenticated encryption key $\mathsf{k}_{\mathsf{ae}} \leftarrow \mathsf{AE.KeyGen}(1^\lambda)$ and sets $\mathsf{k} \leftarrow \mathsf{k}_{\mathsf{ae}}$.
- $\mathsf{Encrypt}(\mathsf{k}, \mathsf{m}) \to (\hat{\mathsf{ct}}, \mathsf{ct})$: On input a key $\mathsf{k} = \mathsf{k}_{\mathsf{ae}}$ and a message $\mathsf{m} \in \mathcal{M}_\lambda$, the encryption algorithm encodes the message m to derive a set of vectors $(\mathsf{m}_1, \dots, \mathsf{m}_\ell) \leftarrow \mathsf{Encode}(\mathsf{m})$ for some $\ell \in \mathbb{N}$. It then samples a PRF key $\mathsf{k}_{\mathsf{prf}} \xleftarrow{\mathrm{R}} \mathcal{K}_{\mathsf{PRF}}$, computes the hash $\mathsf{h} \leftarrow H(\mathsf{m})$, and generates the following ciphertext
$$\hat{\mathsf{ct}} \leftarrow \mathsf{AE.Encrypt}\big(\mathsf{k}_{\mathsf{ae}}, (\mathsf{k}_{\mathsf{prf}}, \mathsf{h})\big).$$
It encrypts each of the message vectors
$$\mathsf{ct}_i \leftarrow \mathsf{m}_i + F(\mathsf{k}_{\mathsf{prf}}, i)$$
for $i = 1, \dots, \ell$, and sets $\mathsf{ct} = (\mathsf{ct}_1, \dots, \mathsf{ct}_\ell)$. It returns $(\hat{\mathsf{ct}}, \mathsf{ct})$.
- $\mathsf{ReKeyGen}(\mathsf{k}_1, \mathsf{k}_2, \hat{\mathsf{ct}}) \to \Delta_{1,2,\hat{\mathsf{ct}}}/\perp$: On input two keys $\mathsf{k}_1, \mathsf{k}_2$ and a ciphertext header $\hat{\mathsf{ct}}$, the re-encryption key generation algorithm first decrypts the header $\mu \leftarrow \mathsf{AE.Decrypt}(\mathsf{k}_1, \hat{\mathsf{ct}})$. If $\mu = \perp$, then it returns $\perp$. Otherwise, it proceeds as follows:
  1. It parses the decrypted header $\mu = (\mathsf{k}_{\mathsf{prf}}, \mathsf{h})$.
  2. It samples a new PRF key $\mathsf{k}'_{\mathsf{prf}} \xleftarrow{\mathrm{R}} \mathcal{K}_{\mathsf{PRF}}$ and defines the PRF update key $\mathsf{k}^{\mathsf{up}}_{\mathsf{prf}} \leftarrow \mathsf{k}'_{\mathsf{prf}} - \mathsf{k}_{\mathsf{prf}}$.

  Finally, it sets $\hat{\mathsf{ct}}' \leftarrow \mathsf{AE.Encrypt}\big(\mathsf{k}_2, (\mathsf{k}'_{\mathsf{prf}}, \mathsf{h})\big)$ and returns $\Delta_{1,2,\hat{\mathsf{ct}}} \leftarrow (\hat{\mathsf{ct}}', \mathsf{k}^{\mathsf{up}}_{\mathsf{prf}})$.
- $\mathsf{ReEncrypt}\big(\Delta_{1,2,\hat{\mathsf{ct}}}, (\hat{\mathsf{ct}}, \mathsf{ct})\big) \to (\hat{\mathsf{ct}}', \mathsf{ct}')/\perp$: On input an update token $\Delta_{1,2,\hat{\mathsf{ct}}}$ and a ciphertext $(\hat{\mathsf{ct}}, \mathsf{ct})$, the re-encryption algorithm parses the update token as $\Delta_{1,2,\hat{\mathsf{ct}}} = (\hat{\mathsf{ct}}', \mathsf{k}^{\mathsf{up}}_{\mathsf{prf}})$ and the ciphertext body as $\mathsf{ct} = (\mathsf{ct}_1, \dots, \mathsf{ct}_\ell)$. Then, it updates each block of the ciphertext
$$\mathsf{ct}'_i \leftarrow \mathsf{ct}_i + F(\mathsf{k}^{\mathsf{up}}_{\mathsf{prf}}, i)$$
for $i = 1, \dots, \ell$. It defines the new ciphertext body $\mathsf{ct}' \leftarrow (\mathsf{ct}'_1, \dots, \mathsf{ct}'_\ell)$ and returns $(\hat{\mathsf{ct}}', \mathsf{ct}')$ as the updated ciphertext.
- $\mathsf{Decrypt}\big(\mathsf{k}, (\hat{\mathsf{ct}}, \mathsf{ct})\big) \to \mathsf{m}/\perp$: On input a key k and a ciphertext $(\hat{\mathsf{ct}}, \mathsf{ct})$, the decryption algorithm first decrypts the ciphertext header $\mu \leftarrow \mathsf{AE.Decrypt}(\mathsf{k}, \hat{\mathsf{ct}})$. If $\mu = \perp$, then it returns $\perp$. Otherwise, it parses the decrypted header $\mu = (\mathsf{k}_{\mathsf{prf}}, \mathsf{h})$, the ciphertext body $\mathsf{ct} = (\mathsf{ct}_1, \dots, \mathsf{ct}_\ell)$, and then decrypts the messages
$$\mathsf{m}_i \leftarrow \mathsf{ct}_i - F(\mathsf{k}_{\mathsf{prf}}, i)$$
for $i = 1, \dots, \ell$. Let $\mathsf{m}' \leftarrow \mathsf{Decode}(\mathsf{m}_1, \dots, \mathsf{m}_\ell)$. If $H(\mathsf{m}') = \mathsf{h}$, then it returns m'. Otherwise, it returns $\perp$.

We formally state the compactness, correctness, and security properties of Construction 5.2 in the following theorem. We provide the formal proof in Appendix F.

Theorem 5.3. *Let Π_{UAE} be the updatable authenticated encryption scheme in Construction 5.2. If Π_{AE} satisfies correctness, $\varepsilon^{\mathsf{conf}}_{\mathsf{ae}}$-confidentiality and $\varepsilon^{\mathsf{int}}_{\mathsf{ae}}$-integrity, $F : \mathcal{K}_{\mathsf{PRF}} \times \{0,1\}^* \to \mathcal{Y}$ satisfies $\varepsilon_{\mathsf{prf}}$-security, and $H : \mathcal{M}_\lambda \to \{0,1\}^\lambda$ 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:
$$\Big| \Pr\big[\mathsf{Expt}^{\mathsf{conf}}_{\Pi_{\mathsf{UAE}}}(\lambda, h, d, \mathcal{A}, 0) = 1\big] - \Pr\big[\mathsf{Expt}^{\mathsf{conf}}_{\Pi_{\mathsf{UAE}}}(\lambda, h, d, \mathcal{A}, 1) = 1\big] \Big| \leq 2h \cdot \varepsilon^{\mathsf{conf}}_{\mathsf{ae}}(\lambda) + 2h \cdot \varepsilon^{\mathsf{int}}_{\mathsf{ae}}(\lambda) + 2Q \cdot \varepsilon_{\mathsf{prf}}(\lambda)$$

For integrity, we have the following bound for all $h, d = \mathsf{poly}(\lambda)$ and efficient adversaries $\mathcal{A}$:
$$\Pr\big[\mathsf{Expt}^{\mathsf{relaxed\text{-}int}}_{\Pi_{\mathsf{UAE}}}(\lambda, h, d, \gamma, \mathcal{A}) = 1\big] \leq h \cdot \varepsilon^{\mathsf{int}}_{\mathsf{ae}}(\lambda) + \varepsilon_{\mathsf{cr}}(\lambda)$$

Remark 5.4 (Regarding relaxed integrity)**.** Construction 5.2 satisfies relaxed integrity as opposed to the full integrity of Definition 3.6. There exists a simple adversary that breaks the integrity experiment when it is provided arbitrary access to the re-encryption oracle $\mathcal{O}_{\mathsf{ReEncrypt}}$. The adversary works as follows:

1. It first uses an 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 encodes 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. It subtracts an arbitrary vector m' from the ciphertext body $\tilde{\mathsf{ct}} \leftarrow \mathsf{ct} - \mathsf{m}'$.

3. It 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. It 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 $(\hat{\mathsf{ct}}', \tilde{\mathsf{ct}}' + \mathsf{m})$ that an adversary returns as a forgery is a valid ciphertext, but it is not an output of any of the oracles that the adversary is provided with. This attack is ruled out in the relaxed integrity experiment. Namely, in the relaxed integrity experiment, the re-encryption oracle $\mathcal{O}_{\mathsf{ReEncrypt}}$ outputs a re-encrypted ciphertext only when the input ciphertexts are well-formed.

6 Key-Homomorphic PRFs from Lattices

In this section, we construct an almost key-homomorphic PRF from the Learning with Errors (LWE) assumption [Reg05]. There are a number of standard variants of the LWE assumption in the literature that give rise to efficient PRF constructions. In this work, we work with the hardness of the *ring* variant of the LWE problem where LWE matrices and vectors are represented as elements over a polynomial ring. This variant of the assumption is often called the Ring Learning with Errors (RLWE) assumption [LPR10].

6.1 Ring Learning with Errors.

The Ring Learning with Errors (RLWE) problem works over a polynomial ring of the form $\mathcal{R} = \mathbb{Z}[X]/(\phi)$ and $\mathcal{R}_q = \mathcal{R}/q\mathcal{R}$ for some polynomial $\phi \in \mathbb{Z}[X]$. The degree of the polynomial ϕ denoted n works as a security parameter for the problem. For simplicity in this work, we restrict to power-of-two positive integers n and cyclotomic polynomials $\phi = X^n + 1 \in \mathbb{Z}[X]$. A ring element $b \in \mathcal{R}$ $(\mathcal{R}_q)$ can be represented as a vector of its polynomial coefficients in $\mathbb{Z}$ $(\mathbb{Z}_q)$. Then, for a ring element $b \in \mathcal{R}$ with vector representation $b = (b_1, \ldots, b_n)$, we define its *norm* $\|b\|$ as the infinity norm of its vector representation $\max_{i \in [n]} |b_i|$. For a positive integer $B \in \mathbb{N}$, we let $\mathcal{E}_B \subseteq \mathcal{R}$ to denote the set of all elements in $\mathcal{R}$ with norm at most B.

Definition 6.1 (Ring Learning with Errors [Reg05, SSTX09, LPR10])**.** Let n, q, B be positive integers, let $\mathcal{R} = \mathbb{Z}[X]/(\phi)$ be a polynomial ring for some $\phi \in \mathbb{Z}[X]$, and $\mathcal{R}_q = \mathcal{R}/q\mathcal{R}$. Then, for an error distribution χ over $\mathcal{E}_B \subseteq \mathcal{R}$, the (decisional) *Ring Learning with Errors* (RLWE) problem $\mathsf{RLWE}_{n,q,\chi}$ asks an adversary to distinguish the following two distributions:

- $\mathcal{O}_s^{\mathsf{Real}}$: On its invocation, the oracle samples a random ring element $a \xleftarrow{R} \mathcal{R}_q$, noise element $e \leftarrow \chi$, and returns $(a, a \cdot s + e) \in \mathcal{R}_q \times \mathcal{R}_q$.
- $\mathcal{O}^{\mathsf{Ideal}}$: On its invocation, the oracle samples random ring elements $a, u \xleftarrow{R} \mathcal{R}_q$ and returns $(a, u) \in \mathcal{R}_q \times \mathcal{R}_q$.

More precisely, we say that $\mathsf{RLWE}_{n,q,\chi}$ is $\varepsilon_{\mathsf{RLWE}}$-secure if for all efficient adversaries $\mathcal{A}$, we have

$$\left| \Pr\left[\mathcal{A}^{\mathcal{O}_s^{\mathsf{Real}}}(1^\lambda) = 1\right] - \Pr\left[\mathcal{A}^{\mathcal{O}_s^{\mathsf{Ideal}}}(1^\lambda) = 1\right] \right| = \varepsilon_{\mathsf{RLWE}}(\lambda),$$

where $s \xleftarrow{\text{R}} \mathcal{R}_q$.

For certain choices of the parameters n, q, B and error distribution χ, the Ring Learning with Errors problem is hard assuming that certain worst-case lattice problems such as approx-SVP on n-dimensional ideal lattices are hard to approximate within $\mathsf{poly}(n \cdot q/B)$ by a quantum algorithm. [Reg05, Pei09, ACPS09, LPR10, MM11, MP12, LPR13, BLP$^+$13, LS15].

6.2 Almost Key-Homomorphic PRFs from RLWE

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

Construction 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 $\mathcal{E}_B \subseteq \mathcal{R}$. We let $\mathsf{Samp}_\chi : \{0,1\}^r \to \mathcal{E}_B$ 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:

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

We define our pseudorandom function $F : \mathcal{R}_q \times \{0,1\}^\ell \to \mathcal{R}_q$ 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 Appendix G.

Theorem 6.3. *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 $\mathsf{RLWE}_{n,q,\chi}$ (Definition 6.1) is $\varepsilon_{\mathsf{RLWE}}$-secure, the pseudorandom function in Construction 6.2 is a $\varepsilon_{\mathsf{prf}}$-secure $2B$-almost key-homomorphic PRF (Definition 2.5) with key space and range $(\mathcal{R}_q, +)$ such that $\varepsilon_{\mathsf{prf}}(\lambda) = \varepsilon_{\mathsf{RLWE}}(\lambda)$.*

6.3 Implementation Considerations

In Section 7, we implement our updatable authenticated encryption schemes in Constructions 4.2 and 5.2. For the scheme in Construction 5.2, we instantiate the (almost) key-homomorphic PRF with the lattice-based PRF in Construction 6.2. For the implementation of Construction 6.2, there are a number of design decisions that must be taken into account. We now discuss a subset of these issues and provide the actual evaluation numbers in Section 7.

Modulus. An important parameter to consider when implementing the key-homomorphic PRF in Construction 6.2 is the modulus q that defines the ring $\mathcal{R}_q$. Naturally, the smaller the modulus q is, the faster the ring operations become and therefore, it is preferable to set q to be as small as possible to optimize the *speed* of the PRF evaluation. At the same time, since Construction 6.2 is an almost key-homomorphic PRF, it is beneficial to set q to be as big as possible to minimize the padding that must be added on to the messages before their encryption, thereby minimizing the *space* required to store these ciphertexts. In Section 7, to

RLWE Parameters

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

Figure 1: RLWE parameters for each value of $|q|$ used in our evaluation.

test for the optimal trade-offs between speed and space, we test Construction 6.2 with a number of different moduli to evaluate their performance.

Number-Theoretic Transform. An evaluation of the PRF in Construction 6.2 consists of a polynomial $y = a \cdot s + e$ where $s \in \mathcal{R}_q$ is the PRF key and $a \in \mathcal{R}_q$, $e \in \mathcal{R}$ are polynomials that are derived from the input to the PRF $x \in \{0,1\}^{\ell}$. Then to multiply two polynomials a and s, it is natural to use fast multiplication via the number-theoretic transform (NTT). A naïve way of implementing the multiplication via NTT is as follows:

1. Convert the polynomials a and s from their "coefficient" representations into their "input-output" (NTT) representations,
2. Multiply the two polynomials in the NTT representation,
3. Convert the result back into the coefficient representation.

For the PRF implementation, one can save on step 1 above. Since the polynomial a is derived from the hash of the PRF input $a \leftarrow H(x)$, one can directly interpret the hash $H(x)$ as a representation of a polynomial already in the NTT representation. Since s is re-used for multiple PRF evaluations, its NTT representation can also be pre-processed once at setup. This allows the PRF evaluation to require only a single NTT conversion as opposed three.

Noise distribution and message encodings. Finally, when instantiating Construction 5.2 with Construction 6.2, an important factor to consider is the noise distribution χ and the message encoding scheme. For the evaluations in Section 7, we chose to use the uniform distribution over a bounded space. We set the norm bounds for the uniform distribution based on the best known attacks on the RLWE problem.

For the message encodings, we chose to trivially pad the messages with additional insignificant bits to cope with noise growth during key-rotation. It is possible to use more sophisticated error correcting codes to achieve better message-to-ciphertext ratios. We considered a number of options such as BCH codes and LDPC codes [Gal62]; however, the actual savings in the ciphertext size appeared to be minimal compared to other optimizations.

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. [EPRS17] 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. [EPRS17] UAE at the cost of an increase in ciphertext size (albeit our ciphertext sizes are still considerably smaller than those of ciphertext-independent schemes [LT18, KLR19, BDGJ19]).

We implemented our constructions in C and evaluated their performance on an 8-core Ubuntu virtual machine with 4GB of RAM running on a Windows 10 computer with 64GB and a 12-core AMD 1920x processor @3.8GHz. 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 [ADPS16, Sei18]. All numbers reported are averages taken over at least 1,000 trials. Our choice of lattice parameters for each modulus size $|q|$ is based

Encrypt and ReEncrypt Throughput (MB/sec)

	KH-PRF UAE					ReCrypt	Nested
	$\lvert q\rvert = 28$	$\lvert q\rvert = 28$ (AVX)	$\lvert q\rvert = 60$	$\lvert q\rvert = 120$	$\lvert q\rvert = 128$	[EPRS17]	$t = 128$
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

Figure 2: Comparing the throughput of our KH-PRF, ReCrypt, and our nested construction configured to allow 128 re-encryptions, for messages of length 4KB, 32KB, and 100KB. 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.

KeyGen and ReKeyGen Time (μsecs)

	KH-PRF UAE $\lvert q\rvert = 60$	ReCrypt [EPRS17]	Nested $t = 128$
32KB Messages			
KeyGen	3.0	1.0	2.6
ReKeyGen	72.7	308.8	10.1

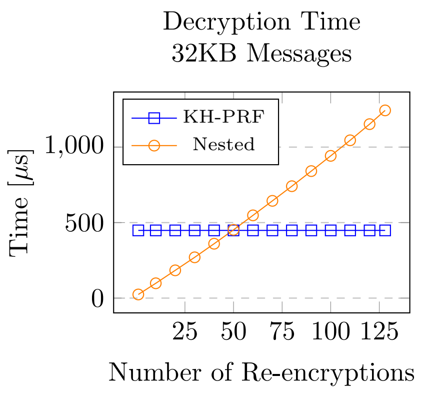
Figure 3: KeyGen and ReKeyGen costs. The main differences in performance are caused by whether the ReKeyGen algorithm needs to sample only AES keys or also KH-PRF keys, the type of KH-PRF used, and the number of ciphertexts contained in the update token.

on the best known attacks on RLWE [APS15], as shown in Figure 1. Our implementation is open source and available at [imp].

Encryption and Re-encryption Costs. Figure 2 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 [EPRS17] 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 4KB to 100KB. The nested AES construction, in turn, has $13 - 47\times$ the encryption throuhgput of our KHPRF-based construction. The nested AES scheme approaches the machine's peak AES throughput of 4.45GB/sec as the message size increases.

We find that for small messages (4KB), 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., 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 best performance for the most message sizes.

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 Figure 3. 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



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 [EPRS17]	3%

Figure 4: 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.

Figure 5: 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.

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.

Decryption Costs. Figure 4 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 occured 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 5 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 32KB message, a ReCrypt ciphertext takes 33KB and a ciphertext under our KH-PRF scheme takes 43.6KB. 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 38.3KB 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 [Bar16] and PCI DSS [PCI18] 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.

- For ciphertexts that will be frequently re-keyed, one should use our KHPRF-based construction where decryption time and ciphertext size do not depend on the number of re-encryptions.
- ReCrypt [EPRS17] can be useful in settings with frequent re-encryptions where ciphertext size is valued much more than encryption and decryption time.

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.

References

[ACPS09] Benny Applebaum, David Cash, Chris Peikert, and Amit Sahai. Fast cryptographic primitives and circular-secure encryption based on hard learning problems. In *CRYPTO*, 2009.

[ADPS16] Erdem Alkim, Léo Ducas, Thomas Pöppelmann, and Peter Schwabe. Post-quantum key exchange - A new hope. In *USENIX Security*, 2016.

[APS15] Martin R Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of learning with errors. *Journal of Mathematical Cryptology*, 9(3):169–203, 2015.

[Bar16] Elaine Barker. Nist special publication 800-57 part 1 revision 4: Recommendation for key management, 2016.

[BDGJ19] Colin Boyd, Gareth T. Davies, Kistian Gjosteen, and Yao Jiang. RISE and SHINE: Fast and secure updatable encryption. *IACR Cryptology ePrint Archive*, 2019:1457, 2019.

[Ber06] Daniel J. Bernstein. Curve25519: New diffie-hellman speed records. In *PKC*, 2006.

[BLMR13] Dan Boneh, Kevin Lewi, Hart William Montgomery, and Ananth Raghunathan. Key homomorphic prfs and their applications. In *CRYPTO*, 2013.

[BLP$^+$13] Zvika Brakerski, Adeline Langlois, Chris Peikert, Oded Regev, and Damien Stehlé. Classical hardness of learning with errors. In *STOC*, 2013.

[BN00] Mihir Bellare and Chanathip Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In *ASIACRYPT*, 2000.

[BP14] Abhishek Banerjee and Chris Peikert. New and improved key-homomorphic pseudorandom functions. In *CRYPTO*, 2014.

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *CCS*, 1993.

[BV15] Zvika Brakerski and Vinod Vaikuntanathan. Constrained key-homomorphic prfs from standard lattice assumptions. In *TCC*, 2015.

[CKN03] Ran Canetti, Hugo Krawczyk, and Jesper B Nielsen. Relaxing chosen-ciphertext security. In *CRYPTO*, 2003.

[EPRS17] Adam Everspaugh, Kenneth G. Paterson, Thomas Ristenpart, and Samuel Scott. Key rotation for authenticated encryption. In *CRYPTO*, 2017.

[FS86] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO*, 1986.

[Gal62] Robert Gallager. Low-density parity-check codes. *IRE Transactions on information theory*, 8(1):21–28, 1962.

[GGM84] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. On the cryptographic applications of random functions. In *CRYPTO*, 1984.

[Goo] Google. Key rotation. https://cloud.google.com/kms/docs/key-rotation.

[imp] Anonymized code repository. https://anonymous.4open.science/r/777b54a8-1946-44ef-b0db-cf27713aa461/.

[Kim20] Sam Kim. Key-homomorphic pseudorandom functions from lwe with small modulus. In *EUROCRYPT*, 2020.

[KLR19] Michael Klooß, Anja Lehmann, and Andy Rupp. (R)CCA secure updatable encryption with integrity protection. In *EUROCRYPT*, 2019.

[Kra01] Hugo Krawczyk. The order of encryption and authentication for protecting communications (or: How secure is ssl?). In *CRYPTO*, 2001.

[LPR10] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In *EUROCRYPT*, 2010.

[LPR13] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. A toolkit for ring-lwe cryptography. In *EUROCRYPT*, 2013.

[LS15] Adeline Langlois and Damien Stehlé. Worst-case to average-case reductions for module lattices. *Designs, Codes and Cryptography*, 75(3):565–599, 2015.

[LT18] Anja Lehmann and Björn Tackmann. Updatable encryption with post-compromise security. In *EUROCRYPT*, 2018.

[MM11] Daniele Micciancio and Petros Mol. Pseudorandom knapsacks and the sample complexity of LWE search-to-decision reductions. In *CRYPTO*, 2011.

[MP12] Daniele Micciancio and Chris Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In *EUROCRYPT*, 2012.

[NPR99] Moni Naor, Benny Pinkas, and Omer Reingold. Distributed pseudo-random functions and kdcs. In *EUROCRYPT*, 1999.

[PCI18] PCI Security Standards Council. Payment card industry data security standard, 2018.

[Pei09] Chris Peikert. Public-key cryptosystems from the worst-case shortest vector problem. In *STOC*, 2009.

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In *STOC*, 2005.

[Sei18] Gregor Seiler. Faster AVX2 optimized NTT multiplication for ring-lwe lattice cryptography. *IACR Cryptology ePrint Archive*, 2018:39, 2018.

[SSTX09] Damien Stehlé, Ron Steinfeld, Keisuke Tanaka, and Keita Xagawa. Efficient public key encryption based on ideal lattices. In *ASIACRYPT*, 2009.

A Everspaugh et al. [EPRS17] Security Definitions

This section states the message confidentiality, re-encryption indistinguishability, and ciphertext integrity definitions of Everspaugh et al. [EPRS17].

The confidentiality security definitions stated here are in fact slight modifications of the definitions of Everspaugh et al. [EPRS17]. The difference between the two sets of definitions is in the way the challenger handles trivial wins. In both sets of definitions (the originals and the ones presented here), an adversary is prohibited from ever receiving a re-encryption of the challenge ciphertext under any of the dishonest keys. Allowing the adversary to do so makes the security game vacuous as the adversary can use these keys to decrypt the challenge ciphertext and always win the game. In Definitions A.1 and A.2, we define the re-encryption oracle $\mathcal{O}_{\mathsf{ReEncrypt}}$ to take in a pair of indices i, j and a ciphertext $(\hat{\mathsf{ct}}, \mathsf{ct})$, and output "$\perp$" if both the target index j is a compromised key (i.e. $j > h$) and $(\hat{\mathsf{ct}}, \mathsf{ct})$ is either a challenge ciphertext or one of its derivations via re-encryption. In the definitions of Everspaugh et al. [EPRS17], if an adversary submits a re-encryption oracle query that satisfies these conditions, the challenger still returns the ciphertext header "$\hat{\mathsf{ct}}'$" (but not the ciphertext body ct') of the re-encrypted ciphertext instead of outputting $\perp$.

The original [EPRS17] definitions provide theoretically stronger security as an adversary in the two security experiments receives more power from the re-encryption oracle. However, it is difficult to deduce what additional security property the stronger variant of the definition captures. In the stronger confidentiality experiments, an adversary has the power to essentially decrypt the ciphertext headers from a challenge ciphertext. Although ciphertext headers are necessarily transmitted during key updates and therefore more likely to be exposed to attackers, these headers are still *ciphertexts* that are designed precisely to provide confidentiality when they are exposed over public channels. Therefore, assuming that an adversary may decrypt a challenge ciphertext header without compromising an honest key in the process is unrealistic and appears to add unnecessary complication to the definition. For this work, we choose to present the simpler and clear variant of the original set of definitions. The limitations of the definitions that we discuss in Section 3 are more fundamental to the way the existing confidentiality experiments are designed and are orthogonal to the issue of handling trivial wins.

We do note that Everspaugh et al. [EPRS17] showed an elegant way of upgrading an updatable encryption scheme that satisfies the simpler variant of the confidentiality security definitions (Definitions A.1 and A.2) to the stronger variant of [EPRS17] via a secret-sharing technique. This transformation incurs minimal efficiency overhead and can also be applied to all of the constructions in this work (Constructions D.1, 4.2 and 5.2). We present our constructions without this transformation purely for simplicity in the notations and proofs.

Definition A.1 (Message Confidentiality [BLMR13, EPRS17]). Let $\Pi_{\mathsf{UAE}} = (\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{ReKeyGen}, \mathsf{ReEncrypt}, \mathsf{Decrypt})$ be an updatable authenticated encryption scheme for a message space $\mathcal{M} = (\mathcal{M})_{\lambda \in \mathbb{N}}$. 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 message confidentiality experiment $\mathsf{Expt}^{\mathsf{msg\text{-}conf}}_{\Pi_{\mathsf{UAE}}}(\lambda, h, d, \mathcal{A}, b)$ as follows:

$\mathsf{Expt}^{\mathsf{msg\text{-}conf}}_{\Pi_{\mathsf{UAE}}}(\lambda, h, d, \mathcal{A}, b)$:

- **Setup phase**: At the start of the experiment, the challenger generates h uncorrupted keys $\mathsf{k}_1, \ldots, \mathsf{k}_h \leftarrow \mathsf{KeyGen}(1^\lambda)$ and d corrupted keys $\mathsf{k}_{h+1}, \ldots, \mathsf{k}_{h+d} \leftarrow \mathsf{KeyGen}(1^\lambda)$. It provides the corrupted keys $\mathsf{k}_{h+1}, \ldots, \mathsf{k}_{h+d}$ to the adversary $\mathcal{A}$.
- **Query phase**: Throughout the query phase of the experiment, the challenger maintains a look-up table T that maps *key index* and *ciphertext header* pairs to *ciphertext bodies*. This table holds the body of the encryption of a challenge message, and its re-encryptions. The adversary is allowed to make the following queries to the challenger:
  - $\mathcal{O}_{\mathsf{Encrypt}}(i, \mathsf{m})$: A query consists of an index $i \in [h]$ and a message $\mathsf{m} \in \mathcal{M}_\lambda$. The challenger computes $(\hat{\mathsf{ct}}, \mathsf{ct}) \leftarrow \mathsf{Encrypt}(\mathsf{k}_i, \mathsf{m})$ and returns $(\hat{\mathsf{ct}}, \mathsf{ct})$ to $\mathcal{A}$.
  - $\mathcal{O}_{\mathsf{ReKeyGen}}(i, j, \hat{\mathsf{ct}})$: A query consists of indices $i, j \in [h + d]$ and a ciphertext header $\hat{\mathsf{ct}}$. If $j > h$ and $\mathsf{T}[i, \hat{\mathsf{ct}}] \neq \perp$, the challenger returns $\perp$. Otherwise, it computes $\Delta_{i,j,\hat{\mathsf{ct}}} \leftarrow$

$\mathsf{ReKeyGen}(\mathsf{k}_i, \mathsf{k}_j, \hat{\mathsf{ct}})$ and returns $\Delta_{i,j,\hat{\mathsf{ct}}}$ to $\mathcal{A}$. If $\mathsf{T}[i, \hat{\mathsf{ct}}] \neq \perp$, then the challenger computes $\mathsf{ct}' \leftarrow \mathsf{ReEncrypt}\big(\Delta_{i,j,\hat{\mathsf{ct}}}, (\hat{\mathsf{ct}}, \mathsf{T}[i, \hat{\mathsf{ct}}])\big)$ and sets $\mathsf{T}[j, \hat{\mathsf{ct}}'] \leftarrow \mathsf{ct}'$.

  - $\mathcal{O}_{\mathsf{ReEncrypt}}\big(i, j, (\hat{\mathsf{ct}}, \mathsf{ct})\big)$: A query consists of indices $i, j \in [h+d]$ and a ciphertext $(\hat{\mathsf{ct}}, \mathsf{ct})$. The challenger computes an update token $\Delta_{i,j,\hat{\mathsf{ct}}} \leftarrow \mathsf{ReKeyGen}(\mathsf{k}_i, \mathsf{k}_j, \hat{\mathsf{ct}})$ and updated ciphertext $(\hat{\mathsf{ct}}', \mathsf{ct}') \leftarrow \mathsf{ReEncrypt}\big(\Delta_{i,j,\hat{\mathsf{ct}}}, (\hat{\mathsf{ct}}, \mathsf{ct})\big)$. If $j > h$ and $\mathsf{T}[i, \hat{\mathsf{ct}}] \neq \perp$, the challenger returns $\perp$. Otherwise, it returns $(\hat{\mathsf{ct}}', \mathsf{ct}')$ to $\mathcal{A}$. Finally, if $j \leq h$ and $\mathsf{T}[i, \hat{\mathsf{ct}}] \neq \perp$, the challenger sets $\mathsf{T}[j, \hat{\mathsf{ct}}'] \leftarrow \mathsf{ct}'$.
  - $\mathcal{O}_{\mathsf{Challenge}}\big(i, \mathsf{m}_0, \mathsf{m}_1\big)$: A query consists of indices $i \in [h]$ and a pair of messages $\mathsf{m}_0, \mathsf{m}_1 \in \mathcal{M}_\lambda$ such that $|\mathsf{m}_0| = |\mathsf{m}_1|$. The challenger computes $(\hat{\mathsf{ct}}_b, \mathsf{ct}_b) \leftarrow \mathsf{Encrypt}(k_i, \mathsf{m}_b)$, sets $\mathsf{T}[i, \hat{\mathsf{ct}}_b] \leftarrow \mathsf{ct}$, and returns $(\hat{\mathsf{ct}}_b, \mathsf{ct}_b)$ to $\mathcal{A}$.

- **Output phase**: At the end of the experiment, the adversary $\mathcal{A}$ outputs a bit $b \in \{0, 1\}$, which is the output of the experiment.

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

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

Definition A.2 (Re-Encryption Indistinguishability [EPRS17]). Let $\Pi_{\mathsf{UAE}} = (\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{ReKeyGen}, \mathsf{ReEncrypt}, \mathsf{Decrypt})$ be an updatable authenticated encryption scheme for a message space $\mathcal{M}_\lambda$. 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 re-encryption indistinguishability experiment $\mathsf{Expt}^{\mathsf{re\text{-}enc\text{-}ind}}_{\Pi_{\mathsf{UAE}}}(\lambda, h, d, \mathcal{A}, b)$ as follows:

$\mathsf{Expt}^{\mathsf{re\text{-}enc\text{-}ind}}_{\Pi_{\mathsf{UAE}}}(\lambda, h, d, \mathcal{A}, b)$:

- **Setup phase**: At the start of the experiment, the challenger generates h uncorrupted keys $\mathsf{k}_1, \ldots, \mathsf{k}_h \leftarrow \mathsf{KeyGen}(1^\lambda)$ and d corrupted keys $\mathsf{k}_{h+1}, \ldots, \mathsf{k}_{h+d} \leftarrow \mathsf{KeyGen}(1^\lambda)$. It provides the corrupted keys $\mathsf{k}_{h+1}, \ldots, \mathsf{k}_{h+d}$ to the adversary $\mathcal{A}$.
- **Query phase**: Throughout the query phase of the experiment, the challenger maintains a look-up table T that maps *key index* and *ciphertext header* pairs to *ciphertext bodies*. The adversary is allowed to make the following queries to the challenger:
  - $\mathcal{O}_{\mathsf{Encrypt}}(i, \mathsf{m})$: A query consists of an index $i \in [h]$ and a message $\mathsf{m} \in \mathcal{M}_\lambda$. The challenger returns $\mathsf{Encrypt}(\mathsf{k}_i, \mathsf{m})$ to $\mathcal{A}$.
  - $\mathcal{O}_{\mathsf{ReKeyGen}}(i, j, \hat{\mathsf{ct}})$: A query consists of indices $i, j \in [h+d]$ and a ciphertext header $\hat{\mathsf{ct}}$. If $j > h$ and $\mathsf{T}[i, \hat{\mathsf{ct}}] \neq \perp$, the challenger returns $\perp$. Otherwise, it computes $\Delta_{i,j,\hat{\mathsf{ct}}} \leftarrow \mathsf{ReKeyGen}(\mathsf{k}_i, \mathsf{k}_j, \hat{\mathsf{ct}})$ and returns $\Delta_{i,j,\hat{\mathsf{ct}}}$ to $\mathcal{A}$. If $\mathsf{T}[i, \hat{\mathsf{ct}}] \neq \perp$, then the challenger computes $\mathsf{ReEncrypt}\big(\Delta_{i,j,\hat{\mathsf{ct}}}, (\hat{\mathsf{ct}}, \mathsf{T}[i, \hat{\mathsf{ct}}])\big)$ and sets $\mathsf{T}[j, \hat{\mathsf{ct}}'] \leftarrow \mathsf{ct}'$.
  - $\mathcal{O}_{\mathsf{ReEncrypt}}\big(i, j, (\hat{\mathsf{ct}}, \mathsf{ct})\big)$: A query consists of indices $i, j \in [h+d]$ and a ciphertext $(\hat{\mathsf{ct}}, \mathsf{ct})$. The challenger computes an update token $\Delta_{i,j,\hat{\mathsf{ct}}} \leftarrow \mathsf{ReKeyGen}(\mathsf{k}_i, \mathsf{k}_j, \hat{\mathsf{ct}})$ and updated ciphertext $(\hat{\mathsf{ct}}', \mathsf{ct}') \leftarrow \mathsf{ReEncrypt}\big(\Delta_{i,j,\hat{\mathsf{ct}}}, (\hat{\mathsf{ct}}, \mathsf{ct})\big)$. If $j > h$ and $\mathsf{T}[i, \hat{\mathsf{ct}}] \neq \perp$, the challenger returns $\perp$. Otherwise, it returns $(\hat{\mathsf{ct}}', \mathsf{ct}')$ to $\mathcal{A}$. Finally, if $j \leq h$ and $\mathsf{T}[i, \hat{\mathsf{ct}}] \neq \perp$, the challenger sets $\mathsf{T}[j, \hat{\mathsf{ct}}'] \leftarrow \mathsf{ct}'$.
  - $\mathcal{O}_{\mathsf{Challenge}}\big(i, j, (\hat{\mathsf{ct}}_0, \mathsf{ct}_0), (\hat{\mathsf{ct}}_1, \mathsf{ct}_1)\big)$: A query consists of indices $i \in [h+d]$, $j \in [h]$, and a pair of ciphertexts $(\hat{\mathsf{ct}}_0, \mathsf{ct}_0)$, $(\hat{\mathsf{ct}}_1, \mathsf{ct}_1)$ such that $|\mathsf{ct}_0| = |\mathsf{ct}_1|$. The challenger first computes $\Delta_{i,j,\hat{\mathsf{ct}}_b} \leftarrow \mathsf{ReKeyGen}(\mathsf{k}_i, \mathsf{k}_j, \hat{\mathsf{ct}}_b)$ and $(\hat{\mathsf{ct}}_b', \mathsf{ct}_b') \leftarrow \mathsf{ReEncrypt}(\Delta_{i,j,\hat{\mathsf{ct}}_b}, (\hat{\mathsf{ct}}_b, \mathsf{ct}_b))$. If any of the algorithms $\mathsf{ReKeyGen}(\mathsf{k}_i, \mathsf{k}_j, \hat{\mathsf{ct}}_b)$ or $\mathsf{ReEncrypt}(\Delta_{i,j,\hat{\mathsf{ct}}_b}, (\hat{\mathsf{ct}}_b, \mathsf{ct}_b))$ outputs $\perp$, then the challenger returns $\perp$. Otherwise, it sets $\mathsf{T}[j, \hat{\mathsf{ct}}_b'] \leftarrow \mathsf{ct}_b'$ and returns $(\hat{\mathsf{ct}}_b', \mathsf{ct}_b')$ to $\mathcal{A}$.
- **Output phase**: At the end of the experiment, the adversary $\mathcal{A}$ outputs a bit $b \in \{0, 1\}$, which is the output of the experiment.

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

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

Definition A.3 (Ciphertext Integrity [EPRS17])**.** 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 $h, d \in \mathbb{N}$, and an adversary $\mathcal{A}$, we define the ciphertext integrity experiment $\mathsf{Expt}^{\text{ctxt-int}}_{\Pi_{\mathsf{UAE}}}(\lambda, h, d, \mathcal{A})$ as follows:

$\mathsf{Expt}^{\text{ctxt-int}}_{\Pi_{\mathsf{UAE}}}(\lambda, h, d, \mathcal{A})$:

- **Setup phase**: At the start of the experiment, the challenger generates h uncorrupted keys $\mathsf{k}_1, \ldots, \mathsf{k}_h \leftarrow \mathsf{KeyGen}(1^\lambda)$ and d corrupted keys $\mathsf{k}_{h+1}, \ldots, \mathsf{k}_{h+d} \leftarrow \mathsf{KeyGen}(1^\lambda)$. It provides the corrupted keys $\mathsf{k}_{h+1}, \ldots, \mathsf{k}_{h+d}$ to the adversary $\mathcal{A}$.
- **Query phase**: Throughout the query phase of the experiment, the challenger maintains a look-up table T that maps *key index* and *ciphertext header* pairs to *ciphertext bodies.* The adversary is allowed to make the following queries to the challenger:
  - $\mathcal{O}_{\mathsf{Encrypt}}(i, \mathsf{m})$: A query consists of an index $i \in [h+d]$ and a message $\mathsf{m} \in \mathcal{M}_\lambda$. The challenger computes $(\hat{\mathsf{ct}}, \mathsf{ct}) \leftarrow \mathsf{Encrypt}(\mathsf{k}_i, \mathsf{m})$ and sets $\mathsf{T}[i, \hat{\mathsf{ct}}] \leftarrow \mathsf{ct}$. It returns $(\hat{\mathsf{ct}}, \mathsf{ct})$ to $\mathcal{A}$.
  - $\mathcal{O}_{\mathsf{ReKeyGen}}(i, j, \hat{\mathsf{ct}})$: A query consists of indices $i, j \in [h+d]$ and a ciphertext header $\hat{\mathsf{ct}}$. If $i > h$ and $j \leq h$, the challenger returns $\perp$. Otherwise, it computes $\Delta_{i,j,\hat{\mathsf{ct}}} \leftarrow \mathsf{ReKeyGen}(\mathsf{k}_i, \mathsf{k}_j, \hat{\mathsf{ct}})$ and returns $\Delta_{i,j,\hat{\mathsf{ct}}}$ to $\mathcal{A}$. If $\mathsf{T}[i, \hat{\mathsf{ct}}] \neq \perp$, the challenger computes $(\hat{\mathsf{ct}}', \mathsf{ct}') \leftarrow \mathsf{ReEncrypt}\big(\Delta_{i,j,\hat{\mathsf{ct}}}, (\hat{\mathsf{ct}}, \mathsf{T}[i, \hat{\mathsf{ct}}])\big)$ and sets $\mathsf{T}[j, \hat{\mathsf{ct}}'] \leftarrow \mathsf{ct}'$.
- **Output phase**: At the end of the experiment, the adversary $\mathcal{A}$ returns an index $i \in [h]$ and a ciphertext $(\hat{\mathsf{ct}}, \mathsf{ct})$. The challenger computes $\mathsf{m} \leftarrow \mathsf{Decrypt}\big(\mathsf{k}_i, (\hat{\mathsf{ct}}, \mathsf{ct})\big)$ and checks the following conditions:
  - $\mathsf{m} = \perp$,
  - $\mathsf{T}[i, \hat{\mathsf{ct}}] = \mathsf{ct}$.

  If any of the conditions above are met, then the challenger returns 0 and otherwise, it returns 1.

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

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

B Comparison to the Ciphertext-Independent Setting

Adaptive vs. Static corruption. Security definitions in the ciphertext-independent setting are largely defined analogously to those in ciphertext-dependent setting. One distinction, however, is in the way an adversary can query the oracles $\mathcal{O}_{\mathsf{ReKeyGen}}$ and $\mathcal{O}_{\mathsf{ReEncrypt}}$. In the security definitions for ciphertext-dependent updatable encryption, an adversary can query these oracles arbitrarily in any order, as long as the adversary does not make a sequence of queries that allows it to trivially win the game. In the ciphertext-independent setting as formulated in [LT18], the confidentiality security experiment is divided into epochs that must be executed in a fixed order. Each key that is generated in the experiment is associated with a unique epoch and therefore, an adversary is required to query these keys in a fixed order. Furthermore, the security definitions in the ciphertext-independent setting allow for separate compromises of keys and update tokens. These distinctions allow the existing updatable encryption constructions in the ciphertext-independent setting to satisfy *adaptive* security where the adversary can corrupt additional malicious keys throughout the progression of the experiment. In contrast, all existing constructions in the ciphertext-dependent setting, including the constructions in this work, satisfy *static* security.

Integrity. The only work that considers integrity for updatable encryption in the ciphertext-independent setting is that of Klooß et al. [KLR19], which provides two integrity definitions. In the first definition, the re-encryption oracle only accepts the ciphertexts that are honestly derived from ciphertexts that were previously provided by the challenger. The second integrity definition removes this restriction on re-encryption oracle queries. The first definition is analogous to the *relaxed* integrity notion that we describe in Definition C.2, and their stronger integrity definition is analogous to the integrity described in Definition 3.6. Klooß et al. [KLR19] also provide two constructions that each satisfy one of these definitions: a practical updatable encryption construction that satisfies the weaker integrity definition, and a theoretical construction that satisfies the stronger definition.

Efficiency. Although none of the existing updatable encryption schemes in the ciphertext-independent setting have yet been implemented, we can compare their asymptotic performance with those of the constructions in this work. The most efficient ciphertext-independent updatable encryption scheme, SHINE [BDGJ19], requires a similar number of symmetric and group operations as the ReCrypt updatable encryption scheme of [EPRS17], which works in the ciphertext-dependent setting. In Section 7, we show that the constructions in this work further improve the performance of ReCrypt and therefore, they would also outperform the existing updatable encryption schemes in the ciphertext-independent setting. Furthermore, unlike ReCrypt and the constructions in this work, SHINE does not provide any integrity protection. The best known ciphertext-independent construction that does provide integrity is that of Klooß et al. [KLR19], which requires plaintext-to-ciphertext ratio of 3, which is prohibitive for most applications.

C Full Definitions of Update Independence and Relaxed Integrity

For completeness, this section states the full definitions of update independence and relaxed integrity, which were omitted from the body of the text due to their similarity to security definitions already described earlier.

Definition C.1 (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 $\mathsf{Expt}^{\mathsf{upd\text{-}ind}}_{\Pi_{\mathsf{UAE}}}(\lambda, h, d, \mathcal{A}, b)$ as follows:

$\mathsf{Expt}^{\mathsf{upd\text{-}ind}}_{\Pi_{\mathsf{UAE}}}(\lambda, h, d, \mathcal{A}, b)$:

- **Setup phase**: At the start of the experiment, the challenger generates h uncorrupted keys $\mathsf{k}_1, \ldots, \mathsf{k}_h \leftarrow \mathsf{KeyGen}(1^\lambda)$ and d corrupted keys $\mathsf{k}_{h+1}, \ldots, \mathsf{k}_{h+d} \leftarrow \mathsf{KeyGen}(1^\lambda)$. It provides the corrupted keys $\mathsf{k}_{h+1}, \ldots, \mathsf{k}_{h+d}$ to the adversary $\mathcal{A}$.
- **Query phase**: Throughout the query phase of the experiment, the challenger maintains a look-up table T that maps *key index* and *ciphertext header* pairs to *ciphertext bodies*. The adversary is allowed to make the following queries to the challenger:
  - $\mathcal{O}_{\mathsf{Encrypt}}(i, \mathsf{m})$: A query consists of an index $i \in [h]$ and a message $\mathsf{m} \in \mathcal{M}_\lambda$. The challenger returns $\mathsf{Encrypt}(\mathsf{k}_i, \mathsf{m})$ to $\mathcal{A}$.
  - $\mathcal{O}_{\mathsf{ReEncrypt}}\big(i, j, (\hat{\mathsf{ct}}, \mathsf{ct})\big)$: A query consists of indices $i, j \in [h+d]$ and a ciphertext $(\hat{\mathsf{ct}}, \mathsf{ct})$. The challenger computes an update token $\Delta_{i,j,\hat{\mathsf{ct}}} \leftarrow \mathsf{ReKeyGen}(\mathsf{k}_i, \mathsf{k}_j, \hat{\mathsf{ct}})$ and updated ciphertext $(\hat{\mathsf{ct}}', \mathsf{ct}') \leftarrow \mathsf{ReEncrypt}\big(\Delta_{i,j,\hat{\mathsf{ct}}}, (\hat{\mathsf{ct}}, \mathsf{ct})\big)$. If $j > h$ and $\mathsf{T}[i, \hat{\mathsf{ct}}] \neq \perp$, the challenger returns $\perp$. Otherwise, it returns $(\hat{\mathsf{ct}}', \mathsf{ct}')$ to $\mathcal{A}$. Finally, if $j \leq h$ and $\mathsf{T}[i, \hat{\mathsf{ct}}] \neq \perp$, the challenger sets $\mathsf{T}[j, \hat{\mathsf{ct}}'] \leftarrow \mathsf{ct}'$.
  - $\mathcal{O}_{\mathsf{Challenge}}\big(i, j, \mathsf{m}, (\hat{\mathsf{ct}}, \mathsf{ct})\big)$: A query consists of indices $i \in [h+d]$, $j \in [h]$, a message $\mathsf{m} \in \mathcal{M}_\lambda$, and a ciphertext $(\hat{\mathsf{ct}}, \mathsf{ct})$. The challenger computes $(\hat{\mathsf{ct}}_0', \mathsf{ct}_0') \leftarrow \mathsf{Encrypt}(k_j, \mathsf{m})$, $\Delta_{i,j,\hat{\mathsf{ct}}} \leftarrow$

$\mathsf{ReKeyGen}(\mathsf{k}_i, \mathsf{k}_j, \hat{\mathsf{ct}})$, and $(\hat{\mathsf{ct}}_1', \mathsf{ct}_1') \leftarrow \mathsf{ReEncrypt}(\Delta_{i,j,\hat{\mathsf{ct}}}, (\hat{\mathsf{ct}}, \mathsf{ct}))$. If $|\hat{\mathsf{ct}}_0'| \neq |\hat{\mathsf{ct}}_1'|$, $|\mathsf{ct}_0'| \neq |\mathsf{ct}_1'|$, or any of the algorithms $\mathsf{ReKeyGen}(\mathsf{k}_i, \mathsf{k}_j, \hat{\mathsf{ct}})$, $\mathsf{ReEncrypt}(\Delta_{i,j,\hat{\mathsf{ct}}}, (\hat{\mathsf{ct}}, \mathsf{ct}))$ outputs $\perp$, then the challenger returns $\perp$. Otherwise, it sets $\mathsf{T}[j, \hat{\mathsf{ct}}_b'] \leftarrow \mathsf{ct}_b'$ and returns $(\hat{\mathsf{ct}}_b', \mathsf{ct}_b')$ to $\mathcal{A}$.

- **Output phase**: At the end of the experiment, the adversary $\mathcal{A}$ outputs a bit $b \in \{0, 1\}$, which is the output of the experiment.

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 $h, d = \mathsf{poly}(\lambda)$ and efficient adversaries $\mathcal{A}$, we have

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

Definition C.2 (Relaxed Integrity)**.** Let n and q be a positive integers, and let $\Pi_{\mathsf{UAE}} = (\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{ReKeyGen}, \mathsf{ReEncrypt}, \mathsf{Decrypt})$ be an updatable authenticated encryption scheme for a message space $\mathcal{M}$ and ciphertext space $(\mathbb{Z}_q^n)^*$. Then, for a security parameter λ, positive integers $h, d, \gamma \in \mathbb{N}$, and an adversary $\mathcal{A}$, we define the *relaxed* integrity experiment $\mathsf{Expt}_{\Pi_{\mathsf{UAE}}}^{\mathsf{relaxed\text{-}int}}(\lambda, h, d, \gamma, \mathcal{A})$ as follows:

$\mathsf{Expt}_{\Pi_{\mathsf{UAE}}}^{\mathsf{relaxed\text{-}int}}(\lambda, h, d, \gamma, \mathcal{A})$:

- **Setup phase**: At the start of the experiment, the challenger generates h uncorrupted keys $\mathsf{k}_1, \ldots, \mathsf{k}_h \leftarrow \mathsf{KeyGen}(1^\lambda)$ and d corrupted keys $\mathsf{k}_{h+1}, \ldots, \mathsf{k}_{h+d} \leftarrow \mathsf{KeyGen}(1^\lambda)$. It provides the corrupted keys $\mathsf{k}_{h+1}, \ldots, \mathsf{k}_{h+d}$ to the adversary $\mathcal{A}$.
- **Query phase**: Throughout the query phase of the experiment, the challenger maintains a look-up table T that maps *key index* and *ciphertext header* pairs to *ciphertext bodies* and *plaintexts*. The adversary is allowed to make the following queries to the challenger:
  - $\mathcal{O}_{\mathsf{Encrypt}}(i, \mathsf{m})$: A query consists of an index $i \in [h + d]$ and a message $\mathsf{m} \in \mathcal{M}$. The challenger computes $(\hat{\mathsf{ct}}, \mathsf{ct}) \leftarrow \mathsf{Encrypt}(\mathsf{k}_i, \mathsf{m})$ and sets $\mathsf{T}[i, \hat{\mathsf{ct}}] \leftarrow (\mathsf{ct}, \mathsf{m})$. It returns $(\hat{\mathsf{ct}}, \mathsf{ct})$ to $\mathcal{A}$.
  - $\mathcal{O}_{\mathsf{ReKeyGen}}(i, j, \hat{\mathsf{ct}})$: A query consists of indices $i, j \in [h + d]$ and a ciphertext header $\hat{\mathsf{ct}}$. If $i > h$ and $j \leq h$, the challenger returns $\perp$. Otherwise, it computes $\Delta_{i,j,\hat{\mathsf{ct}}} \leftarrow \mathsf{ReKeyGen}(\mathsf{k}_i, \mathsf{k}_j, \hat{\mathsf{ct}})$ and returns $\Delta_{i,j,\hat{\mathsf{ct}}}$ to $\mathcal{A}$. If $\mathsf{T}[i, \hat{\mathsf{ct}}] \neq \perp$, let $(\mathsf{ct}, \mathsf{m}) \leftarrow \mathsf{T}[i, \hat{\mathsf{ct}}]$. The challenger computes $(\hat{\mathsf{ct}}', \mathsf{ct}') \leftarrow \mathsf{ReEncrypt}\big(\Delta_{i,j,\hat{\mathsf{ct}}}, (\hat{\mathsf{ct}}, \mathsf{ct})\big)$ and sets $\mathsf{T}[j, \hat{\mathsf{ct}}'] \leftarrow (\mathsf{ct}', \mathsf{m})$.
  - $\mathcal{O}_{\mathsf{ReEncrypt}}\big(i, j, (\hat{\mathsf{ct}}, \mathsf{ct})\big)$: A query consists of indices $i, j \in [h + d]$ and a ciphertext $(\hat{\mathsf{ct}}, \mathsf{ct})$. The challenger first decrypts $\mathsf{m} \leftarrow \mathsf{Decrypt}\big(\mathsf{k}_i, (\hat{\mathsf{ct}}, \mathsf{ct})\big)$ and returns $\perp$ if $\mathsf{m} = \perp$. Otherwise, the challenger computes an update token $\Delta_{i,j,\hat{\mathsf{ct}}} \leftarrow \mathsf{ReKeyGen}(\mathsf{k}_i, \mathsf{k}_j, \hat{\mathsf{ct}})$, updated ciphertext $(\hat{\mathsf{ct}}', \mathsf{ct}') \leftarrow \mathsf{ReEncrypt}\big(\Delta_{i,j,\hat{\mathsf{ct}}}, (\hat{\mathsf{ct}}, \mathsf{ct})\big)$, and returns $(\hat{\mathsf{ct}}', \mathsf{ct}')$ to $\mathcal{A}$. If $j \leq h$, it sets $\mathsf{T}[j, \hat{\mathsf{ct}}'] \leftarrow (\mathsf{ct}', \mathsf{m})$.
- **Output phase**: At the end of the experiment, the adversary $\mathcal{A}$ returns an index $i \in [h]$ and a ciphertext $(\hat{\mathsf{ct}}, \mathsf{ct})$ for $\mathsf{ct} = (\mathsf{ct}_1, \ldots, \mathsf{ct}_\ell) \in (\mathbb{Z}_q^n)^\ell$. The challenger computes $\mathsf{m} \leftarrow \mathsf{Decrypt}\big(\mathsf{k}_i, (\hat{\mathsf{ct}}, \mathsf{ct})\big)$ and checks the following conditions:
  - $\mathsf{m} = \perp$,
  - For $(\mathsf{ct}', \mathsf{m}') \leftarrow \mathsf{T}[i, \hat{\mathsf{ct}}], \mathsf{ct}' = (\mathsf{ct}_1', \ldots, \mathsf{ct}_{\ell'}')$, we have that $\mathsf{m} = \mathsf{m}'$, $\ell = \ell'$, and $\big\|\mathsf{ct}_j - \mathsf{ct}_j'\big\| \leq \gamma$ for all $j \in [\ell]$.

  If either of the conditions above are met, then the challenger returns 0. Otherwise, it returns 1.

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

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

D Full Construction of Simple Nested Scheme

Construction D.1. Our construction uses the following building block:

- A standard 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 construct an updatable authenticated encryption scheme $\Pi_{\mathsf{UAE}} = (\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{ReKeyGen}, \mathsf{ReEncrypt}, \mathsf{Decrypt})$ for message space $\mathcal{M} = (\mathcal{M}_\lambda)_{\lambda \in \mathbb{N}}$ as follows:

- $\mathsf{KeyGen}(1^\lambda) \to \mathsf{k}$: On input the security parameter λ, the key generation algorithm samples an authenticated encryption key $\hat{\mathsf{k}} \leftarrow \mathsf{AE.KeyGen}(1^\lambda)$ and sets $\mathsf{k} \leftarrow \hat{\mathsf{k}}$.

- $\mathsf{Encrypt}(\mathsf{k}, \mathsf{m}) \to (\hat{\mathsf{ct}}, \mathsf{ct})$: On input a key $\mathsf{k} = \hat{\mathsf{k}}$, and a message $\mathsf{m} \in \mathcal{M}_\lambda$, the encryption algorithm first samples a new authenticated encryption key $\mathsf{k}_{\mathsf{ae}} \leftarrow \mathsf{AE.KeyGen}(1^\lambda)$. It then encrypts the key $\hat{\mathsf{ct}} \leftarrow \mathsf{AE.Encrypt}\big(\hat{\mathsf{k}}, (\mathsf{k}_{\mathsf{ae}}, \perp)\big)$, encrypts the message $\mathsf{ct} \leftarrow \mathsf{AE.Encrypt}\big(\mathsf{k}_{\mathsf{ae}}, (\mathsf{m}, \perp)\big)$, and returns $(\hat{\mathsf{ct}}, \mathsf{ct})$.

- $\mathsf{ReKeyGen}(\mathsf{k}_1, \mathsf{k}_2, \hat{\mathsf{ct}}) \to \Delta_{1,2,\hat{\mathsf{ct}}}/\perp$: On input two keys $\mathsf{k}_1 = \hat{\mathsf{k}}_1$, $\mathsf{k}_2 = \hat{\mathsf{k}}_2$, and a ciphertext header $\hat{\mathsf{ct}}$, the re-encryption key generation algorithm first samples new authenticated encryption keys $\hat{\mathsf{k}}_{\mathsf{history}}, \mathsf{k}'_{\mathsf{ae}} \leftarrow \mathsf{AE.KeyGen}(1^\lambda)$. It then re-encrypts the ciphertext header $\hat{\mathsf{ct}}_{\mathsf{history}} \leftarrow \mathsf{AE.Encrypt}\big(\hat{\mathsf{k}}_{\mathsf{history}}, \mathsf{AE.Decrypt}(\hat{\mathsf{k}}_1, \hat{\mathsf{ct}})\big)$, generates a new ciphertext header $\hat{\mathsf{ct}} \leftarrow \mathsf{AE.Encrypt}\big(\hat{\mathsf{k}}_2, (\mathsf{k}'_{\mathsf{ae}}, \hat{\mathsf{k}}_{\mathsf{history}})\big)$, and returns $\Delta_{1,2,\hat{\mathsf{ct}}} \leftarrow (\mathsf{k}'_{\mathsf{ae}}, \hat{\mathsf{ct}}', \hat{\mathsf{ct}}_{\mathsf{history}})$.

- $\mathsf{ReEncrypt}\big(\Delta_{1,2,\hat{\mathsf{ct}}}, (\hat{\mathsf{ct}}, \mathsf{ct})\big)$: On input an update token $\Delta_{1,2,\hat{\mathsf{ct}}}$ and a ciphertext $(\hat{\mathsf{ct}}, \mathsf{ct})$, the re-encryption algorithm first parses $\Delta_{1,2,\hat{\mathsf{ct}}} = (\mathsf{k}'_{\mathsf{ae}}, \hat{\mathsf{ct}}', \hat{\mathsf{ct}}_{\mathsf{history}})$. Then, it encrypts the ciphertext $\mathsf{ct}' \leftarrow \mathsf{AE.Encrypt}\big(\mathsf{k}'_{\mathsf{ae}}, (\mathsf{ct}, \hat{\mathsf{ct}}_{\mathsf{history}})\big)$ and returns $(\hat{\mathsf{ct}}', \mathsf{ct}')$.

- $\mathsf{Decrypt}\big(\mathsf{k}, (\hat{\mathsf{ct}}, \mathsf{ct})\big) \to \mathsf{m}/\perp$: On input a key $\mathsf{k} = \hat{\mathsf{k}}$ and a ciphertext $(\hat{\mathsf{ct}}, \mathsf{ct})$, the decryption algorithm proceeds as follows:

  1. If $\mathsf{AE.Decrypt}(\hat{\mathsf{k}}, \hat{\mathsf{ct}}) = \perp$, then return $\perp$. Otherwise, set $(\mathsf{k}'_{\mathsf{ae}}, \hat{\mathsf{k}}'_{\mathsf{history}}) \leftarrow \mathsf{AE.Decrypt}(\hat{\mathsf{k}}, \hat{\mathsf{ct}})$.
  2. If $\mathsf{AE.Decrypt}(\mathsf{k}'_{\mathsf{ae}}, \mathsf{ct}) = \perp$, then return $\perp$. Otherwise, set $(\mathsf{ct}', \hat{\mathsf{ct}}'_{\mathsf{history}}) \leftarrow \mathsf{AE.Decrypt}(\mathsf{k}'_{\mathsf{ae}}, \mathsf{ct})$.
  3. If $\hat{\mathsf{k}}'_{\mathsf{history}} \neq \perp$, then set $\hat{\mathsf{k}} \leftarrow \hat{\mathsf{k}}'_{\mathsf{history}}$, $\mathsf{ct} \leftarrow \mathsf{ct}'$, $\hat{\mathsf{ct}} \leftarrow \hat{\mathsf{ct}}'_{\mathsf{history}}$, and return to step 1.
  4. Otherwise, set $\mathsf{m} \leftarrow \mathsf{ct}'$.

  If the decryption algorithm does not abort above, then it returns m.

We state the compactness, correctness, and the security properties of Construction D.1 as follows.

Theorem D.2 (Compactness)**.** *The updatable authenticated encryption scheme* Π_{UAE} *in Construction D.1 satisfies compactness (Definition 3.2).*

Theorem D.3 (Correctness)**.** *Suppose that* Π_{AE} *satisfies correctness (Definition 2.7). Then, the updatable authenticated encryption scheme* Π_{UAE} *in Construction D.1 satisfies correctness (Definition 3.3).*

Theorem D.4 (Message Confidentiality)**.** *Let* Π_{UAE} *be the updatable authenticated encryption scheme in Construction D.1. If* Π_{AE} *satisfies correctness,* $\varepsilon^{\mathsf{conf}}_{\mathsf{ae}}$*-confidentiality, and* $\varepsilon^{\mathsf{int}}_{\mathsf{ae}}$*-integrity, then for all* $h, d = \mathsf{poly}(\lambda)$ *and efficient adversaries* $\mathcal{A}$ *that make at most* Q *oracle queries, we have*

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

In particular, if Π_{AE} *satisfies correctness, confidentiality, and integrity, then* Π_{UAE} *satisfies message confidentiality (Definition A.1).*

Theorem D.5 (Re-Encryption Indistinguishability)**.** *Let Π_{UAE} be the updatable authenticated encryption scheme in Construction D.1. If Π_{AE} satisfies correctness, $\varepsilon_{\mathsf{ae}}^{\mathsf{conf}}$-confidentiality, and $\varepsilon_{\mathsf{ae}}^{\mathsf{int}}$-integrity, then for all $h, d = \mathsf{poly}(\lambda)$ and efficient adversaries $\mathcal{A}$ that make at most Q oracle queries, we have*

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

In particular, if Π_{AE} satisfies correctness, confidentiality, and integrity, then Π_{UAE} satisfies confidentiality (Definition A.2).

Theorem D.6 (Ciphertext Integrity)**.** *Let Π_{UAE} be the updatable authenticated encryption scheme in Construction D.1. If Π_{AE} satisfies correctness, $\varepsilon_{\mathsf{ae}}^{\mathsf{conf}}$-confidentiality, $\varepsilon_{\mathsf{ae}}^{\mathsf{int}}$-integrity, then for all $h, d = \mathsf{poly}(\lambda)$ and efficient adversaries $\mathcal{A}$ that make at most Q challenge,* $\mathsf{ReKeyGen}$*, or* $\mathsf{ReEncrypt}$ *queries, we have*

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

In particular, if Π_{AE} satisfies correctness, confidentiality, and integrity then Π_{UAE} satisfies integrity (Definition 3.6).

In Section 4 (Construction 4.2), we augment Construction D.1 such that it achieves our new security definitions of *body compactness* (Definition 3.5) and *update independence* (Definition C.1). As the proofs of compactness, correctness, and security for Construction D.1 are identical to the proofs of compactness, correctness, and security for Construction 4.2, we refrain from duplicating the proofs.

E Proof of Theorem 4.3

E.1 Proof of Strong Compactness

Header compactness. Fix the security parameter λ, any message $\mathsf{m} \in \mathcal{M}_\lambda$, and let $\mathsf{k}_1, \mathsf{k}_2 \leftarrow \mathsf{KeyGen}(1^\lambda)$, $(\hat{\mathsf{ct}}, \mathsf{ct}) \leftarrow \mathsf{Encrypt}(\mathsf{k}_1, \mathsf{m})$, and $\Delta_{1,2,\hat{\mathsf{ct}}} \leftarrow \mathsf{ReKeyGen}(\mathsf{k}_1, \mathsf{k}_2, \hat{\mathsf{ct}})$. By construction, the ciphertext header $\hat{\mathsf{ct}}$ and the re-encryption key $\Delta_{1,2,\hat{\mathsf{ct}}}$ has the following form:

- *Ciphertext header*: By the specification of $\mathsf{Encrypt}$ and $\mathsf{ReKeyGen}$, we have $\hat{\mathsf{ct}} = \mathsf{AE.Encrypt}\big(\mathsf{k}_1, (s, \ell, \mathsf{k}_{\mathsf{ae}}, \hat{\mathsf{k}}_{\mathsf{history}})\big)$. Here, $\mathsf{k}_{\mathsf{ae}}, \hat{\mathsf{k}}_{\mathsf{history}}$ are authenticated encryption keys and s is a PRG seed whose lengths depend only on the security parameter λ. The positive integer ℓ specifies the length of the ciphertext body, whose length can fixed to be $\mathsf{poly}(\lambda)$.

- *Re-encryption key*: The re-encryption key $\Delta_{1,2,\hat{\mathsf{ct}}}$ consists of a new ciphertext header $\hat{\mathsf{ct}}'$, a ciphertext $\hat{\mathsf{ct}}_{\mathsf{history}} \leftarrow \mathsf{AE.Encrypt}(\hat{\mathsf{k}}'_{\mathsf{history}}, (\mathsf{k}_{\mathsf{ae}}, \hat{\mathsf{k}}_{\mathsf{history}}))$, length bound ℓ, a PRG seed s', and an encryption key $\mathsf{k}'_{\mathsf{ae}}$. As described above, the lengths of $\hat{\mathsf{ct}}'$, ℓ, k_{ae}, and s' are all either fixed or depend only on λ. This is also true of $|\hat{\mathsf{ct}}_{\mathsf{history}}|$ because it is an encryption of two encryption keys, each of which has a length that depends only on λ. Therefore, $|\Delta_{1,2,\hat{\mathsf{ct}}}| = \mathsf{poly}_2(\lambda)$ for some polynomial poly_2.

Body compactness. Fix the security parameter λ, number of updates $N \in \mathbb{N}$, and any message $\mathsf{m} \in \mathcal{M}_\lambda$. Let $\mathsf{k}_1, \ldots, \mathsf{k}_t \leftarrow \mathsf{KeyGen}(1^\lambda, 1^t)$, $(\hat{\mathsf{ct}}_1, \mathsf{ct}_1) \leftarrow \mathsf{Encrypt}(\mathsf{k}_1, \mathsf{m})$, 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 $i = 1, \ldots, t-1$. By construction, the ciphertext body ct_i for $i > 1$ always has the same size $|\mathsf{ct}_{i-1}|$ as the preceding ciphertext because padding is added until the new ciphertext matches the length of the previous ciphertext. Thus we only need to consider the size of the first ciphertext $|\mathsf{ct}_1|$. The ciphertext

ct_1 consists of $\mathsf{ct}_{\mathsf{payload}} \leftarrow \mathsf{AE.Encrypt}(\mathsf{k}_{\mathsf{ae}}, \mathsf{m})$ and $t(2\rho+\nu)$ bits of padding generated by the PRG G, for ρ and ν determined by λ and the AE scheme used. Thus the size of the padding $|\mathsf{ct}_{\mathsf{pad}}| = \mathsf{poly}_1(t, \lambda)$ for some polynomial poly_1. We have that $|\mathsf{AE.Encrypt}(\mathsf{k}_{\mathsf{ae}}, \mathsf{m})| = \mathsf{poly}_2(\lambda, \mathsf{m})$ for some polynomial poly_2. Then $|\mathsf{ct}| = \mathsf{poly}_1(t, \lambda) + \mathsf{poly}_2(\lambda, \mathsf{m}) = \mathsf{poly}(\lambda, t, \mathsf{m})$ for another polynomial poly, completing the proof of bounded body compactness.

E.2 Proof of Correctness

Fix the security parameter λ, number of updates $N \in \mathbb{N}$, and any message $\mathsf{m} \in \mathcal{M}_\lambda$. Let $\mathsf{k}_1, \ldots, \mathsf{k}_N \leftarrow \mathsf{KeyGen}(1^\lambda)$, $(\hat{\mathsf{ct}}_1, \mathsf{ct}_1) \leftarrow \mathsf{Encrypt}(\mathsf{k}_1, \mathsf{m})$, 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 must show that

$$\mathsf{Decrypt}\big(\mathsf{k}_N, (\hat{\mathsf{ct}}_N, \mathsf{ct}_N)\big) = \mathsf{m}.$$

We will prove correctness of Construction 4.2 by induction on i, the number of layers of encryption on the ciphertext body ct. First we consider the base case of $i = 1$, where $(\hat{\mathsf{ct}}_1, \mathsf{ct}_1) \leftarrow \mathsf{Encrypt}(\mathsf{k}_1, \mathsf{m})$. In this case we have

- $\mathsf{ct}_{\mathsf{payload}} \leftarrow \mathsf{AE.Encrypt}\big(\mathsf{k}_{\mathsf{ae}}, \mathsf{m}\big)$
- $\mathsf{ct}_{\mathsf{pad}} \leftarrow G(s)$ such that $\mathsf{ct}_{\mathsf{pad}} \in \{0,1\}^{t\cdot(2\rho+\nu)}$
- $\mathsf{ct}_1 \leftarrow (\mathsf{ct}_{\mathsf{payload}}, \mathsf{ct}_{\mathsf{pad}})$
- $\hat{\mathsf{ct}}_1 \leftarrow \mathsf{AE.Encrypt}\big(\hat{\mathsf{k}}_1, (s, |\mathsf{ct}_{\mathsf{payload}}|, \mathsf{k}_{\mathsf{ae}}, \perp)\big)$

During decryption, by the correctness of Π_{AE}, we have that $\mathsf{AE.Decrypt}(\hat{\mathsf{k}}_1, \hat{\mathsf{ct}}_1) = (s, |\mathsf{ct}_{\mathsf{payload}}|, \mathsf{k}_{\mathsf{ae}}, \perp)$. Thus the $\mathsf{Decrypt}$ algorithm correctly regenerates $G(s) = \mathsf{ct}_{\mathsf{pad}}$ and strips it from ct_1. Finally, again using the correctness of Π_{AE}, $\mathsf{AE.Decrypt}(\mathsf{k}_{\mathsf{ae}}, \mathsf{ct}_{\mathsf{payload}}) = \mathsf{m}$.

Next, assuming $\mathsf{Decrypt}(\mathsf{k}_i, (\hat{\mathsf{ct}}_i, \mathsf{ct}_i)) = \mathsf{m}$, and that decryption of $\mathsf{ct}_{\mathsf{payload}\ ,i}$ under $\mathsf{k}_{\mathsf{ae}\ ,i}$ outputs m, we show that $\mathsf{Decrypt}(\mathsf{k}_{i+1}, (\hat{\mathsf{ct}}_{i+1}, \mathsf{ct}_{i+1})) = \mathsf{m}$ as well. For consistency of notation, we will denote ciphertexts corresponding to the encryption under k_i without a subscript and denote ciphertexts corresponding to the encryption under k_{i+1} with a prime, e.g., $\hat{\mathsf{ct}}'$. Now we have

- $\hat{\mathsf{ct}}_{\mathsf{history}} \leftarrow \mathsf{AE.Encrypt}(\hat{\mathsf{k}}'_{\mathsf{history}}, (\mathsf{k}_{\mathsf{ae}}, \hat{\mathsf{k}}_{\mathsf{history}}))$
- $\mathsf{ct}'_{\mathsf{payload}} \leftarrow \mathsf{AE.Encrypt}\big(\mathsf{k}'_{\mathsf{ae}}, (\mathsf{ct}_{\mathsf{payload}}, \hat{\mathsf{ct}}_{\mathsf{history}})\big)$
- $\mathsf{ct}'_{\mathsf{pad}} \leftarrow G(s')$
- $\mathsf{ct}' \leftarrow (\mathsf{ct}'_{\mathsf{payload}}, \mathsf{ct}'_{\mathsf{pad}})$
- $\hat{\mathsf{ct}}' \leftarrow \mathsf{AE.Encrypt}\big(\hat{\mathsf{k}}_{i+1}, (s', \ell, \mathsf{k}'_{\mathsf{ae}}, \hat{\mathsf{k}}'_{\mathsf{history}})\big)$

As above, during decryption, by the correctness of Π_{AE}, we have that $\mathsf{AE.Decrypt}(\hat{\mathsf{k}}', \hat{\mathsf{ct}}') = (s', \ell, \mathsf{k}'_{\mathsf{ae}}, \hat{\mathsf{k}}'_{\mathsf{history}})$. Thus the $\mathsf{Decrypt}$ algorithm correctly regenerates $G(s') = \mathsf{ct}_{\mathsf{pad}}$ and strips it from ct_1. By again using the correctness of Π_{AE}, we have that $\mathsf{AE.Decrypt}(\mathsf{k}'_{\mathsf{ae}}, \mathsf{ct}'_{\mathsf{payload}}) = (\mathsf{ct}_{\mathsf{payload}}, \hat{\mathsf{ct}}_{\mathsf{history}})$ and that $\mathsf{AE.Decrypt}(\hat{\mathsf{k}}'_{\mathsf{history}}, \hat{\mathsf{ct}}_{\mathsf{history}}) = (\mathsf{k}_{\mathsf{ae}}, \hat{\mathsf{k}}_{\mathsf{history}})$. Now, by the induction hypothesis, we have $\mathsf{ct}_{\mathsf{payload}}$ under k_{ae} decrypts to m.

E.3 Proofs of Update Independence, Message Confidentiality, and Re-encryption Indistinguishability

The proofs of update independence, message confidentiality, and re-encryption indistinguishability are almost identical. Therefore, we provide a full proof of update independence and discuss the minor modifications that are required to adapt the proof for message confidentiality and re-encryption indistinguishability.

Update independence. We proceed via a sequence of hybrid experiments that are defined as follows:

- Hyb_0: This hybrid experiment corresponds to the real updatable authenticated encryption update independence experiment $\mathsf{Expt}^{\mathsf{upd\text{-}ind}}_{\Pi_{\mathsf{UAE}}}(\lambda, h, d, \mathcal{A}, 0)$ that is instantiated with Construction 4.2.

- Hyb_1: In this hybrid experiment, we introduce an additional abort condition to the challenger's simulation. Namely, throughout the query phase of the experiment, the challenger maintains an additional look-up table $\mathsf{T}_{\mathsf{header}}$ that keeps track of all of the "well-formed" ciphertext headers under honest keys that $\mathcal{A}$ receives from the challenger throughout the experiment. The table is initially set to be empty, and the challenger answers each of $\mathcal{A}$'s oracle queries as follows:

  - $\mathcal{O}_{\mathsf{Encrypt}}(i, \mathsf{m})$: The challenger answers the oracle exactly as in Hyb_0 by generating the ciphertext header
  $$\hat{\mathsf{ct}} \leftarrow \mathsf{AE.Encrypt}\big(\hat{\mathsf{k}}_i, (s, |\mathsf{ct}_{\mathsf{payload}}|, \mathsf{k}_{\mathsf{ae}}, \perp)\big),$$
  and the ciphertext body as specified in construction 4.2. However, after returning the ciphertext $(\hat{\mathsf{ct}}, \mathsf{ct})$ to $\mathcal{A}$, it additionally adds the mapping $\mathsf{T}_{\mathsf{header}}[i, \hat{\mathsf{ct}}] \leftarrow (s, |\mathsf{ct}_{\mathsf{payload}}|, \mathsf{k}_{\mathsf{ae}}, \perp)$ to the table if $i < h$.

  - $\mathcal{O}_{\mathsf{ReEncrypt}}\big(i, j, (\hat{\mathsf{ct}}, \mathsf{ct})\big)$: The challenger first checks if $\mathsf{T}_{\mathsf{header}}[i, \hat{\mathsf{ct}}] = \perp$ and outputs $\perp$ if so. This check is skipped if $i > h$ (if i does not correspond to an honest key). If it does not abort, then the challenger proceeds exactly as in Hyb_0 by setting $\hat{\mathsf{ct}}' \leftarrow \mathsf{AE.Encrypt}\big(\hat{\mathsf{k}}_j, (s', l, \mathsf{k}'_{\mathsf{ae}}, \hat{\mathsf{k}}'_{\mathsf{history}})\big)$, computing $\Delta_{i,j,\hat{\mathsf{ct}}}$, and returning $\mathsf{ReEncrypt}\big(\Delta_{i,j,\hat{\mathsf{ct}}}, (\hat{\mathsf{ct}}, \mathsf{ct})\big)$ to $\mathcal{A}$. However, after returning the updated ciphertext to $\mathcal{A}$, it adds the mapping $\mathsf{T}_{\mathsf{header}}[j, \hat{\mathsf{ct}}'] \leftarrow (s', l, \mathsf{k}'_{\mathsf{ae}}, \hat{\mathsf{k}}'_{\mathsf{history}})$ to the table.

  - $\mathcal{O}_{\mathsf{Challenge}}(i, j, \mathsf{m}, (\hat{\mathsf{ct}}, \mathsf{ct}))$: The challenger first checks if $\mathsf{T}_{\mathsf{header}}[i, \hat{\mathsf{ct}}] = \perp$ and outputs $\perp$ if so. This check is skipped if $i > h$ (if i does not correspond to an honest key). If it does not abort, then the challenger answers the oracle query exactly as in Hyb_0 by generating the ciphertext header
  $$\hat{\mathsf{ct}}' \leftarrow \mathsf{AE.Encrypt}\big(\mathsf{k}_j, (s, |\mathsf{ct}_{\mathsf{payload}}|, \mathsf{k}_{\mathsf{ae}}, \perp)\big),$$
  and the ciphertext body ct' according to the specification of $\mathsf{Encrypt}(\mathsf{k}_j, \mathsf{m})$ when $b = 0$ or by generating the ciphertext header
  $$\hat{\mathsf{ct}}' \leftarrow \mathsf{AE.Encrypt}\big(\mathsf{k}_j, (s', l, \mathsf{k}'_{\mathsf{ae}}, \hat{\mathsf{k}}'_{\mathsf{history}})\big)$$
  and ciphertext body ct' according to $\mathsf{ReEncrypt}\big(\Delta_{i,j,\hat{\mathsf{ct}}}, (\hat{\mathsf{ct}}, \mathsf{ct})\big)$ when $b = 1$. However, after returning the ciphertext $(\hat{\mathsf{ct}}', \mathsf{ct}')$ to $\mathcal{A}$, it additionally adds the mapping $\mathsf{T}_{\mathsf{header}}[j, \hat{\mathsf{ct}}'] \leftarrow (s, |\mathsf{ct}_{\mathsf{payload}}|, \mathsf{k}_{\mathsf{ae}}, \perp)$ to the table when $b = 0$ or $\mathsf{T}_{\mathsf{header}}[j, \hat{\mathsf{ct}}'] \leftarrow (s', l, \mathsf{k}'_{\mathsf{ae}}, \hat{\mathsf{k}}'_{\mathsf{history}})$ when $b = 1$.

  The rest of the experiment remains unchanged from Hyb_0.

  In Lemma E.1 below, we show that the hybrid experiments Hyb_0 and Hyb_1 are computationally indistinguishable assuming that Π_{AE} satisfies integrity.

- Hyb_2: In this hybrid experiment, we erase the decryption algorithm $\mathsf{AE.Decrypt}$ from the challenger's simulation for honest keys in ciphertext headers. Namely, the challenger answers $\mathcal{A}$'s re-encryption oracle and challenge oracle as follows when $i \leq h$:

  - $\mathcal{O}_{\mathsf{ReEncrypt}}\big(i, j, (\hat{\mathsf{ct}}, \mathsf{ct})\big)$: The challenger answers the oracle exactly as in Hyb_1, but instead of decrypting the ciphertext header $\mu \leftarrow \mathsf{AE.Decrypt}(\mathsf{k}_i, \hat{\mathsf{ct}})$, it sets $\mu \leftarrow \mathsf{T}_{\mathsf{header}}[i, \hat{\mathsf{ct}}]$. If no such entry exists in $\mathsf{T}_{\mathsf{header}}$, then it immediately aborts the experiment and outputs $\perp$.

  - $\mathcal{O}_{\mathsf{Challenge}}(i, j, \mathsf{m}, (\hat{\mathsf{ct}}, \mathsf{ct}))$: The challenger answers the oracle exactly as in Hyb_1, but instead of decrypting the ciphertext header $\mu \leftarrow \mathsf{AE.Decrypt}(\mathsf{k}_i, \hat{\mathsf{ct}})$ in the call to $\mathsf{ReKeyGen}$, it sets $\mu \leftarrow \mathsf{T}_{\mathsf{header}}[i, \hat{\mathsf{ct}}]$. If no such entry exists in $\mathsf{T}_{\mathsf{header}}$, then it immediately aborts the experiment and outputs $\perp$.

The rest of the experiment remains unchanged from Hyb_1.

In Lemma E.2 below, we show that the hybrid experiments Hyb_1 and Hyb_2 are perfectly indistinguishable assuming that Π_{AE} is correct.

- Hyb_3: In this hybrid experiment, we erase the contents of ciphertext headers for honest keys. Namely, the challenger answers each of $\mathcal{A}$'s oracle queries as follows, where intlen represents the number of bits required to represent an integer:
  - $\mathcal{O}_{\mathsf{Encrypt}}(i, \mathsf{m})$: If $i \leq h$, the challenger answers the oracle query exactly as in Hyb_2, but it sets the ciphertext header $\hat{\mathsf{ct}}$ to be
  $$\hat{\mathsf{ct}} \leftarrow \mathsf{AE.Encrypt}\big(\hat{\mathsf{k}}_i, (0^{|s|}, 0^{\mathsf{intlen}}, 0^{|\mathsf{k}_{\mathsf{ae}}|}, 0^{\lambda})\big).$$
  The rest of the simulation in answering $\mathcal{A}$'s queries remain unchanged.
  - $\mathcal{O}_{\mathsf{ReEncrypt}}\big(i, j, (\hat{\mathsf{ct}}, \mathsf{ct})\big)$: If $j \leq h$, the challenger answers the oracle query exactly as in Hyb_2, but it sets the ciphertext header $\hat{\mathsf{ct}}'$ to be
  $$\hat{\mathsf{ct}}' \leftarrow \mathsf{AE.Encrypt}\big(\hat{\mathsf{k}}_j, (0^{|s'|}, 0^{\mathsf{intlen}}, 0^{|\mathsf{k}'_{\mathsf{ae}}|}, 0^{|\hat{\mathsf{k}}'_{\mathsf{history}}|})\big).$$
  The rest of the simulation in answering $\mathcal{A}$'s queries remain unchanged.
  - $\mathcal{O}_{\mathsf{Challenge}}(i, j, \mathsf{m}, (\hat{\mathsf{ct}}, \mathsf{ct}))$: If $j \leq h$, the challenger answers the oracle query exactly as in Hyb_2, but it sets the ciphertext header $\hat{\mathsf{ct}}'$ to be
  $$\hat{\mathsf{ct}}' \leftarrow \mathsf{AE.Encrypt}\big(\hat{\mathsf{k}}_j, ((0^{|s|}, 0^{\mathsf{intlen}}, 0^{|\mathsf{k}_{\mathsf{ae}}|}, 0^{\lambda})\big)$$
  when $b = 0$ or
  $$\hat{\mathsf{ct}}' \leftarrow \mathsf{AE.Encrypt}\big(\hat{\mathsf{k}}_j, ((0^{|s'|}, 0^{\mathsf{intlen}}, 0^{|\mathsf{k}'_{\mathsf{ae}}|}, 0^{\hat{\mathsf{k}}'_{\mathsf{history}}})\big)$$
  when $b = 1$. The rest of the simulation in answering $\mathcal{A}$'s queries remain unchanged.

  The rest of the experiment remains unchanged from Hyb_2.

  In Lemma E.3 below, we show that the hybrid experiments Hyb_2 and Hyb_3 are computationally indistinguishable assuming that Π_{AE} satisfies confidentiality.

- Hyb_4: In this hybrid experiment, we replace the output of $G(\cdot)$ with a random string. Namely, the challenger answers each of $\mathcal{A}$'s oracle queries as follows:
  - $\mathcal{O}_{\mathsf{Encrypt}}(i, \mathsf{m})$: If $i \leq h$, the challenger answers the oracle exactly as in Hyb_3, but instead of evaluating $\mathsf{ct}_{\mathsf{pad}} \leftarrow G(s)$, it sets $\mathsf{ct}_{\mathsf{pad}} \xleftarrow{\mathrm{R}} \{0,1\}^{t(2\rho+\nu)}$. The rest of the simulation in answering $\mathcal{A}$'s queries remains unchanged.
  - $\mathcal{O}_{\mathsf{ReEncrypt}}\big(i, j, (\hat{\mathsf{ct}}, \mathsf{ct})\big)$: If $j \leq h$, the challenger answers the oracle exactly as in Hyb_3, but instead of evaluating $\mathsf{ct}'_{\mathsf{pad}} \leftarrow G(s')$, it sets $\mathsf{ct}'_{\mathsf{pad}} \xleftarrow{\mathrm{R}} \{0,1\}^{|\mathsf{ct}|-|\mathsf{ct}'_{\mathsf{payload}}|}$. The rest of the simulation in answering $\mathcal{A}$'s queries remains unchanged.
  - $\mathcal{O}_{\mathsf{Challenge}}(i, j, \mathsf{m}, (\hat{\mathsf{ct}}, \mathsf{ct}))$: If $j \leq h$, the challenger answers the oracle exactly as in Hyb_3, but instead of evaluating $\mathsf{ct}'_{\mathsf{pad}} \leftarrow G(s')$ when $b = 1$, it sets $\mathsf{ct}'_{\mathsf{pad}} \xleftarrow{\mathrm{R}} \{0,1\}^{|\mathsf{ct}|-|\mathsf{ct}'_{\mathsf{payload}}|}$. The rest of the simulation in answering $\mathcal{A}$'s queries remains unchanged.

  The rest of the experiment remains unchanged from Hyb_3.

  In Lemma E.5 below, we show that the hybrid experiments Hyb_3 and Hyb_4 are computationally indistinguishable assuming that G satisfies PRG security.

- Hyb_5: In this hybrid we replace $\mathsf{AE.Encrypt}(\hat{\mathsf{k}}'_{\mathsf{history}}, \cdot)$ in oracle queries made under honest keys with a completely random function. Namely, the challenger answers each of $\mathcal{A}$'s oracle queries as follows:

- $\mathcal{O}_{\mathsf{ReEncrypt}}\big(i, j, (\hat{\mathsf{ct}}, \mathsf{ct})\big)$: If $j \leq h$, the challenger answers the oracle exactly as in Hyb_4, but instead of setting $\hat{\mathsf{ct}}_{\mathsf{history}} \leftarrow \mathsf{AE.Encrypt}(\hat{\mathsf{k}}'_{\mathsf{history}}, (\mathsf{k}_{\mathsf{ae}}, \hat{\mathsf{k}}_{\mathsf{history}}))$, it sets $\hat{\mathsf{ct}}_{\mathsf{history}} \leftarrow f_{\hat{\mathsf{k}}'_{\mathsf{history}}}((\mathsf{k}_{\mathsf{ae}}, \hat{\mathsf{k}}_{\mathsf{history}}))$, for a random function $f_{\hat{\mathsf{k}}'_{\mathsf{history}}}(\cdot)$. The rest of the simulation in answering $\mathcal{A}$'s queries remains unchanged.
- $\mathcal{O}_{\mathsf{Challenge}}(i, j, \mathsf{m}, (\hat{\mathsf{ct}}, \mathsf{ct}))$: If $j \leq h$, the challenger answers the oracle exactly as in Hyb_4, but instead of setting $\hat{\mathsf{ct}}_{\mathsf{history}} \leftarrow \mathsf{AE.Encrypt}(\hat{\mathsf{k}}'_{\mathsf{history}}, (\mathsf{k}_{\mathsf{ae}}, \hat{\mathsf{k}}_{\mathsf{history}}))$ when $b = 1$, it sets $\hat{\mathsf{ct}}_{\mathsf{history}} \leftarrow f_{\hat{\mathsf{k}}'_{\mathsf{history}}}((\mathsf{k}_{\mathsf{ae}}, \hat{\mathsf{k}}_{\mathsf{history}}))$, for a random function $f_{\hat{\mathsf{k}}'_{\mathsf{history}}}(\cdot)$. The rest of the simulation in answering $\mathcal{A}$'s queries remains unchanged.

In Lemma E.7 below, we show that the hybrid experiments Hyb_4 and Hyb_5 are computationally indistinguishable assuming that $\mathsf{AE.Encrypt}$ has ciphertext pseudorandomness.

- Hyb_6: In this hybrid experiment, we replace the output of $\mathsf{AE.Encrypt}(\mathsf{k}_{\mathsf{ae}}, \cdot)$ in oracle queries made under honest keys with that of a completely random function. Namely, the challenger answers each of $\mathcal{A}$'s oracle queries as follows:
  - $\mathcal{O}_{\mathsf{Encrypt}}(i, \mathsf{m})$: If $i \leq h$, the challenger answers the oracle exactly as in Hyb_5, but instead of setting $\mathsf{ct}_{\mathsf{payload}} \leftarrow \mathsf{AE.Encrypt}(\mathsf{k}_{\mathsf{ae}}, \mathsf{m})$, it sets $\mathsf{ct}_{\mathsf{payload}} \leftarrow f_{\mathsf{k}_{\mathsf{ae}}}(\mathsf{m})$ for a random function $f_{\mathsf{k}_{\mathsf{ae}}}(\cdot)$. The rest of the simulation in answering $\mathcal{A}$'s queries remains unchanged.
  - $\mathcal{O}_{\mathsf{ReEncrypt}}\big(i, j, (\hat{\mathsf{ct}}, \mathsf{ct})\big)$: If $j \leq h$, the challenger answers the oracle exactly as in Hyb_5, but instead of setting $\mathsf{ct}'_{\mathsf{payload}} \leftarrow \mathsf{AE.Encrypt}(\mathsf{k}'_{\mathsf{ae}}, (\mathsf{ct}_{\mathsf{payload}}, \hat{\mathsf{ct}}_{\mathsf{history}}))$, it sets $\mathsf{ct}'_{\mathsf{payload}} \leftarrow f_{\mathsf{k}'_{\mathsf{ae}}}((\mathsf{ct}_{\mathsf{payload}}, \hat{\mathsf{ct}}_{\mathsf{history}}))$ for a random function $f_{\mathsf{k}'_{\mathsf{ae}}}(\cdot)$. The rest of the simulation in answering $\mathcal{A}$'s queries remains unchanged.
  - $\mathcal{O}_{\mathsf{Challenge}}(i, j, \mathsf{m}, (\hat{\mathsf{ct}}, \mathsf{ct}))$: If $j \leq h$, the challenger answers the oracle exactly as in Hyb_5, but instead of setting $\mathsf{ct}'_{\mathsf{payload}} \leftarrow \mathsf{AE.Encrypt}(\mathsf{k}'_{\mathsf{ae}}, (\mathsf{ct}_{\mathsf{payload}}, \hat{\mathsf{ct}}_{\mathsf{history}}))$ when $b = 1$, it sets $\mathsf{ct}'_{\mathsf{payload}} \leftarrow f_{\mathsf{k}'_{\mathsf{ae}}}((\mathsf{ct}_{\mathsf{payload}}, \hat{\mathsf{ct}}_{\mathsf{history}}))$ for a random function $f_{\mathsf{k}'_{\mathsf{ae}}}(\cdot)$. The rest of the simulation in answering $\mathcal{A}$'s queries remains unchanged.

  In Lemma E.9 below, we show that the hybrid experiments Hyb_5 and Hyb_6 are computationally indistinguishable assuming that $\mathsf{AE.Encrypt}$ has ciphertext pseudorandomness.

- Hyb_7: In this hybrid experiment, we modify the challenger from directly encrypting the message m to *re-encrypting* the ciphertext $(\hat{\mathsf{ct}}, \mathsf{ct})$ when answering $\mathcal{A}$'s challenge query $\mathcal{O}_{\mathsf{Challenge}}\big(i, j, \mathsf{m}, (\hat{\mathsf{ct}}, \mathsf{ct})\big)$. Namely, on a query $\mathcal{O}_{\mathsf{Challenge}}\big(i, j, \mathsf{m}, (\hat{\mathsf{ct}}, \mathsf{ct})\big)$, instead of proceeding with $b = 0$, the challenge is handled with $b = 1$. The rest of the experiment remains unchanged from Hyb_6.

  In Lemma E.10 below, we show that the hybrid experiments Hyb_6 and Hyb_7 are perfectly indistinguishable.

- Hyb_8: Starting from this hybrid, we start unrolling back the changes that we made from Hyb_0. In this hybrid experiment, we undo the changes that we made in hyb_6 by replacing the evaluations of random functions $f_{\mathsf{k}_{\mathsf{ae}}}$ and $f_{\mathsf{k}'_{\mathsf{ae}}}$ with evaluations of $\mathsf{AE.Encrypt}(\mathsf{k}_{\mathsf{ae}}, \cdot)$ and $\mathsf{AE.Encrypt}(\mathsf{k}'_{\mathsf{ae}}, \cdot)$ respectively.

  In Lemma E.11 beloew, we show that the hybrid experiments hyb_7 and hyb_8 are computationally indistinguishable assuming that $\mathsf{AE.Encrypt}$ has ciphertext pseudorandomness.

- Hyb_9: In this hybrid experiment, we undo the changes that we made in Hyb_5 by replacing the evaluations of random functions $f_{\hat{\mathsf{k}}'_{\mathsf{history}}}$ with evaluations of $\mathsf{AE.Encrypt}(\hat{\mathsf{k}}'_{\mathsf{history}}, \cdot)$.

  In Lemma E.12 below, we show that the hybrid experiments Hyb_8 and Hyb_9 are computationally indistinguishable assuming that $\mathsf{AE.Encrypt}$ has ciphertext pseudorandomness.

- Hyb_{10}: In this hybrid experiment, we undo the changes that we made in Hyb_4 by replacing the random strings in oracle responses under honest keys with true PRG outputs.

  In Lemma E.13 below, we show that the hybrid experiments Hyb_9 and Hyb_{10} are computationally indistinguishable assuming that $G : \{0,1\}^\lambda \to \{0,1\}^*$ is a secure PRG.

- Hyb_{11}: In this hybrid experiment, we undo the changes that we made in Hyb_3 by including the real contents of the ciphertext headers instead of encryptions of zero.

  In Lemma E.14 below, we show that the hybrid experiments Hyb_{10} and Hyb_{11} are computationally indistinguishable assuming that Π_{AE} satisfies confidentiality.

- Hyb_{12}: In this hybrid experiment, we undo the changes that we made in Hyb_2 by re-introducing the decryption algorithm $\mathsf{AE.Decrypt}$ in the challenger's simulation.

  In Lemma E.15 below, we show that the hybrid experiments Hyb_{11} and Hyb_{12} are perfectly indistinguishable assuming that Π_{AE} is correct.

- Hyb_{13}: In this hybrid experiment, we undo the changes that we made in Hyb_1 by removing the additional abort condition.

  In Lemma E.16 below, we show that the hybrid experiments Hyb_{12} and Hyb_{13} are computationally indistinguishable assuming that Π_{AE} satisfies integrity.

  This hybrid experiment corresponds to the real updatable authenticated encryption update independence experiment $\mathsf{Expt}^{\mathsf{upd\text{-}ind}}_{\Pi_{\mathsf{UAE}}}(\lambda, h, d, \mathcal{A}, 1)$.

We now show that each of the consecutive hybrid experiments are indistinguishable. For a hybrid experiment Hyb and an adversary $\mathcal{A}$, we use $\mathsf{Hyb}(\mathcal{A})$ to denote the random variable that represents the output of experiment Hyb with adversary $\mathcal{A}$.

Lemma E.1. *Suppose that Π_{AE} satisfies $\varepsilon^{\mathsf{int}}_{\mathsf{ae}}$-integrity (Definition 2.9). Then, for all efficient adversaries $\mathcal{A}$, we have*

$$\left|\Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1]\right| \leq h \cdot \varepsilon^{\mathsf{int}}_{\mathsf{ae}}(\lambda).$$

Proof. Let $\mathcal{A}$ be an adversary that distinguishes experiments Hyb_0 and Hyb_1. We construct an algorithm $\mathcal{B}$ that uses $\mathcal{A}$ to break the integrity of Π_{AE} (Definition 2.9). Algorithm $\mathcal{B}$ works as follows:

- **Setup phase**: At the start of the experiment, algorithm $\mathcal{B}$ samples a random index $i^* \xleftarrow{\text{R}} [h]$. It generates the keys k_i for $i \in [h+d] \setminus \{i^*\}$ according to the (identical) specifications of Hyb_0 and Hyb_1. For k_{i^*}, algorithm $\mathcal{B}$ leaves it unspecified.

- **Query phase**: Algorithm $\mathcal{B}$ simulates the responses to $\mathcal{A}$'s oracle queries as follows:

  - $\mathcal{O}_{\mathsf{Encrypt}}(i, \mathsf{m})$: If $i \neq i^*$, algorithm $\mathcal{B}$ proceeds according to Hyb_0. If $i = i^*$, it proceeds according to the specification in Hyb_1. Whenever $\mathcal{B}$ must use k_{i^*} to generate the ciphertext header $\hat{\mathsf{ct}} \leftarrow \mathsf{AE.Encrypt}\big(\hat{\mathsf{k}}_{i^*}, (s, |\mathsf{ct}_{\mathsf{payload}}|, \mathsf{k}_{\mathsf{ae}}, \perp)\big)$, it uses the encryption oracle $\mathcal{O}_{\mathsf{k}^*_i}(\cdot)$ for Π_{AE}.
  - $\mathcal{O}_{\mathsf{ReEncrypt}}\big(i, j, (\hat{\mathsf{ct}}, \mathsf{ct})\big)$: If $i = i^*$, algorithm $\mathcal{B}$ proceeds according to the specification in Hyb_1. Otherwise, it proceeds according to Hyb_0. In both cases, whenever $\mathcal{B}$ must use k_{i^*} to generate the ciphertext header $\hat{\mathsf{ct}}' \leftarrow \mathsf{AE.Encrypt}\big(\hat{\mathsf{k}}_{i^*}, (s', \ell, \mathsf{k}'_{\mathsf{ae}}, \hat{\mathsf{k}}'_{\mathsf{history}})\big)$, it uses the encryption oracle $\mathcal{O}_{\mathsf{k}^*_i}(\cdot)$ for Π_{AE} (Definition 2.9). If $i = i^*$ and $\mathcal{B}$ must abort, it submits $\hat{\mathsf{ct}}$ as a forgery for Π_{AE}.
  - $\mathcal{O}_{\mathsf{Challenge}}(i, j, \mathsf{m}, (\hat{\mathsf{ct}}, \mathsf{ct}))$: If $i = i^*$, algorithm $\mathcal{B}$ proceeds according to the specification in Hyb_1. Otherwise, it proceeds according to Hyb_0. In both cases, whenever $\mathcal{B}$ must use k_{i^*} to generate the ciphertext header $\hat{\mathsf{ct}}' \leftarrow \mathsf{AE.Encrypt}\big(\hat{\mathsf{k}}_{i^*}, (s', \ell, \mathsf{k}'_{\mathsf{ae}}, \hat{\mathsf{k}}'_{\mathsf{history}})\big)$, it uses the encryption oracle $\mathcal{O}_{\mathsf{k}^*_i}(\cdot)$ for Π_{AE} (Definition 2.9). If $i = i^*$ and $\mathcal{B}$ must abort, then it submits $\hat{\mathsf{ct}}$ as a forgery for Π_{AE}.

- **Output phase**: At the end of the experiment, adversary $\mathcal{A}$ outputs a bit $b \in \{0,1\}$, which algorithm $\mathcal{B}$ returns as the output of the experiment.

By definition, the only difference between the hybrid experiments Hyb_0 and Hyb_1 is the additional abort condition when the challenger answers an adversary's re-encryption or challenge queries in Hyb_1. Therefore, by definition, algorithm $\mathcal{B}$ perfectly simulates $\mathcal{A}$'s views of the experiments Hyb_0 and Hyb_1 modulo the abort conditions. Furthermore, by the specification of $\mathcal{B}$, if $\mathcal{A}$ forces $\mathcal{B}$ to abort in any of these queries, then $\mathcal{B}$ successfully forges a new ciphertext for Π_{AE}.

To formally analyze the probability that $\mathcal{B}$ successfully forges a new ciphertext, let us define the following set of random variables:

- Let Z denote the event that $\mathcal{B}$ successfully forges a ciphertext at the end of the simulation above.
- Let X_i for $i \in [h]$ denote the event that $i = i^*$ during $\mathcal{B}$'s simulation above.
- Let Y_i for $i \in [h]$ denote the event that adversary $\mathcal{A}$ forces the challenger to abort in Hyb_1 by submitting a query $\mathcal{O}_{\mathsf{ReEncrypt}}\big(i, j, (\hat{\mathsf{ct}}, \mathsf{ct})\big)$ or $\mathcal{O}_{\mathsf{Challenge}}(i, j, \mathsf{m}, (\hat{\mathsf{ct}}, \mathsf{ct}))$.

Then, by definition, algorithm $\mathcal{B}$ successfully forges a new authenticated encryption ciphertext when $\mathcal{A}$ forces algorithm $\mathcal{B}$ to abort on a query $\mathcal{O}_{\mathsf{ReEncrypt}}\big(i^*, j, (\hat{\mathsf{ct}}, \mathsf{ct})\big)$ or $\mathcal{O}_{\mathsf{Challenge}}(i^*, j, \mathsf{m}, (\hat{\mathsf{ct}}, \mathsf{ct}))$:

$$\begin{aligned}
\Pr\big[Z\big] &= \sum_{i\in[h]} \Pr\big[X_i \cap Y_i\big] \\
&= \sum_{i\in[h]} \Pr\big[X_i \mid Y_i\big] \cdot \Pr\big[Y_i\big] \\
&= \sum_{i\in[h]} \frac{1}{h} \cdot \Pr\big[Y_i\big] \\
&= \frac{1}{h} \sum_{i\in[h]} \Pr\big[Y_i\big] \\
&\geq \frac{1}{h} \cdot \Pr\big[Y_1 \cup \ldots \cup Y_h\big],
\end{aligned}$$

where the last inequality follows by the union bound. Now, since the only difference between the hybrid experiments Hyb_0 and Hyb_1 is the additional abort condition when the challenger answers $\mathcal{O}_{\mathsf{ReEncrypt}}\big(i, j, (\hat{\mathsf{ct}}, \mathsf{ct})\big)$ or $\mathcal{O}_{\mathsf{Challenge}}(i, j, \mathsf{m}, (\hat{\mathsf{ct}}, \mathsf{ct}))$, an adversary's advantage in distinguishing the two experiments is bounded by the probability of the event $Y_1 \cup \ldots \cup Y_h$:

$$\big|\Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1]\big| \leq \Pr\big[Y_1 \cup \ldots \cup Y_h\big].$$

Putting the two inequalities together, we have

$$\begin{aligned}
\big|\Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1]\big| &\leq \Pr\big[Y_1 \cup \ldots \cup Y_h\big] \\
&\leq h \cdot \Pr\big[Z\big] \\
&\leq h \cdot \varepsilon_{\mathsf{ae}}^{\mathsf{int}}(\lambda),
\end{aligned}$$

and the lemma follows. ☐

Lemma E.2. *Suppose that Π_{AE} is correct (Definition 2.7). Then, for all (unbounded) adversaries $\mathcal{A}$, we have*

$$\big|\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1]\big| = 0.$$

Proof. The only difference between the two hybrid experiments is in the way the challenger decrypts the ciphertext headers. For each query to $\mathcal{O}_{\mathsf{ReEncrypt}}\big(i, j, (\hat{\mathsf{ct}}, \mathsf{ct})\big)$ and $\mathcal{O}_{\mathsf{Challenge}}(i, j, \mathsf{m}, (\hat{\mathsf{ct}}, \mathsf{ct}))$, the challenger in Hyb_1 computes $\mu \leftarrow \mathsf{AE.Decrypt}(\mathsf{k}_i, \hat{\mathsf{ct}})$ while the challenger in Hyb_2 sets $\mu \leftarrow \mathsf{T}_{\mathsf{header}}[i, \hat{\mathsf{ct}}]$. The rest of the experiments remain identical.

By the correctness condition for Π_{AE}, these two distributions of μ in the two experiments are identically distributed as long as $(i, \hat{\mathsf{ct}})$ is contained in $\mathsf{T}_{\mathsf{header}}$. However, in both Hyb_1 and Hyb_2, if $\hat{\mathsf{ct}}$ is not contained in $\mathsf{T}_{\mathsf{header}}$, the challenger returns $\perp$. Therefore, the view of $\mathcal{A}$ in the two experiments are identically distributed. ☐

Lemma E.3. *Suppose that* Π_{AE} *satisfies* $\varepsilon_{\mathsf{ae}}^{\mathsf{conf}}$*-confidentiality (Definition 2.8). Then, for all efficient adversaries* $\mathcal{A}$*, we have*

$$\big|\Pr[\mathsf{Hyb}_2(\mathcal{A})=1]-\Pr[\mathsf{Hyb}_3(\mathcal{A})=1]\big| \le h\cdot \varepsilon_{\mathsf{ae}}^{\mathsf{conf}}(\lambda).$$

Proof. To prove the lemma, we proceed via a sequence of inner hybrid experiments. For $\gamma = 0,\dots,h$, we define the hybrid experiments $\mathsf{Hyb}_{2,\gamma}$ as follows:

- $\mathsf{Hyb}_{2,\gamma}$: The challenger proceeds through the setup phase of the experiment according to the specifications in Hyb_2 and Hyb_3 (which are identical). The challenger answers each of $\mathcal{A}$'s queries during the query phase of the experiment as follows:
  - $\mathcal{O}_{\mathsf{Encrypt}}(i,\mathsf{m})$: If $i>\gamma$, then the challenger proceeds as in Hyb_2. Otherwise, the challenger proceeds as in Hyb_3.
  - $\mathcal{O}_{\mathsf{ReEncrypt}}\big(i,j,(\hat{\mathsf{ct}},\mathsf{ct})\big)$: If $i>\gamma$, then the challenger proceeds as in Hyb_2. Otherwise, the challenge proceeds as in Hyb_3.
  - $\mathcal{O}_{\mathsf{Challenge}}(i,j,\mathsf{m},(\hat{\mathsf{ct}},\mathsf{ct}))$: If $i>\gamma$, then the challenger proceeds as in Hyb_2. Otherwise, the challenger proceeds as in Hyb_3.

  At the end of the experiment, adversary $\mathcal{A}$ outputs a bit b, which the challenger returns as the output of the experiment.

By definition, experiment $\mathsf{Hyb}_{2,0}$ corresponds to experiment Hyb_2, and experiment $\mathsf{Hyb}_{2,h}$ correponds to experiment Hyb_3. To prove the lemma, we show that each consecutive hybrid experiments $\mathsf{Hyb}_{2,\gamma-1}$ and $\mathsf{Hyb}_{2,\gamma}$ for $\gamma=1,\dots,h$ are computationally indistinguishable.

Claim E.4. *Suppose that* Π_{AE} *satisfies* $\varepsilon_{\mathsf{ae}}^{\mathsf{conf}}$*-confidentiality. Then, for all* $\gamma\in[h]$ *and all efficient adversaries* $\mathcal{A}$*, we have*

$$\big|\Pr[\mathsf{Hyb}_{2,\gamma-1}(\mathcal{A})=1]-\Pr[\mathsf{Hyb}_{2,\gamma}(\mathcal{A})]\big| \le \varepsilon_{\mathsf{ae}}^{\mathsf{conf}}(\lambda).$$

Proof. Let $\mathcal{A}$ be an adversary that distinguishes experiments $\mathsf{Hyb}_{2,\gamma-1}$ and $\mathsf{Hyb}_{2,\gamma}$. We construct an algorithm $\mathcal{B}$ that uses $\mathcal{A}$ to break the confidentiality of Π_{AE}. Algorithm $\mathcal{B}$ works as follows:

- **Setup phase**: For the setup phase, algorithm $\mathcal{B}$ proceeds according to the specifications in $\mathsf{Hyb}_{2,\gamma-1}$ and $\mathsf{Hyb}_{2,\gamma}$ (which are identical). However, for the key k_γ, it leaves it unspecified.
- **Query phase**: Algorithm $\mathcal{B}$ simulates the responses to $\mathcal{A}$'s oracle queries as follows:
  - $\mathcal{O}_{\mathsf{Encrypt}}(i,\mathsf{m})$: Algorithm $\mathcal{B}$ follows the exact specification of the two experiments. However, since k_γ is unspecified, it uses the encryption oracle

    $$\mathcal{O}_{\mathsf{k}_\gamma,b}\big((s,|\mathsf{ct}_{\mathsf{payload}}|,\mathsf{k}_{\mathsf{ae}},\perp),(0^{|s|},0^{\mathsf{intlen}},0^{|\mathsf{k}_{\mathsf{ae}}|},0^{\lambda})\big),$$

    in place of $\mathsf{AE.Encrypt}\big(\mathsf{k}_\gamma,(s,|\mathsf{ct}_{\mathsf{payload}}|,\mathsf{k}_{\mathsf{ae}},\perp)\big)$ or $\mathsf{AE.Encrypt}\big(\mathsf{k}_\gamma,(0^{|s|},0^{\mathsf{intlen}},0^{|\mathsf{k}_{\mathsf{ae}}|},0^{\lambda})\big)$.
  - $\mathcal{O}_{\mathsf{ReEncrypt}}\big(i,j,(\hat{\mathsf{ct}},\mathsf{ct})\big)$: Algorithm $\mathcal{B}$ follows the exact specification of the two experiments. However, since k_γ is unspecified, it uses the encryption oracle

    $$\mathcal{O}_{\mathsf{k}_\gamma,b}\big((s',\ell,\mathsf{k}'_{\mathsf{ae}},\hat{\mathsf{k}}'_{\mathsf{history}}),(\hat{\mathsf{k}}_j,(0^{|s'|},0^{\mathsf{intlen}},0^{|\mathsf{k}'_{\mathsf{ae}}|},0^{|\hat{\mathsf{k}}'_{\mathsf{history}}|})\big),$$

    in place of $\mathsf{AE.Encrypt}\big(\mathsf{k}_\gamma,(s',\ell,\mathsf{k}'_{\mathsf{ae}},\hat{\mathsf{k}}'_{\mathsf{history}})\big)$ or $\mathsf{AE.Encrypt}\big(\mathsf{k}_\gamma,(0^{|s'|},0^{\mathsf{intlen}},0^{|\mathsf{k}'_{\mathsf{ae}}|},0^{|\hat{\mathsf{k}}'_{\mathsf{history}}|})\big)$.
  - $\mathcal{O}_{\mathsf{Challenge}}(i,j,\mathsf{m},(\hat{\mathsf{ct}},\mathsf{ct}))$: Algorithm $\mathcal{B}$ follows the exact specification of the two experiments. However, since k_γ is unspecified, it uses the encryption oracle $\mathcal{O}_{\mathsf{k}_\gamma,b}$ instead of $\mathsf{AE.Encrypt}(\mathsf{k}_\gamma,\cdot)$ as described above for encryption and re-encryption.
- **Output phase**: At the end of the experiment, adversary $\mathcal{A}$ outputs a bit $b\in\{0,1\}$, which $\mathcal{B}$ returns as its own output.

By specification, algorithm $\mathcal{B}$ perfectly simulates the experiments $\mathsf{Hyb}_{2,\gamma}$ and $\mathsf{Hyb}_{2,\gamma-1}$ as long as the output of the oracle $\mathcal{O}_{\mathsf{k}_\gamma,b}(\cdot,\cdot)$ is consistent with the specifications of the two experiments. By specification (Definition 2.8), we have

$$\mathcal{O}_{\mathsf{k}_\gamma,0}\big((s,|\mathsf{ct}_{\mathsf{payload}}|,\mathsf{k}_{\mathsf{ae}},\perp),(0^{|s|},0^{\mathsf{intlen}},0^{|\mathsf{k}_{\mathsf{ae}}|},0^\lambda)\big)=\mathsf{AE.Encrypt}\big(\mathsf{k}_\gamma,(s,|\mathsf{ct}_{\mathsf{payload}}|,\mathsf{k}_{\mathsf{ae}},\perp)\big),$$

and

$$\mathcal{O}_{\mathsf{k}_\gamma,1}\big((s,|\mathsf{ct}_{\mathsf{payload}}|,\mathsf{k}_{\mathsf{ae}},\perp),(0^{|s|},0^{\mathsf{intlen}},0^{|\mathsf{k}_{\mathsf{ae}}|},0^\lambda)\big)=\mathsf{AE.Encrypt}\big(\mathsf{k}_\gamma,(0^{|s|},0^{\mathsf{intlen}},0^{|\mathsf{k}_{\mathsf{ae}}|},0^\lambda)\big).$$

This means that if $\mathcal{B}$ is interacting with the oracle $\mathcal{O}_{\mathsf{k}_\gamma,0}$, then it perfectly simulates $\mathsf{Hyb}_{2,\gamma-1}$, and if it is interacting with the oracle $\mathcal{O}_{\mathsf{k}_\gamma,1}$, then it perfectly simulates $\mathsf{Hyb}_{2,\gamma}$. Therefore, with the same distinguishing advantage of the two experiments by $\mathcal{A}$, algorithm $\mathcal{B}$ breaks the confidentiality of Π_{AE}. The claim now follows. ☐

The statement of the lemma now follows from Claim E.4 and the triangle inequality. ☐

Lemma E.5. *Suppose that $G:\{0,1\}^\lambda\to\{0,1\}^*$ satisfies $\varepsilon_{\mathsf{prg}}$ PRG security (Definition 2.2). Then, for all efficient adversaries $\mathcal{A}$ that make at most Q oracle queries, we have*

$$\big|\Pr[\mathsf{Hyb}_3(\mathcal{A})=1]-\Pr[\mathsf{Hyb}_4(\mathcal{A})=1]\big|\le Q\cdot\varepsilon_{\mathsf{prg}}(\lambda).$$

Proof. To prove the lemma, we proceed via a sequence of inner hybrid experiments. For $\gamma=0,\ldots,Q$, we define the hybrid experiments $\mathsf{Hyb}_{3,\gamma}$ as follows:

- $\mathsf{Hyb}_{3,\gamma}$: The challenger proceeds through the setup phase of the experiment according to the specifications in Hyb_3 and Hyb_4 (which are identical). The challenger numbers $\mathcal{A}$'s oracle queries and answers each of $\mathcal{A}$'s kth query during the query phase of the experiment as follows:
  - $\mathcal{O}_{\mathsf{Encrypt}}(i,\mathsf{m})$: If $k>\gamma$, then the challenger proceeds as in Hyb_3. Otherwise, the challenger proceeds as in Hyb_4.
  - $\mathcal{O}_{\mathsf{ReEncrypt}}\big(i,j,(\hat{\mathsf{ct}},\mathsf{ct})\big)$: If $k>\gamma$, then the challenger proceeds as in Hyb_3. Otherwise, the challenge proceeds as in Hyb_4.
  - $\mathcal{O}_{\mathsf{Challenge}}(i,j,\mathsf{m},(\hat{\mathsf{ct}},\mathsf{ct}))$: If $k>\gamma$, then the challenger proceeds as in Hyb_3. Otherwise, the challenger proceeds as in Hyb_4.

  At the end of the experiment, adversary $\mathcal{A}$ outputs a bit b, which the challenger returns as the output of the experiment.

By definition, experiment $\mathsf{Hyb}_{3,0}$ corresponds to experiment Hyb_3, and experiment $\mathsf{Hyb}_{3,Q}$ correponds to experiment Hyb_3. To prove the lemma, we show that each consecutive hybrid experiments $\mathsf{Hyb}_{3,\gamma-1}$ and $\mathsf{Hyb}_{3,\gamma}$ for $\gamma=1,\ldots,Q$ are computationally indistinguishable.

Claim E.6. *Suppose that $G:\{0,1\}^\lambda\to\{0,1\}^*$ satisfies $\varepsilon_{\mathsf{prg}}$ PRG security (Definition 2.2). Then, for all $\gamma\in[Q]$ and all efficient adversaries $\mathcal{A}$, we have*

$$\big|\Pr[\mathsf{Hyb}_{3,\gamma-1}(\mathcal{A})=1]-\Pr[\mathsf{Hyb}_{3,\gamma}(\mathcal{A})]\big|\le\varepsilon_{\mathsf{prg}}(\lambda).$$

Proof. Let $\mathcal{A}$ be an adversary that distinguishes experiments $\mathsf{Hyb}_{3,\gamma-1}$ and $\mathsf{Hyb}_{3,\gamma}$. We construct an algorithm $\mathcal{B}$ that uses $\mathcal{A}$ to break the PRG security of G. Algorithm $\mathcal{B}$ proceeds according to the specification of $\mathsf{Hyb}_{3,\gamma-1}$, except in the γth call to G, if it is in an execution of $\mathcal{O}_{\mathsf{Encrypt}}(i,\mathsf{m})$ where $i\le h$ or of $\mathcal{O}_{\mathsf{ReEncrypt}}\big(i,j,(\hat{\mathsf{ct}},\mathsf{ct})\big)$ or $\mathcal{O}_{\mathsf{Challenge}}(i,j,\mathsf{m},(\hat{\mathsf{ct}},\mathsf{ct}))$ where $j\le h$, it passes on its input instead of evaluating G. At the end of the experiment, $\mathcal{B}$ passes on $\mathcal{A}$'s output as its own output.

Since the seed on which G is evaluated by the challenger does not appear in the view of the adversary $\mathcal{A}$, $\mathcal{B}$ provides a perfect simulation of $\mathsf{Hyb}_{3,\gamma-1}$ if its input is a PRG evaluation and a perfect simulation of $\mathsf{Hyb}_{3,\gamma}$ if its input is a truly random string. Therefore, with the same distinguishing advantage of the two experiments by $\mathcal{A}$, algorithm $\mathcal{B}$ breaks the PRG security of G. The claim now follows. ☐

The statement of the lemma now follows from Claim E.6 and the triangle inequality. □

Lemma E.7. *Suppose that* $\mathsf{AE.Encrypt}$ *satisfies* $\varepsilon_{\mathsf{ae}}^{\mathsf{rand}}$ *ciphertext pseudorandomness (Definition 2.10). Then, for all efficient adversaries* $\mathcal{A}$ *that make at most* Q *oracle queries, we have*

$$\left|\Pr[\mathsf{Hyb}_4(\mathcal{A})=1]-\Pr[\mathsf{Hyb}_5(\mathcal{A})=1]\right| \leq Q\cdot\varepsilon_{\mathsf{ae}}^{\mathsf{rand}}(\lambda).$$

Proof. To prove the lemma, we proceed via a sequence of inner hybrid experiments. For $\gamma = 0,\ldots,Q$, we define the hybrid experiments $\mathsf{Hyb}_{4,\gamma}$ as follows:

- $\mathsf{Hyb}_{4,\gamma}$: The challenger proceeds through the setup phase of the experiment according to the specifications in Hyb_4 and Hyb_5 (which are identical). The challenger numbers $\mathcal{A}$'s oracle queries, *starting from* Q *and counting backwards*, and answers $\mathcal{A}$'s kth last query during the query phase of the experiment as follows:
  - $\mathcal{O}_{\mathsf{Encrypt}}(i,\mathsf{m})$: If $k > \gamma$, then the challenger proceeds as in Hyb_4. Otherwise, the challenger proceeds as in Hyb_5.
  - $\mathcal{O}_{\mathsf{ReEncrypt}}\big(i,j,(\hat{\mathsf{ct}},\mathsf{ct})\big)$: If $k > \gamma$, then the challenger proceeds as in Hyb_4. Otherwise, the challenge proceeds as in Hyb_5.
  - $\mathcal{O}_{\mathsf{Challenge}}(i,j,\mathsf{m},(\hat{\mathsf{ct}},\mathsf{ct}))$: If $k > \gamma$, then the challenger proceeds as in Hyb_4. Otherwise, the challenger proceeds as in Hyb_5.

  At the end of the experiment, adversary $\mathcal{A}$ outputs a bit b, which the challenger returns as the output of the experiment.

By definition, experiment $\mathsf{Hyb}_{4,0}$ corresponds to experiment Hyb_4, and experiment $\mathsf{Hyb}_{4,Q}$ correponds to experiment Hyb_5. To prove the lemma, we show that each consecutive hybrid experiments $\mathsf{Hyb}_{4,\gamma-1}$ and $\mathsf{Hyb}_{4,\gamma}$ for $\gamma = 1,\ldots,Q$ are computationally indistinguishable.

Claim E.8. *Suppose that* $\mathsf{AE.Encrypt}$ *satisfies* $\varepsilon_{\mathsf{ae}}^{\mathsf{rand}}$ *ciphertext pseudorandomness (Definition 2.10). Then, for all* $\gamma \in [Q]$ *and all efficient adversaries* $\mathcal{A}$*, we have*

$$\left|\Pr[\mathsf{Hyb}_{4,\gamma-1}(\mathcal{A})=1]-\Pr[\mathsf{Hyb}_{4,\gamma}(\mathcal{A})]\right| \leq \varepsilon_{\mathsf{ae}}^{\mathsf{rand}}(\lambda).$$

Proof. Let $\mathcal{A}$ be an adversary that distinguishes experiments $\mathsf{Hyb}_{4,\gamma-1}$ and $\mathsf{Hyb}_{4,\gamma}$. We construct an algorithm $\mathcal{B}$ that uses $\mathcal{A}$ to break the ciphertext pseudorandomness of $\mathsf{AE.Encrypt}$. Algorithm $\mathcal{B}$ answers each of $\mathcal{A}$'s oracle queries exactly as the challenger in $\mathsf{Hyb}_{4,\gamma}$, except it answers the γth query (starting counting queries from Q and counting downward) as follows:

- $\mathcal{O}_{\mathsf{Encrypt}}(i,\mathsf{m})$: This query is handled identically to challenger $\mathsf{Hyb}_{4,\gamma}$.
- $\mathcal{O}_{\mathsf{ReEncrypt}}\big(i,j,(\hat{\mathsf{ct}},\mathsf{ct})\big)$: If $j \leq h$, $\mathcal{B}$ answers the oracle query exactly as in $\mathsf{Hyb}_{4,\gamma}$, but it sets the value $\hat{\mathsf{ct}}_{\mathsf{history}} \leftarrow F((\mathsf{k}_{\mathsf{ae}},\hat{\mathsf{k}}_{\mathsf{history}}))$ where $F(\cdot)$ is the oracle provided to $\mathcal{B}$ in the ciphertext pseudorandomness experiment. The rest of the simulation in answering $\mathcal{A}$'s queries remains unchanged.
- $\mathcal{O}_{\mathsf{Challenge}}(i,j,\mathsf{m},(\hat{\mathsf{ct}},\mathsf{ct}))$: If $j \leq h$, $\mathcal{B}$ answers the oracle query exactly as in $\mathsf{Hyb}_{4,\gamma}$, but it sets the value $\hat{\mathsf{ct}}_{\mathsf{history}} \leftarrow F((\mathsf{k}_{\mathsf{ae}},\hat{\mathsf{k}}_{\mathsf{history}}))$ where $F(\cdot)$ is the oracle provided to $\mathcal{B}$ in the ciphertext pseudorandomness experiment. The rest of the simulation in answering $\mathcal{A}$'s queries remains unchanged.

Apart from the changes described above, $\mathcal{B}$ simulates the challenger of $\mathsf{Hyb}_{4,\gamma}$ exactly. At the end of the experiment, adversary $\mathcal{A}$ outputs a bit $b \in \{0,1\}$, which $\mathcal{B}$ returns as its own output.

By specification, algorithm $\mathcal{B}$ perfectly simulates the experiment $\mathsf{Hyb}_{4,\gamma}$ or $\mathsf{Hyb}_{4,\gamma-1}$ depending on whether the output of the oracle $F(\cdot)$ corresponds to evaluation of $\mathsf{AE.Encrypt}(\cdot,\cdot)$ or a randomly chosen function $f(\cdot)$. This is the case because if $j \leq h$, the key $\hat{\mathsf{k}}'_{\mathsf{history}}$ does not appear in the adversary $\mathcal{A}$'s view: it was erased from the ciphertext header in Hyb_3, and the value of $\hat{\mathsf{ct}}_{\mathsf{history}}$ that is based on it was replaced with the output of a random function in a previous subhybrid $\mathsf{Hyb}_{4,\gamma'}$ (this is why we begin counting queries from the last query instead of the first). If $j \geq h$, the hybrids $\mathsf{Hyb}_{4,\gamma}$ and $\mathsf{Hyb}_{4,\gamma-1}$ are defined identically. Therefore, with the same distinguishing advantage of the two experiments by $\mathcal{A}$, algorithm $\mathcal{B}$ breaks the ciphertext pseudorandomness of $\mathsf{AE.Encrypt}$. The claim now follows. □

The statement of the lemma now follows from Claim E.8 and the triangle inequality. □

Lemma E.9. *Suppose that* $\mathsf{AE.Encrypt}$ *satisfies* $\varepsilon_{\mathsf{ae}}^{\mathsf{rand}}$ *ciphertext pseudorandomness (Definition 2.10). Then, for all efficient adversaries* $\mathcal{A}$ *that make at most* Q *oracle queries, we have*

$$\left|\Pr[\mathsf{Hyb}_5(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_6(\mathcal{A}) = 1]\right| \leq Q \cdot \varepsilon_{\mathsf{ae}}^{\mathsf{rand}}(\lambda).$$

Proof. This proof is analagous to the proof of Lemma E.7, so we omit it. □

Lemma E.10. *For all (unbounded) adversaries* $\mathcal{A}$*, we have*

$$\left|\Pr[\mathsf{Hyb}_6(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_7(\mathcal{A}) = 1]\right| = 0.$$

Proof. The only difference between the two hybrid experiments is in the way the challenger responds to $\mathcal{A}$'s challenge oracle queries to $\mathcal{O}_{\mathsf{Challenge}}$. Namely, to generate a ciphertext body ct' on a query $\mathcal{O}_{\mathsf{Challenge}}\big(i, j, \mathsf{m}, (\hat{\mathsf{ct}}, \mathsf{ct})\big)$, the challenger in Hyb_6 computes $\mathsf{ct}'_{\mathsf{payload}} \leftarrow f_{\mathsf{k}_{\mathsf{ae}}}(\mathsf{m})$ and $\mathsf{ct}'_{\mathsf{pad}} \xleftarrow{\mathrm{R}} \{0,1\}^{t\cdot(2\rho+\nu)}$, and then it sets $\mathsf{ct}' \leftarrow (\mathsf{ct}'_{\mathsf{payload}}, \mathsf{ct}'_{\mathsf{pad}})$. The challenger in Hyb_7 computes $\Delta_{i,j,\hat{\mathsf{ct}}} \leftarrow \mathsf{ReKeyGen}(\mathsf{k}_i, \mathsf{k}_j, \hat{\mathsf{ct}})$, $\mathsf{ct}'_{\mathsf{payload}} \leftarrow f_{\mathsf{k}'_{\mathsf{ae}}}(\mathsf{ct}_{\mathsf{payload}}, \hat{\mathsf{ct}}_{\mathsf{history}})$, and $\mathsf{ct}'_{\mathsf{pad}} \xleftarrow{\mathrm{R}} \{0,1\}^{|\mathsf{ct}|-|\mathsf{ct}'_{\mathsf{payload}}|}$ and then sets $\mathsf{ct}' \leftarrow (\mathsf{ct}'_{\mathsf{payload}}, \mathsf{ct}'_{\mathsf{pad}})$. However, since $f_{\mathsf{k}_{\mathsf{ae}}}(\cdot)$ and $f_{\mathsf{k}'_{\mathsf{ae}}}(\cdot)$ are completely random functions, these two distributions of the ciphertext body components are identically distributed independent of m or ct as long as the resulting ciphertexts have the same length. By specification, the challenger returns ct' only when this is the case. Therefore, the view of $\mathcal{A}$ in Hyb_6 and Hyb_7 are identically distributed and the lemma follows. □

Lemma E.11. *Suppose that* $\mathsf{AE.Encrypt}$ *satisfies* $\varepsilon_{\mathsf{ae}}^{\mathsf{rand}}$ *ciphertext pseudorandomness (Definition 2.10). Then, for all efficient adversaries* $\mathcal{A}$ *that make at most* Q *oracle queries, we have*

$$\left|\Pr[\mathsf{Hyb}_7(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_8(\mathcal{A}) = 1]\right| \leq Q \cdot \varepsilon_{\mathsf{ae}}^{\mathsf{rand}}(\lambda).$$

Proof. The proof is identical to the proof of Lemma E.9. □

Lemma E.12. *Suppose that* $\mathsf{AE.Encrypt}$ *satisfies* $\varepsilon_{\mathsf{ae}}^{\mathsf{rand}}$ *ciphertext pseudorandomness (Definition 2.10). Then, for all efficient adversaries* $\mathcal{A}$ *that make at most* Q *oracle queries, we have*

$$\left|\Pr[\mathsf{Hyb}_8(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_9(\mathcal{A}) = 1]\right| \leq Q \cdot \varepsilon_{\mathsf{ae}}^{\mathsf{rand}}(\lambda).$$

Proof. The proof is identical to the proof of Lemma E.7. □

Lemma E.13. *Suppose that* $G : \{0,1\}^\lambda \to \{0,1\}^*$ *satisfies* $\varepsilon_{\mathsf{prg}}$ *PRG security (Definition 2.2). Then, for all efficient adversaries* $\mathcal{A}$ *that make at most* Q *oracle queries, we have*

$$\left|\Pr[\mathsf{Hyb}_9(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{10}(\mathcal{A}) = 1]\right| \leq Q \cdot \varepsilon_{\mathsf{prg}}(\lambda).$$

Proof. The proof is identical to the proof of Lemma E.5. □

Lemma E.14. *Suppose that* Π_{AE} *satisfies* $\varepsilon_{\mathsf{ae}}^{\mathsf{conf}}$*-confidentiality (Definition 2.8). Then, for all efficient adversaries* $\mathcal{A}$*, we have*

$$\left|\Pr[\mathsf{Hyb}_{10}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{11}(\mathcal{A}) = 1]\right| \leq h \cdot \varepsilon_{\mathsf{ae}}^{\mathsf{conf}}(\lambda).$$

Proof. The proof is identical to the proof of Lemma E.3. □

Lemma E.15. *Suppose that Π_{AE} is correct (Definition 2.7). Then, for all (unbounded) adversaries $\mathcal{A}$, we have*

$$\big|\Pr[\mathsf{Hyb}_{11}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{12}(\mathcal{A}) = 1]\big| = 0.$$

Proof. The proof is identical to the proof of Lemma E.2. □

Lemma E.16. *Suppose that Π_{AE} satisfies $\varepsilon^{\mathsf{int}}_{\mathsf{ae}}$-integrity (Definition 2.9). Then, for all efficient adversaries $\mathcal{A}$, we have*

$$\big|\Pr[\mathsf{Hyb}_{12}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{13}(\mathcal{A}) = 1]\big| \leq h \cdot \varepsilon^{\mathsf{int}}_{\mathsf{ae}}(\lambda).$$

Proof. The proof is identical to the proof of Lemma E.1. □

By combining the lemmas above and using the triangle inequality, the proof of update independence follows.

Message confidentiality and re-encryption indistinguishability. The proofs of message confidentiality and re-encryption indistinguishability are almost identical to the proof of update independence. In addition to modifying the first and the final hybrid experiments, we can adapt the proof of update independence as follows:

- The proofs of message confidentiality and re-encryption indistinguishability do not include hybrids Hyb_4 and Hyb_{10}, which rely on the security of the PRG G. The definitions of message confidentiality and re-encryption indistinguishability (Definition A.1 and Definition A.2) do not require hiding whether a ciphertext is a fresh encryption or a re-encryption.
- The proofs of message confidentiality and re-encryption indistinguishability rely on the confidentiality of Π_{AE} instead of ciphertext pseudorandomness in the hybrid experiments Hyb_5, Hyb_6, Hyb_8 and Hyb_9. The definitions of message confidentiality and re-encryption indistinguishability do not hide whether a ciphertext is a fresh encryption or a re-encryption, so the content of an encryption need not appear random to hide where the encryption ends and a PRG output begins.
- The definitions of message confidentiality and re-encryption indistinguishability include an additional $\mathsf{ReKeyGen}$ oracle $\mathcal{O}_{\mathsf{ReKeyGen}}$, so the the proofs of message confidentiality and re-encryption indistinguishability must also discuss this oracle. Changes made to this oracle in all hybrids are identical to changes made to the $\mathcal{O}_{\mathsf{ReEncrypt}}$ oracle in the proof above.

E.4 Proof of Integrity

We proceed via a sequence of hybrid experiments that are defined as follows:

- Hyb_0: This hybrid experiment corresponds to the real updatable authenticated encryption update independence experiment $\mathsf{Expt}^{\mathsf{int}}_{\Pi_{\mathsf{UAE}}}(\lambda, h, d, \mathcal{A})$ that is instantiated with Construction 4.2.
- Hyb_1: In this hybrid experiment, we introduce an additional abort condition to the challenger's simulation. Namely, throughout the query phase of the experiment, the challenger maintains an additional look-up table $\mathsf{T}_{\mathsf{header}}$ that keeps track of all of the "well-formed" ciphertext headers (under honest keys) that $\mathcal{A}$ receives from the challenger throughout the experiment. The table is initially set to be empty, and the challenger answers each of $\mathcal{A}$'s oracle queries as follows:
  - $\mathcal{O}_{\mathsf{Encrypt}}(i, \mathsf{m})$: The challenger answers the oracle exactly as in Hyb_0 by generating the ciphertext header
  $$\hat{\mathsf{ct}} \leftarrow \mathsf{AE.Encrypt}\big(\hat{\mathsf{k}}_i, (s, |\mathsf{ct}_{\mathsf{payload}}|, \mathsf{k}_{\mathsf{ae}}, \perp)\big),$$
  and the ciphertext body as specified in construction 4.2. After returning the ciphertext $(\hat{\mathsf{ct}}, \mathsf{ct})$ to $\mathcal{A}$, it additionally adds the mapping $\mathsf{T}_{\mathsf{header}}[i, \hat{\mathsf{ct}}] \leftarrow (s, |\mathsf{ct}_{\mathsf{payload}}|, \mathsf{k}_{\mathsf{ae}}, \perp)$ to the table if $i \leq h$.

- $\mathcal{O}_{\mathsf{ReKeyGen}}(i, j, \hat{\mathsf{ct}})$: If $i \leq h$, then the challenger first checks if $\mathsf{T}_{\mathsf{header}}[i, \hat{\mathsf{ct}}] = \perp$ and outputs $\perp$ if this is the case. If $\mathsf{T}_{\mathsf{header}}[i, \hat{\mathsf{ct}}] \neq \perp$ or $i > h$, then the challenger proceeds exactly as in Hyb_0, by setting $\hat{\mathsf{ct}}' \leftarrow \mathsf{AE.Encrypt}(\hat{\mathsf{k}}_j, (s', l, \mathsf{k}'_{\mathsf{ae}}, \hat{\mathsf{k}}'_{\mathsf{history}}))$, computing $\Delta_{i,j,\hat{\mathsf{ct}}}$, and returning it to $\mathcal{A}$. After returning $\Delta_{i,j,\hat{\mathsf{ct}}}$ to $\mathcal{A}$, it adds the mapping $\mathsf{T}_{\mathsf{header}}[j, \hat{\mathsf{ct}}'] \leftarrow (s', l, \mathsf{k}'_{\mathsf{ae}}, \hat{\mathsf{k}}'_{\mathsf{history}})$ to the table.
- $\mathcal{O}_{\mathsf{ReEncrypt}}(i, j, (\hat{\mathsf{ct}}, \mathsf{ct}))$: If $i \leq h$, then the challenger first checks if $\mathsf{T}_{\mathsf{header}}[i, \hat{\mathsf{ct}}] = \perp$ and output $\perp$ if this is the case. If $\mathsf{T}_{\mathsf{header}}[i, \hat{\mathsf{ct}}] \neq \perp$ or $i > h$, then the challenger proceeds exactly as in Hyb_0 by setting $\hat{\mathsf{ct}}' \leftarrow \mathsf{AE.Encrypt}(\hat{\mathsf{k}}_j, (s', l, \mathsf{k}'_{\mathsf{ae}}, \hat{\mathsf{k}}'_{\mathsf{history}}))$, computing $\Delta_{i,j,\hat{\mathsf{ct}}}$, and returning $\mathsf{ReEncrypt}(\Delta_{i,j,\hat{\mathsf{ct}}}, (\hat{\mathsf{ct}}, \mathsf{ct}))$ to $\mathcal{A}$. After returning the updated ciphertext to $\mathcal{A}$, it adds the mapping $\mathsf{T}_{\mathsf{header}}[j, \hat{\mathsf{ct}}'] \leftarrow (s', l, \mathsf{k}'_{\mathsf{ae}}, \hat{\mathsf{k}}'_{\mathsf{history}})$ to the table.

At the end of the experiment, adversary $\mathcal{A}$ outputs an index $i \leq h$ and a ciphertext $(\hat{\mathsf{ct}}, \mathsf{ct})$. As specified in Hyb_0, the challenger checks the following conditions, outputting $\perp$ if either are met:

- $\mathsf{m} = \perp$,
- $\mathsf{T}[i, \hat{\mathsf{ct}}] = \mathsf{ct}$.

In addition, when invoking the decryption algorithm $\mathsf{Decrypt}$ above, the challenger verifies whether $\mathsf{T}_{\mathsf{header}}[i, \hat{\mathsf{ct}}] = \perp$. If this is the case, then the challenger also outputs $\perp$. If none of the conditions are met, then the challenger returns 1 as the output of the experiment.

During the output phase of the experiment, when the challenger is decrypting the purported forged ciphertext $(\hat{\mathsf{ct}}, \mathsf{ct})$, when the challenger computes $\mu \leftarrow \mathsf{AE.Decrypt}(\hat{\mathsf{k}}, \hat{\mathsf{ct}})$ (which occurs in step 1 of decryption), it also checks the table $\mathsf{T}_{\mathsf{header}}$ and sets $\mu' \leftarrow \mathsf{T}_{\mathsf{header}}[i, \hat{\mathsf{ct}}]$. If $\mu \neq \mu'$, the challenger aborts the experiment and outputs $\perp$.

The rest of the experiment remains unchanged from Hyb_0.

In Lemma E.17 below, we show that the hybrid experiments Hyb_0 and Hyb_1 are computationally indistinguishable assuming that Π_{AE} satisfies integrity.

- Hyb_2: In this hybrid experiment, we erase the decryption algorithm $\mathsf{AE.Decrypt}$ from the challenger's simulation for honest keys in ciphertext headers. Namely, the challenger answers $\mathcal{A}$'s $\mathsf{ReKeyGen}$ and $\mathsf{ReEncrypt}$ oracle queries as follows when $i \leq h$:

  - $\mathcal{O}_{\mathsf{ReKeyGen}}(i, j, \hat{\mathsf{ct}})$: The challenger answers the oracle exactly as in Hyb_1, but instead of decrypting the ciphertext header $\mu \leftarrow \mathsf{AE.Decrypt}(\mathsf{k}_i, \hat{\mathsf{ct}})$, it sets $\mu \leftarrow \mathsf{T}_{\mathsf{header}}[i, \hat{\mathsf{ct}}]$. If no such entry exists in $\mathsf{T}_{\mathsf{header}}$, then it immediately aborts the experiment and outputs $\perp$.
  - $\mathcal{O}_{\mathsf{ReEncrypt}}(i, j, (\hat{\mathsf{ct}}, \mathsf{ct}))$: The challenger answers the oracle exactly as in Hyb_1, but instead of decrypting the ciphertext header $\mu \leftarrow \mathsf{AE.Decrypt}(\mathsf{k}_i, \hat{\mathsf{ct}})$, it sets $\mu \leftarrow \mathsf{T}_{\mathsf{header}}[i, \hat{\mathsf{ct}}]$. If no such entry exists in $\mathsf{T}_{\mathsf{header}}$, then it immediately aborts the experiment and outputs $\perp$.

  In the output phase, the challenger answers the oracle exactly as in Hyb_1, but instead of decrypting the ciphertext header $\mu \leftarrow \mathsf{AE.Decrypt}(\mathsf{k}_i, \hat{\mathsf{ct}})$, it sets $\mu \leftarrow \mathsf{T}_{\mathsf{header}}[i, \hat{\mathsf{ct}}]$. If no such entry exists in $\mathsf{T}_{\mathsf{header}}$, then it immediately aborts the experiment and outputs $\perp$.

  The rest of the experiment remains unchanged from Hyb_1.

  In Lemma E.18 below, we show that the hybrid experiments Hyb_1 and Hyb_2 are perfectly indistinguishable assuming that Π_{AE} is correct.

- Hyb_3: In this hybrid experiment, we erase the contents of ciphertext headers for honest keys. Namely, the challenger answers each of $\mathcal{A}$'s oracle queries as follows, where intlen represents the number of bits required to represent an integer:

– $\mathcal{O}_{\mathsf{Encrypt}}(i, \mathsf{m})$: If $i \leq h$, the challenger answers the oracle query exactly as in Hyb_2, but it sets the ciphertext header $\hat{\mathsf{ct}}$ to be

$$\hat{\mathsf{ct}} \leftarrow \mathsf{AE.Encrypt}\big(\hat{\mathsf{k}}_i, (0^{|s|}, 0^{\mathsf{intlen}}, 0^{|\mathsf{k}_{\mathsf{ae}}|}, 0^{\lambda})\big).$$

The rest of the simulation in answering $\mathcal{A}$'s queries remain unchanged.

– $\mathcal{O}_{\mathsf{ReKeyGen}}(i, j, \hat{\mathsf{ct}})$: If $j \leq h$, the challenger answers the oracle query exactly as in Hyb_2, but it sets the ciphertext header $\hat{\mathsf{ct}}'$ to be

$$\hat{\mathsf{ct}}' \leftarrow \mathsf{AE.Encrypt}\big(\hat{\mathsf{k}}_j, (0^{|s'|}, 0^{\mathsf{intlen}}, 0^{|\mathsf{k}'_{\mathsf{ae}}|}, 0^{|\hat{\mathsf{k}}'_{\mathsf{history}}|})\big).$$

The rest of the simulation in answering $\mathcal{A}$'s queries remain unchanged.

– $\mathcal{O}_{\mathsf{ReEncrypt}}(i, j, (\hat{\mathsf{ct}}, \mathsf{ct}))$: If $j \leq h$, the challenger answers the oracle query exactly as in Hyb_2, but it sets the ciphertext header $\hat{\mathsf{ct}}'$ to be

$$\hat{\mathsf{ct}}' \leftarrow \mathsf{AE.Encrypt}\big(\hat{\mathsf{k}}_j, (0^{|s'|}, 0^{\mathsf{intlen}}, 0^{|\mathsf{k}'_{\mathsf{ae}}|}, 0^{|\hat{\mathsf{k}}'_{\mathsf{history}}|})\big).$$

The rest of the simulation in answering $\mathcal{A}$'s queries remain unchanged.

The rest of the experiment remains unchanged from Hyb_2.

In Lemma E.19 below, we show that the hybrid experiments Hyb_2 and Hyb_3 are computationally indistinguishable assuming that Π_{AE} satisfies confidentiality.

- Hyb_4: In this hybrid we erase the contents of $\hat{\mathsf{ct}}_{\mathsf{history}}$ ciphertexts produced under honest keys. Namely, the challenger answers each of $\mathcal{A}$'s $\mathsf{ReKeyGen}$ and $\mathsf{ReEncrypt}$ oracle queries as follows:

  – $\mathcal{O}_{\mathsf{ReKeyGen}}(i, j, \hat{\mathsf{ct}})$: If $j \leq h$, the challenger answers the oracle exactly as in Hyb_3, but instead of setting $\hat{\mathsf{ct}}_{\mathsf{history}} \leftarrow \mathsf{AE.Encrypt}(\hat{\mathsf{k}}'_{\mathsf{history}}, (\mathsf{k}_{\mathsf{ae}}, \hat{\mathsf{k}}_{\mathsf{history}}))$, it sets $\hat{\mathsf{ct}}_{\mathsf{history}}$ to be

  $$\hat{\mathsf{ct}}_{\mathsf{history}} \leftarrow \mathsf{AE.Encrypt}(\hat{\mathsf{k}}'_{\mathsf{history}}, (0^{|\mathsf{k}_{\mathsf{ae}}|}, 0^{|\hat{\mathsf{k}}_{\mathsf{history}}|})).$$

  The rest of the simulation in answering $\mathcal{A}$'s queries remains unchanged.

  – $\mathcal{O}_{\mathsf{ReEncrypt}}(i, j, (\hat{\mathsf{ct}}, \mathsf{ct}))$: If $j < h$, the challenger answers the oracle exactly as in Hyb_3, but instead of setting $\hat{\mathsf{ct}}_{\mathsf{history}} \leftarrow \mathsf{AE.Encrypt}(\hat{\mathsf{k}}'_{\mathsf{history}}, (\mathsf{k}_{\mathsf{ae}}, \hat{\mathsf{k}}_{\mathsf{history}}))$, it sets $\hat{\mathsf{ct}}_{\mathsf{history}}$ to be

  $$\hat{\mathsf{ct}}_{\mathsf{history}} \leftarrow \mathsf{AE.Encrypt}(\hat{\mathsf{k}}'_{\mathsf{history}}, (0^{|\mathsf{k}_{\mathsf{ae}}|}, 0^{|\hat{\mathsf{k}}_{\mathsf{history}}|})).$$

  The rest of the simulation in answering $\mathcal{A}$'s queries remains unchanged.

  In Lemma E.20 below, we show that the hybrid experiments Hyb_3 and Hyb_4 are computationally indistinguishable assuming that Π_{AE} satisfies confidentiality.

- Hyb_5: In this hybrid experiment, we introduce an additional abort condition to the challenger's simulation. Namely, throughout the query phase of the experiment, the challenger maintains an additional look-up table $\mathsf{T}_{\mathsf{payload}}$ that keeps track of all of the "well-formed" payload ciphertexts $\mathsf{ct}_{\mathsf{payload}}$ under honest keys that $\mathcal{A}$ receives from the challenger throughout the experiment. The table is initially set to be empty, and the challenger answers $\mathcal{A}$'s $\mathsf{Encrypt}$ and $\mathsf{ReEncrypt}$ oracle queries as follows:

  – $\mathcal{O}_{\mathsf{Encrypt}}(i, \mathsf{m})$: The challenger proceeds exactly as in Hyb_4 and sets $\mathsf{ct}_{\mathsf{payload}} \leftarrow \mathsf{AE.Encrypt}(\mathsf{k}_{\mathsf{ae}}, \mathsf{m})$. However, it also adds the mapping $\mathsf{T}_{\mathsf{payload}}[\mathsf{k}_{\mathsf{ae}}, \mathsf{ct}_{\mathsf{payload}}] \leftarrow \mathsf{m}$ to the table.

  – $\mathcal{O}_{\mathsf{ReEncrypt}}(i, j, (\hat{\mathsf{ct}}, \mathsf{ct}))$: The challenger proceeds exactly as in Hyb_4 and sets $\mathsf{ct}'_{\mathsf{payload}} \leftarrow \mathsf{AE.Encrypt}\big(\mathsf{k}'_{\mathsf{ae}}, (\mathsf{ct}_{\mathsf{payload}}, \hat{\mathsf{ct}}_{\mathsf{history}})\big)$. However, it also adds the mapping $\mathsf{T}_{\mathsf{payload}}[\mathsf{k}'_{\mathsf{ae}}, \mathsf{ct}'_{\mathsf{payload}}] \leftarrow (\mathsf{ct}_{\mathsf{payload}}, \hat{\mathsf{ct}}_{\mathsf{history}})$ to the table.

During the output phase of the experiment, when the challenger is decrypting the purported forged ciphertext $(\hat{\mathsf{ct}}, \mathsf{ct})$, each time the challenger computes $\mu \leftarrow \mathsf{AE.Decrypt}(\mathsf{k}'_{\mathsf{ae}}, \mathsf{ct}_{\mathsf{payload}})$ or $\mu \leftarrow \mathsf{AE.Decrypt}(\mathsf{k}_{\mathsf{ae}}, \mathsf{ct})$ (which occurs in steps 4 and 6 of decryption), it also checks the table $\mathsf{T}_{\mathsf{payload}}$ and sets $\mu' \leftarrow \mathsf{T}_{\mathsf{payload}}[\mathsf{k}'_{\mathsf{ae}}, \mathsf{ct}_{\mathsf{payload}}]$ or $\mu' \leftarrow \mathsf{T}_{\mathsf{payload}}[\mathsf{k}_{\mathsf{ae}}, \mathsf{ct}]$ respectively. If $\mu \neq \mu'$, the challenger aborts the experiment and outputs $\perp$. This check is skipped if there is no entry in $\mathsf{T}_{\mathsf{payload}}[\mathsf{k}'_{\mathsf{ae}}, \cdot]$ or $\mathsf{T}_{\mathsf{payload}}[\mathsf{k}_{\mathsf{ae}}, \cdot]$ respectively, i.e., if the payload key in question was never used by the challenger. The rest of the experiment remains unchanged from Hyb_4.

In Lemma E.22 below, we show that the hybrid experiments Hyb_4 and Hyb_5 are computationally indistinguishable assuming that Π_{AE} satisfies integrity.

Finally, in Lemma E.23 below, we show that an adversary has negligible advantage in forcing the challenger to output 1 in Hyb_5.

We now show that each of the consecutive hybrid experiments are indistinguishable. For a hybrid experiment Hyb and an adversary $\mathcal{A}$, we use $\mathsf{Hyb}(\mathcal{A})$ to denote the random variable that represents the output of experiment Hyb with adversary $\mathcal{A}$.

Lemma E.17. *Suppose that Π_{AE} satisfies $\varepsilon_{\mathsf{ae}}^{\mathsf{int}}$-integrity (Definition 2.9). Then, for all efficient adversaries $\mathcal{A}$, we have*

$$\left|\Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1]\right| \leq h \cdot \varepsilon_{\mathsf{ae}}^{\mathsf{int}}(\lambda).$$

Proof. This proof is almost identical to that of Lemma E.1, so we omit the full proof. □

Lemma E.18. *Suppose that Π_{AE} is correct (Definition 2.7). Then, for all (unbounded) adversaries $\mathcal{A}$, we have*

$$\left|\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1]\right| = 0.$$

Proof. The only difference between the two hybrid experiments is in the way the challenger decrypts the ciphertext headers. For each query to $\mathcal{O}_{\mathsf{ReKeyGen}}(i, j, \hat{\mathsf{ct}})$ or $\mathcal{O}_{\mathsf{ReEncrypt}}(i, j, (\hat{\mathsf{ct}}, \mathsf{ct}))$, as well as in the output phase, the challenger in Hyb_1 computes $\mu \leftarrow \mathsf{AE.Decrypt}(\mathsf{k}_i, \hat{\mathsf{ct}})$ while the challenger in Hyb_2 sets $\mu \leftarrow \mathsf{T}_{\mathsf{header}}[i, \hat{\mathsf{ct}}]$. The rest of the experiments remains identical.

By the correctness condition for Π_{AE}, these two distributions of μ in the two experiments are identically distributed as long as $(i, \hat{\mathsf{ct}})$ is contained in $\mathsf{T}_{\mathsf{header}}$. However, in both Hyb_1 and Hyb_2, if $\hat{\mathsf{ct}}$ is not contained in $\mathsf{T}_{\mathsf{header}}$, the challenger returns $\perp$. Therefore, the view of $\mathcal{A}$ in the two experiments are identically distributed. □

Lemma E.19. *Suppose that Π_{AE} satisfies $\varepsilon_{\mathsf{ae}}^{\mathsf{conf}}$-confidentiality (Definition 2.8). Then, for all efficient adversaries $\mathcal{A}$, we have*

$$\left|\Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1]\right| \leq h \cdot \varepsilon_{\mathsf{ae}}^{\mathsf{conf}}(\lambda).$$

Proof. This proof is almost identical to that of Lemma E.3, so we omit the full proof. □

Lemma E.20. *Suppose that Π_{AE} satisfies $\varepsilon_{\mathsf{ae}}^{\mathsf{conf}}$-confidentiality (Definition 2.8). Then, for all efficient adversaries $\mathcal{A}$ that make at most Q oracle queries, we have*

$$\left|\Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_4(\mathcal{A}) = 1]\right| \leq Q \cdot \varepsilon_{\mathsf{ae}}^{\mathsf{conf}}(\lambda).$$

Proof. To prove the lemma, we proceed via a sequence of inner hybrid experiments. For $\gamma = 0, \ldots, Q$, we define the hybrid experiments $\mathsf{Hyb}_{3,\gamma}$ as follows:

- $\mathsf{Hyb}_{3,\gamma}$: The challenger proceeds through the setup phase of the experiment according to the specifications in Hyb_3 and Hyb_4 (which are identical). The challenger numbers $\mathcal{A}$'s oracle queries, *starting from Q and counting backwards*, and answers $\mathcal{A}$'s kth last query during the query phase of the experiment as follows:

– $\mathcal{O}_{\mathsf{Encrypt}}(i, \mathsf{m})$: If $k > \gamma$, then the challenger proceeds as in Hyb_3. Otherwise, the challenger proceeds as in Hyb_4.

– $\mathcal{O}_{\mathsf{ReKeyGen}}(i, j, \hat{\mathsf{ct}})$: If $k > \gamma$, then the challenger proceeds as in Hyb_3. Otherwise, the challenger proceeds as in Hyb_4.

– $\mathcal{O}_{\mathsf{ReEncrypt}}(i, j, (\hat{\mathsf{ct}}, \mathsf{ct}))$: If $k > \gamma$, then the challenger proceeds as in Hyb_3. Otherwise, the challenge proceeds as in Hyb_4.

At the end of the experiment, adversary $\mathcal{A}$ outputs a bit b, which the challenger returns as the output of the experiment.

By definition, experiment $\mathsf{Hyb}_{3,0}$ corresponds to experiment Hyb_3, and experiment $\mathsf{Hyb}_{3,Q}$ correponds to experiment Hyb_4. To prove the lemma, we show that each consecutive hybrid experiments $\mathsf{Hyb}_{3,\gamma-1}$ and $\mathsf{Hyb}_{3,\gamma}$ for $\gamma = 1, \ldots, Q$ are computationally indistinguishable.

Claim E.21. *Suppose that Π_{AE} satisfies $\varepsilon_{\mathsf{ae}}^{\mathsf{conf}}$-confidentiality. Then, for all $\gamma \in [h]$ and all efficient adversaries $\mathcal{A}$, we have*

$$\big| \Pr[\mathsf{Hyb}_{3,\gamma-1}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{3,\gamma}(\mathcal{A})] \big| \leq \varepsilon_{\mathsf{ae}}^{\mathsf{conf}}(\lambda).$$

Proof. Let $\mathcal{A}$ be an adversary that distinguishes experiments $\mathsf{Hyb}_{3,\gamma-1}$ and $\mathsf{Hyb}_{3,\gamma}$. We construct an algorithm $\mathcal{B}$ that uses $\mathcal{A}$ to break the confidentiality of Π_{AE}. Algorithm $\mathcal{B}$ answers each of $\mathcal{A}$'s oracle queries exactly as the challenger in $\mathsf{Hyb}_{3,\gamma}$, except it answers the γth query (starting couting queries from Q and counting downwards) as follows:

- $\mathcal{O}_{\mathsf{Encrypt}}(i, \mathsf{m})$: This query is handled identically to challenger $\mathsf{Hyb}_{3,\gamma}$.
- $\mathcal{O}_{\mathsf{ReKeyGen}}(i, j, \hat{\mathsf{ct}})$: If $j \leq h$, $\mathcal{B}$ answers the oracle query exactly as in $\mathsf{Hyb}_{3,\gamma}$, but it sets

$$\hat{\mathsf{ct}}_{\mathsf{history}} \leftarrow \mathcal{O}_{k_\gamma,b}((\mathsf{k}_{\mathsf{ae}}, \hat{\mathsf{k}}_{\mathsf{history}}), (0^{|\mathsf{k}_{\mathsf{ae}}|}, 0^{|\hat{\mathsf{k}}_{\mathsf{history}}|}))$$

  instead of setting

$$\hat{\mathsf{ct}}_{\mathsf{history}} \leftarrow \mathsf{AE.Encrypt}(\hat{\mathsf{k}}'_{\mathsf{history}}, (\mathsf{k}_{\mathsf{ae}}, \hat{\mathsf{k}}_{\mathsf{history}}))$$

  or

$$\hat{\mathsf{ct}}_{\mathsf{history}} \leftarrow \mathsf{AE.Encrypt}(\hat{\mathsf{k}}'_{\mathsf{history}}, (0^{|\mathsf{k}_{\mathsf{ae}}|}, 0^{|\hat{\mathsf{k}}_{\mathsf{history}}|})).$$

  The rest of the simulation in answering $\mathcal{A}$'s queries remains unchanged.
- $\mathcal{O}_{\mathsf{ReEncrypt}}(i, j, (\hat{\mathsf{ct}}, \mathsf{ct}))$: If $j \leq h$, $\mathcal{B}$ answers the oracle query exactly as in $\mathsf{Hyb}_{3,\gamma}$, but it sets

$$\hat{\mathsf{ct}}_{\mathsf{history}} \leftarrow \mathcal{O}_{k_\gamma,b}((\mathsf{k}_{\mathsf{ae}}, \hat{\mathsf{k}}_{\mathsf{history}}), (0^{|\mathsf{k}_{\mathsf{ae}}|}, 0^{|\hat{\mathsf{k}}_{\mathsf{history}}|}))$$

  instead of setting

$$\hat{\mathsf{ct}}_{\mathsf{history}} \leftarrow \mathsf{AE.Encrypt}(\hat{\mathsf{k}}'_{\mathsf{history}}, (\mathsf{k}_{\mathsf{ae}}, \hat{\mathsf{k}}_{\mathsf{history}}))$$

  or

$$\hat{\mathsf{ct}}_{\mathsf{history}} \leftarrow \mathsf{AE.Encrypt}(\hat{\mathsf{k}}'_{\mathsf{history}}, (0^{|\mathsf{k}_{\mathsf{ae}}|}, 0^{|\hat{\mathsf{k}}_{\mathsf{history}}|})).$$

  The rest of the simulation in answering $\mathcal{A}$'s queries remains unchanged.

Apart from the changes described above, $\mathcal{B}$ simulates the challenger of $\mathsf{Hyb}_{3,\gamma}$ exactly. At the end of the experiment, adversary $\mathcal{A}$ outputs a bit $b \in \{0, 1\}$, which $\mathcal{B}$ returns as its own output.

By specification, algorithm $\mathcal{B}$ perfectly simulates the experiments $\mathsf{Hyb}_{3,\gamma}$ and $\mathsf{Hyb}_{3,\gamma-1}$ as long as the output of the oracle $\mathcal{O}_{\mathsf{k}_\gamma,b}(\cdot,\cdot)$ is consistent with the specifications of the two experiments. By specification (Definition 2.8), we have

$$\mathcal{O}_{k_\gamma,0}((\mathsf{k}_{\mathsf{ae}}, \hat{\mathsf{k}}_{\mathsf{history}}), (0^{|\mathsf{k}_{\mathsf{ae}}|}, 0^{|\hat{\mathsf{k}}_{\mathsf{history}}|})) = \hat{\mathsf{ct}}_{\mathsf{history}} \leftarrow \mathsf{AE.Encrypt}(\hat{\mathsf{k}}'_{\mathsf{history}}, (\mathsf{k}_{\mathsf{ae}}, \hat{\mathsf{k}}_{\mathsf{history}})),$$

and

$$\mathcal{O}_{k_\gamma,1}((\mathsf{k}_{\mathsf{ae}}, \hat{\mathsf{k}}_{\mathsf{history}}), (0^{|\mathsf{k}_{\mathsf{ae}}|}, 0^{|\hat{\mathsf{k}}_{\mathsf{history}}|})) = \mathsf{AE.Encrypt}(\hat{\mathsf{k}}'_{\mathsf{history}}, (0^{|\mathsf{k}_{\mathsf{ae}}|}, 0^{|\hat{\mathsf{k}}_{\mathsf{history}}|})).$$

This means that if $\mathcal{B}$ is interacting with the oracle $\mathcal{O}_{\mathsf{k}_\gamma,0}$, then it perfectly simulates $\mathsf{Hyb}_{3,\gamma-1}$, and if it is interacting with the oracle $\mathcal{O}_{\mathsf{k}_\gamma,1}$, then it perfectly simulates $\mathsf{Hyb}_{3,\gamma}$. This is the case because if $j \leq h$, the key $\hat{\mathsf{k}}'_{\mathsf{history}}$ does not appear in the adversary $\mathcal{A}$'s view: it was erased from the ciphertext header in Hyb_3, and it was erased from $\hat{\mathsf{ct}}_{\mathsf{history}}$ in a previous subhybrid $\mathsf{Hyb}_{3,\gamma'}$ (this is why we begin counting queries from the last query instead of the first). If $j \geq h$, the hybrids $\mathsf{Hyb}_{3,\gamma}$ and $\mathsf{Hyb}_{3,\gamma-1}$ are defined identically. Therefore, with the same distinguishing advantage of the two experiments by $\mathcal{A}$, algorithm $\mathcal{B}$ breaks the confidentiality of Π_{AE}. The claim now follows. ☐

The statement of the lemma now follows from Claim E.21 and the triangle inequality. ☐

Lemma E.22. *Suppose that Π_{AE} satisfies $\varepsilon_{\mathsf{ae}}^{\mathsf{int}}$-integrity (Definition 2.9). Then, for all efficient adversaries $\mathcal{A}$ that make at most Q oracle queries, we have*

$$\left|\Pr[\mathsf{Hyb}_4(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_5(\mathcal{A}) = 1]\right| \leq Q \cdot \varepsilon_{\mathsf{ae}}^{\mathsf{int}}(\lambda).$$

Proof. Let $\mathcal{A}$ be an adversary that distinguishes experiments Hyb_4 and Hyb_5. We construct an algorithm $\mathcal{B}$ that uses $\mathcal{A}$ to break the integrity of Π_{AE} (Definition 2.9). Algorithm $\mathcal{B}$ proceeds through the setup phase of the experiment according to the specifications in Hyb_4 and Hyb_5 (which are identical). Then it samples a random index $c^* \xleftarrow{\text{R}} [Q]$. The challenger counts $\mathcal{A}$'s oracle queries and answers each of $\mathcal{A}$'s cth query during the query phase of the experiment as follows:

- $\mathcal{O}_{\mathsf{Encrypt}}(i, \mathsf{m})$: If $c = c^*$, algorithm $\mathcal{B}$ proceeds according to the specification in Hyb_5. Otherwise, it proceeds according to Hyb_4. In both cases, whenever $\mathcal{B}$ must use c^*th value of k_{ae} to generate the ciphertext $\mathsf{ct}_{\mathsf{payload}} \leftarrow \mathsf{AE.Encrypt}(\mathsf{k}_{\mathsf{ae}}, \mathsf{m})$, it uses the encryption oracle $\mathcal{O}_{\mathsf{k}_{c^*}}(\cdot)$ for Π_{AE} (Definition 2.9). If $c = c^*$ and $\mathcal{B}$ must abort, it submits the c^*th value of $\hat{\mathsf{ct}}_{\mathsf{history}}$ as a forgery for Π_{AE}.
- $\mathcal{O}_{\mathsf{ReKeyGen}}\big(i, j, \hat{\mathsf{ct}}\big)$: If $c = c^*$, algorithm $\mathcal{B}$ proceeds according to the specification in Hyb_5. Otherwise, it proceeds according to Hyb_4. In both cases, whenever $\mathcal{B}$ must use c^*th value of $\mathsf{k}'_{\mathsf{ae}}$ to generate the ciphertext $\mathsf{ct}'_{\mathsf{payload}} \leftarrow \mathsf{AE.Encrypt}(\mathsf{k}'_{\mathsf{ae}}, (\mathsf{ct}_{\mathsf{payload}}, \hat{\mathsf{ct}}_{\mathsf{history}}))$, it uses the encryption oracle $\mathcal{O}_{\mathsf{k}_{c^*}}(\cdot)$ for Π_{AE} (Definition 2.9). If $c = c^*$ and $\mathcal{B}$ must abort, it submits the c^*th value of $\hat{\mathsf{ct}}_{\mathsf{history}}$ as a forgery for Π_{AE}.
- $\mathcal{O}_{\mathsf{ReEncrypt}}\big(i, j, (\hat{\mathsf{ct}}, \mathsf{ct})\big)$: If $c = c^*$, algorithm $\mathcal{B}$ proceeds according to the specification in Hyb_5. Otherwise, it proceeds according to Hyb_4. In both cases, whenever $\mathcal{B}$ must use c^*th value of $\mathsf{k}'_{\mathsf{ae}}$ to generate the ciphertext $\mathsf{ct}'_{\mathsf{payload}} \leftarrow \mathsf{AE.Encrypt}(\mathsf{k}'_{\mathsf{ae}}, (\mathsf{ct}_{\mathsf{payload}}, \hat{\mathsf{ct}}_{\mathsf{history}}))$, it uses the encryption oracle $\mathcal{O}_{\mathsf{k}_{c^*}}(\cdot)$ for Π_{AE} (Definition 2.9). If $c = c^*$ and $\mathcal{B}$ must abort, it submits the c^*th value of $\hat{\mathsf{ct}}_{\mathsf{history}}$ as a forgery for Π_{AE}.

At the end of the experiment, adversary $\mathcal{A}$ outputs a bit $b \in \{0, 1\}$, which algorithm $\mathcal{B}$ returns as the output of the experiment.

By definition, the only difference between the hybrid experiments Hyb_4 and Hyb_5 is the additional abort condition when the challenger answers an adversary's oracle queries in Hyb_5. Therefore, by definition, algorithm $\mathcal{B}$ perfectly simulates $\mathcal{A}$'s views of the experiments Hyb_4 and Hyb_5 modulo the abort conditions. Furthermore, by the specification of $\mathcal{B}$, if $\mathcal{A}$ forces $\mathcal{B}$ to abort in any of these queries, then $\mathcal{B}$ successfully forges a new ciphertext for Π_{AE}.

To formally analyze the probability that $\mathcal{B}$ successfully forges a new ciphertext, let us define the following set of random variables:

- Let Z denote the event that $\mathcal{B}$ successfully forges a ciphertext at the end of the simulation above.
- Let X_c for $c \in [Q]$ denote the event that $c = c^*$ during $\mathcal{B}$'s simulation above.
- Let Y_c for $c \in [Q]$ denote the event that adversary $\mathcal{A}$ forces the challenger to abort in Hyb_5 by submitting a query $\mathcal{O}_{\mathsf{Encrypt}}(i, \mathsf{m})$, $\mathcal{O}_{\mathsf{ReKeyGen}}\big(i, j, \hat{\mathsf{ct}}\big)$, or $\mathcal{O}_{\mathsf{ReEncrypt}}\big(i, j, (\hat{\mathsf{ct}}, \mathsf{ct})\big)$.

The lemma now follows from the identical argument used to prove Lemma E.1.

□

Lemma E.23. *For all (unbounded) adversaries $\mathcal{A}$, we have*

$$\big|\Pr[\mathsf{Hyb}_5(\mathcal{A}) = 1]\big| \le Q/2^\lambda.$$

Proof. We now prove that the adversary in Hyb_5 has an at most negligible probability of forging a UAE ciphertext $(\hat{\mathsf{ct}}, \mathsf{ct})$ under a key k_i of its choosing, $i \le h$. For the sake of contradiction, suppose that $\mathcal{A}$ successfully produces such a forgery.

First, recall that if the experiment completes without aborting, then $\hat{\mathsf{ct}}$ must appear in $\mathsf{T}_{\mathsf{header}}$. Thus we can recover $(s', \ell, \mathsf{k}'_{\mathsf{ae}}, \hat{\mathsf{k}}'_{\mathsf{history}}) \leftarrow \mathsf{T}_{\mathsf{header}}[i, \hat{\mathsf{ct}}]$, the plaintext of $\hat{\mathsf{ct}}$. If $G(s') \ne \mathsf{ct}_{\mathsf{pad}}$, then decryption would fail (in step 3), so it must be that $G(s') = \mathsf{ct}_{\mathsf{pad}}$. This means we can strip off $\mathsf{ct}_{\mathsf{pad}}$ from ct to recover $\mathsf{ct}_{\mathsf{payload}}$.

Now consider $\mu \leftarrow \mathsf{T}_{\mathsf{payload}}[\mathsf{k}'_{\mathsf{ae}}, \mathsf{ct}_{\mathsf{payload}}]$. Since decryption completes without aborting, this entry must exist. Observe that since $\mathsf{k}'_{\mathsf{ae}}$ contains λ bits of entropy, the probability of $\mathsf{T}_{\mathsf{payload}}$ containing two or more entries of the form $\mathsf{T}_{\mathsf{payload}}[\mathsf{k}'_{\mathsf{ae}}, \cdot]$ is at most $Q/2^\lambda$. We will only consider the case where there is only one such entry. By construction, $\mathsf{T}_{\mathsf{payload}}[\mathsf{k}'_{\mathsf{ae}}, \mathsf{ct}_{\mathsf{payload}}]$ must be added to $\mathsf{T}_{\mathsf{payload}}$ in the same oracle call that $\mathsf{T}_{\mathsf{header}}[i, \hat{\mathsf{ct}}]$ is added to $\mathsf{T}_{\mathsf{header}}$. This means that, in the same oracle call, the entry $\mathsf{T}[i, \hat{\mathsf{ct}}] \leftarrow \mathsf{ct}$ was added to T, for $\mathsf{ct} = (\mathsf{ct}_{\mathsf{payload}}, \mathsf{ct}_{\mathsf{pad}})$. But this is a contradiction because $(\hat{\mathsf{ct}}, \mathsf{ct})$ cannot be a forgery for k_i if it appears in the table. Thus a successful forgery in Hyb_5 leads to a contradiction with probability at least $1 - Q/2^\lambda$, completing the proof. □

By combining the lemmas above and using the triangle inequality, the proof of integrity follows.

F Proof of Theorem 5.3

F.1 Proof of Strong Compactness

Header compactness. Fix the security parameter λ, any message $\mathsf{m} \in \mathcal{M}_\lambda$, and let $\mathsf{k}_1, \mathsf{k}_2 \leftarrow \mathsf{KeyGen}(1^\lambda)$, $(\hat{\mathsf{ct}}, \mathsf{ct}) \leftarrow \mathsf{Encrypt}(\mathsf{k}_1, \mathsf{m})$, and $\Delta_{1,2,\hat{\mathsf{ct}}} \leftarrow \mathsf{ReKeyGen}(\mathsf{k}_1, \mathsf{k}_2, \hat{\mathsf{ct}})$. By construction, the ciphertext header $\hat{\mathsf{ct}}$ and the re-encryption key $\Delta_{1,2,\hat{\mathsf{ct}}}$ has the following form:

- *Ciphertext header*: By the specification of $\mathsf{Encrypt}$, we have $\hat{\mathsf{ct}} = \mathsf{AE.Encrypt}\big(\mathsf{k}_1, (\mathsf{k}_{\mathsf{prf}}, \mathsf{h})\big)$ where $\mathsf{k}_{\mathsf{prf}} \in \mathcal{K}_{\mathsf{PRF}}$ and $\mathsf{h} = H(\mathsf{m})$. The PRF key space $\mathcal{K}_{\mathsf{PRF}}$ depends only on the security parameter, and by definition of $\mathcal{H}$, we have $|\mathsf{h}| = \lambda$. Therefore, we have $|\hat{\mathsf{ct}}| = \mathsf{poly}_1(\lambda)$ for some polynomial poly_1.

- *Re-encryption key*: The re-encryption key $\Delta_{1,2,\hat{\mathsf{ct}}}$ consists of a new ciphertext header $\hat{\mathsf{ct}}'$ and a PRF key $\mathsf{k}^{\mathsf{up}}_{\mathsf{prf}}$. By the specification of $\mathsf{ReEncrypt}$, we have $\hat{\mathsf{ct}}' = \big(\mathsf{k}_2, (\mathsf{k}'_{\mathsf{prf}}, \mathsf{h})\big)$ where $\mathsf{k}'_{\mathsf{prf}} \in \mathcal{K}_{\mathsf{PRF}}$ and $\mathsf{h} = H(\mathsf{m})$, and therefore, we have $|\hat{\mathsf{ct}}'| = \mathsf{poly}'_2(\lambda)$ for some polynomial poly'_2 as above. Furthermore, by specification, $\mathsf{k}^{\mathsf{up}}_{\mathsf{prf}} \xleftarrow{\mathrm{R}} \mathcal{K}_{\mathsf{PRF}}$ and therefore, $|\mathsf{k}^{\mathsf{up}}_{\mathsf{prf}}| = \mathsf{poly}''_2(\lambda)$ for some polynomial poly''_2. This shows that $|\Delta_{1,2,\hat{\mathsf{ct}}}| = \mathsf{poly}'_2(\lambda) + \mathsf{poly}''_2(\lambda) = \mathsf{poly}_2(\lambda)$ for some polynomial poly_2.

Compactness now follows.

Body compactness. Fix the security parameter λ, number of updates $N \in \mathbb{N}$, and any message $\mathsf{m} \in \mathcal{M}_\lambda$. Let $\mathsf{k}_1, \ldots, \mathsf{k}_N \leftarrow \mathsf{KeyGen}(1^\lambda)$, $(\hat{\mathsf{ct}}_1, \mathsf{ct}_1) \leftarrow \mathsf{Encrypt}(\mathsf{k}_1, \mathsf{m})$, 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 $i = 1, \ldots, N-1$. By construction, the ciphertext body ct_i has the form $\mathsf{ct}_i = (\mathsf{ct}_1, ..., \mathsf{ct}_\ell)$ where $\ell \in \mathbb{N}$ is the size of the output, in blocks, of $(\mathsf{m}_1, ..., \mathsf{m}_\ell) \leftarrow \mathsf{Pad}(\mathsf{m})$, which depends only on m. Therefore, we have $|\mathsf{ct}_i| = \mathsf{poly}(\mathsf{m})$ for some polynomial poly, completing the proof of body compactness.

F.2 Proof of Correctness

Fix the security parameter $\lambda \in \mathbb{N}$, number of updates $N = \mathsf{poly}(\lambda)$, and any message $\mathsf{m} \in \mathcal{M}_\lambda$. Let $\mathsf{k}_1, \ldots, \mathsf{k}_N \leftarrow \mathsf{KeyGen}(1^\lambda)$, $(\hat{\mathsf{ct}}_1, \mathsf{ct}_1) \leftarrow \mathsf{Encrypt}(\mathsf{k}_1, \mathsf{m})$, 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 must show that

$$\mathsf{Decrypt}\big(\mathsf{k}_N, (\hat{\mathsf{ct}}_N, \mathsf{ct}_N)\big) = \mathsf{m}.$$

To do this, we show that $(\hat{\mathsf{ct}}_i, \mathsf{ct}_i)$ for $i \in [N]$ satisfies:

- $\hat{\mathsf{ct}}_i = \mathsf{AE.Encrypt}\big(\mathsf{k}, (\mathsf{k}_{\mathsf{prf}}, \mathsf{h})\big)$ for a PRF key $\mathsf{k}_{\mathsf{prf}} \in \mathcal{K}_{\mathsf{PRF}}$ and $\mathsf{h} = H(\mathsf{m})$.
- $\mathsf{ct}_i = (\mathsf{ct}_{i,1}, \ldots, \mathsf{ct}_{i,\ell})$ where each ciphertext block $\mathsf{ct}_{i,j}$ for $j \in [\ell]$ has the form

  $$\mathsf{ct}_{i,j} \leftarrow \mathsf{m}_j + F(\mathsf{k}_{\mathsf{prf},i}, j) + e_{i,j},$$

  for some $|e_{i,j}| \leq i \cdot \beta$ and $(\mathsf{m}_1, \ldots, \mathsf{m}_\ell) \leftarrow \mathsf{Pad}(\mathsf{m})$.

This implies that $\mathsf{Decrypt}\big(\mathsf{k}_N, (\hat{\mathsf{ct}}_N, \mathsf{ct}_N)\big) = \mathsf{m}$ by the specification of $\mathsf{Decrypt}$ and the correctness of the encoding scheme.

The show that $(\hat{\mathsf{ct}}_i, \mathsf{ct}_i)$ for $i \in [N]$ has the form as above, we proceed via induction.

- The ciphertext $(\hat{\mathsf{ct}}_1, \mathsf{ct}_1)$ has the form above simply by the specification of the $\mathsf{Encrypt}$ algorithm.
- Now assuming that $(\hat{\mathsf{ct}}_{i-1}, \mathsf{ct}_{i-1})$ for some $i \in [N]$ has the form above, let us consider $(\hat{\mathsf{ct}}_i, \mathsf{ct}_i)$. We know that $(\hat{\mathsf{ct}}_{i-1}, \mathsf{ct}_{i-1}$ satisfies
  - $\hat{\mathsf{ct}}_{i-1} = \mathsf{AE.Encrypt}\big(\mathsf{k}_{i-1}, (\mathsf{k}_{\mathsf{prf},i-1}, \mathsf{h})\big)$ for a PRF key $\mathsf{k}_{\mathsf{prf},i-1} \in \mathcal{K}_{\mathsf{PRF}}$ and $\mathsf{h} = H(\mathsf{m})$.
  - $\mathsf{ct}_{i-1} = (\mathsf{ct}_{i-1,1}, \ldots, \mathsf{ct}_{i-1,\ell})$ where each ciphertext block for $j \in [\ell]$ has the form

    $$\mathsf{ct}_{i-1,j} \leftarrow \mathsf{m}_{i-1,j} + F(\mathsf{k}_{\mathsf{prf}}, j) + e_{i-1,j},$$

    for some noise $|e_{i-1,j}| \leq (i-1) \cdot \beta$.

  On input two keys $\mathsf{k}_{i-1}, \mathsf{k}_i$, and ciphertext header $\hat{\mathsf{ct}}_{i-1}$, the $\mathsf{ReKeyGen}$ algorithm proceeds as follows:

  1. It decrypts $(\mathsf{k}'_{\mathsf{prf},i-1}, \mathsf{h}') \leftarrow \mathsf{AE.Decrypt}(\mathsf{k}_{i-1}, \hat{\mathsf{ct}}_{i-1})$.
  2. It samples a new PRF key $\mathsf{k}_{\mathsf{prf},i} \xleftarrow{\text{R}} \mathcal{K}_{\mathsf{PRF}}$ and defines $\mathsf{k}^{\mathsf{up}}_{\mathsf{prf}} \leftarrow \mathsf{k}_{\mathsf{prf},i} - \mathsf{k}'_{\mathsf{prf},i-1}$.
  3. It sets $\hat{\mathsf{ct}}'_i \leftarrow \mathsf{AE.Encrypt}\big(\mathsf{k}_i, (\mathsf{k}_{\mathsf{prf},i}, \mathsf{h}')\big)$ and $\Delta_{i-1,i} \leftarrow (\hat{\mathsf{ct}}'_i, \mathsf{k}^{\mathsf{up}}_{\mathsf{prf}})$.

  By the correctness of Π_{AE} (Definition 2.7), we have $\mathsf{k}'_{\mathsf{prf},i-1} = \mathsf{k}_{\mathsf{prf},i-1}$ and $\mathsf{h}' = \mathsf{h}$. Therefore, we have $\hat{\mathsf{ct}}'_i = \mathsf{AE.Encrypt}\big(\mathsf{k}_i, (\mathsf{k}_{\mathsf{prf},i}, \mathsf{h})\big)$ and $\mathsf{k}^{\mathsf{up}}_{\mathsf{prf}} = \mathsf{k}_{\mathsf{prf},i} - \mathsf{k}_{\mathsf{prf},i-1}$.

  Now, on input $\Delta_{i-1,i} = (\hat{\mathsf{ct}}_i, \mathsf{k}^{\mathsf{up}}_{\mathsf{prf}})$ and ciphertext $(\hat{\mathsf{ct}}_{i-1}, \mathsf{ct}_{i-1})$, the $\mathsf{ReEncrypt}$ algorithm proceeds as follows:

  1. It sets $\hat{\mathsf{ct}}_i \leftarrow \hat{\mathsf{ct}}'_i = \mathsf{AE.Encrypt}\big(\mathsf{k}_i, (\mathsf{k}_{\mathsf{prf},i}, \mathsf{h})\big)$.
  2. For $\mathsf{ct}_i = (\mathsf{ct}_{i,1}, \ldots, \mathsf{ct}_{i,\ell})$, it sets

     $$\begin{aligned}\mathsf{ct}_{i,j} \leftarrow \mathsf{ct}_{i-1,j} + F(\mathsf{k}^{\mathsf{up}}_{\mathsf{prf}}, j) &= \Big(\mathsf{m}_i + F(\mathsf{k}_{\mathsf{prf},i-1}, j) + e_{i-1,j}\Big) + \Big(F(\mathsf{k}_{\mathsf{prf},i} - \mathsf{k}_{\mathsf{prf},i-1}, j)\Big) \\ &= \mathsf{m}_i + F(\mathsf{k}_{\mathsf{prf},i}, j) + e_{i-1,j} + e'_{i,j},\end{aligned}$$

     for error $|e'_{i,j}| \leq \beta$ for $j = 1, \ldots, \ell$. This means that $|e_{i-1,j} + e'_{i,j}| \leq i \cdot \beta$.

Correctness now follows.

F.3 Proof of Confidentiality

We proceed via a sequence of hybrid experiments that are defined as follows:

- Hyb_0: This hybrid experiment corresponds to the real updatable authenticated encryption confidentiality experiment $\mathsf{Expt}^{\mathsf{conf}}_{\Pi_{\mathsf{UAE}}}(\lambda, h, d, \mathcal{A}, 0)$ that is instantiated with Construction 5.2.

- Hyb_1: In this hybrid experiment, we introduce an abort condition to the challenger's simulation. Namely, throughout the query phase of the experiment, the challenger maintains an additional look-up table $\mathsf{T}_{\mathsf{header}}$ that keeps track of all of the "well-formed" ciphertext headers under honest keys that $\mathcal{A}$ receives from the challenger throughout the experiment. The table is initially set empty and the challenger answers each of $\mathcal{A}$'s oracle queries as follows:

  - $\mathcal{O}_{\mathsf{Encrypt}}(i, \mathsf{m})$: The challenger answers the oracle exactly as in Hyb_0 by generating the ciphertext header
  $$\hat{\mathsf{ct}} \leftarrow \mathsf{AE.Encrypt}\big(\mathsf{k}_i, (\mathsf{k}_{\mathsf{prf}}, \mathsf{h})\big),$$
  and the ciphertext body $\mathsf{ct} = (\mathsf{ct}_1, \ldots, \mathsf{ct}_\ell)$ according to the specification of $\mathsf{Encrypt}(\mathsf{k}_i, \mathsf{m})$. Furthermore, after returning the ciphertext $(\hat{\mathsf{ct}}, \mathsf{ct})$ to $\mathcal{A}$, it additionally adds the mapping $\mathsf{T}_{\mathsf{header}}[i, \hat{\mathsf{ct}}] \leftarrow (\mathsf{k}_{\mathsf{prf}}, \mathsf{h})$ to the table.
  - $\mathcal{O}_{\mathsf{ReKeyGen}}(i, j, \hat{\mathsf{ct}})$: If $i \leq h$, then the challenger checks if $\mathsf{T}_{\mathsf{header}}[i, \hat{\mathsf{ct}}] = \perp$ and outputs $\perp$ if this is the case. If $\mathsf{T}_{\mathsf{header}}[i, \hat{\mathsf{ct}}] \neq \perp$ or $i > h$, then the challenger proceeds exactly as in Hyb_0 by computing $\hat{\mathsf{ct}}' \leftarrow \mathsf{AE.Encrypt}\big(\mathsf{k}_j, (\mathsf{k}'_{\mathsf{prf}}, \mathsf{h})\big)$, and returning $\Delta_{i,j,\hat{\mathsf{ct}}} \leftarrow (\hat{\mathsf{ct}}', \mathsf{k}^{\mathsf{up}}_{\mathsf{prf}})$ to $\mathcal{A}$. Furthermore, if $j \leq h$, then it adds the mapping $\mathsf{T}_{\mathsf{header}}[j, \hat{\mathsf{ct}}'] \leftarrow (\mathsf{k}'_{\mathsf{prf}}, \mathsf{h})$ to the table.
  - $\mathcal{O}_{\mathsf{ReEncrypt}}\big(i, j, (\hat{\mathsf{ct}}, \mathsf{ct})\big)$: If $i \leq h$, then the challenger checks if $\mathsf{T}_{\mathsf{header}}[i, \hat{\mathsf{ct}}] = \perp$ and outputs $\perp$ if this is the case. If $\mathsf{T}_{\mathsf{header}}[i, \hat{\mathsf{ct}}] \neq \perp$ or $i > h$, then the challenger proceeds exactly as in Hyb_0 by computing $\hat{\mathsf{ct}}' \leftarrow \mathsf{AE.Encrypt}\big(\mathsf{k}_j, (\mathsf{k}'_{\mathsf{prf}}, \mathsf{h})\big)$, setting $\Delta_{i,j,\hat{\mathsf{ct}}} \leftarrow (\hat{\mathsf{ct}}', \mathsf{k}^{\mathsf{up}}_{\mathsf{prf}})$, and returning $\mathsf{ReEncrypt}\big(\Delta_{i,j,\hat{\mathsf{ct}}}, (\hat{\mathsf{ct}}, \mathsf{ct})\big)$ to $\mathcal{A}$. Furthermore, if $j \leq h$, then it adds the mapping $\mathsf{T}_{\mathsf{header}}[j, \hat{\mathsf{ct}}'] \leftarrow (\mathsf{k}'_{\mathsf{prf}}, \mathsf{h})$ to the table.
  - $\mathcal{O}_{\mathsf{Challenge}}(i, j, \mathsf{m}, (\hat{\mathsf{ct}}, \mathsf{ct}))$: If $i \leq h$, then the challenger checks if $\mathsf{T}_{\mathsf{header}}[i, \hat{\mathsf{ct}}] = \perp$ and outputs $\perp$ if this is the case. If $\mathsf{T}_{\mathsf{header}}[i, \hat{\mathsf{ct}}] \neq \perp$ or $i > h$, then the challenger answers the oracle query exactly as in Hyb_0 by generating the ciphertext header
  $$\hat{\mathsf{ct}}' \leftarrow \mathsf{AE.Encrypt}\big(\mathsf{k}_j, (\mathsf{k}_{\mathsf{prf}}, \mathsf{h})\big),$$
  and the ciphertext body $\mathsf{ct}' = (\mathsf{ct}'_1, \ldots, \mathsf{ct}'_\ell)$ according to the specification of $\mathsf{Encrypt}(\mathsf{k}_j, \mathsf{m})$. After returning the ciphertext $(\hat{\mathsf{ct}}', \mathsf{ct}')$ to $\mathcal{A}$, it adds the mapping $\mathsf{T}_{\mathsf{header}}[j, \hat{\mathsf{ct}}'] \leftarrow (\mathsf{k}_{\mathsf{prf}}, \mathsf{h})$ to the table.

  The rest of the experiment remains unchanged from Hyb_0.

  In Lemma F.1 below, we show that the hybrid experiments Hyb_0 and Hyb_1 are computationally indistinguishable assuming that Π_{AE} satisfies integrity.

- Hyb_2: In this hybrid experiment, we erase the decryption algorithm $\mathsf{AE.Decrypt}$ from the challenger's simulation. Namely, the challenger answers $\mathcal{A}$'s oracle queries exactly as in Hyb_1. However, for the re-encryption key generation, re-encryption, and challenge oracle queries where the input index $i \leq h$, the challenger works as follows:

  - $\mathcal{O}_{\mathsf{ReKeyGen}}(i, j, \hat{\mathsf{ct}})$: The challenger answers the oracle exactly as in Hyb_1, but instead of decrypting the ciphertext header $\mu \leftarrow \mathsf{AE.Decrypt}(\mathsf{k}_i, \hat{\mathsf{ct}})$, it sets $\mu \leftarrow \mathsf{T}_{\mathsf{header}}[i, \hat{\mathsf{ct}}]$. If no such entry exists in $\mathsf{T}_{\mathsf{header}}$, then it aborts the experiment and outputs $\perp$.
  - $\mathcal{O}_{\mathsf{ReEncrypt}}\big(i, j, (\hat{\mathsf{ct}}, \mathsf{ct})\big)$: The challenger answers the oracle exactly as in Hyb_1, but instead of decrypting the ciphertext header $\mu \leftarrow \mathsf{AE.Decrypt}(\mathsf{k}_i, \hat{\mathsf{ct}})$, it sets $\mu \leftarrow \mathsf{T}_{\mathsf{header}}[i, \hat{\mathsf{ct}}]$. If no such entry exists in $\mathsf{T}_{\mathsf{header}}$, then it aborts the experiment and outputs $\perp$.

  - $\mathcal{O}_{\mathsf{Challenge}}(i, j, \mathsf{m}, (\hat{\mathsf{ct}}, \mathsf{ct}))$: The challenger answers the oracle exactly as in Hyb_1, but whenever it must compute $\mu \leftarrow \mathsf{AE.Decrypt}(\mathsf{k}_i, \hat{\mathsf{ct}})$ in the call to $\mathsf{ReKeyGen}$, it sets $\mu \leftarrow \mathsf{T}_{\mathsf{header}}[i, \hat{\mathsf{ct}}]$. If no such entry exists in $\mathsf{T}_{\mathsf{header}}$, then it aborts the experiment and outputs $\perp$.

  The rest of the experiment remains unchanged from Hyb_1.

  In Lemma F.2 below, we show that the hybrid experiments Hyb_1 and Hyb_2 are perfectly indistinguishable assuming that Π_{AE} is correct.

- Hyb_3: In this hybrid experiment, we erase the PRF keys $\mathsf{k}_{\mathsf{prf}}$ and any information about the plaintext m from the ciphertext headers. Namely, the challenger answers each of $\mathcal{A}$'s oracle queries as follows:

  - $\mathcal{O}_{\mathsf{Encrypt}}(i, \mathsf{m})$: The challenger answers the oracle query exactly as in Hyb_2, but instead of setting the ciphertext header $\hat{\mathsf{ct}} \leftarrow \mathsf{AE.Encrypt}\big(\mathsf{k}_i, (\mathsf{k}_{\mathsf{prf}}, \mathsf{h})\big)$, it sets

  $$\hat{\mathsf{ct}} \leftarrow \mathsf{AE.Encrypt}\big(\mathsf{k}_i, (0^{|\mathsf{k}_{\mathsf{prf}}|}, 0^{|\mathsf{h}|})\big).$$

  The rest of the simulation in answering $\mathcal{A}$'s queries remains unchanged.

  - $\mathcal{O}_{\mathsf{ReKeyGen}}(i, j, \hat{\mathsf{ct}})$: If $j > h$, then the challenger answers the oracle query exactly as in Hyb_2. If $j \leq h$, then instead of setting the new ciphertext header $\hat{\mathsf{ct}}' \leftarrow \mathsf{AE.Encrypt}\big(\mathsf{k}_2, (\mathsf{k}'_{\mathsf{prf}}, \mathsf{h})\big)$, it sets

  $$\hat{\mathsf{ct}}' \leftarrow \mathsf{AE.Encrypt}\big(\mathsf{k}_2, (0^{|\mathsf{k}'_{\mathsf{prf}}|}, 0^{|\mathsf{h}|})\big).$$

  - $\mathcal{O}_{\mathsf{ReEncrypt}}\big(i, j, (\hat{\mathsf{ct}}, \mathsf{ct})\big)$: If $j > h$, then the challenger answers the oracle query exactly as in Hyb_2. If $j \leq h$, then instead of setting the new ciphertext header $\hat{\mathsf{ct}}' \leftarrow \mathsf{AE.Encrypt}\big(\mathsf{k}_2, (\mathsf{k}'_{\mathsf{prf}}, \mathsf{h})\big)$, it sets

  $$\hat{\mathsf{ct}}' \leftarrow \mathsf{AE.Encrypt}\big(\mathsf{k}_2, (0^{|\mathsf{k}'_{\mathsf{prf}}|}, 0^{|\mathsf{h}|})\big).$$

  - $\mathcal{O}_{\mathsf{Challenge}}(i, j, \mathsf{m}, (\hat{\mathsf{ct}}, \mathsf{ct}))$: The challenger answers the oracle query exactly as in Hyb_2, but instead of setting the ciphertext header $\hat{\mathsf{ct}}' \leftarrow \mathsf{AE.Encrypt}\big(\mathsf{k}_j, (\mathsf{k}_{\mathsf{prf}}, \mathsf{h})\big)$, it sets

  $$\hat{\mathsf{ct}}' \leftarrow \mathsf{AE.Encrypt}\big(\mathsf{k}_j, (0^{|\mathsf{k}_{\mathsf{prf}}|}, 0^{|\mathsf{h}|})\big).$$

  The rest of the simulation in answering $\mathcal{A}$'s queries remain unchanged.

  The rest of the experiment remains unchanged from Hyb_2.

  In Lemma F.3 below, we show that the hybrid experiments Hyb_2 and Hyb_3 are computationally indistinguishable assuming that Π_{AE} satisfies confidentiality.

- Hyb_4: In this hybrid experiment, we replace the PRF F with a completely random function. Namely, when answering $\mathcal{A}$'s challenge oracle queries $\mathcal{O}_{\mathsf{Challenge}}(i, j, \mathsf{m}, (\hat{\mathsf{ct}}, \mathsf{ct}))$, instead of evaluating the PRFs $F(\mathsf{k}_{\mathsf{prf}}, \cdot)$ and $F(\mathsf{k}^{\mathsf{up}}_{\mathsf{prf}}, \cdot)$, it uses a random function $f_{\mathsf{k}_{\mathsf{prf}}}(\cdot)$ or $f_{\mathsf{k}^{\mathsf{up}}_{\mathsf{prf}}}(\cdot)$ instead (for the PRF output that affects the returned ciphertext). The rest of the experiment remains unchanged from Hyb_3.

  In Lemma F.5 below, we show that the hybrid experiments Hyb_3 and Hyb_4 are computationally indistinguishable assuming that $F : \mathcal{K}_{\mathsf{PRF}} \times \{0,1\}^* \rightarrow \mathcal{Y}$ is a secure PRF.

- Hyb_5: In this hybrid experiment, we modify the challenger from directly encrypting the message m to *re-encrypting* the ciphertext $(\hat{\mathsf{ct}}, \mathsf{ct})$ when answering $\mathcal{A}$'s challenge query $\mathcal{O}_{\mathsf{Challenge}}\big(i, j, \mathsf{m}, (\hat{\mathsf{ct}}, \mathsf{ct})\big)$. Namely, on a query $\mathcal{O}_{\mathsf{Challenge}}\big(i, j, \mathsf{m}, (\hat{\mathsf{ct}}, \mathsf{ct})\big)$, instead of computing $(\mathsf{m}_1, \ldots, \mathsf{m}_\ell) \leftarrow \mathsf{Pad}(\mathsf{m})$ and setting $\mathsf{ct}'_i \leftarrow \mathsf{m}_i + f_{\mathsf{k}_{\mathsf{prf}}}(i)$ for $i \in [\ell]$, the challenger proceeds as follows:

  1. It computes $\Delta_{i,j,\hat{\mathsf{ct}}} \leftarrow \mathsf{ReKeyGen}(\mathsf{k}_i, \mathsf{k}_j, \hat{\mathsf{ct}})$,
  2. It parses $\Delta_{i,j,\hat{\mathsf{ct}}} = (\hat{\mathsf{ct}}', \mathsf{k}^{\mathsf{up}}_{\mathsf{prf}})$, $\mathsf{ct} = (\mathsf{ct}_1, \ldots, \mathsf{ct}_\ell)$,

3. It sets $\mathsf{ct}'_i \leftarrow \mathsf{ct}_i + f_{\mathsf{k}_{\mathsf{prf}}^{\mathsf{up}}}(i)$ for $i \in [\ell]$.

The rest of the experiment remains unchanged from Hyb_3.

In Lemma F.7 below, we show that the hybrid experiments Hyb_4 and Hyb_5 are perfectly indistinguishable.

- Hyb_6: Starting from this hybrid, we start unrolling back the changes that we made from Hyb_0. In this hybrid experiment, we undo the changes that we made in Hyb_4 by replacing the random function evaluations with true PRF evaluations.

  In Lemma F.8 below, we show that the hybrid experiments Hyb_5 and Hyb_6 are computationally indistinguishable assuming that $F : \mathcal{K}_{\mathsf{PRF}} \times \{0,1\}^* \to \mathcal{Y}$ is a secure PRF.

- Hyb_7: In this hybrid experiment, we undo the changes that we made in Hyb_3 by including the PRF keys $\mathsf{k}_{\mathsf{prf}}$ and information about the plaintext m in the ciphertext headers.

  In Lemma F.9 below, we show that the hybrid experiments Hyb_6 and Hyb_7 are computationally indistinguishable assuming that Π_{AE} satisfies confidentiality.

- Hyb_8: In this hybrid experiment, we undo the changes that we made in Hyb_2 by re-introducing the decryption algorithm $\mathsf{AE.Decrypt}$ in the challenger's simulation.

  In Lemma F.10 below, we show that the hybrid experiments Hyb_7 and Hyb_8 are perfectly indistinguishable assuming that Π_{AE} is correct.

- Hyb_9: In this hybrid experiment, we undo the changes that we made in Hyb_1 by removing the additional abort condition.

  In Lemma F.11 below, we show that the hybrid experiments Hyb_8 and Hyb_9 are computationally indistinguishable assuming that Π_{AE} satisfies integrity.

  This hybrid experiment corresponds to the real updatable authenticated encryption confidentiality experiment $\mathsf{Expt}^{\mathsf{conf}}_{\Pi_{\mathsf{UAE}}}(\lambda, h, d, \mathcal{A}, 1)$.

We now show that each of the consecutive hybrid experiments are indistinguishable. For a hybrid experiment Hyb and an adversary $\mathcal{A}$, we use $\mathsf{Hyb}(\mathcal{A})$ to denote the random variable representing the output of experiment Hyb with adversary $\mathcal{A}$.

Lemma F.1. *Suppose that Π_{AE} satisfies $\varepsilon^{\mathsf{int}}_{\mathsf{ae}}$-integrity (Definition 2.9). Then, for all efficient adversaries $\mathcal{A}$, we have*

$$\big|\Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1]\big| \le h \cdot \varepsilon^{\mathsf{int}}_{\mathsf{ae}}(\lambda).$$

Proof. Let $\mathcal{A}$ be an adversary that distinguishes experiments Hyb_0 and Hyb_1. We construct an algorithm $\mathcal{B}$ that uses $\mathcal{A}$ to break the integrity of Π_{AE} (Definition 2.9). Algorithm $\mathcal{B}$ works as follows:

- **Setup phase**: At the start of the experiment, algorithm $\mathcal{B}$ samples a random index $i^* \xleftarrow{\mathrm{R}} [h]$. It generates the keys k_i for $i \in [h+d] \setminus \{i^*\}$ according to the (identical) specifications of Hyb_0 and Hyb_1. For k_{i^*}, algorithm $\mathcal{B}$ leaves it unspecified.

- **Query phase**: Algorithm $\mathcal{B}$ simulates the responses to $\mathcal{A}$'s oracle queries as follows:

  - $\mathcal{O}_{\mathsf{Encrypt}}(i, \mathsf{m})$: If $i \neq i^*$, algorithm $\mathcal{B}$ proceeds according to Hyb_0. If $i = i^*$, it proceeds according to the specification in Hyb_1. Whenever $\mathcal{B}$ must use k_{i^*} to generate the ciphertext header $\hat{\mathsf{ct}} \leftarrow \mathsf{AE.Encrypt}\big(\mathsf{k}_{i^*}, (\mathsf{k}_{\mathsf{prf}}, \mathsf{h})\big)$, it uses the encryption oracle $\mathcal{O}_{\mathsf{k}_i^*}(\cdot)$ for Π_{AE}.
  - $\mathcal{O}_{\mathsf{ReKeyGen}}(i, j, \hat{\mathsf{ct}})$: If $i \neq i^*$, then $\mathcal{B}$ proceeds according to the specification in Hyb_0. If $i = i^*$, then it proceeds according to the specification in Hyb_1. In both cases, whenever $\mathcal{B}$ must use k_{i^*} to generate the ciphertext header $\hat{\mathsf{ct}}' \leftarrow \mathsf{AE.Encrypt}\big(\mathsf{k}_{i^*}, (\mathsf{k}'_{\mathsf{prf}}, \mathsf{h})\big)$, it uses the encryption oracle $\mathcal{O}_{\mathsf{k}_i^*}(\cdot)$ for Π_{AE} (Definition 2.9). If $i = i^*$ and $\mathcal{B}$ must abort, it submits $\hat{\mathsf{ct}}$ as a forgery for Π_{AE}.

  - $\mathcal{O}_{\mathsf{ReEncrypt}}\big(i, j, (\hat{\mathsf{ct}}, \mathsf{ct})\big)$: If $i = i^*$, algorithm $\mathcal{B}$ proceeds according to the specification in Hyb_1. Otherwise, it proceeds according to Hyb_0. In both cases, whenever $\mathcal{B}$ must use k_{i^*} to generate the ciphertext header $\hat{\mathsf{ct}}' \leftarrow \mathsf{AE.Encrypt}\big(\mathsf{k}_{i^*}, (\mathsf{k}'_{\mathsf{prf}}, \mathsf{h})\big)$, it uses the encryption oracle $\mathcal{O}_{\mathsf{k}^*_i}(\cdot)$ for Π_{AE} (Definition 2.9). If $i = i^*$ and $\mathcal{B}$ must abort, it submits $\hat{\mathsf{ct}}$ as a forgery for Π_{AE}.
  - $\mathcal{O}_{\mathsf{Challenge}}(i, j, \mathsf{m}, (\hat{\mathsf{ct}}, \mathsf{ct}))$: If $i = i^*$, algorithm $\mathcal{B}$ proceeds according to the specification in Hyb_1. Otherwise, it proceeds according to Hyb_0. In both cases, whenever $\mathcal{B}$ must use k_{i^*} to generate the ciphertext header $\hat{\mathsf{ct}}' \leftarrow \mathsf{AE.Encrypt}\big(\mathsf{k}_{i^*}, (\mathsf{k}'_{\mathsf{prf}}, \mathsf{h})\big)$, it uses the encryption oracle $\mathcal{O}_{\mathsf{k}^*_i}(\cdot)$ for Π_{AE} (Definition 2.9). If $i = i^*$ and $\mathcal{B}$ must abort, then it submits $\hat{\mathsf{ct}}$ as a forgery for Π_{AE}.

- **Output phase**: At the end of the experiment, adversary $\mathcal{A}$ outputs a bit $b \in \{0, 1\}$, which algorithm $\mathcal{B}$ returns as the output of the experiment.

By definition, the only difference between the hybrid experiments Hyb_0 and Hyb_1 is the additional abort condition when the challenger answers an adversary's re-encryption key generation, re-encryption, or challenge queries in Hyb_1. Therefore, by definition, algorithm $\mathcal{B}$ perfectly simulates $\mathcal{A}$'s views of the experiments Hyb_0 and Hyb_1 modulo the abort conditions. Furthermore, by the specification of $\mathcal{B}$, if $\mathcal{A}$ forces $\mathcal{B}$ to abort in any of these queries, then $\mathcal{B}$ successfully forges a new ciphertext for Π_{AE}.

To formally analyze the probability that $\mathcal{B}$ successfully forges a new ciphertext, let us define the following set of random variables:

- Let Z denote the event that $\mathcal{B}$ successfully forges a ciphertext at the end of the simulation above.
- Let X_i for $i \in [h]$ denote the event that $i = i^*$ during $\mathcal{B}$'s simulation above.
- Let Y_i for $i \in [h]$ denote the event that adversary $\mathcal{A}$ forces the challenger to abort in Hyb_1 by submitting a query $\mathcal{O}_{\mathsf{ReKeyGen}}(i, j, \hat{\mathsf{ct}})$, $\mathcal{O}_{\mathsf{ReEncrypt}}\big(i, j, (\hat{\mathsf{ct}}, \mathsf{ct})\big)$, or $\mathcal{O}_{\mathsf{Challenge}}(i, j, \mathsf{m}, (\hat{\mathsf{ct}}, \mathsf{ct}))$.

Then, by definition, algorithm $\mathcal{B}$ successfully forges a new authenticated encryption ciphertext when $\mathcal{A}$ forces algorithm $\mathcal{B}$ to abort on a query $\mathcal{O}_{\mathsf{ReKeyGen}}(i^*, j, \hat{\mathsf{ct}})$, $\mathcal{O}_{\mathsf{ReEncrypt}}\big(i^*, j, (\hat{\mathsf{ct}}, \mathsf{ct})\big)$, or $\mathcal{O}_{\mathsf{Challenge}}(i^*, j, \mathsf{m}, (\hat{\mathsf{ct}}, \mathsf{ct}))$:

$$\begin{aligned}
\Pr\big[Z\big] &= \sum_{i\in[h]} \Pr\big[X_i \cap Y_i\big] \\
&= \sum_{i\in[h]} \Pr\big[X_i \mid Y_i\big] \cdot \Pr\big[Y_i\big] \\
&= \sum_{i\in[h]} \frac{1}{h} \cdot \Pr\big[Y_i\big] \\
&= \frac{1}{h} \sum_{i\in[h]} \Pr\big[Y_i\big] \\
&\geq \frac{1}{h} \cdot \Pr\big[Y_1 \cup \ldots \cup Y_h\big],
\end{aligned}$$

where the last inequality follows by the union bound. Now, since the only difference between the hybrid experiments Hyb_0 and Hyb_1 is the additional abort condition when the challenger answers $\mathcal{O}_{\mathsf{ReKeyGen}}(i, j, \hat{\mathsf{ct}})$, $\mathcal{O}_{\mathsf{ReEncrypt}}\big(i, j, (\hat{\mathsf{ct}}, \mathsf{ct})\big)$, or $\mathcal{O}_{\mathsf{Challenge}}(i, j, \mathsf{m}, (\hat{\mathsf{ct}}, \mathsf{ct}))$, an adversary's advantage in distinguishing the two experiments is bounded by the probability of the event $Y_1 \cup \ldots \cup Y_h$:

$$\big|\Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1]\big| \leq \Pr\big[Y_1 \cup \ldots \cup Y_h\big].$$

Putting the two inequalities together, we have

$$\begin{aligned}
\big|\Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1]\big| &\leq \Pr\big[Y_1 \cup \ldots \cup Y_h\big] \\
&\leq h \cdot \Pr\big[Z\big] \\
&\leq h \cdot \varepsilon_{\mathsf{ae}}^{\mathsf{int}}(\lambda),
\end{aligned}$$

and the lemma follows. □

Lemma F.2. *Suppose that* Π_{AE} *is correct (Definition 2.7). Then, for all (unbounded) adversaries* $\mathcal{A}$, *we have*

$$\big|\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1]\big| = 0.$$

Proof. The only difference between the two hybrid experiments is in the way the challenger decrypts the ciphertext headers. For each queries $\mathcal{O}_{\mathsf{ReKeyGen}}(i, j, \hat{\mathsf{ct}})$, $\mathcal{O}_{\mathsf{ReEncrypt}}\big(i, j, (\hat{\mathsf{ct}}, \mathsf{ct})\big)$, and $\mathcal{O}_{\mathsf{Challenge}}\big(i, j, \mathsf{m}, (\hat{\mathsf{ct}}, \mathsf{ct})\big)$, the challenger in Hyb_1 computes $\mu \leftarrow \mathsf{AE.Decrypt}(\mathsf{k}_i, \hat{\mathsf{ct}})$ while the challenger in Hyb_2 sets $\mu \leftarrow \mathsf{T}_{\mathsf{header}}[i, \hat{\mathsf{ct}}]$. The rest of the experiments remains identical.

By the correctness condition for Π_{AE}, these two distributions of μ in the two experiments are identically distributed as long as $(i, \hat{\mathsf{ct}})$ is contained in $\mathsf{T}_{\mathsf{header}}$. However, in both Hyb_1 and Hyb_2, if $\hat{\mathsf{ct}}$ is not contained in $\mathsf{T}_{\mathsf{header}}$, the challenger returns $\perp$. Therefore, the view of $\mathcal{A}$ in the two experiments are identically distributed. ☐

Lemma F.3. *Suppose that* Π_{AE} *satisfies* $\varepsilon_{\mathsf{ae}}^{\mathsf{conf}}$*-confidentiality (Definition 2.8). Then, for all efficient adversaries* $\mathcal{A}$, *we have*

$$\big|\Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1]\big| \leq h \cdot \varepsilon_{\mathsf{ae}}^{\mathsf{conf}}(\lambda).$$

Proof. To prove the lemma, we proceed via a sequence of inner hybrid experiment. For $\gamma = 0, \ldots, h$, we define the hybrid experiments $\mathsf{Hyb}_{2,\gamma}$ as follows:

- $\mathsf{Hyb}_{2,\gamma}$: The challenger proceeds through the setup phase of the experiment according to the specifications in Hyb_2 and Hyb_3 (which are identical). The challenger answers each of $\mathcal{A}$'s queries during the query phase of the experiment as follows:
  - $\mathcal{O}_{\mathsf{Encrypt}}(i, \mathsf{m})$: If $i > \gamma$, then the challenger proceeds as in Hyb_2. Otherwise, the challenger proceeds as in Hyb_3.
  - $\mathcal{O}_{\mathsf{ReKeyGen}}(i, j, \hat{\mathsf{ct}})$: If $i > \gamma$, then the challenger proceeds as in Hyb_2. Otherwise, the challenger proceeds as in Hyb_3.
  - $\mathcal{O}_{\mathsf{ReEncrypt}}\big(i, j, (\hat{\mathsf{ct}}, \mathsf{ct})\big)$: If $i > \gamma$, then the challenger proceeds as in Hyb_2. Otherwise, the challenge proceeds as in Hyb_3.
  - $\mathcal{O}_{\mathsf{Challenge}}(i, j, \mathsf{m}, (\hat{\mathsf{ct}}, \mathsf{ct}))$: If $i > \gamma$, then the challenger proceeds as in Hyb_2. Otherwise, the challenger proceeds as in Hyb_3.

  At the end of the experiment, adversary $\mathcal{A}$ outputs a bit b, which the challenger returns as the output of the experiment.

By definition, experiment $\mathsf{Hyb}_{2,0}$ corresponds to experiment Hyb_2, and experiment $\mathsf{Hyb}_{2,\gamma}$ correponds to experiment Hyb_3. To prove the lemma, we show that each consecutive hybrid experiments $\mathsf{Hyb}_{2,\gamma-1}$ and $\mathsf{Hyb}_{2,\gamma}$ for $\gamma = 1, \ldots, h$ are computationally indistinguishable.

Claim F.4. *Suppose that* Π_{AE} *satisfies* $\varepsilon_{\mathsf{ae}}^{\mathsf{conf}}$*-confidentiality. Then, for all* $\gamma \in [h]$ *and all efficient adversaries* $\mathcal{A}$, *we have*

$$\big|\Pr[\mathsf{Hyb}_{2,\gamma-1}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{3,\gamma}(\mathcal{A})]\big| \leq \varepsilon_{\mathsf{ae}}^{\mathsf{conf}}(\lambda).$$

Proof. Let $\mathcal{A}$ be an adversary that distinguishes experiments $\mathsf{Hyb}_{2,\gamma-1}$ and $\mathsf{Hyb}_{2,\gamma}$. We construct an algorithm $\mathcal{B}$ that uses $\mathcal{A}$ to break the confidentiality of Π_{AE}. Algorithm $\mathcal{B}$ works as follows:

- **Setup phase**: For the setup phase, algorithm $\mathcal{B}$ proceeds according to the specifications in $\mathsf{Hyb}_{2,\gamma-1}$ and $\mathsf{Hyb}_{2,\gamma}$ (which are identical). However, for the key k_γ, it leaves it unspecified.
- **Query phase**: Algorithm $\mathcal{B}$ simulates the responses to $\mathcal{A}$'s oracle queries as follows:

– $\mathcal{O}_{\mathsf{Encrypt}}(i, \mathsf{m})$: Algorithm $\mathcal{B}$ follows the exact specification of the two experiments. However, since k_γ is unspecified, it uses the encryption oracle

$$\mathcal{O}_{\mathsf{k}_\gamma, b}\big((\mathsf{k}_{\mathsf{prf}}, \mathsf{h}), (0^{|\mathsf{k}_{\mathsf{prf}}|}, 0^{|\mathsf{h}|})\big),$$

in place of $\mathsf{AE.Encrypt}\big(\mathsf{k}_\gamma, (\mathsf{k}_{\mathsf{prf}}, \mathsf{h})\big)$ or $\mathsf{AE.Encrypt}\big(\mathsf{k}_\gamma, (0^{|\mathsf{k}_{\mathsf{prf}}|}, 0^{|\mathsf{h}|})\big)$.

– $\mathcal{O}_{\mathsf{ReKeyGen}}(i, j, \hat{\mathsf{ct}})$: Algorithm $\mathcal{B}$ follows the exact specification of the two experiments. However, since k_γ is unspecified, it uses the encryption oracle

$$\mathcal{O}_{\mathsf{k}_\gamma, b}\big((\mathsf{k}'_{\mathsf{prf}}, \mathsf{h}), (0^{|\mathsf{k}'_{\mathsf{prf}}|}, 0^{|\mathsf{h}|})\big),$$

in place of $\mathsf{AE.Encrypt}\big(\mathsf{k}_\gamma, (\mathsf{k}_{\mathsf{prf}}, \mathsf{h})\big)$ or $\mathsf{AE.Encrypt}\big(\mathsf{k}_\gamma, (0^{|\mathsf{k}'_{\mathsf{prf}}|}, 0^{|\mathsf{h}|})\big)$.

– $\mathcal{O}_{\mathsf{ReEncrypt}}\big(i, j, (\hat{\mathsf{ct}}, \mathsf{ct})\big)$: Algorithm $\mathcal{B}$ follows the exact specification of the two experiments. However, since k_γ is unspecified, it uses the encryption oracle

$$\mathcal{O}_{\mathsf{k}_\gamma, b}\big((\mathsf{k}'_{\mathsf{prf}}, \mathsf{h}), (0^{|\mathsf{k}'_{\mathsf{prf}}|}, 0^{|\mathsf{h}|})\big),$$

in place of $\mathsf{AE.Encrypt}\big(\mathsf{k}_\gamma, (\mathsf{k}_{\mathsf{prf}}, \mathsf{h})\big)$ or $\mathsf{AE.Encrypt}\big(\mathsf{k}_\gamma, (0^{|\mathsf{k}'_{\mathsf{prf}}|}, 0^{|\mathsf{h}|})\big)$.

– $\mathcal{O}_{\mathsf{Challenge}}(i, j, \mathsf{m}, (\hat{\mathsf{ct}}, \mathsf{ct}))$: Algorithm $\mathcal{B}$ follows the exact specification of the two experiments. However, since k_γ is unspecified, it uses the encryption oracle

$$\mathcal{O}_{\mathsf{k}_\gamma, b}\big((\mathsf{k}_{\mathsf{prf}}, \mathsf{h}), (0^{|\mathsf{k}_{\mathsf{prf}}|}, 0^{|\mathsf{h}|})\big),$$

in place of $\mathsf{AE.Encrypt}\big(\mathsf{k}_\gamma, (\mathsf{k}_{\mathsf{prf}}, \mathsf{h})\big)$ or $\mathsf{AE.Encrypt}\big(\mathsf{k}_\gamma, (0^{|\mathsf{k}_{\mathsf{prf}}|}, 0^{|\mathsf{h}|})\big)$.

- **Output phase**: At the end of the experiment, adversary $\mathcal{A}$ outputs a bit $b \in \{0, 1\}$, which $\mathcal{B}$ returns as its own output.

By specification, algorithm $\mathcal{B}$ perfectly simulates the experiments $\mathsf{Hyb}_{2,\gamma}$ and $\mathsf{Hyb}_{2,\gamma-1}$ as long as the output of the oracle $\mathcal{O}_{\mathsf{k}_\gamma, b}(\cdot, \cdot)$ is consistent with the specifications of the two experiments. By specification (Definition 2.8), we have

$$\mathcal{O}_{\mathsf{k}_\gamma, 0}\big((\mathsf{k}_{\mathsf{prf}}, \mathsf{h}), (0^{|\mathsf{k}_{\mathsf{prf}}|}, 0^{|\mathsf{h}|})\big) = \mathsf{AE.Encrypt}\big(\mathsf{k}_\gamma, (\mathsf{k}_{\mathsf{prf}}, \mathsf{h})\big),$$

and

$$\mathcal{O}_{\mathsf{k}_\gamma, 1}\big((\mathsf{k}_{\mathsf{prf}}, \mathsf{h}), (0^{|\mathsf{k}_{\mathsf{prf}}|}, 0^{|\mathsf{h}|})\big) = \mathsf{AE.Encrypt}\big(\mathsf{k}_\gamma, (0^{|\mathsf{k}_{\mathsf{prf}}|}, 0^{|\mathsf{h}|})\big).$$

This means that if $\mathcal{B}$ is interacting with the oracle $\mathcal{O}_{\mathsf{k}_\gamma, 0}$, then it perfectly simulates $\mathsf{Hyb}_{2,\gamma-1}$, and if it is interacting with the oracle $\mathcal{O}_{\mathsf{k}_\gamma, 1}$, then it perfectly simulates $\mathsf{Hyb}_{2,\gamma}$. Therefore, with the same distinguishing advantage of the two experiments by $\mathcal{A}$, algorithm $\mathcal{B}$ breaks the confidentiality of Π_{AE}. The claim now follows. ☐

The statement of the lemma now follows from Claim F.4 and the triangle inequality. ☐

Lemma F.5. *Suppose that $F : \mathcal{K}_{\mathsf{PRF}} \times \{0,1\}^* \to \mathcal{Y}$ satisfies $\varepsilon_{\mathsf{prf}}$-security (Definition 2.3). Then, for all efficient adversaries $\mathcal{A}$ that makes Q challenge oracle queries to $\mathcal{O}_{\mathsf{Challenge}}$, we have*

$$\big|\Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_4(\mathcal{A}) = 1]\big| \leq Q \cdot \varepsilon_{\mathsf{prf}}(\lambda).$$

Proof. To prove the lemma, we proceed via a sequence of inner hybrid experiments. For $\gamma = 0, \ldots, Q$, we define the hybrid experiments $\mathsf{Hyb}_{3,\gamma}$ as follows:

- $\mathsf{Hyb}_{3,\gamma}$: The challenger proceeds through the setup phase of the experiments according to the specifications in Hyb_3 and Hyb_4 (which are identical). The challenger also answers each of $\mathcal{A}$'s oracle queries $\mathcal{O}_{\mathsf{ReKeyGen}}$, $\mathcal{O}_{\mathsf{Encrypt}}$, $\mathcal{O}_{\mathsf{ReEncrypt}}$ queries according the specifications in the two experiments. Algorithm $\mathcal{B}$ answers $\mathcal{A}$'s challenge oracle queries as follows:

– $\mathcal{O}_{\mathsf{Challenge}}(i, j, \mathsf{m}, (\hat{\mathsf{ct}}, \mathsf{ct}))$: For $\mathcal{A}$'s first γ queries, the challenger proceeds exactly as in Hyb_3. For the rest of the queries, it proceeds as in Hyb_4.

At the end of the experiment, adversary $\mathcal{A}$ outptus a bit b, which the challenger returns as the output of the experiment.

By definition, experiment $\mathsf{Hyb}_{3,0}$ corresponds to experiment Hyb_3, and experiment $\mathsf{Hyb}_{3,Q}$ corresponds to experiment Hyb_4. The lemma follows by the computational indistinguishability of the consecutive hybrid experiments $\mathsf{Hyb}_{3,\gamma-1}$ and $\mathsf{Hyb}_{3,\gamma}$ for $\gamma = 1, \ldots, Q$. The proof of the following claim follows immediately from the definition of PRF security (Definition 2.3) and the fact that the adversary's view is independent of the PRF key $\mathsf{k}_{\mathsf{prf}}$ or $\mathsf{k}_{\mathsf{prf}}^{\mathsf{up}}$ used in the output of $\mathcal{O}_{\mathsf{Challenge}}(i, j, \mathsf{m}, (\hat{\mathsf{ct}}, \mathsf{ct}))$ as enforced by the checks on the table T in the confidentiality definition.

Claim F.6. *Suppose that $F : \mathcal{K}_{\mathsf{PRF}} \times \{0,1\}^* \to \mathcal{Y}$ satisfies $\varepsilon_{\mathsf{prf}}$-securitry. Then, for all $\gamma \in [Q]$ and all efficient adversaries $\mathcal{A}$ that makes at most Q challenge oracle queries $\mathcal{O}_{\mathsf{Challenge}}$, we have*

$$\big|\Pr[\mathsf{Hyb}_{3,\gamma-1}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{3,\gamma}(\mathcal{A}) = 1]\big| \leq \varepsilon_{\mathsf{prf}}(\lambda).$$

The statement of the lemma now follows from Claim F.6 and the triangle inequality. □

Lemma F.7. *For all (unbounded) adversaries $\mathcal{A}$, we have*

$$\big|\Pr[\mathsf{Hyb}_4(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_5(\mathcal{A}) = 1]\big| = 0.$$

Proof. The only difference between the two hybrid experiments is in the way the challenger responds to $\mathcal{A}$'s challenge oracle queries to $\mathcal{O}_{\mathsf{Challenge}}$. Namely, to generate a ciphertext body ct' on a query $\mathcal{O}_{\mathsf{Challenge}}\big(i, j, \mathsf{m}, (\hat{\mathsf{ct}}, \mathsf{ct})\big)$, the challenger in Hyb_4 computes $(\mathsf{m}_1, \ldots, \mathsf{m}_\ell) \leftarrow \mathsf{Pad}(\mathsf{m})$ and sets

$$\mathsf{ct}'_i \leftarrow \mathsf{m}_i + f_{\mathsf{k}_{\mathsf{prf}}}(i)$$

for $i \in [\ell]$, while the challenger in Hyb_5 computes $\Delta_{i,j,\hat{\mathsf{ct}}} \leftarrow \mathsf{ReKeyGen}(\mathsf{k}_i, \mathsf{k}_j, \hat{\mathsf{ct}})$ and sets

$$\mathsf{ct}'_i \leftarrow \mathsf{ct}_i + f_{\mathsf{k}_{\mathsf{prf}}^{\mathsf{up}}}(i)$$

for $i \in [\ell]$ where $\Delta_{i,j,\hat{\mathsf{ct}}} = (\hat{\mathsf{ct}}', \mathsf{k}_{\mathsf{prf}}^{\mathsf{up}})$, $\mathsf{ct} = (\mathsf{ct}_1, \ldots, \mathsf{ct}_\ell)$. However, since $f_{\mathsf{k}_{\mathsf{prf}}}(\cdot)$ and $f_{\mathsf{k}_{\mathsf{prf}}^{\mathsf{up}}}(\cdot)$ are completely random functions, these two distributions of the ciphertext body components are identically distributed independent of m or ct as long as the resulting ciphertexts have the same length. By specification, the challenger returns $\mathsf{ct}' = (\mathsf{ct}'_1, \ldots, \mathsf{ct}'_\ell)$ only when this is the case. Therefore, the view of $\mathcal{A}$ in Hyb_4 and Hyb_5 are identically distributed and the lemma follows. □

Lemma F.8. *Suppose that $F : \mathcal{K}_{\mathsf{PRF}} \times \{0,1\}^* \to \mathcal{Y}$ satisfies $\varepsilon_{\mathsf{prf}}$-security (Definition 2.3). Then, for all efficient adversaries $\mathcal{A}$ that makes Q challenge oracle queries to $\mathcal{O}_{\mathsf{Challenge}}$, we have*

$$\big|\Pr[\mathsf{Hyb}_5(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_6(\mathcal{A}) = 1]\big| \leq Q \cdot \varepsilon_{\mathsf{prf}}(\lambda).$$

Proof. The proof is identical to the proof of Lemma F.5. □

Lemma F.9. *Suppose that Π_{AE} satisfies $\varepsilon_{\mathsf{ae}}^{\mathsf{conf}}$-confidentiality (Definition 2.8). Then, for all efficient adversaries $\mathcal{A}$, we have*

$$\big|\Pr[\mathsf{Hyb}_6(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_7(\mathcal{A}) = 1]\big| \leq h \cdot \varepsilon_{\mathsf{ae}}^{\mathsf{conf}}(\lambda).$$

Proof. The proof is identical to the proof of Lemma F.3. □

Lemma F.10. *Suppose that Π_{AE} is correct (Definition 2.7). Then, for all (unbounded) adversaries $\mathcal{A}$, we have*

$$\big|\Pr[\mathsf{Hyb}_7(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_8(\mathcal{A}) = 1]\big| = 0.$$

Proof. The proof is identical to the proof of Lemma F.2. ☐

Lemma F.11. *Suppose that* Π_{AE} *satisfies* $\varepsilon_{\mathsf{ae}}^{\mathsf{int}}$*-integrity (Definition 2.9). Then, for all efficient adversaries* $\mathcal{A}$*, we have*

$$\left|\Pr[\mathsf{Hyb}_8(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_9(\mathcal{A}) = 1]\right| \le h \cdot \varepsilon_{\mathsf{ae}}^{\mathsf{int}}(\lambda).$$

Proof. The proof is identical to the proof of Lemma F.1. ☐

By combining the lemmas above and using the triangle inequality, the proof of confidentiality follows.

F.4 Proof of Relaxed Integrity

We proceed via a sequence of hybrid experiments that are defined as follows:

- Hyb_0: This hybrid experiment corresponds to the real updatable authenticated encryption relaxed integrity experiment $\mathsf{Expt}_{\Pi_{\mathsf{UAE}}}^{\mathsf{relaxed\text{-}int}}(\lambda, h, d, \gamma, \mathcal{A})$ that is instantiated with Construction 5.2.
- Hyb_1: In this hybrid experiment, we introduce an additional abort condition to the challenger's simulation. Namely, throughout the experiment, the challenger maintains an additional look-up table $\mathsf{T}_{\mathsf{header}}$ that keeps track of all of the "well-formed" ciphertext headers (under the honest keys) that $\mathcal{A}$ receives from the challenger. Then, it answers each of $\mathcal{A}$'s oracle queries as follows:
  - $\mathcal{O}_{\mathsf{Encrypt}}(i, \mathsf{m})$: The challenger answers the oracle exactly as in Hyb_0 by generating the ciphertext header
  $$\hat{\mathsf{ct}} \leftarrow \mathsf{AE.Encrypt}\big(\mathsf{k}_i, (\mathsf{k}_{\mathsf{prf}}, \mathsf{h})\big),$$
  and the ciphertext body $\mathsf{ct} = (\mathsf{ct}_1, \ldots, \mathsf{ct}_\ell)$ according to the specification of $\mathsf{Encrypt}(\mathsf{k}_i, \mathsf{m})$. After returning the ciphertext $(\hat{\mathsf{ct}}, \mathsf{ct})$ to $\mathcal{A}$, it adds the mapping $\mathsf{T}_{\mathsf{header}}[i, \hat{\mathsf{ct}}] \leftarrow \big((\mathsf{k}_{\mathsf{prf}}, \mathsf{h}), \mathsf{m}\big)$ to the table if $i \le h$.
  - $\mathcal{O}_{\mathsf{ReKeyGen}}(i, j, \hat{\mathsf{ct}})$: If $i > h$, then the challenger proceeds exactly as in Hyb_0. Otherwise, if $i \le h$, then the challenger first checks if $\mathsf{T}_{\mathsf{header}}[i, \hat{\mathsf{ct}}] = \perp$ and outputs $\perp$ if this is the case. If $\mathsf{T}_{\mathsf{header}}[i, \hat{\mathsf{ct}}] \ne \perp$, then the challenger proceeds as in Hyb_0 by computing $\hat{\mathsf{ct}} \leftarrow \mathsf{AE.Encrypt}\big(\mathsf{k}_j, (\mathsf{k}'_{\mathsf{prf}}, \mathsf{h})\big)$, and returning $\Delta_{i,j,\hat{\mathsf{ct}}} \leftarrow (\hat{\mathsf{ct}}', \mathsf{k}_{\mathsf{prf}}^{\mathsf{up}})$ to $\mathcal{A}$. Furthermore, if $j \le h$, then it sets $(\mu, \mathsf{m}) \leftarrow \mathsf{T}_{\mathsf{header}}[i, \hat{\mathsf{ct}}]$ and adds the mapping $\mathsf{T}_{\mathsf{header}}[j, \hat{\mathsf{ct}}'] \leftarrow \big((\mathsf{k}'_{\mathsf{prf}}, \mathsf{h}), \mathsf{m}\big)$ to the table.
  - $\mathcal{O}_{\mathsf{ReEncrypt}}\big(i, j, (\hat{\mathsf{ct}}, \mathsf{ct})\big)$: The challenger answers the query as follows:
    - * If $i > h$, then the challenger proceeds exactly as in Hyb_0 by first computing $\mathsf{m} \leftarrow \mathsf{Decrypt}\big(\mathsf{k}_i, (\hat{\mathsf{ct}}, \mathsf{ct})\big)$. If $\mathsf{m} = \perp$, then the challenger returns $\perp$. Otherwise, it computes $\hat{\mathsf{ct}} \leftarrow \mathsf{AE.Encrypt}\big(\mathsf{k}_j, (\mathsf{k}'_{\mathsf{prf}}, \mathsf{h})\big)$, sets $\Delta_{i,j,\hat{\mathsf{ct}}} \leftarrow (\hat{\mathsf{ct}}', \mathsf{k}_{\mathsf{prf}}^{\mathsf{up}})$, and returns $(\hat{\mathsf{ct}}', \mathsf{ct}') \leftarrow \mathsf{ReEncrypt}\big(\Delta_{i,j,\hat{\mathsf{ct}}}, (\hat{\mathsf{ct}}, \mathsf{ct})\big)$ to $\mathcal{A}$. If $j \le h$, then it additionally adds $\mathsf{T}_{\mathsf{header}}[j, \hat{\mathsf{ct}}'] \leftarrow \big((\mathsf{k}'_{\mathsf{prf}}, \mathsf{h}), \mathsf{m}\big)$ to the table.
    - * If $i \le h$, then the challenger first checks if $\mathsf{T}_{\mathsf{header}}[i, \hat{\mathsf{ct}}] = \perp$ and outputs $\perp$ if this is the case. If $\mathsf{T}_{\mathsf{header}}[i, \hat{\mathsf{ct}}] \ne \perp$, then the challenger proceeds as in Hyb_0 by computing $\hat{\mathsf{ct}} \leftarrow \mathsf{AE.Encrypt}\big(\mathsf{k}_j, (\mathsf{k}'_{\mathsf{prf}}, \mathsf{h})\big)$, and returning $(\hat{\mathsf{ct}}', \mathsf{ct}') \leftarrow \big(\mathsf{ReEncrypt}\big(\Delta_{i,j,\hat{\mathsf{ct}}}, (\hat{\mathsf{ct}}, \mathsf{ct})\big)$ to $\mathcal{A}$. If $j \le h$, then it additionally adds $\mathsf{T}_{\mathsf{header}}[j, \hat{\mathsf{ct}}'] \leftarrow \big((\mathsf{k}'_{\mathsf{prf}}, \mathsf{h}), \mathsf{m}\big)$ to the table.

  At the end of the experiment, adversary $\mathcal{A}$ outputs an index $i \le h$ and a ciphertext $(\hat{\mathsf{ct}}, \mathsf{ct})$. As specified in Hyb_0, the challenger computes $\mathsf{m} \leftarrow \mathsf{Decrypt}\big(\mathsf{k}_i, (\hat{\mathsf{ct}}, \mathsf{ct})\big)$ and checks the following conditions, outputting $\perp$ if either is met:
  - $\mathsf{m} = \perp$,
  - For $(\mathsf{ct}', \mathsf{m}') \leftarrow \mathsf{T}[i, \hat{\mathsf{ct}}]$, $\mathsf{ct}' = (\mathsf{ct}'_1, \ldots, \mathsf{ct}'_\ell)$, we have that $\mathsf{m} = \mathsf{m}'$, $\ell = \ell'$, and $\left\|\mathsf{ct}_j - \mathsf{ct}'_j\right\| \le \gamma$ for all $j \in [\ell]$.

In addition, the challenger verifies whether $\mathsf{T}_{\mathsf{header}}[i, \hat{\mathsf{ct}}] = \perp$. If this is the case, then the challenger also outputs $\perp$. If none of the conditions above are met, then the challenger returns 1 as the output of the experiment.

In Lemma F.12 below, we show that the hybrid experiments Hyb_0 and Hyb_1 are computationally indistinguishable assuming that Π_{AE} satisfies integrity.

- Hyb_2: In this hybrid experiment, we erase the decryption algorithm $\mathsf{AE.Decrypt}$ from the challenger's simulation for honest keys. Namely, the challenger answers $\mathcal{A}$'s re-encryption key generation oracle $\mathcal{O}_{\mathsf{ReKeyGen}}$ and re-encryption oracle $\mathcal{O}_{\mathsf{ReEncrypt}}$ as follows:

  – $\mathcal{O}_{\mathsf{ReKeyGen}}(i, j, \hat{\mathsf{ct}})$: The challenger answers the oracle exactly as in Hyb_1, but instead of decrypting the ciphertext header $\mu \leftarrow \mathsf{AE.Decrypt}(\mathsf{k}_i, \hat{\mathsf{ct}})$, it looks up $(\mu, \mathsf{m}) \leftarrow \mathsf{T}[i, \hat{\mathsf{ct}}]$. If no such entry exists in $\mathsf{T}_{\mathsf{header}}$, then it immediately aborts the experiments and outputs $\perp$.

  – $\mathcal{O}_{\mathsf{ReEncrypt}}\big(i, j, (\hat{\mathsf{ct}}, \mathsf{ct})\big)$: The challenger answers the oracle exactly as in Hyb_1, but instead of decrypting the ciphertext header $\mu \leftarrow \mathsf{AE.Decrypt}(\mathsf{k}_i, \hat{\mathsf{ct}})$, it looks up $(\mu, \mathsf{m}) \leftarrow \mathsf{T}_{\mathsf{header}}[i, \hat{\mathsf{ct}}]$. If no such entry exists in $\mathsf{T}_{\mathsf{header}}$, then it immediately aborts the experiments and outputs $\perp$.

  At the end of the experiment, adversary $\mathcal{A}$ outputs an index $i \leq h$ and a ciphertext $(\hat{\mathsf{ct}}, \mathsf{ct})$. As specified in Hyb_1, the challenger computes $\mathsf{m} \leftarrow \mathsf{Decrypt}\big(\mathsf{k}_i, (\hat{\mathsf{ct}}, \mathsf{ct})\big)$ and checks the following conditions, outputting $\perp$ if either is met:

  – $\mathsf{m} = \perp$,
  – For $(\mathsf{ct}', \mathsf{m}') \leftarrow \mathsf{T}[i, \hat{\mathsf{ct}}]$, $\mathsf{ct}' = (\mathsf{ct}'_1, \ldots, \mathsf{ct}'_\ell)$, we have that $\mathsf{m} = \mathsf{m}'$, $\ell = \ell'$, and $\big\|\mathsf{ct}_j - \mathsf{ct}'_j\big\| \leq \gamma$ for all $j \in [\ell]$.

  However, when computing $\mu \leftarrow \mathsf{Decrypt}\big(\mathsf{k}_i, (\hat{\mathsf{ct}}, \mathsf{ct})\big)$, instead of decrypting the header $\mu \leftarrow \mathsf{AE.Decrypt}(\mathsf{k}_i, \hat{\mathsf{ct}})$, it sets $\mu \leftarrow \mathsf{T}_{\mathsf{header}}[i, \hat{\mathsf{ct}}]$. If no such entry exists in $\mathsf{T}_{\mathsf{header}}$, then it aborts the experiment and outputs $\perp$. The rest of the output phase remains unchanged.

  In Lemma F.13 below, we show that the hybrid experiments Hyb_1 and Hyb_2 are perfectly indistinguishable assuming that Π_{AE} satisfies correctness.

  Then in Lemma F.14, we show that an adversary's advantage in forcing the challenger to output 1 in Hyb_2 is negligible assuming that H is collision-resistant.

We now show that each of the consecutive hybrid experiments are indistinguishable. For a hybrid experiment Hyb and an adversary $\mathcal{A}$, we use $\mathsf{Hyb}(\mathcal{A})$ to denote the random variable that represents the output of experiment Hyb with adversary $\mathcal{A}$.

Lemma F.12. *Suppose that Π_{AE} satisfies $\varepsilon^{\mathsf{int}}_{\mathsf{ae}}$-integrity (Definition 2.9). Then, for all efficient adversaries $\mathcal{A}$, we have*

$$\big|\Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1]\big| \leq h \cdot \varepsilon^{\mathsf{int}}_{\mathsf{ae}}(\lambda).$$

Proof. The proof is almost identical to the proof of Lemma F.1 aside from the different set of oracles that are available to the adversary in the integrity game. The only distinction between Hyb_1 that we consider for integrity and Hyb_1 that we consider for the confidentiality security proof in Lemma F.1 is the content of $\mathsf{T}_{\mathsf{header}}$. In Hyb_1 for integrity, the challenger additionally includes a plaintext message m that is encrypted by the ciphertext body of each corresponding ciphertext headers. However, this distinction does not actually impact the security proof as long as there always exists a well-defined plaintext message m for each of $\mathcal{A}$'s oracle queries to $\mathcal{O}_{\mathsf{ReEncrypt}}$. This condition is guaranteed by the *relaxed* integrity security game (Definition C.2) and therefore, the lemma follows by the same argument as in the proof of Lemma F.1. □

Lemma F.13. *Suppose that Π_{AE} is correct (Definition 2.7). Then, for all (unbounded) adversaries $\mathcal{A}$, we have*

$$\big|\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1]\big| = 0.$$

Proof. The proof is identical to the proof of Lemma F.2. □

Lemma F.14. *Suppose that $\mathcal{H}$ is a $\varepsilon_{\mathsf{cr}}$-secure collision resistant hash function. Then, for all efficient adversaries $\mathcal{A}$, we have*

$$\Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1] \leq \varepsilon_{\mathsf{cr}}(\lambda).$$

Proof. Let $\mathcal{A}$ be an adversary that produces a ciphertext forgery in Hyb_2. We construct an algorithm $\mathcal{B}$ that uses $\mathcal{A}$ to produce a collision for the hash function H. Algorithm $\mathcal{B}$ works as follows:

- Algorithm $\mathcal{B}$ simply follows the specification of the challenger in Hyb_2 until $\mathcal{A}$ produces a forgery. Once $\mathcal{A}$ produces a forgery, which consists of an index $i \in [h]$ and a ciphertext $(\hat{\mathsf{ct}}, \mathsf{ct})$, $\mathcal{B}$ verifies if $\mathsf{T}_{\mathsf{header}}[i, \hat{\mathsf{ct}}] = \perp$. If this is the case, then it aborts and returns $\perp$. Otherwise, it decrypts the message $\mathsf{m} \leftarrow \mathsf{Decrypt}\big(\mathsf{k}_i, (\hat{\mathsf{ct}}, \mathsf{ct})\big)$ and also looks up $(\mu, \mathsf{m}') \leftarrow \mathsf{T}_{\mathsf{header}}[i, \hat{\mathsf{ct}}]$. It returns $(\mathsf{m}, \mathsf{m}')$ as the collision for the hash function H.

We now show that if $\mathcal{A}$ successfully produces a valid forgery for Hyb_2, then $\mathcal{B}$'s output $(\mathsf{m}, \mathsf{m}')$ is a valid collision for H: $H(\mathsf{m}) = H(\mathsf{m}')$ and $\mathsf{m} \neq \mathsf{m}'$. If $\mathcal{A}$'s forgery, which consists of an index $i \in [h]$ and a ciphertext $(\hat{\mathsf{ct}}, \mathsf{ct})$, is a valid forgery, then it must be the case that for $\mathsf{m} \leftarrow \mathsf{Decrypt}\big(\mathsf{k}_i, (\hat{\mathsf{ct}}, \mathsf{ct})\big)$, we have

- $\mathsf{m} \neq \perp$,
- For $(\mathsf{ct}', \mathsf{m}') \leftarrow \mathsf{T}[i, \hat{\mathsf{ct}}]$, $\mathsf{ct}' = (\mathsf{ct}'_1, \ldots, \mathsf{ct}'_\ell)$, we have that $\mathsf{m} = \mathsf{m}'$, $\ell = \ell'$, and $\big\|\mathsf{ct}_j - \mathsf{ct}'_j\big\| \leq \gamma$ for all $j \in [\ell]$.

Now, let $\big((\mathsf{k}_{\mathsf{prf}}, \mathsf{h}), \mathsf{m}\big) \leftarrow \mathsf{T}_{\mathsf{header}}[i, \hat{\mathsf{ct}}]$. We first note that by the specification of the challenger in Hyb_2, we have $\mathsf{h} = H(\mathsf{m}')$. Furthermore, by the specification of the $\mathsf{Decrypt}$ algorithm, in order for $\mathsf{m} = \mathsf{Decrypt}\big(\mathsf{k}_i, (\hat{\mathsf{ct}}, \mathsf{ct})\big) \neq \perp$, we must have $\mathsf{m} = H(\mathsf{m})$. Finally, since $\mathsf{T}[i, \hat{\mathsf{ct}}] \neq \mathsf{ct}$, the forged ciphertext $(\hat{\mathsf{ct}}, \mathsf{ct})$ was never output by the challenger during the simulation of the experiment. Since ct is uniquely determined by the underlying plaintext m for a fixed PRF key $\mathsf{k}_{\mathsf{prf}}$, we have $\mathsf{m} \neq \mathsf{m}'$. The lemma follows. □

By combining the lemmas above and using the triangle inequality, the proof of relaxed integrity follows.

G Proof of Theorem 6.3

G.1 Security

Let $\mathcal{A}$ be an adversary that distinguishes the PRF $F(s, x)$ from a truly random function $f \xleftarrow{\text{R}} \mathsf{Funs}[\{0,1\}^\ell, \mathcal{R}_q]$. We construct an algorithm $\mathcal{B}$ that uses $\mathcal{A}$ to break the Ring Learning with Errors problem $\mathsf{RLWE}_{n,q,\chi}$. Algorithm $\mathcal{B}$ proceeds as follows:

- **Setup phase**: At the start of the experiment, algorithm $\mathcal{B}$ initiates a look-up tables T_0 and T_1:
  - T_0 is indexed by PRF inputs $x \in \{0,1\}^\ell$ and stores elements $(a, v) \in \mathcal{R}_q \times \mathcal{R}_q$.
  - T_1 is indexed by elements (s, x) and stores elements $r \in \{0,1\}^\ell$.
- **Query phase**: Algorithm $\mathcal{B}$ answers each of $\mathcal{A}$'s oracle queries as follows:
  - *RO queries to H_0*: For each query $x \in \{0,1\}^\ell$ to H_0, algorithm $\mathcal{B}$ checks if $\mathsf{T}_1[x] = \perp$. If this is the case, then it queries the $\mathsf{RLWE}_{n,q,\chi}$ oracle to receive a sample $(a, v) \in \mathcal{R}_q \times \mathcal{R}_q$, sets $\mathsf{T}_1[x] \leftarrow (a, v)$, and returns a to $\mathcal{A}$. If $\mathsf{T}_1[x] \neq \perp$, then it looks up $(a, v) \leftarrow \mathsf{T}[x]$, and returns a to $\mathcal{A}$.
  - *RO Queries to H_1*: For each query $(s', x) \in \mathcal{R}_q \times \{0,1\}^\ell$, algorithm $\mathcal{B}$ checks if $\mathsf{T}_2[s', x] = \perp$. If this is the case, then it samples a random element $r \xleftarrow{\text{R}} \{0,1\}^\ell$, sets $T[s', x] \leftarrow r$, and returns r. If $\mathsf{T}_2[s', x] \neq \perp$, then it sets $r \leftarrow \mathsf{T}_2[s', x]$, and returns r.
  - *Evaluation queries*: On an evaluation query $x \in \{0,1\}^\ell$, algorithm $\mathcal{B}$ checks if $\mathsf{T}[x] = \perp$. If this is the case, then it queries the $\mathsf{RLWE}_{n,q,\chi}$ oracle to receive a sample $(a, v) \in \mathcal{R}_q \times \mathcal{R}_q$, sets $\mathsf{T}[x] \leftarrow (a, v)$, and returns v to $\mathcal{A}$. If $\mathsf{T}[x] \neq \perp$, then it looks up $(a, v) \leftarrow \mathsf{T}[x]$, and returns v to $\mathcal{A}$.

- **Output phase**: At the end of the experiment, adversary $\mathcal{A}$ outputs a bit $b \in \{0,1\}$. Now, algorithm $\mathcal{B}$ proceeds as follows.
  - For each $(s', x) \in \mathcal{R}_q \times \{0,1\}^\ell$ for which $\mathsf{T}_2[s', x] \neq \perp$, it checks if $\|v - a \cdot s'\| \leq 2B$ for each $(a, v) \in \mathsf{T}_1$. If there exists such entry $(s', x) \in \mathcal{R}_q \times \{0,1\}^\ell$, algorithm $\mathcal{B}$ outputs 0.
  - If no such entry $(s', x) \in \mathcal{R}_q \times \{0,1\}^\ell$ exists, then algorithm $\mathcal{B}$ outputs b that $\mathcal{A}$ returned.

Suppose that $\mathcal{B}$ is interacting with the real RLWE oracle $\mathcal{O}_s^{\mathsf{Real}}$, which provides $(a, a \cdot s + e)$ for $a \xleftarrow{\text{R}} \mathcal{R}_q$ and $e \leftarrow \chi$ to $\mathcal{B}$. Then, by definition, algorithm $\mathcal{B}$ perfectly simulates the responses to all of $\mathcal{A}$'s oracle queries as long as $\mathcal{A}$ never submits (s, x) as a query to H_1 for some $x \in \{0,1\}^\ell$. However, if $\mathcal{A}$ submits such a query to H_1, then algorithm $\mathcal{B}$ uses s to verify whether it is interacting with the real oracle $\mathcal{O}_s^{\mathsf{Real}}$ or $\mathcal{O}^{\mathsf{Ideal}}$ at the output phase of the simulation.

Now, suppose that $\mathcal{B}$ is interacting with the ideal RLWE oracle $\mathcal{O}^{\mathsf{Ideal}}$, which provides $(a, u) \xleftarrow{\text{R}} \mathcal{R}_q \times \mathcal{R}_q$. Then, by definition, algorithm $\mathcal{B}$ perfectly simulates the responses to $\mathcal{A}$'s queries. At the end of the simulation, algorithm $\mathcal{B}$ returns the output of $\mathcal{A}$ as long as $\mathcal{A}$ never made a query (s', x) to H_1 for which s' is a valid $\mathsf{RLWE}_{n,q,\chi}$ secret. However, when $\mathcal{B}$ is interacting with $\mathcal{O}^{\mathsf{Ideal}}$, no such s' can exist. Therefore, algorithm $\mathcal{B}$ distinguishes oracles $\mathcal{O}_s^{\mathsf{Real}}$ and $\mathcal{O}^{\mathsf{Ideal}}$ with at least the distinguishing advantage of $\mathcal{A}$ in distinguishing $F(s, \cdot)$ and $f(\cdot)$.

G.2 Key-Homomorphism

Key-homomorphism of the PRF follows straightforwardly from the following relation:

$$\begin{aligned} F(s_1, x) + F(s_2, x) - F(s_1 + s_2, x) &= a \cdot s_1 + e_1 + a \cdot s_2 + e_2 - a \cdot (s_1 + s_2) + e_3 \\ &= e_1 + e_2 - e_3, \end{aligned}$$

where $e_1 = \mathsf{Samp}\big(H_1(s_1, x)\big)$, $e_2 = \mathsf{Samp}\big(H_1(s_2, x)\big)$, and $e_3 = \mathsf{Samp}\big(H_1(s_1 + s_2, x)\big)$. By definition of Samp, the norm of ring elements e_1, e_2, and e_3 are bounded by B. Therefore, $\|e_1 + e_2 - e_3\| \leq \|e_1\| + \|e_2\| + \|e_3\| \leq 3B$. The statement of the theorem now follows.